

Investigating for bias in healthcare algorithms: a sex-stratified analysis of supervised machine learning models in liver disease prediction

Isabel Straw , Honghan Wu

To cite: Straw I, Wu H. Investigating for bias in healthcare algorithms: a sex-stratified analysis of supervised machine learning models in liver disease prediction. *BMJ Health Care Inform* 2022;**29**:e100457. doi:10.1136/bmjhci-2021-100457

► Additional supplemental material is published online only. To view, please visit the journal online (<http://dx.doi.org/10.1136/bmjhci-2021-100457>).

Received 31 July 2021
Accepted 06 April 2022



© Author(s) (or their employer(s)) 2022. Re-use permitted under CC BY-NC. No commercial re-use. See rights and permissions. Published by BMJ.

Institute of Health Informatics, University College London, London, UK

Correspondence to

Dr Isabel Straw;
isabelstraw@doctors.org.uk

ABSTRACT

Objectives The Indian Liver Patient Dataset (ILPD) is used extensively to create algorithms that predict liver disease. Given the existing research describing demographic inequities in liver disease diagnosis and management, these algorithms require scrutiny for potential biases. We address this overlooked issue by investigating ILPD models for sex bias.

Methods Following our literature review of ILPD papers, the models reported in existing studies are recreated and then interrogated for bias. We define four experiments, training on sex-unbalanced/balanced data, with and without feature selection. We build random forests (RFs), support vector machines (SVMs), Gaussian Naïve Bayes and logistic regression (LR) classifiers, running experiments 100 times, reporting average results with SD.

Results We reproduce published models achieving accuracies of >70% (LR 71.31% (2.37 SD) – SVM 79.40% (2.50 SD)) and demonstrate a previously unobserved performance disparity. Across all classifiers females suffer from a higher false negative rate (FNR). Presently, RF and LR classifiers are reported as the most effective models, yet in our experiments they demonstrate the greatest FNR disparity (RF; –21.02%; LR; –24.07%).

Discussion We demonstrate a sex disparity that exists in published ILPD classifiers. In practice, the higher FNR for females would manifest as increased rates of missed diagnosis for female patients and a consequent lack of appropriate care. Our study demonstrates that evaluating biases in the initial stages of machine learning can provide insights into inequalities in current clinical practice, reveal pathophysiological differences between the male and females, and can mitigate the digitisation of inequalities into algorithmic systems.

Conclusion Our findings are important to medical data scientists, clinicians and policy-makers involved in the implementation of medical artificial intelligence systems. An awareness of the potential biases of these systems is essential in preventing the digital exacerbation of healthcare inequalities.

BACKGROUND

Liver cirrhosis accounts for 1.8% of deaths in Europe, a number which has grown significantly over the past decade as rates of alcohol consumption, chronic hepatitis infections and

Summary

What is already known on this topic

► Machine learning models that leverage biochemical data for modelling patient trajectories are rapidly increasing, yet these algorithms are rarely scrutinised for demographic bias or their impact on health inequalities.

What this study adds

► Our study demonstrates a previously unobserved sex disparity in model performance for algorithms built from a commonly used liver disease dataset. We highlight how biochemical algorithms may reinforce and exacerbate existing healthcare inequalities.

How this study might affect research, practice or policy

► Bias in biochemical algorithms is an overlooked issue. In clinical practice, the higher rate of false negatives for female patients would manifest as an increased rate of missed diagnosis for female patients and a consequent lack of appropriate care.
► Furthermore, sex differences in biochemical feature importance reinforces existing research that suggests unisex biochemical thresholds may disadvantage female patients in current practice. These findings are important to medical data scientists, clinicians and policy-makers involved in the implementation of medical artificial intelligence systems. An awareness of the potential biases of these systems is essential in preventing the digital exacerbation of healthcare inequalities

obesity-related liver disease have increased.¹ Yet, liver disease does not affect all populations equally. Recent research has demonstrated sex differences in the prevalence, diagnosis and management of various hepatic illnesses.^{2–5} A key determinant of patient outcomes from liver disease is the early detection of pathology, yet when it comes to diagnosis and referral, female patients appear to be at a significant disadvantage.^{2–5}

In alcohol related liver disease, Vatsalya *et al* report that women are less likely to be

suspected of alcohol abuse, diagnosed and often experience more severe disease with worse outcomes.^{2 3} Sex differences in diagnosis are compounded by inequalities in the liver disease management. Mathur *et al* report disparities in access to liver transplantation that result in females having markedly lower transplant rates than their male counterparts.⁴ The problem extends beyond hepatology. In 2021, the UK parliamentary report on the gender health gap highlighted that the UK has the largest female health gap in the G20 and the 12th largest globally.⁵ The exclusion of females from research trials (extending to animal research), the neglect of female bodies throughout medical pedagogy and the unconscious biases of practitioners are a few of the intersecting factors that result in worse health outcomes for female patients.^{6–10}

Liver function tests are integral to patient diagnosis and monitoring. These ‘biochemical markers’ include proteins made by the liver (eg, albumin), and enzymes required for metabolism (eg, aspartate aminotransferase (AST)). Bias research has illustrated that biochemical markers are not equally effective for all patient groups.^{3 7 10–12} Suthahar *et al* describe how sex differences in biomarker thresholds affect objectivity in management, as what is considered ‘normal’ in one sex, may not be so in the other.¹² Grimm *et al* investigate the relationship between albumin and mortality, reporting that albumin offers a higher predictive power for males compared with females.¹¹ Furthermore, Vatsalya *et al* and Stepien *et al* describe sex differences in biochemical cut offs, highlighting that the milder expression of liver injury for females may result in female disease going undetected.^{3 13} Such disparities in the predictive potential of clinical biomarkers have the potential to exacerbate healthcare inequalities.^{6 7 10 12}

The rise in healthcare artificial intelligence (AI) has resulted in the increasing use of large clinical datasets for machine learning (ML).¹⁴ ML classifiers that use biochemical markers to model patient trajectories have consistently outperformed traditional statistical models.¹⁴ However, despite the promise of ML tools, the presence of demographic biases in AI algorithms has indicated that historical harms may materialise in digital systems and worsen population inequalities.^{7 15–17} The development of predictive models from biomarkers is one area in which medical ML models are at risk of encoding the errors of current practice. In our paper we explore for this possibility in liver disease prediction by examining models built from a commonly cited dataset: The Indian Liver Patient Dataset (ILPD).

The ILPD is a widely used open-source dataset that provides the biochemical markers of a sample of patients, some of whom have liver disease.^{18–22} BanuPriya and Tamilselvi provide an overview of classification models built from this dataset, since which time further models have been published from both academics and major industry.^{18 19 21} Authors consistently report accuracies of >70% for identifying liver patients, with logistic regression (LR) models and random forests (RFs) giving the

best results. Jin *et al*²³ demonstrate accuracies of 72.7% with LR models, similarly Adil *et al* achieve 74% accuracy with their LR model, outperforming artificial neural networks and support vector machines (SVMs).²⁴ A recent study from Intel reproduces these models and performs additional feature selection giving model accuracies of 74.6% (RF) and 71.2% (SVM).¹⁹

Predictive ML models may benefit patient care if they can diagnose liver disease at an earlier stage.²⁵ Yet, despite the existing literature that describes biases in clinical medicine, biochemical tests and algorithmic performance, none of the ML studies on the ILPD focus on sex disparities in model performance.^{4 7 8 10–12 16 17} We seek to address this gap in the research by investigating the ILPD dataset and its respective models for sex bias.^{18–20}

METHODOLOGY

The ILPD was originally collected from India and consists of 583 patient records, of which 416 have liver disease. We imported the ILPD from the UCI repository (full codebook available in online supplemental material C).^{19 22}

Data exploration and initial analysis

Data exploration is the primary stage of the ML process and involves file importation, formatting, descriptive statistics and configuring datatypes. Online supplemental table 1 gives the variables included in our dataset and their initial datatypes.

Feature exploration

Online supplemental table 2 presents the sex-stratified feature importance ranked by Pearson’s correlation coefficient. For females, the enzymes ALT and AST are ranked fourth and fifth, whereas for males they are ranked seventh and eighth. Further, albumin and A/G ratio are ranked higher for male patients compared with female patients. These subtle differences in feature importance may reflect underlying sex differences in hepatic pathophysiology and biomarker expression.^{3 4 26} Further, online supplemental table 2 demonstrates that the mean IQR across all biomarkers is less for females, suggesting that these biomarkers may have less of a predictive power for female patients overall (mean IQR; female 0.145, male 0.175).

Data preprocessing

Data preparation steps reflected existing studies.^{19 20} Mean imputation was used to address missing values, gender was mapped to a 0/1 numerical datatype, normalisation was performed using minimum-maximum scaler function and the target variable was recoded to binary variable, such that 1 represents diseased patients (n=416).

Addressing class imbalance

The original dataset demonstrated significant class imbalance (167 healthy vs 416) diseased patients) and sex imbalance (142 females vs 441 males). Similarly to existing models, we implement the imblearn SMOTE()

Table 1 Summary counts of classes in the Indian liver patient dataset dataset, including counts after the dataset is balanced

	Target (disease=1)	Dataset 1 (original)	Total counts for sexes	Dataset 2 (oversampled minority class)	Total counts for sexes	Dataset 3 (sex balanced, oversampled females)	Total counts for sexes
Female	0	50	142	145	237	408	595
	1	92		92		187	
Male	0	117	441	271	595	271	595
	1	324		324		324	
Total			583		832		1190

package to address these imbalances; oversampling both the minority class and under-represented females as detailed in table 1.¹⁹ The sex-unbalanced dataset is retained to compare the impact of female representation in the training data on sex disparities in performance.

Model development and implementation

Gulia and Praveen Rani review the classification algorithms that have been built from the ILPD, including RFs and SVMs.²⁰ A more recent review from BanuPriya and Tamilselvi describe the accuracies of additional models including Bayesian Networks, which is further built on by the work of Aswathy who evaluates the performance of LR models on the ILPD.^{18 19} We replicate the methods of these studies, reproducing RF, SVM, Gaussian Naïve Bayes (GNB) and LR classifiers. We implement these models across four experiments, in which we evaluate the overall and sex-stratified performance of the classifiers.

Experiment 1: models trained on unbalanced dataset, without feature selection

Initially, we reproduce existing studies, building a predictive algorithm on the full unbalanced dataset to predict liver disease. Data were divided into test and training subsets (30%/70%), hyperparameters were tuned using GridSearchCV(), the model was trained on the mixed-sex data and results were stratified by sex to give the evaluation metrics for males/females separately. We do this 100 times (building, training and testing separate models) and report average results with SD over the 100 runs. This is done for all four classifiers resulting in four results tables (online supplemental material B Spreadsheets, ‘Experiment 3.1.1—RF’—‘Experiment 3.1.1 GNB’).

Experiment 2: models trained on sex-balanced dataset, without feature selection

The methodology of experiment 1 is repeated using the sex-balanced dataset defined in Table 1. We ensure sex balance in the training data by taking random subsets from the male and females separately, which are appended together to form the full sex-balanced training data for each individual experiment (online supplemental file 3 Spreadsheets, ‘Experiment 3.1.2—RF’—‘Experiment 3.1.2 GNB’).

Experiment 3: models trained on unbalanced dataset, with feature selection

In experiment 3, we perform feature selection based on the unbalanced dataset, in experiment 4, we perform feature selection on the sex-balanced dataset. Feature selection is performed using Recursive Feature Elimination (RFE) sklearn package, which returns the top five ranked features (online supplemental material B Spreadsheets, ‘Experiment 3.1.3—RF’—‘Experiment 3.1.3 GNB’).

Experiment 4: models trained on balanced dataset, with feature selection

Lastly, models and feature selection are fitted to the sex-balanced dataset. Our aim was to investigate whether feature selection would differ once the representation of females was addressed, and whether this would influence any performance disparities.

Model evaluation

Evaluation metrics are reported for all patients and separately for the sexes (equations 1–3). We examine the mean difference between the male and females for each evaluation metric to demonstrate any disparities (equation 4). Two-sample paired t-tests are run on the series of 100 experiments for the male and female patients to assess whether the mean difference between sexes, for each of the evaluation metrics, is statistically significant ($p < 0.05$).

Equation 1: accuracy evaluation metric

Accuracy gives the proportion of correct predictions produced by a model.

$$\text{Accuracy} = \frac{\text{True positives} + \text{True Negatives}}{\text{True positives} + \text{True Negatives} + \text{False Positives} + \text{False Negatives}}$$

Equation 2: F-score evaluation metric, precision and recall

The F-score is the average of precision and recall, with a value of 1 being a perfect score.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{F Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Equation 3: performance error rates

The following error rates are used throughout our evaluation.²¹

- ▶ True positive: Predicted yes and they do have disease.
- ▶ True negative: Predicted no and they do not have disease.
- ▶ False positive: Predicted yes, but they do not have disease.
- ▶ False negative: Predicted no, but they actually do have disease.

$$\text{True Negative Rate (TNR)} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

$$\text{True Positive Rate (TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Equation 4: sex performance disparity

Sex performance disparity = Male evaluation metric (mean) – Female evaluation metric (mean)

RESULTS

We ran 16 experiments: experiments 1–4, with each of the four classifiers. The detailed results tables with the 100 experiment runs are provided in the spreadsheet files in online supplemental material B. In online supplemental material A ‘Tables in Text’, we provide summary in several condensed tables, which give the average evaluation metrics and the statistical significance of any male–female differences.

Results for experiment 1

Online supplemental table 3 demonstrates that our four models reflect the existing literature, achieving accuracies >70% (71.31% (2.37 SD) LR – 79.40% (2.50 SD)

SVM). Table 2 details the disparities for each evaluation metric, from which we observe a statistically significant sex disparity in Accuracy for all classifiers, with mixed results regarding the direction of the disparity (performance disparity –2.98 SVM to 2.96% RF, $p < 0.05$). In the case of the ROC_AUC score, we observe a significant disparity that negatively impacts females for the RF (6.80%, $p < 0.05$), LR (2.93%, $p < 0.05$) and GNB (5.53%, $p < 0.05$) classifiers.

The accuracy and ROC_AUC disparities fluctuate depending on the balance between the different error rates, however, on examining the error rates individually, we see a consistency in error trends for each sex. Across all classifiers females suffer from a higher false negative rate (FNR), while males suffer from a higher false positive rate. The disparity demonstrates a consistently higher recall for males, with females experience a lower recall and correspondingly higher FNR disparity, –2.58% to –24.07%, table 2)

Results for experiment 2

In experiment 2, we trained on sex-balanced data, improving overall accuracy across all four classifiers (RF 81.66% (2.33 SD) vs 78.17 (2.36 SD); LR 74.53% (1.96 SD) vs 71.31% (2.37 SD); SVM 83.30% (1.75 SD) vs 79.40% (2.50 SD); GNB 74.75% (1.9 SD) vs 71.53% (2.61 SD)—online supplemental table 4). We now see a consistent accuracy disparity that benefits females across all four classifiers (–11.47% to –6.17%, $p < 0.05$ —table 3). Disparities in the ROC_AUC scores are less consistent (LR unbalanced ROC disparity 2.93%, LR balanced ROC disparity 4.79%; GNB unbalanced ROC disparity 5.53%, GNB balanced disparity 5.45%).

Table 2 Experiment 3.1.1—unbalanced training data without feature selection, sex performance disparities

Mean difference averaged over n=100	Random forest classifier		Logistic regression classifier		Support vector machine		Gaussian Naïve Bayes	
	Sex performance disparities (%)	t-test p value	Sex performance disparities (%)	t-test p value	Sex performance disparities (%)	t-test p value	Sex performance disparities (%)	t-test p Value
Accuracy	2.96	0.00	–2.85	0.01	–2.98	0.02	–2.72	0.02
FScore	15.63	0.00	15.86	0.00	4.14	0.00	16.19	0.00
ROC_AUC*	6.80	0.00	2.93	0.00	–2.41	0.08	5.53	0.00
Precision	5.25	0.00	–4.87	0.00	3.41	0.00	–3.13	0.05
Recall	21.02	0.00	24.07	0.00	2.58	0.04	19.31	0.00
False negative rate	–21.02	0.00	–24.07	0.00	–2.58	0.08	–19.31	0.00
True negative rate	–7.42	0.00	–18.20	0.00	–7.40	0.00	–8.24	0.00
False positive rate	7.42	0.00	18.20	0.00	7.40	0.00	8.24	0.00
True positive rate	21.02	0.00	24.07	0.00	2.58	0.04	19.31	0.00

*ROC AUC score is a measure of the separation between classes in a binary classifier, derived from the area under the ROC curve.

Table 3 Experiment 3.1.2—balanced training data without feature selection, sex performance disparities

Mean difference averaged over n=100	Random forest classifier		Logistic regression classifier		Support vector machine		Gaussian Naïve Bayes	
	Sex performance disparities (%)	t-test p value	Sex performance disparities (%)	t-test p value	Sex performance disparities (%)	t-test p value	Sex performance disparities (%)	t-test p value
Accuracy	-6.17	0.00	-6.36	0.00	-11.47	0.00	-7.43	0.00
FScore	7.69	0.00	20.17	0.00	-3.40	0.00	16.65	0.00
ROC_AUC	0.60	0.13	4.79	0.00	-9.06	0.00	5.45	0.00
Precision	-0.94	0.88	-4.75	0.00	-2.32	0.14	0.24	0.37
Recall	12.88	0.00	29.22	0.00	-4.64	0.00	19.82	0.00
False negative rate	-12.88	0.00	-29.22	0.00	4.64	0.00	-19.82	0.00
True negative rate	-11.69	0.00	-19.65	0.00	-13.49	0.00	-8.93	0.00
False positive rate	11.69	0.00	19.65	0.00	13.49	0.00	8.93	0.00
True positive rate	12.88	0.00	29.22	0.00	-4.64	0.00	19.82	0.00

Online supplemental table 5 presents a comparison of the evaluation metrics with/without balancing of the training data. In one case, we observe an improvement in performance for all patients. When trained on the balanced dataset, the LR accuracy improves overall (74.53% (1.96 SD) vs 71.31% (2.37 SD)), for females (77.71% (2.42 SD) vs 73.33% (3.95 SD)) and for males (71.35% (3.22 SD) vs 70.49% (2.74 SD)).

Results for experiment 3

We did not see an improvement in overall performance or a reduction in disparities with RFE. A significant ROC_AUC disparity is apparent across all four

classifiers (3.60%–6.61%, $p < 0.05$) that negatively impacts females. We see the same error rate findings as earlier, with a higher FNR for females (FNR Disparity -18.21 to -21.24%, $p < 0.05$, table 4 and online supplemental table 6).

Results for experiment 4

Experiment 4 gives mixed results. The accuracy disparity benefits females across all classifiers (-4.64% to -6.80%, $p < 0.05$), whereas the ROC_AUC disparity demonstrates a benefit for males in three out of four classifiers (-0.05% to 5.95%, $p < 0.05$, table 5) The results relate to the subtle changes in error rates with each model, however, across

Table 4 Experiment 3.1.3—unbalanced training data with feature selection, sex performance disparities

	Random forest classifier		Logistic regression classifier		Support vector machine		Gaussian Naïve Bayes	
	Sex performance disparities (%)	t-test p value	Sex performance disparities (%)	t-test p value	Sex performance disparities (%)	t-test p value	Sex performance disparities (%)	t-test p value
Accuracy	3.42	0.00	-2.90	0.01	-2.75	0.01	-3.31	0.00
FScore	15.36	0.00	15.79	0.00	16.50	0.00	15.29	0.00
ROC_AUC	6.61	0.00	3.60	0.00	4.90	0.00	4.99	0.00
Precision	9.85	0.00	0.24	0.44	-0.87	0.90	-3.41	0.03
Recall	18.21	0.00	21.24	0.00	20.30	0.00	18.54	0.00
False negative rate	-18.21	0.00	-21.24	0.00	-20.30	0.00	-18.54	0.00
True negative rate	-4.99	0.00	-14.04	0.00	-10.50	0.00	-8.57	0.00
False positive rate	4.99	0.00	14.04	0.00	10.50	0.00	8.57	0.00
True positive rate	18.21	0.00	21.24	0.00	20.30	0.00	18.54	0.00

Table 5 Experiment 3.1.4—balanced training data with feature selection, sex performance disparities

	Random forest classifier		Logistic regression classifier		Support vector machine		Gaussian Naïve Bayes	
	Sex performance disparities (%)	t-test p value	Sex performance disparities (%)	t-test p value	Sex performance disparities (%)	t-test p value	Sex performance disparities (%)	t-test p value
Accuracy	-5.62	0.00	-6.80	0.00	-6.19	0.00	-4.64	0.00
FScore	7.86	0.00	14.39	0.00	16.46	0.00	21.63	0.00
ROC_AUC	-0.05%	0.46	3.57%	0.00	5.95%	0.00	8.17%	0.00
Precision	4.60%	0.00	9.28%	0.00	12.82%	0.00	9.35%	0.00
Recall	9.70%	0.00	15.51%	0.00	15.38%	0.00	22.78%	0.00
False negative rate	-9.70	0.00	-15.51	0.00	-15.38	0.00	-22.78	0.00
True negative rate	-9.79	0.00	-8.37	0.00	-3.47	0.00	-6.44	0.00
False positive rate	9.79	0.00	8.37	0.00	3.47	0.00	6.44	0.00
True positive rate	9.70	0.00	15.51	0.00	15.38	0.00	22.78	0.00

all classifiers the FNR is consistently higher for females (-9.70% to -22.78%, $p < 0.05$ (online supplemental table 7)).

Analysis of feature selection

Online supplemental table 8 gives the feature rankings assigned by the RFE model when fitted to unbalanced and balanced data, focusing on RF classifiers. When we address the under-representation of females in the training data, ALP and gender are included as the top two features, while A/G ratio and total bilirubin are removed. This finding may reflect existing research that describes sex differences in biomarker expression. In their analysis gender-specific references intervals for hepatic biomarkers, Li *et al* highlight sex differences in ALP, ALT and GGT, indicating that differing thresholds may be appropriate for diagnosis.²⁷ Sex differences in biochemical disease profiles may explain why integrating more female patients affects the feature selection in experiment 4.

DISCUSSION

In recent years, research has highlighted that medical biases and female under-representation may significantly contribute to differences in healthcare outcomes; in our paper, we have examined how this phenomena may extend into ML.^{6–8 10 28} We present several key findings:

- ▶ **Model reproduction and demonstration of disparity:** We have demonstrated a previously unobserved sex disparity that exists in published ML classifiers based on the ILPD dataset.
- ▶ **Error disparities:** Sex disparities in Accuracy and ROC_AUC fluctuate depending on model and the balance between error rates, however, sex differences

in specific error rates are consistent. We observe a consistently lower recall and correspondingly higher FNR for females. Of note, RF and LR classifiers are reported as the most effective on the ILPD dataset, however, these models demonstrate the greatest disparity in the FNR when trained on the original dataset (RF, FNR disparity -21.02% ($p < 0.05$); LR, FNR disparity -24.07%, ($p < 0.05$)). Clinically, this FNR disparity would materialise as an inequality in disease detection that negatively impacts females, with higher instances of missed disease.

- ▶ **Balanced training:** Training on sex-balanced data improved overall performance for all classifiers. In the case of the LR classifier, accuracy improves overall and for the sexes separately, indicating that with the right model selection addressing poor performance for the under-represented group does not need to come at the expense of the majority group.
- ▶ **Impact of model architecture on disparity:** Our experimental outcomes were not consistent across models, indicating that bias mitigation techniques may need to be tailored to model choice.
- ▶ **Analysis of feature ranking:** Our comparison of feature importance reinforces existing clinical research that highlights the sex differences in the role of liver biomarkers.

Implications for data science

Our experiments demonstrated that sex-specific feature selection and addressing under-representation of females may be an important bias mitigation technique when developing ML algorithms in medicine. Furthermore, we illustrate that there is no consistent solution across all classifiers, suggesting techniques need to be tailored

to model choice. ML models also present novel opportunities for improving existing practice and addressing health disparities that relate to biochemical discrepancies between the sexes. Given the evolving evidence that critiques the use of ‘unisex’ biochemical thresholds, ML models that do not rely on these defined thresholds may pose a superior alternative if developed with an awareness of the subtle sex differences in disease manifestation.

Implications for clinical medicine and public health

Classification algorithms are being increasingly used in healthcare settings to assist clinicians in medical diagnosis.²⁰ Unless these algorithms are evaluated for biases, they may only improve care for a subset of patients and consequently increase healthcare inequalities.⁷ By evaluating ML models for demographic biases before they are implemented in digital medicine, we can mitigate the perpetuation of these inequalities into digital systems.

Furthermore, insights from model development can be used to inform current clinical care. Our data exploration of feature correlation demonstrated sex differences in feature importance. Such research can inform practising clinicians on the relevance of different indicators for the patient in front of them, for example, albumin may be more indicative of pathology in males.¹¹ Lastly, examining disparities in algorithmic performance offers an opportunity to reflect on which patients may be being missed in current practice. Throughout our analysis, we demonstrated a persistently high FNR for females, suggesting that female disease is at risk of being overlooked. Examining the physiological profile of algorithmic false negatives presents an opportunity to better understand which patients are at risk of being misdiagnosed.

It should be noted that the ILPD does not include demographic information on race or ethnicity.²² Racial biases have been reported in the biochemical tests used across different subspecialties, resulting in worse care for marginalised racial groups.^{29–30} A key limitation of our study is that we cannot perform a race stratified analysis. Furthermore, we are unable to evaluate the relevance of other demographic features. An intersectional approach to healthcare inequalities would consider the mediating impact of socioeconomic class, or the compounding impact of gender (as opposed to sex) and sexuality on marginalised patients. Accounting for the complex nature of these intersectional relationships requires more advanced modelling and new bias evaluation techniques.

CONCLUSIONS

The historic absence of women from the healthcare profession and from clinical research resulted in domain knowledge that centres around the male body and neglects female physiological differences. To ensure sex-based inequalities do not manifest in medical AI, an evaluation of demographic performance disparities must be integrated into model development. Evaluating biases in the initial stages of ML can provide insights into

inequalities in existing practice, reveal pathophysiological differences between the sexes and can mitigate the digitisation of healthcare inequalities in algorithmic systems.

Twitter Isabel Straw @IsabelStrawMD and Honghan Wu @hhwu

Contributors IS conceived of the presented idea and is the guarantor responsible for overall content. IS developed the theory and performed the computations. HW verified the analytical methods.

Funding This work was supported by UK Research and Innovation (UKRI Grant Reference Number EP/S021612/1).

Competing interests None declared.

Patient consent for publication Not applicable.

Provenance and peer review Not commissioned; externally peer reviewed.

Data availability statement Data are available in a public, open access repository.

Supplemental material This content has been supplied by the author(s). It has not been vetted by BMJ Publishing Group Limited (BMJ) and may not have been peer-reviewed. Any opinions or recommendations discussed are solely those of the author(s) and are not endorsed by BMJ. BMJ disclaims all liability and responsibility arising from any reliance placed on the content. Where the content includes any translated material, BMJ does not warrant the accuracy and reliability of the translations (including but not limited to local regulations, clinical guidelines, terminology, drug names and drug dosages), and is not responsible for any error and/or omissions arising from translation and adaptation or otherwise.

Open access This is an open access article distributed in accordance with the Creative Commons Attribution Non Commercial (CC BY-NC 4.0) license, which permits others to distribute, remix, adapt, build upon this work non-commercially, and license their derivative works on different terms, provided the original work is properly cited, appropriate credit is given, any changes made indicated, and the use is non-commercial. See: <http://creativecommons.org/licenses/by-nc/4.0/>.

ORCID iD

Isabel Straw <http://orcid.org/0000-0003-0003-3550>

REFERENCES

- Blachier M, Leleu H, Peck-Radosavljevic M, *et al*. The burden of liver disease in Europe: a review of available epidemiological data. *J Hepatol* 2013;58:593–608.
- Morgan MY, Sherlock S. Sex-Related differences among 100 patients with alcoholic liver disease. *Br Med J* 1977;1:939–41.
- Vatsalya V, Liaquat HB, Ghosh K, *et al*. A review on the sex differences in organ and system pathology with alcohol drinking. *Curr Drug Abuse Rev* 2016;9:87–92.
- Mathur AK, Schaubel DE, Gong Q, *et al*. Sex-based disparities in liver transplant rates in the United States. *Am J Transplant* 2011;11:1435–43.
- UK Parliament, Women’s health outcomes: Is there a gender gap?, House of Lords Library, Editor. 2021, House of Lords. Available: <https://lordslibrary.parliament.uk/womens-health-outcomes-is-there-a-gender-gap/>
- Cleghorn E. *Unwell women: misdiagnosis and myth in a man-made world*. New York, NY: Dutton, 2021.
- Straw I. The automation of bias in medical artificial intelligence (AI): decoding the past to create a better future. *Artif Intell Med* 2020;110:101965.
- Krieger N, Fee E. Man-made medicine and women’s health: the biopolitics of sex/gender and race/ethnicity. *Int J Health Serv* 1994;24:265–83.
- Hoffmann DE, Tarzian AJ. The girl who cried pain: a bias against women in the treatment of pain. *J Law Med Ethics* 2001;29:13–27.
- Hamberg K. Gender bias in medicine. *Womens Health* 2008;4:237–43.
- Grimm G, Haslacher H, Kampitsch T, *et al*. Sex differences in the association between albumin and all-cause and vascular mortality. *Eur J Clin Invest* 2009;39:860–5.
- Suthahar N, Meems LMG, Ho JE, *et al*. Sex-Related differences in contemporary biomarkers for heart failure: a review. *Eur J Heart Fail* 2020;22:775–88.
- Stepien M, Fedirko V, Duarte-Salles T, *et al*. Prospective association of liver function biomarkers with development of hepatobiliary cancers. *Cancer Epidemiol* 2016;40:179–87.



- 14 Sidey-Gibbons JAM, Sidey-Gibbons CJ. Machine learning in medicine: a practical introduction. *BMC Med Res Methodol* 2019;19:1–18.
- 15 Cirillo D, Catuara-Solarz S, Morey C, et al. Sex and gender differences and biases in artificial intelligence for biomedicine and healthcare. *NPJ Digit Med* 2020;3:81.
- 16 O’Neil C. *Weapons of math destruction*. Harlow, England: Penguin Books, 2017.
- 17 Straw I, Callison-Burch C. Artificial intelligence in mental health and the biases of language based models. *PLoS One* 2020;15:e0240376.
- 18 M. BanuPriya, Tamilselvi PR. *Performance analysis of liver disease prediction using machine learning algorithms*. 5, 2018.
- 19 Aswathy C. Liver patient dataset classification using the Intel® distribution for python. Intel, specialized development tools, 2018. Available: <https://www.intel.com/content/www/us/en/developer/articles/technical/liver-patient-dataset-classification-using-the-intel-distribution-for-python.html>
- 20 Gulia A, Praveen Rani DRV. Liver patient classification using intelligence techniques. *Int J Comput Sci Inf Technol Res* 2014;5:5110–5.
- 21 Ramana BV, Boddu RSK. *Performance comparison of classification algorithms on medical datasets*. 2019 IEEE 9th Annual computing and communication workshop and conference (CCWC), 2019: 140–5.
- 22 Dua D, Graff C. UCI machine learning Repository. Irvine, Ca: University of California, school of information and computer science. ILPD dataset, 2019. Available: <https://archive.ics.uci.edu/ml/datasets/ILPD+%28Indian+Liver+Patient+Dataset%29#>
- 23 Jin H, Kim S, Kim J. Decision factors on effective liver patient data prediction. *International Journal of Bio-Science and Bio-Technology* 2014;6:167–78.
- 24 Adil SH, Ebrahim M, Raza K, et al. *Liver patient classification using logistic regression*. 4th International Conference on Computer and Information Sciences (ICCOINS). IEEE, 2018:2018.
- 25 Auxilla LA. *Accuracy prediction using machine learning techniques for Indian patient liver disease*. 2nd International Conference on Trends in Electronics and Informatics (ICOEI), 2018: 45–50.
- 26 Guy J, Peters MG. Liver disease in women: the influence of gender on epidemiology, natural history, and patient outcomes. *Gastroenterol Hepatol* 2013;9:633.
- 27 Li X, Wang D, Yang C, et al. Establishment of age- and gender-specific pediatric reference intervals for liver function tests in healthy Han children. *World J Pediatr* 2018;14:151–9.
- 28 Vyas DA, Eisenstein LG, Jones DS. Hidden in plain sight — reconsidering the use of race correction in clinical algorithms. *N Engl J Med Overseas Ed* 2020;383:874–82.
- 29 Eneanya ND, Boulware LE, Tsai J, et al. Health inequities and the inappropriate use of race in nephrology. *Nat Rev Nephrol* 2022;18:84–94.
- 30 Powe NR. Black kidney function matters: use or misuse of race? *JAMA* 2020;324:737–8.

Investigating for bias in healthcare algorithms: A sex stratified analysis of supervised machine learning models in liver disease prediction

Dr I Straw, Dr Honghan Wu

University College London, Institute of Health Informatics

Codebook and manuscript currently under submission for the British Medical Journal (BMJ).

```
!pip install imbalanced-learn

pip install --upgrade scipy

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC

import imblearn
from imblearn.over_sampling import SMOTE
from tqdm import tqdm
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA

from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.preprocessing import PowerTransformer
from sklearn.feature_selection import RFE
import seaborn as sns
from sklearn.utils import resample
from sklearn.preprocessing import MinMaxScaler
from sklearn import neighbors
from sklearn.metrics import precision_score, recall_score,
classification_report, confusion_matrix, roc_auc_score, auc,
roc_curve, make_scorer
from sklearn.base import clone
from itertools import combinations
from sklearn.model_selection import train_test_split, cross_val_score,
cross_validate
from keras.models import Sequential
from keras.layers import Dense
```

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import linear_model
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from scipy.stats import kendalltau
from scipy.stats import ttest_ind

import warnings
warnings.filterwarnings("ignore")
```

Data Import and Exploration

```
# Importing dataset from URL
URL =
"https://archive.ics.uci.edu/ml/machine-learning-databases/00225/India
n%20Liver%20Patient%20Dataset%20(ILPD).csv"
ILPD = pd.read_csv(URL, names=["Age", "Gender", "Total Bilirubin
(TB)", "Direct Bilirubin (DB)", "ALP (Alkphos Alkaline Phosphotase)",
"SGPT Alamine Aminotransferate", "SGOT Aspartate Aminotransferase",
"Total Protiens (TP)", "Albumin (ALB)", "A/G Ratio", "Target"])
ILPD.head(5)

print("The Shape of the Dataset:", ILPD.shape)
ILPD.describe()
ILPD.info() # gender datatype needs to be changed

ILPD['Target'] = (ILPD['Target']).astype(int)
ILPD.info()
```

Descriptives statistics performed before data pre-processing

```
ILPD.groupby(['Gender', 'Target']).size()

# Descriptive statistics
ILPD.groupby(['Gender']).mean()

# Addressing null values
ILPD['A/G Ratio'].fillna(ILPD['A/G Ratio'].mean(), inplace=True)
ILPD[ILPD["A/G Ratio"].isnull()]

ILPD.isnull().sum()

# Converting Gender to a numerical variable
gender_mapping = {'Male':1, 'Female':0}

# target 2 = healthy, 1 = diseased
```

```
target_mapping = {2:0, 1:1}
ILPD['Gender'] = ILPD['Gender'].map(target_mapping)
ILPD['Target'] = ILPD['Target'].map(target_mapping)
ILPD.info()
# Need to be an integer, or becomes decimals in the upsampling
ILPD['Target'] = (ILPD['Target']).astype(int)

ILPD.groupby(['Gender', 'Target']).size()

F_ILPD1 = ILPD[ILPD['Gender']==0]
print('Female data:', len(F_ILPD1))
M_ILPD1 = ILPD[ILPD['Gender']==1]
print('Male data:', len(M_ILPD1))

# Correlating features for males and females
ILPDCorrelation = pd.DataFrame(ILPD.corr(method='pearson'))
ILPDCorrelation['Target Correlations'] =
ILPDCorrelation['Target'].abs()
ILPDCorrelation = ILPDCorrelation.iloc[:, -1:]
ILPDCorrelation.sort_values(by='Target Correlations', ascending=False,
inplace=True)
ILPDCorrelation.reset_index(level=0, inplace=True)
ILPDCorrelation.reset_index(level=0, inplace=True)
ILPDCorrelation.rename(columns = {'level_0':'All Rank',
'index':'Feature'}, inplace=True)

F_ILPDCorrelation = pd.DataFrame(F_ILPD1.corr(method='pearson'))
F_ILPDCorrelation['F Target Correlations'] =
F_ILPDCorrelation['Target'].abs()
F_ILPDCorrelation = F_ILPDCorrelation.iloc[:, -1:]
F_ILPDCorrelation.sort_values(by='F Target Correlations',
ascending=False, inplace=True)
F_ILPDCorrelation.reset_index(level=0, inplace=True)
F_ILPDCorrelation.reset_index(level=0, inplace=True)
F_ILPDCorrelation.rename(columns = {'level_0':'Female Rank',
'index':'Feature'}, inplace=True)

M_ILPDCorrelation = pd.DataFrame(M_ILPD1.corr(method='pearson'))
M_ILPDCorrelation['M Target Correlations'] =
M_ILPDCorrelation['Target'].abs()
M_ILPDCorrelation = M_ILPDCorrelation.iloc[:, -1:]
M_ILPDCorrelation.sort_values(by='M Target Correlations',
ascending=False, inplace=True)
M_ILPDCorrelation.reset_index(level=0, inplace=True)
M_ILPDCorrelation.reset_index(level=0, inplace=True)
M_ILPDCorrelation.rename(columns = {'level_0':'Male Rank',
'index':'Feature'}, inplace=True)
ILPDCorrelation
```

```

FinalTable1 = pd.merge(ILPDCorrelation, F_ILPDCorrelation,
on='Feature', how='left')
FinalTable = pd.merge(FinalTable1, M_ILPDCorrelation, on='Feature',
how='left')
FinalTable = FinalTable.iloc[1:, :]
FRanks = FinalTable['Female Rank']
MRanks = FinalTable['Male Rank']
AllRanks = FinalTable['All Rank']
print(AllRanks)
print(FRanks)
print(MRanks)
FinalTable
Mcorr, _ = kendalltau(MRanks, AllRanks)
Fcorr, _ = kendalltau(FRanks, AllRanks)
print(Mcorr, Fcorr)

```

Data Preperation

Addressing Skewed Data

```

skewed = ['Total Bilirubin (TB)', 'Direct Bilirubin (DB)', 'ALP
(Alkphos Alkaline Phosphotase)', 'SGPT Alamine Aminotransferate',
'SGOT Aspartate Aminotransferase', 'Total Protiens (TP)', 'A/G Ratio']
skewed
for column in skewed:
    ILPD[column] = ILPD[column].apply('log1p')

```

Addressing null values

```

ILPD['A/G Ratio'].fillna(ILPD['A/G Ratio'].mean(), inplace=True)
ILPD[ILPD["A/G Ratio"].isnull()]

ILPD.isnull().sum()

Scaler = MinMaxScaler()
ScaledILPD = ILPD
ScaledILPD[['Total Bilirubin (TB)', 'Direct Bilirubin (DB)',
'ALP (Alkphos Alkaline Phosphotase)', 'SGPT Alamine
Aminotransferate',
'SGOT Aspartate Aminotransferase', 'Total Protiens (TP)',
'Albumin (ALB)', 'A/G Ratio',
'Target']] = Scaler.fit_transform(ILPD[['Total Bilirubin (TB)', 'Direct
Bilirubin (DB)',
'ALP (Alkphos Alkaline Phosphotase)', 'SGPT Alamine
Aminotransferate',
'SGOT Aspartate Aminotransferase', 'Total Protiens (TP)',
'Albumin (ALB)', 'A/G Ratio', 'Target']])

```

Defining Datasets

A described in the accompanying thesis, we separate the Indian Liver Patient Dataset (ILPD) into sex-stratified datasets to perform our model evaluation.


```
print('Female data:', len(F_ILPD1))
print('Male data:', len(M_ILPD1))

# Oversampling to address class imbalance
oversample = SMOTE()

y_ILPDTarget = ILPD.iloc[:, -1]
X_AllILPD = ILPD.iloc[:, :-1]
All_ILPDX_over, All_ILPDy_over = oversample.fit_resample(X_AllILPD,
y_ILPDTarget)

# Reform this into a dataframe
ILPD_ClassBal = All_ILPDX_over
ILPD_ClassBal['Target'] = All_ILPDy_over
ILPD_ClassBal

# Print after class imbalance address

print("Before addressing class imbalance")
print(ILPD.groupby(['Target']).size())

print("After addressing class imbalance")
print(ILPD_ClassBal.groupby(['Target']).size())

ILPD = ILPD_ClassBal

ILPD_ClassBal.groupby(['Gender', 'Target']).size()

ILPD['Target'] = ILPD['Target'].astype(int)
ILPD.info()

oversample = SMOTE()

y_ILPDSexTarget = ILPD.iloc[:, 1]
X_ILPD = ILPD.loc[:, ['Age', 'Total Bilirubin (TB)', 'Direct Bilirubin
(DB)',
    'ALP (Alkphos Alkaline Phosphotase)', 'SGPT Alamine
Aminotransferate',
    'SGOT Aspartate Aminotransferase', 'Total Protiens (TP)',
    'Albumin (ALB)', 'A/G Ratio', 'Target']]

ILPDX_over, ILPDy_over = oversample.fit_resample(X_ILPD,
y_ILPDSexTarget)
ILPD_Final = ILPDX_over
ILPD_Final['Gender'] = ILPDy_over
ILPD_Final
F_ILPD2 = ILPD_Final[ILPD_Final['Gender']==0]
print('Female data:', len(F_ILPD2))
M_ILPD2 = ILPD_Final[ILPD_Final['Gender']==1] # With 1 as suffix as
will reasmples for balancing
```

```
print('Female data:', len(F_ILPD2))
print('Male data:', len(M_ILPD2))

# These results may differ slightly to the manuscript, as the exact
counts will differ depending on the SMOTE Sampling
F_Sick1 = F_ILPD2[F_ILPD2['Target']==1]
F_Well1 = F_ILPD2[F_ILPD2['Target']==0]
M_Sick1 = M_ILPD2[M_ILPD2['Target']==1]
M_Well1 = M_ILPD2[M_ILPD2['Target']==0]

print('Sick Females:', len(F_Sick1), 'Sick Males:', len(M_Sick1))
print('Well Females:', len(F_Well1), 'Well Males', len(M_Well1))
```

3.0 Defining Datasets: Balanced and Unbalanced

```
AllX = ILPD.iloc[:, :-1]
Ally = ILPD.iloc[:, -1]

print(M_ILPD2.groupby(['Gender', 'Target']).size())
print(F_ILPD2.groupby(['Gender', 'Target']).size())
F_ILPD2

# For the balanced experiments we need Fx Fy separate
FX = F_ILPD2.loc[:, ['Age', 'Total Bilirubin (TB)', 'Direct Bilirubin
(DB)',
    'ALP (Alkphos Alkaline Phosphotase)', 'SGPT Alamine
Aminotransferate',
    'SGOT Aspartate Aminotransferase', 'Total Protiens (TP)',
    'Albumin (ALB)', 'A/G Ratio', 'Gender']]
Fy = F_ILPD2.loc[:, ['Target']]

MX = M_ILPD2.loc[:, ['Age', 'Total Bilirubin (TB)', 'Direct Bilirubin
(DB)',
    'ALP (Alkphos Alkaline Phosphotase)', 'SGPT Alamine
Aminotransferate',
    'SGOT Aspartate Aminotransferase', 'Total Protiens (TP)',
    'Albumin (ALB)', 'A/G Ratio', 'Gender']]
My = M_ILPD2.loc[:, ['Target']]
print(len(FX), len(MX), len(Fy), len(My))

Bal_ILPD = M_ILPD2.append(F_ILPD2)

## Define these so can fit to balanced data for hyperparameter tuning
Bal_X = Bal_ILPD.iloc[:, :-1]
Bal_y = Bal_ILPD.iloc[:, -1]

Bal_ILPD.groupby(['Gender', 'Target']).size()
print(len(FX), len(Fy), len(MX), len(My))

Bal_ILPD.groupby(['Gender', 'Target']).size()
```

Model Development and Hyperparameter Tuning

(Deleted previous way of tuning, see downloadeds in file - now done in function)

```
# For reference we now have our Unbalanced Data and Balanced Data
print(Bal_ILPD.shape, len(Bal_X), len(Bal_y))
print(ILPD.shape, len(AllX), len(Ally))

print(AllX.shape, Ally.shape)
print(AllX.groupby(['Gender']).size())
```

3.0 Experiments

Model Experiments: Defining functions for model experiments

3.1.1: Experiment 1 - Unbalanced without feature selection (4 models) 3.1.2: Experiment 2 - Balanced without feature selection (4 models) 3.1.3: Experiment 3 - Unbalanced **with** feature selection (4 models) 3.1.4: Experiment 4 - Balanced **with** feature selection (4 models)

```
# Defined Feature Set
## Defining feature subsets
AllFeatures = ['Age', 'Gender', 'Total Bilirubin (TB)', 'Direct
Bilirubin (DB)',
              'ALP (Alkphos Alkaline Phosphotase)', 'SGPT Alamine
Aminotransferate',
              'SGOT Aspartate Aminotransferase', 'Total Protiens (TP)',
              'Albumin (ALB)', 'A/G Ratio']

NB = GaussianNB()
LR = LogisticRegression()
SVM = SVC()
RF = RandomForestClassifier()
```

3.1 Unbalanced Training Data, No Feature Selection

```
# 1.
def unbalanced_run(X, y, FeatureSet, Classifier):
    AllX_train, AllX_test, Ally_train, Ally_test = train_test_split(X,
y, test_size=0.30)
    X_train = AllX_train[FeatureSet]
    X_test = AllX_test[FeatureSet]

    # Experiment specific hyperparameter tuning - Needed for
parameters
    depth=[2, 8, 16]
    n_estimators = [64, 128, 256]
    RFParams = dict(max_depth=depth, n_estimators=n_estimators)
    LRParams = {"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}
```

```
SVMParams = {'C': [0.001,0.01,0.1,1,10,100,1000],
             'gamma': [0.001,0.01,0.1,1,10,100,1000]
            }
NBParams = {'var_smoothing': np.logspace(0,-9, num=100)}

# Name of Parameters
if Classifier == LR:
    params = LRParams
if Classifier == RF:
    params = RFParams
if Classifier == SVM:
    params = SVMParams
if Classifier == NB:
    params = NBParams

ClassifierParams = GridSearchCV(Classifier, params, cv=5,
scoring='accuracy')
ClassifierParams.fit(AllX_train, Ally_train)

if Classifier == LR:
    Modell = LogisticRegression(**ClassifierParams.best_params_)
if Classifier == SVM:
    Modell = SVC(**ClassifierParams.best_params_)
if Classifier == RF:
    Modell =
RandomForestClassifier(**ClassifierParams.best_params_)
if Classifier == NB:
    Modell = GaussianNB(**ClassifierParams.best_params_)

Model = Modell.fit(AllX_train, Ally_train)
Ally_pred = Model.predict(AllX_test)

# Printing out results - same for each function
Accuracy = (accuracy_score(Ally_test,Ally_pred))*100
FScore = (f1_score(Ally_test, Ally_pred))*100
ROC_AUC = roc_auc_score(Ally_test, Ally_pred, multi_class='ovo')
Precision = precision_score(Ally_test, Ally_pred)
Recall = recall_score(Ally_test, Ally_pred)

Table = AllX_test
Table['By_pred']=Ally_pred
Table['By_true']=Ally_test
Table['Accuracy']=Accuracy
Table['FScore']=FScore
Table['ROC_AUC']=ROC_AUC
Table['Precision']=Precision
Table['Recall']=Recall

# Calculating true positives and negatives
conditions = [
```



```

    Table['By_pred'].eq(0) & Table['By_true'].eq(0),
    Table['By_pred'].eq(0) & Table['By_true'].eq(1),
    Table['By_pred'].eq(1) & Table['By_true'].eq(0),
    Table['By_pred'].eq(1) & Table['By_true'].eq(1)
]
choices = ['TN', 'FN', 'FP', 'TP']
Table['Result'] = np.select(conditions, choices, default=0)

FemaleDF = Table[Table['Gender']==0.0]
F_Accuracy = (accuracy_score(FemaleDF['By_true'],
FemaleDF['By_pred']))*100
F_FScore = (f1_score(FemaleDF['By_true'],
FemaleDF['By_pred']))*100
F_ROC_AUC = roc_auc_score(FemaleDF['By_true'],
FemaleDF['By_pred'], multi_class='ovo')
F_Precision = precision_score(FemaleDF['By_true'],
FemaleDF['By_pred'])
F_Recall = recall_score(FemaleDF['By_true'], FemaleDF['By_pred'])

F_FNs = len(FemaleDF[FemaleDF['Result']=='FN'])
F_TNs = len(FemaleDF[FemaleDF['Result']=='TN'])
F_FPs = len(FemaleDF[FemaleDF['Result']=='FP'])
F_TPs = len(FemaleDF[FemaleDF['Result']=='TP'])
F_All = FemaleDF['Result'].sum()

# ** EDITING IN HERE, TRUE VALUES FOR FALSE POSITIVE ETC
F_FalseNegativeRate = (F_FNs/(F_FNs+F_TPs))*100
F_TrueNegativeRate = (F_TNs/(F_TNs+F_FPs))*100
F_FalsePositiveRate = (F_FPs/(F_FPs+F_TNs))*100
F_TruePositiveRate = (F_TPs/(F_TPs+F_FNs))*100

# Males
MaleDF = Table[Table['Gender']==1.0]
M_Accuracy = (accuracy_score(MaleDF['By_true'],
MaleDF['By_pred']))*100
M_FScore = (f1_score(MaleDF['By_true'], MaleDF['By_pred']))*100
M_ROC_AUC = roc_auc_score(MaleDF['By_true'], MaleDF['By_pred'],
multi_class='ovo')
M_Precision = precision_score(MaleDF['By_true'],
MaleDF['By_pred'])
M_Recall = recall_score(MaleDF['By_true'], MaleDF['By_pred'])

M_FNs = len(MaleDF[MaleDF['Result']=='FN'])
M_TNs = len(MaleDF[MaleDF['Result']=='TN'])
M_FPs = len(MaleDF[MaleDF['Result']=='FP'])
M_TPs = len(MaleDF[MaleDF['Result']=='TP'])
M_All = MaleDF['Result'].sum()

M_FalseNegativeRate = (M_FNs/(M_FNs+M_TPs))*100
M_TrueNegativeRate = (M_TNs/(M_TNs+M_FPs))*100

```

```

M_FalsePositiveRate = (M_FPs/(M_FPs+M_TNs))*100
M_TruePositiveRate = (M_TPs/(M_TPs+M_FNs))*100

#print("Female False Negative Rate", F_FalseNegativeRate)
#print("Male False Negative Rate", M_FalseNegativeRate)

#print("Female True Positive Rate", F_TruePositiveRate)
#print("Male True PositiveRate", M_TruePositiveRate)

#print("Female False Positive Rate", F_FalsePositiveRate)
#print("Male False Positive Rate", M_FalsePositiveRate)

    return Accuracy, FScore, ROC_AUC, Precision, Recall, F_Accuracy,
    F_FScore, F_ROC_AUC, F_Precision, F_Recall, F_FNs, F_TNs, F_FPs,
    F_TPs, F_FalseNegativeRate, F_TrueNegativeRate, F_FalsePositiveRate,
    F_TruePositiveRate, M_Accuracy, M_FScore, M_ROC_AUC, M_Precision,
    M_Recall, M_FNs, M_TNs, M_FPs, M_TPs, M_FalseNegativeRate,
    M_TrueNegativeRate, M_FalsePositiveRate, M_TruePositiveRate

# Order of results for reference
#Accuracy, FScore, ROC_AUC, Precision, Recall

#F_Accuracy, F_FScore, F_ROC_AUC, F_Precision, F_Recall,
#F_FNs, F_TNs, F_FPs, F_TPs
#F_FalseNegativeRate, F_TrueNegativeRate, F_FalsePositiveRate,
F_TruePositiveRate

#M_Accuracy, M_FScore, M_ROC_AUC, M_Precision, M_Recall,
#M_FNs, M_TNs, M_FPs, M_TPs
#M_FalseNegativeRate, M_TrueNegativeRate, M_FalsePositiveRate,
M_TruePositiveRate

# Example - Can swap in the classifiers here
unbalanced_run(AllX, Ally, AllFeatures, LR)

3.1.1 Random Forest - Unbalanced, No Feature Selection
experiment_number = np.arange(100)

## RF Results
cols = ['UB_Accuracy', 'UB_FScore', 'UB_ROC_AUC', 'UB_Precision',
'UB_Recall',

        'UB_F_Accuracy', 'UB_F_FScore', 'UB_F_ROC_AUC',
'UB_F_Precision', 'UB_F_Recall',
        'UB_F_FNs', 'UB_F_TNs', 'UB_F_FPs', 'UB_F_TPs',
        'UB_F_FNR', 'UB_F_TNR', 'UB_F_FPR', 'UB_F_TPR',

        'UB_M_Accuracy', 'UB_M_FScore', 'UB_M_ROC_AUC',
'UB_M_Precision', 'UB_M_Recall',

```

```

        'UB_M_FNs', 'UB_M_TNs', 'UB_M_FPs', 'UB_M_TPs',
        'UB_M_FNR', 'UB_M_TNR', 'UB_M_FPR', 'UB_M_TPR',
    ]

Unbal_No_FS_RFdf = pd.DataFrame(columns=cols)

# Scroll left here to see the function that defines it as a balanced
# experiment
for i in tqdm(experiment_number):
    UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall,
    UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall,
    UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR, UB_F_TNR, UB_F_FPR,
    UB_F_TPR, UB_M_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision,
    UB_M_Recall, UB_M_FNs, UB_M_TNs, UB_M_FPs, UB_M_TPs, UB_M_FNR,
    UB_M_TNR, UB_M_FPR, UB_M_TPR = unbalanced_run(AllX, Ally, AllFeatures,
    RF)

    list = UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision,
    UB_Recall, UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision,
    UB_F_Recall, UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR,
    UB_F_TNR, UB_F_FPR, UB_F_TPR, UB_M_Accuracy, UB_M_FScore,
    UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNs, UB_M_TNs,
    UB_M_FPs, UB_M_TPs, UB_M_FNR, UB_M_TNR, UB_M_FPR, UB_M_TPR
    Unbal_No_FS_RFdf.loc[len(Unbal_No_FS_RFdf)] = list

# reset the name of the df so can make multiple outputs with same code
df = Unbal_No_FS_RFdf
print('All Accuracy (Mean)', df['UB_Accuracy'].mean(), 'SD',
df['UB_Accuracy'].std())
print('All FScore (Mean)', df['UB_FScore'].mean(), 'SD',
df['UB_FScore'].std())
print('All ROC_AUC (Mean)', df['UB_ROC_AUC'].mean(), 'SD',
df['UB_ROC_AUC'].std())
print('All Precision (Mean)', df['UB_Precision'].mean(), 'SD',
df['UB_Precision'].std())
print('All Recall (Mean)', df['UB_Recall'].mean(), 'SD',
df['UB_Recall'].std())
print('')

print('F Accuracy (Mean)', df['UB_F_Accuracy'].mean(), 'SD',
df['UB_F_Accuracy'].std())
print('F FScore (Mean)', df['UB_F_FScore'].mean(), 'SD',
df['UB_F_FScore'].std())
print('F ROC_AUC (Mean)', df['UB_F_ROC_AUC'].mean(),
df['UB_F_ROC_AUC'].std())
print('F Precision (Mean)', df['UB_F_Precision'].mean(),
df['UB_F_Precision'].std())
print('F Recall (Mean)', df['UB_F_Recall'].mean(),
df['UB_F_Recall'].std())

```

```

print('F FNR (Mean)', df['UB_F_FNR'].mean(), df['UB_F_FNR'].std())
print('F TNR (Mean)', df['UB_F_TNR'].mean(), df['UB_F_TNR'].std())
print('F FPR (Mean)', df['UB_F_FPR'].mean(), df['UB_F_FPR'].std())
print('F TPR (Mean)', df['UB_F_TPR'].mean(), df['UB_F_TPR'].std())
print('')

```

```

print('M Accuracy (Mean)', df['UB_M_Accuracy'].mean(),
df['UB_M_Accuracy'].std())
print('M FScore (Mean)', df['UB_M_FScore'].mean(),
df['UB_M_FScore'].std())
print('M ROC_AUC (Mean)', df['UB_M_ROC_AUC'].mean(),
df['UB_M_ROC_AUC'].std())
print('M Precision (Mean)', df['UB_M_Precision'].mean(),
df['UB_M_Precision'].std())
print('M Recall (Mean)', df['UB_M_Recall'].mean(),
df['UB_M_Recall'].std())

```

```

print('M FNR (Mean)', df['UB_M_FNR'].mean(), df['UB_M_FNR'].std())
print('M TNR (Mean)', df['UB_M_TNR'].mean(), df['UB_M_TNR'].std())
print('M FPR (Mean)', df['UB_M_FPR'].mean(), df['UB_M_FPR'].std())
print('M TPR (Mean)', df['UB_M_TPR'].mean(), df['UB_M_TPR'].std())
print('')

```

```
print("Female/Male T Tests:")
```

```
# This gives us each run
```

```
Unbal_No_FS_RFdf.to_csv('Unbal_No_FS_RFdf.csv')
```

3.1.2 Logistic Regression - Unbalanced, No Feature Selection

```
# Logistic Regression, No Feature Selection, Unbalanced Data
```

```
experiment_number = np.arange(100)
```

```
## RF Results
```

```
cols = ['UB_Accuracy', 'UB_FScore', 'UB_ROC_AUC', 'UB_Precision',
'UB_Recall',
```

```

    'UB_F_Accuracy', 'UB_F_FScore', 'UB_F_ROC_AUC',
'UB_F_Precision', 'UB_F_Recall',
    'UB_F_FNs', 'UB_F_TNs', 'UB_F_FPs', 'UB_F_TPs',
    'UB_F_FNR', 'UB_F_TNR', 'UB_F_FPR', 'UB_F_TPR',
```

```

    'UB_M_Accuracy', 'UB_M_FScore', 'UB_M_ROC_AUC',
'UB_M_Precision', 'UB_M_Recall',
    'UB_M_FNs', 'UB_M_TNs', 'UB_M_FPs', 'UB_M_TPs',
    'UB_M_FNR', 'UB_M_TNR', 'UB_M_FPR', 'UB_M_TPR',
]

```

```
]
```



```

Unbal_No_FS_LRdf = pd.DataFrame(columns=cols)

# Scroll left here to see the function that defines it as a balanced
experiment

# NEEDED TO ENSURE THERE WAS AN INDENT

for i in tqdm(experiment_number):
    UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall,
    UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall,
    UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR, UB_F_TNR, UB_F_FPR,
    UB_F_TPR, UB_M_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision,
    UB_M_Recall, UB_M_FNs, UB_M_TNs, UB_M_FPs, UB_M_TPs, UB_M_FNR,
    UB_M_TNR, UB_M_FPR, UB_M_TPR = unbalanced_run(AllX, Ally, AllFeatures,
    LR)

    list = UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision,
    UB_Recall, UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision,
    UB_F_Recall, UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR,
    UB_F_TNR, UB_F_FPR, UB_F_TPR, UB_M_Accuracy, UB_M_FScore,
    UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNs, UB_M_TNs,
    UB_M_FPs, UB_M_TPs, UB_M_FNR, UB_M_TNR, UB_M_FPR, UB_M_TPR
    Unbal_No_FS_LRdf.loc[len(Unbal_No_FS_LRdf)] = list

# reset the name of the df so can make multiple outputs with same code
df = Unbal_No_FS_LRdf
print('All Accuracy (Mean)', df['UB_Accuracy'].mean(), 'SD',
df['UB_Accuracy'].std())
print('All FScore (Mean)', df['UB_FScore'].mean(), 'SD',
df['UB_FScore'].std())
print('All ROC_AUC (Mean)', df['UB_ROC_AUC'].mean(), 'SD',
df['UB_ROC_AUC'].std())
print('All Precision (Mean)', df['UB_Precision'].mean(), 'SD',
df['UB_Precision'].std())
print('All Recall (Mean)', df['UB_Recall'].mean(), 'SD',
df['UB_Recall'].std())
print('
    ')

print('F Accuracy (Mean)', df['UB_F_Accuracy'].mean(), 'SD',
df['UB_F_Accuracy'].std())
print('F FScore (Mean)', df['UB_F_FScore'].mean(), 'SD',
df['UB_F_FScore'].std())
print('F ROC_AUC (Mean)', df['UB_F_ROC_AUC'].mean(),
df['UB_F_ROC_AUC'].std())
print('F Precision (Mean)', df['UB_F_Precision'].mean(),
df['UB_F_Precision'].std())
print('F Recall (Mean)', df['UB_F_Recall'].mean(),
df['UB_F_Recall'].std())

print('F FNR (Mean)', df['UB_F_FNR'].mean(), df['UB_F_FNR'].std())

```

```

print('F TNR (Mean)', df['UB_F_TNR'].mean(), df['UB_F_TNR'].std())
print('F FPR (Mean)', df['UB_F_FPR'].mean(), df['UB_F_FPR'].std())
print('F TPR (Mean)', df['UB_F_TPR'].mean(), df['UB_F_TPR'].std())
print('      ')

print('M Accuracy (Mean)', df['UB_M_Accuracy'].mean(),
df['UB_M_Accuracy'].std())
print('M FScore (Mean)', df['UB_M_FScore'].mean(),
df['UB_M_FScore'].std())
print('M ROC_AUC (Mean)', df['UB_M_ROC_AUC'].mean(),
df['UB_M_ROC_AUC'].std())
print('M Precision (Mean)', df['UB_M_Precision'].mean(),
df['UB_M_Precision'].std())
print('M Recall (Mean)', df['UB_M_Recall'].mean(),
df['UB_M_Recall'].std())

print('M FNR (Mean)', df['UB_M_FNR'].mean(), df['UB_M_FNR'].std())
print('M TNR (Mean)', df['UB_M_TNR'].mean(), df['UB_M_TNR'].std())
print('M FPR (Mean)', df['UB_M_FPR'].mean(), df['UB_M_FPR'].std())
print('M TPR (Mean)', df['UB_M_TPR'].mean(), df['UB_M_TPR'].std())
print('      ')

print("Female/Male T Tests:")

# This gives us each run
Unbal_No_FS_LRdf.to_csv('Unbal_No_FS_LRdf.csv')

3.1.3 Support Vector Machine - Unbalanced, No Feature Selection
## SVM Results
cols = ['UB_Accuracy', 'UB_FScore', 'UB_ROC_AUC', 'UB_Precision',
'UB_Recall',

        'UB_F_Accuracy', 'UB_F_FScore', 'UB_F_ROC_AUC',
'UB_F_Precision', 'UB_F_Recall',
        'UB_F_FNs', 'UB_F_TNs', 'UB_F_FPs', 'UB_F_TPs',
        'UB_F_FNR', 'UB_F_TNR', 'UB_F_FPR', 'UB_F_TPR',

        'UB_M_Accuracy', 'UB_M_FScore', 'UB_M_ROC_AUC',
'UB_M_Precision', 'UB_M_Recall',
        'UB_M_FNs', 'UB_M_TNs', 'UB_M_FPs', 'UB_M_TPs',
        'UB_M_FNR', 'UB_M_TNR', 'UB_M_FPR', 'UB_M_TPR',
]

Unbal_No_FS_SVMdf = pd.DataFrame(columns=cols)
for i in tqdm(experiment_number):
    UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall,
    UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall,

```

```
UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR, UB_F_TNR, UB_F_FPR,
UB_F_TPR, UB_M_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision,
UB_M_Recall, UB_M_FNs, UB_M_TNs, UB_M_FPs, UB_M_TPs, UB_M_FNR,
UB_M_TNR, UB_M_FPR, UB_M_TPR = unbalanced_run(AllX, Ally, AllFeatures,
SVM)
```

```
list = UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision,
UB_Recall, UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision,
UB_F_Recall, UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR,
UB_F_TNR, UB_F_FPR, UB_F_TPR, UB_M_Accuracy, UB_M_FScore,
UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNs, UB_M_TNs,
UB_M_FPs, UB_M_TPs, UB_M_FNR, UB_M_TNR, UB_M_FPR, UB_M_TPR
Unbal_No_FS_SVMdf.loc[len(Unbal_No_FS_SVMdf)] = list
```

```
# reset the name of the df so can make multiple outputs with same code
```

```
df = Unbal_No_FS_SVMdf
print('All Accuracy (Mean)', df['UB_Accuracy'].mean(), 'SD',
df['UB_Accuracy'].std())
print('All FScore (Mean)', df['UB_FScore'].mean(), 'SD',
df['UB_FScore'].std())
print('All ROC_AUC (Mean)', df['UB_ROC_AUC'].mean(), 'SD',
df['UB_ROC_AUC'].std())
print('All Precision (Mean)', df['UB_Precision'].mean(), 'SD',
df['UB_Precision'].std())
print('All Recall (Mean)', df['UB_Recall'].mean(), 'SD',
df['UB_Recall'].std())
print(' ')

print('F Accuracy (Mean)', df['UB_F_Accuracy'].mean(), 'SD',
df['UB_F_Accuracy'].std())
print('F FScore (Mean)', df['UB_F_FScore'].mean(), 'SD',
df['UB_F_FScore'].std())
print('F ROC_AUC (Mean)', df['UB_F_ROC_AUC'].mean(),
df['UB_F_ROC_AUC'].std())
print('F Precision (Mean)', df['UB_F_Precision'].mean(),
df['UB_F_Precision'].std())
print('F Recall (Mean)', df['UB_F_Recall'].mean(),
df['UB_F_Recall'].std())

print('F FNR (Mean)', df['UB_F_FNR'].mean(), df['UB_F_FNR'].std())
print('F TNR (Mean)', df['UB_F_TNR'].mean(), df['UB_F_TNR'].std())
print('F FPR (Mean)', df['UB_F_FPR'].mean(), df['UB_F_FPR'].std())
print('F TPR (Mean)', df['UB_F_TPR'].mean(), df['UB_F_TPR'].std())
print(' ')

print('M Accuracy (Mean)', df['UB_M_Accuracy'].mean(),
df['UB_M_Accuracy'].std())
print('M FScore (Mean)', df['UB_M_FScore'].mean(),
df['UB_M_FScore'].std())
```

```

print('M ROC_AUC (Mean)', df['UB_M_ROC_AUC'].mean(),
df['UB_M_ROC_AUC'].std())
print('M Precision (Mean)', df['UB_M_Precision'].mean(),
df['UB_M_Precision'].std())
print('M Recall (Mean)', df['UB_M_Recall'].mean(),
df['UB_M_Recall'].std())

print('M FNR (Mean)', df['UB_M_FNR'].mean(), df['UB_M_FNR'].std())
print('M TNR (Mean)', df['UB_M_TNR'].mean(), df['UB_M_TNR'].std())
print('M FPR (Mean)', df['UB_M_FPR'].mean(), df['UB_M_FPR'].std())
print('M TPR (Mean)', df['UB_M_TPR'].mean(), df['UB_M_TPR'].std())
print('          ')

print("Female/Male T Tests:")

# This gives us each run
Unbal_No_FS_SVMdf.to_csv('Unbal_No_FS_SVMdf.csv')

3.1.3 Gaussian NB - Unbalanced, No Feature Selection
# Gaussian NB, No Feature Selection, Unbalanced Data
experiment_number = np.arange(100)

## RF Results
cols = ['UB_Accuracy', 'UB_FScore', 'UB_ROC_AUC', 'UB_Precision',
'UB_Recall',
        'UB_F_Accuracy', 'UB_F_FScore', 'UB_F_ROC_AUC',
'UB_F_Precision', 'UB_F_Recall',
        'UB_F_FNs', 'UB_F_TNs', 'UB_F_FPs', 'UB_F_TPs',
        'UB_F_FNR', 'UB_F_TNR', 'UB_F_FPR', 'UB_F_TPR',
        'UB_M_Accuracy', 'UB_M_FScore', 'UB_M_ROC_AUC',
'UB_M_Precision', 'UB_M_Recall',
        'UB_M_FNs', 'UB_M_TNs', 'UB_M_FPs', 'UB_M_TPs',
        'UB_M_FNR', 'UB_M_TNR', 'UB_M_FPR', 'UB_M_TPR',
]

Unbal_No_FS_GNBdf = pd.DataFrame(columns=cols)

# Scroll left here to see the function that defines it as a balanced
experiment

# NEEDED TO ENSURE THERE WAS AN INDENT

for i in tqdm(experiment_number):
    UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall,
    UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall,
    UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR, UB_F_TNR, UB_F_FPR,

```



```
UB_F_TPR, UB_M_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision,  
UB_M_Recall, UB_M_FNs, UB_M_TNs, UB_M_FPs, UB_M_TPs, UB_M_FNR,  
UB_M_TNR, UB_M_FPR, UB_M_TPR = unbalanced_run(AllX, Ally, AllFeatures,  
NB)
```

```
list = UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision,  
UB_Recall, UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision,  
UB_F_Recall, UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR,  
UB_F_TNR, UB_F_FPR, UB_F_TPR, UB_M_Accuracy, UB_M_FScore,  
UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNs, UB_M_TNs,  
UB_M_FPs, UB_M_TPs, UB_M_FNR, UB_M_TNR, UB_M_FPR, UB_M_TPR  
Unbal_No_FS_GNBdf.loc[len(Unbal_No_FS_GNBdf)] = list
```

```
# reset the name of the df so can make multiple outputs with same code
```

```
df = Unbal_No_FS_GNBdf  
print('All Accuracy (Mean)', df['UB_Accuracy'].mean(), 'SD',  
df['UB_Accuracy'].std())  
print('All FScore (Mean)', df['UB_FScore'].mean(), 'SD',  
df['UB_FScore'].std())  
print('All ROC_AUC (Mean)', df['UB_ROC_AUC'].mean(), 'SD',  
df['UB_ROC_AUC'].std())  
print('All Precision (Mean)', df['UB_Precision'].mean(), 'SD',  
df['UB_Precision'].std())  
print('All Recall (Mean)', df['UB_Recall'].mean(), 'SD',  
df['UB_Recall'].std())  
print('')  
  
print('F Accuracy (Mean)', df['UB_F_Accuracy'].mean(), 'SD',  
df['UB_F_Accuracy'].std())  
print('F FScore (Mean)', df['UB_F_FScore'].mean(), 'SD',  
df['UB_F_FScore'].std())  
print('F ROC_AUC (Mean)', df['UB_F_ROC_AUC'].mean(),  
df['UB_F_ROC_AUC'].std())  
print('F Precision (Mean)', df['UB_F_Precision'].mean(),  
df['UB_F_Precision'].std())  
print('F Recall (Mean)', df['UB_F_Recall'].mean(),  
df['UB_F_Recall'].std())  
  
print('F FNR (Mean)', df['UB_F_FNR'].mean(), df['UB_F_FNR'].std())  
print('F TNR (Mean)', df['UB_F_TNR'].mean(), df['UB_F_TNR'].std())  
print('F FPR (Mean)', df['UB_F_FPR'].mean(), df['UB_F_FPR'].std())  
print('F TPR (Mean)', df['UB_F_TPR'].mean(), df['UB_F_TPR'].std())  
print('')
```

```
print('M Accuracy (Mean)', df['UB_M_Accuracy'].mean(),  
df['UB_M_Accuracy'].std())  
print('M FScore (Mean)', df['UB_M_FScore'].mean(),  
df['UB_M_FScore'].std())  
print('M ROC_AUC (Mean)', df['UB_M_ROC_AUC'].mean(),
```

```

df['UB_M_ROC_AUC'].std())
print('M Precision (Mean)', df['UB_M_Precision'].mean(),
df['UB_M_Precision'].std())
print('M Recall (Mean)', df['UB_M_Recall'].mean(),
df['UB_M_Recall'].std())

print('M FNR (Mean)', df['UB_M_FNR'].mean(), df['UB_M_FNR'].std())
print('M TNR (Mean)', df['UB_M_TNR'].mean(), df['UB_M_TNR'].std())
print('M FPR (Mean)', df['UB_M_FPR'].mean(), df['UB_M_FPR'].std())
print('M TPR (Mean)', df['UB_M_TPR'].mean(), df['UB_M_TPR'].std())
print('          ')

print("Female/Male T Tests:")

# This gives us each run
Unbal_No_FS_GNBdf.to_csv('Unbal_No_FS_GNBdf.csv')

3.2 Balanced, without Feature Selection
# 2. BALANCED, WITHOUT FEATURE SELECTION
def balanced_run(MX, My, FX, Fy, FeatureSet, Classifier):
    MX_train, MX_test, My_train, My_test = train_test_split(MX, My,
test_size=0.30)
    FX_train, FX_test, Fy_train, Fy_test = train_test_split(FX, Fy,
test_size=0.30)

    # Redefine X train so its just desired features (i.e. No Sex)
    BX_train = MX_train.append(FX_train)
    BX_test = MX_test.append(FX_test)
    By_train = My_train.append(Fy_train)
    By_test = My_test.append(Fy_test)

    X_train = BX_train[FeatureSet]
    X_test = BX_test[FeatureSet]

    # Experiment specific hyperparameter tuning - Needed for
parameters
    depth=[2, 8, 16]
    n_estimators = [64, 128, 256]
    RFParams = dict(max_depth=depth, n_estimators=n_estimators)
    LRParams = {"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}
    SVMParams = {'C': [0.001,0.01,0.1,1,10,100,1000],
'gamma': [0.001,0.01,0.1,1,10,100,1000]
}
    NBParams = {'var_smoothing': np.logspace(0,-9, num=100)}

    # Name of Parameters
    if Classifier == LR:
        params = LRParams
    if Classifier == RF:

```

```
    params = RFParams
  if Classifier == SVM:
    params = SVMParams
  if Classifier == NB:
    params = NBParams

ClassifierParams = GridSearchCV(Classifier, params, cv=5,
scoring='accuracy')
ClassifierParams.fit(BX_train, By_train)

  if Classifier == LR:
    Modell = LogisticRegression(**ClassifierParams.best_params_)
  if Classifier == SVM:
    Modell = SVC(**ClassifierParams.best_params_)
  if Classifier == RF:
    Modell =
RandomForestClassifier(**ClassifierParams.best_params_)
  if Classifier == NB:
    Modell = GaussianNB(**ClassifierParams.best_params_)

Model = Modell.fit(BX_train, By_train)
By_pred = Model.predict(BX_test)

# Printing out results - same for each function
Accuracy = (accuracy_score(By_test,By_pred))*100
FScore = (f1_score(By_test, By_pred))*100
ROC_AUC = roc_auc_score(By_test, By_pred, multi_class='ovo')
Precision = precision_score(By_test, By_pred)
Recall = recall_score(By_test, By_pred)

Table = BX_test
Table['By_pred']=By_pred
Table['By_true']=By_test
Table['Accuracy']=Accuracy
Table['FScore']=FScore
Table['ROC_AUC']=ROC_AUC
Table['Precision']=Precision
Table['Recall']=Recall

# Calculating true positives and negatives
conditions = [
    Table['By_pred'].eq(0) & Table['By_true'].eq(0),
    Table['By_pred'].eq(0) & Table['By_true'].eq(1),
    Table['By_pred'].eq(1) & Table['By_true'].eq(0),
    Table['By_pred'].eq(1) & Table['By_true'].eq(1)
]
choices = ['TN', 'FN', 'FP', 'TP']
Table['Result'] = np.select(conditions, choices, default=0)

FemaleDF = Table[Table['Gender']==0.0]
```

```
F_Accuracy = (accuracy_score(FemaleDF['By_true'],
FemaleDF['By_pred']))*100
F_FScore = (f1_score(FemaleDF['By_true'],
FemaleDF['By_pred']))*100
F_ROC_AUC = roc_auc_score(FemaleDF['By_true'],
FemaleDF['By_pred'], multi_class='ovo')
F_Precision = precision_score(FemaleDF['By_true'],
FemaleDF['By_pred'])
F_Recall = recall_score(FemaleDF['By_true'], FemaleDF['By_pred'])

F_FNs = len(FemaleDF[FemaleDF['Result']=='FN'])
F_TNs = len(FemaleDF[FemaleDF['Result']=='TN'])
F_FPs = len(FemaleDF[FemaleDF['Result']=='FP'])
F_TPs = len(FemaleDF[FemaleDF['Result']=='TP'])
F_All = FemaleDF['Result'].sum()

# ** EDITING IN HERE, TRUE VALUES FOR FALSE POSITIVE ETC
F_FalseNegativeRate = (F_FNs/(F_FNs+F_TPs))*100
F_TrueNegativeRate = (F_TNs/(F_TNs+F_FPs))*100
F_FalsePositiveRate = (F_FPs/(F_FPs+F_TNs))*100
F_TruePositiveRate = (F_TPs/(F_TPs+F_FNs))*100

# Males
MaleDF = Table[Table['Gender']==1.0]
M_Accuracy = (accuracy_score(MaleDF['By_true'],
MaleDF['By_pred']))*100
M_FScore = (f1_score(MaleDF['By_true'], MaleDF['By_pred']))*100
M_ROC_AUC = roc_auc_score(MaleDF['By_true'], MaleDF['By_pred'],
multi_class='ovo')
M_Precision = precision_score(MaleDF['By_true'],
MaleDF['By_pred'])
M_Recall = recall_score(MaleDF['By_true'], MaleDF['By_pred'])

M_FNs = len(MaleDF[MaleDF['Result']=='FN'])
M_TNs = len(MaleDF[MaleDF['Result']=='TN'])
M_FPs = len(MaleDF[MaleDF['Result']=='FP'])
M_TPs = len(MaleDF[MaleDF['Result']=='TP'])
M_All = MaleDF['Result'].sum()

M_FalseNegativeRate = (M_FNs/(M_FNs+M_TPs))*100
M_TrueNegativeRate = (M_TNs/(M_TNs+M_FPs))*100
M_FalsePositiveRate = (M_FPs/(M_FPs+M_TNs))*100
M_TruePositiveRate = (M_TPs/(M_TPs+M_FNs))*100

return Accuracy, FScore, ROC_AUC, Precision, Recall, F_Accuracy,
F_FScore, F_ROC_AUC, F_Precision, F_Recall, F_FNs, F_TNs, F_FPs,
F_TPs, F_FalseNegativeRate, F_TrueNegativeRate, F_FalsePositiveRate,
F_TruePositiveRate, M_Accuracy, M_FScore, M_ROC_AUC, M_Precision,
M_Recall, M_FNs, M_TNs, M_FPs, M_TPs, M_FalseNegativeRate,
M_TrueNegativeRate, M_FalsePositiveRate, M_TruePositiveRate
```

```

# testing experiment
balanced_run(MX, My, FX, Fy, AllFeatures, RF)

3.2.1 Random Forest - Balanced, without Feature Selection
# RANDOM FOREST, No Feature Selection, Unbalanced Data
experiment_number = np.arange(100)

## RF Results
cols = ['B_Accuracy', 'B_FScore', 'B_ROC_AUC', 'B_Precision',
        'B_Recall',

        'B_F_Accuracy', 'B_F_FScore', 'B_F_ROC_AUC', 'B_F_Precision',
        'B_F_Recall',
        'B_F_FNs', 'B_F_TNs', 'B_F_FPs', 'B_F_TPs',
        'B_F_FNR', 'B_F_TNR', 'B_F_FPR', 'B_F_TPR',

        'B_M_Accuracy', 'B_M_FScore', 'B_M_ROC_AUC', 'B_M_Precision',
        'B_M_Recall',
        'B_M_FNs', 'B_M_TNs', 'B_M_FPs', 'B_M_TPs',
        'B_M_FNR', 'B_M_TNR', 'B_M_FPR', 'B_M_TPR',
        ]

Bal_No_FS_RFdf = pd.DataFrame(columns=cols)
for i in tqdm(experiment_number):
    B_Accuracy, B_FScore, B_ROC_AUC, B_Precision, B_Recall,
    B_F_Accuracy, B_F_FScore, B_F_ROC_AUC, B_F_Precision, B_F_Recall,
    B_F_FNs, B_F_TNs, B_F_FPs, B_F_TPs, B_F_FNR, B_F_TNR, B_F_FPR,
    B_F_TPR, B_M_Accuracy, B_M_FScore, B_M_ROC_AUC, B_M_Precision,
    B_M_Recall, B_M_FNs, B_M_TNs, B_M_FPs, B_M_TPs, B_M_FNR, B_M_TNR,
    B_M_FPR, B_M_TPR = balanced_run(MX, My, FX, Fy, AllFeatures, RF)

    list = B_Accuracy, B_FScore, B_ROC_AUC, B_Precision, B_Recall,
    B_F_Accuracy, B_F_FScore, B_F_ROC_AUC, B_F_Precision, B_F_Recall,
    B_F_FNs, B_F_TNs, B_F_FPs, B_F_TPs, B_F_FNR, B_F_TNR, B_F_FPR,
    B_F_TPR, B_M_Accuracy, B_M_FScore, B_M_ROC_AUC, B_M_Precision,
    B_M_Recall, B_M_FNs, B_M_TNs, B_M_FPs, B_M_TPs, B_M_FNR, B_M_TNR,
    B_M_FPR, B_M_TPR
    Bal_No_FS_RFdf.loc[len(Bal_No_FS_RFdf)] = list

# reset the name of the df so can make multiple outputs with same code
df = Bal_No_FS_RFdf
print('All Accuracy (Mean)', df['B_Accuracy'].mean(), 'SD',
df['B_Accuracy'].std())
print('All FScore (Mean)', df['B_FScore'].mean(), 'SD',
df['B_FScore'].std())
print('All ROC_AUC (Mean)', df['B_ROC_AUC'].mean(), 'SD',
df['B_ROC_AUC'].std())
print('All Precision (Mean)', df['B_Precision'].mean(), 'SD',
df['B_Precision'].std())

```

```

print('All Recall (Mean)', df['B_Recall'].mean(), 'SD',
df['B_Recall'].std())
print('      ')

print('F Accuracy (Mean)', df['B_F_Accuracy'].mean(), 'SD',
df['B_F_Accuracy'].std())
print('F FScore (Mean)', df['B_F_FScore'].mean(), 'SD',
df['B_F_FScore'].std())
print('F ROC_AUC (Mean)', df['B_F_ROC_AUC'].mean(),
df['B_F_ROC_AUC'].std())
print('F Precision (Mean)', df['B_F_Precision'].mean(),
df['B_F_Precision'].std())
print('F Recall (Mean)', df['B_F_Recall'].mean(),
df['B_F_Recall'].std())

print('F FNR (Mean)', df['B_F_FNR'].mean(), df['B_F_FNR'].std())
print('F TNR (Mean)', df['B_F_TNR'].mean(), df['B_F_TNR'].std())
print('F FPR (Mean)', df['B_F_FPR'].mean(), df['B_F_FPR'].std())
print('F TPR (Mean)', df['B_F_TPR'].mean(), df['B_F_TPR'].std())
print('      ')

print('M Accuracy (Mean)', df['B_M_Accuracy'].mean(),
df['B_M_Accuracy'].std())
print('M FScore (Mean)', df['B_M_FScore'].mean(),
df['B_M_FScore'].std())
print('M ROC_AUC (Mean)', df['B_M_ROC_AUC'].mean(),
df['B_M_ROC_AUC'].std())
print('M Precision (Mean)', df['B_M_Precision'].mean(),
df['B_M_Precision'].std())
print('M Recall (Mean)', df['B_M_Recall'].mean(),
df['B_M_Recall'].std())

print('M FNR (Mean)', df['B_M_FNR'].mean(), df['B_M_FNR'].std())
print('M TNR (Mean)', df['B_M_TNR'].mean(), df['B_M_TNR'].std())
print('M FPR (Mean)', df['B_M_FPR'].mean(), df['B_M_FPR'].std())
print('M TPR (Mean)', df['B_M_TPR'].mean(), df['B_M_TPR'].std())
print('      ')

# This gives us each run
Bal_No_FS_RFdf.to_csv('Bal_No_FS_RFdf.csv')

3.2.2 Logistic Regression - Balanced No Feature Selection
# RANDOM FOREST, No Feature Selection, Unbalanced Data
experiment_number = np.arange(100)

## RF Results
cols = ['B_Accuracy', 'B_FScore', 'B_ROC_AUC', 'B_Precision',
'B_Recall',

        'B_F_Accuracy', 'B_F_FScore', 'B_F_ROC_AUC', 'B_F_Precision',

```



```

'B_F_Recall',
  'B_F_FNs', 'B_F_TNs', 'B_F_FPs', 'B_F_TPs',
  'B_F_FNR', 'B_F_TNR', 'B_F_FPR', 'B_F_TPR',

  'B_M_Accuracy', 'B_M_FScore', 'B_M_ROC_AUC', 'B_M_Precision',
'B_M_Recall',
  'B_M_FNs', 'B_M_TNs', 'B_M_FPs', 'B_M_TPs',
  'B_M_FNR', 'B_M_TNR', 'B_M_FPR', 'B_M_TPR',
]

Bal_No_FS_LRdf = pd.DataFrame(columns=cols)
for i in tqdm(experiment_number):
    B_Accuracy, B_FScore, B_ROC_AUC, B_Precision, B_Recall,
    B_F_Accuracy, B_F_FScore, B_F_ROC_AUC, B_F_Precision, B_F_Recall,
    B_F_FNs, B_F_TNs, B_F_FPs, B_F_TPs, B_F_FNR, B_F_TNR, B_F_FPR,
    B_F_TPR, B_M_Accuracy, B_M_FScore, B_M_ROC_AUC, B_M_Precision,
    B_M_Recall, B_M_FNs, B_M_TNs, B_M_FPs, B_M_TPs, B_M_FNR, B_M_TNR,
    B_M_FPR, B_M_TPR = balanced_run(MX, My, FX, Fy, AllFeatures, LR)

    list = B_Accuracy, B_FScore, B_ROC_AUC, B_Precision, B_Recall,
    B_F_Accuracy, B_F_FScore, B_F_ROC_AUC, B_F_Precision, B_F_Recall,
    B_F_FNs, B_F_TNs, B_F_FPs, B_F_TPs, B_F_FNR, B_F_TNR, B_F_FPR,
    B_F_TPR, B_M_Accuracy, B_M_FScore, B_M_ROC_AUC, B_M_Precision,
    B_M_Recall, B_M_FNs, B_M_TNs, B_M_FPs, B_M_TPs, B_M_FNR, B_M_TNR,
    B_M_FPR, B_M_TPR
    Bal_No_FS_LRdf.loc[len(Bal_No_FS_LRdf)] = list

# reset the name of the df so can make multiple outputs with same code
df = Bal_No_FS_LRdf
print('All Accuracy (Mean)', df['B_Accuracy'].mean(), 'SD',
df['B_Accuracy'].std())
print('All FScore (Mean)', df['B_FScore'].mean(), 'SD',
df['B_FScore'].std())
print('All ROC_AUC (Mean)', df['B_ROC_AUC'].mean(), 'SD',
df['B_ROC_AUC'].std())
print('All Precision (Mean)', df['B_Precision'].mean(), 'SD',
df['B_Precision'].std())
print('All Recall (Mean)', df['B_Recall'].mean(), 'SD',
df['B_Recall'].std())
print('

')

print('F Accuracy (Mean)', df['B_F_Accuracy'].mean(), 'SD',
df['B_F_Accuracy'].std())
print('F FScore (Mean)', df['B_F_FScore'].mean(), 'SD',
df['B_F_FScore'].std())
print('F ROC_AUC (Mean)', df['B_F_ROC_AUC'].mean(),
df['B_F_ROC_AUC'].std())
print('F Precision (Mean)', df['B_F_Precision'].mean(),
df['B_F_Precision'].std())
print('F Recall (Mean)', df['B_F_Recall'].mean(),

```

```

df['B_F_Recall'].std())

print('F FNR (Mean)', df['B_F_FNR'].mean(), df['B_F_FNR'].std())
print('F TNR (Mean)', df['B_F_TNR'].mean(), df['B_F_TNR'].std())
print('F FPR (Mean)', df['B_F_FPR'].mean(), df['B_F_FPR'].std())
print('F TPR (Mean)', df['B_F_TPR'].mean(), df['B_F_TPR'].std())
print('')

print('M Accuracy (Mean)', df['B_M_Accuracy'].mean(),
df['B_M_Accuracy'].std())
print('M FScore (Mean)', df['B_M_FScore'].mean(),
df['B_M_FScore'].std())
print('M ROC_AUC (Mean)', df['B_M_ROC_AUC'].mean(),
df['B_M_ROC_AUC'].std())
print('M Precision (Mean)', df['B_M_Precision'].mean(),
df['B_M_Precision'].std())
print('M Recall (Mean)', df['B_M_Recall'].mean(),
df['B_M_Recall'].std())

print('M FNR (Mean)', df['B_M_FNR'].mean(), df['B_M_FNR'].std())
print('M TNR (Mean)', df['B_M_TNR'].mean(), df['B_M_TNR'].std())
print('M FPR (Mean)', df['B_M_FPR'].mean(), df['B_M_FPR'].std())
print('M TPR (Mean)', df['B_M_TPR'].mean(), df['B_M_TPR'].std())
print('')

# This gives us each run
Bal_No_FS_LRdf.to_csv('Bal_No_FS_LRdf.csv')

3.2.3 Support Vector Machine - Balanced No Feature Selection
experiment_number = np.arange(100)
cols = ['B_Accuracy', 'B_FScore', 'B_ROC_AUC', 'B_Precision',
'B_Recall',

        'B_F_Accuracy', 'B_F_FScore', 'B_F_ROC_AUC', 'B_F_Precision',
'B_F_Recall',
        'B_F_FNs', 'B_F_TNs', 'B_F_FPs', 'B_F_TPs',
        'B_F_FNR', 'B_F_TNR', 'B_F_FPR', 'B_F_TPR',

        'B_M_Accuracy', 'B_M_FScore', 'B_M_ROC_AUC', 'B_M_Precision',
'B_M_Recall',
        'B_M_FNs', 'B_M_TNs', 'B_M_FPs', 'B_M_TPs',
        'B_M_FNR', 'B_M_TNR', 'B_M_FPR', 'B_M_TPR',
]

Bal_No_FS_SVMdf = pd.DataFrame(columns=cols)
for i in tqdm(experiment_number):
    B_Accuracy, B_FScore, B_ROC_AUC, B_Precision, B_Recall,
    B_F_Accuracy, B_F_FScore, B_F_ROC_AUC, B_F_Precision, B_F_Recall,
    B_F_FNs, B_F_TNs, B_F_FPs, B_F_TPs, B_F_FNR, B_F_TNR, B_F_FPR,
    B_F_TPR, B_M_Accuracy, B_M_FScore, B_M_ROC_AUC, B_M_Precision,

```

```
B_M_Recall, B_M_FNs, B_M_TNs, B_M_FPs, B_M_TPs, B_M_FNR, B_M_TNR,
B_M_FPR, B_M_TPR = balanced_run(MX, My, FX, Fy, AllFeatures, SVM)
```

```
list = B_Accuracy, B_FScore, B_ROC_AUC, B_Precision, B_Recall,
B_F_Accuracy, B_F_FScore, B_F_ROC_AUC, B_F_Precision, B_F_Recall,
B_F_FNs, B_F_TNs, B_F_FPs, B_F_TPs, B_F_FNR, B_F_TNR, B_F_FPR,
B_F_TPR, B_M_Accuracy, B_M_FScore, B_M_ROC_AUC, B_M_Precision,
B_M_Recall, B_M_FNs, B_M_TNs, B_M_FPs, B_M_TPs, B_M_FNR, B_M_TNR,
B_M_FPR, B_M_TPR
Bal_No_FS_SVMdf.loc[len(Bal_No_FS_SVMdf)] = list
```

```
# reset the name of the df so can make multiple outputs with same code
```

```
df = Bal_No_FS_SVMdf
print('All Accuracy (Mean)', df['B_Accuracy'].mean(), 'SD',
df['B_Accuracy'].std())
print('All FScore (Mean)', df['B_FScore'].mean(), 'SD',
df['B_FScore'].std())
print('All ROC_AUC (Mean)', df['B_ROC_AUC'].mean(), 'SD',
df['B_ROC_AUC'].std())
print('All Precision (Mean)', df['B_Precision'].mean(), 'SD',
df['B_Precision'].std())
print('All Recall (Mean)', df['B_Recall'].mean(), 'SD',
df['B_Recall'].std())
print('')

print('F Accuracy (Mean)', df['B_F_Accuracy'].mean(), 'SD',
df['B_F_Accuracy'].std())
print('F FScore (Mean)', df['B_F_FScore'].mean(), 'SD',
df['B_F_FScore'].std())
print('F ROC_AUC (Mean)', df['B_F_ROC_AUC'].mean(),
df['B_F_ROC_AUC'].std())
print('F Precision (Mean)', df['B_F_Precision'].mean(),
df['B_F_Precision'].std())
print('F Recall (Mean)', df['B_F_Recall'].mean(),
df['B_F_Recall'].std())

print('F FNR (Mean)', df['B_F_FNR'].mean(), df['B_F_FNR'].std())
print('F TNR (Mean)', df['B_F_TNR'].mean(), df['B_F_TNR'].std())
print('F FPR (Mean)', df['B_F_FPR'].mean(), df['B_F_FPR'].std())
print('F TPR (Mean)', df['B_F_TPR'].mean(), df['B_F_TPR'].std())
print('')

print('M Accuracy (Mean)', df['B_M_Accuracy'].mean(),
df['B_M_Accuracy'].std())
print('M FScore (Mean)', df['B_M_FScore'].mean(),
df['B_M_FScore'].std())
print('M ROC_AUC (Mean)', df['B_M_ROC_AUC'].mean(),
df['B_M_ROC_AUC'].std())
print('M Precision (Mean)', df['B_M_Precision'].mean(),
df['B_M_Precision'].std())
```

```
print('M Recall (Mean)', df['B_M_Recall'].mean(),
df['B_M_Recall'].std())

print('M FNR (Mean)', df['B_M_FNR'].mean(), df['B_M_FNR'].std())
print('M TNR (Mean)', df['B_M_TNR'].mean(), df['B_M_TNR'].std())
print('M FPR (Mean)', df['B_M_FPR'].mean(), df['B_M_FPR'].std())
print('M TPR (Mean)', df['B_M_TPR'].mean(), df['B_M_TPR'].std())
print('')
```

```
# This gives us each run
```

```
Bal_No_FS_SVMdf.to_csv('Bal_No_FS_SVMdf.csv')
```

3.2.4 Gaussian NB - Balanced No Feature Selection

```
experiment_number = np.arange(100)
```

```
## RF Results
```

```
cols = ['B_Accuracy', 'B_FScore', 'B_ROC_AUC', 'B_Precision',
'B_Recall',

        'B_F_Accuracy', 'B_F_FScore', 'B_F_ROC_AUC', 'B_F_Precision',
'B_F_Recall',
        'B_F_FNs', 'B_F_TNs', 'B_F_FPs', 'B_F_TPs',
        'B_F_FNR', 'B_F_TNR', 'B_F_FPR', 'B_F_TPR',

        'B_M_Accuracy', 'B_M_FScore', 'B_M_ROC_AUC', 'B_M_Precision',
'B_M_Recall',
        'B_M_FNs', 'B_M_TNs', 'B_M_FPs', 'B_M_TPs',
        'B_M_FNR', 'B_M_TNR', 'B_M_FPR', 'B_M_TPR',
]
```

```
Bal_No_FS_NBdf = pd.DataFrame(columns=cols)
```

```
for i in tqdm(experiment_number):
```

```
    B_Accuracy, B_FScore, B_ROC_AUC, B_Precision, B_Recall,
    B_F_Accuracy, B_F_FScore, B_F_ROC_AUC, B_F_Precision, B_F_Recall,
    B_F_FNs, B_F_TNs, B_F_FPs, B_F_TPs, B_F_FNR, B_F_TNR, B_F_FPR,
    B_F_TPR, B_M_Accuracy, B_M_FScore, B_M_ROC_AUC, B_M_Precision,
    B_M_Recall, B_M_FNs, B_M_TNs, B_M_FPs, B_M_TPs, B_M_FNR, B_M_TNR,
    B_M_FPR, B_M_TPR = balanced_run(MX, My, FX, Fy, AllFeatures, NB)
```

```
    list = B_Accuracy, B_FScore, B_ROC_AUC, B_Precision, B_Recall,
    B_F_Accuracy, B_F_FScore, B_F_ROC_AUC, B_F_Precision, B_F_Recall,
    B_F_FNs, B_F_TNs, B_F_FPs, B_F_TPs, B_F_FNR, B_F_TNR, B_F_FPR,
    B_F_TPR, B_M_Accuracy, B_M_FScore, B_M_ROC_AUC, B_M_Precision,
    B_M_Recall, B_M_FNs, B_M_TNs, B_M_FPs, B_M_TPs, B_M_FNR, B_M_TNR,
    B_M_FPR, B_M_TPR
```

```
    Bal_No_FS_NBdf.loc[len(Bal_No_FS_NBdf)] = list
```

```
# reset the name of the df so can make multiple outputs with same code
```

```
df = Bal_No_FS_NBdf
```

```
print('All Accuracy (Mean)', df['B_Accuracy'].mean(), 'SD',
df['B_Accuracy'].std())
print('All FScore (Mean)', df['B_FScore'].mean(), 'SD',
df['B_FScore'].std())
print('All ROC_AUC (Mean)', df['B_ROC_AUC'].mean(), 'SD',
df['B_ROC_AUC'].std())
print('All Precision (Mean)', df['B_Precision'].mean(), 'SD',
df['B_Precision'].std())
print('All Recall (Mean)', df['B_Recall'].mean(), 'SD',
df['B_Recall'].std())
print('
')

print('F Accuracy (Mean)', df['B_F_Accuracy'].mean(), 'SD',
df['B_F_Accuracy'].std())
print('F FScore (Mean)', df['B_F_FScore'].mean(), 'SD',
df['B_F_FScore'].std())
print('F ROC_AUC (Mean)', df['B_F_ROC_AUC'].mean(),
df['B_F_ROC_AUC'].std())
print('F Precision (Mean)', df['B_F_Precision'].mean(),
df['B_F_Precision'].std())
print('F Recall (Mean)', df['B_F_Recall'].mean(),
df['B_F_Recall'].std())

print('F FNR (Mean)', df['B_F_FNR'].mean(), df['B_F_FNR'].std())
print('F TNR (Mean)', df['B_F_TNR'].mean(), df['B_F_TNR'].std())
print('F FPR (Mean)', df['B_F_FPR'].mean(), df['B_F_FPR'].std())
print('F TPR (Mean)', df['B_F_TPR'].mean(), df['B_F_TPR'].std())
print('
')

print('M Accuracy (Mean)', df['B_M_Accuracy'].mean(),
df['B_M_Accuracy'].std())
print('M FScore (Mean)', df['B_M_FScore'].mean(),
df['B_M_FScore'].std())
print('M ROC_AUC (Mean)', df['B_M_ROC_AUC'].mean(),
df['B_M_ROC_AUC'].std())
print('M Precision (Mean)', df['B_M_Precision'].mean(),
df['B_M_Precision'].std())
print('M Recall (Mean)', df['B_M_Recall'].mean(),
df['B_M_Recall'].std())

print('M FNR (Mean)', df['B_M_FNR'].mean(), df['B_M_FNR'].std())
print('M TNR (Mean)', df['B_M_TNR'].mean(), df['B_M_TNR'].std())
print('M FPR (Mean)', df['B_M_FPR'].mean(), df['B_M_FPR'].std())
print('M TPR (Mean)', df['B_M_TPR'].mean(), df['B_M_TPR'].std())
print('
')

# This gives us each run
Bal_No_FS_NBdf.to_csv('Bal_No_FS_NBdf.csv')
```

Experiment 3.3.1 Unbalanced Training Data with Feature Selection

Feature Selection Experiment

1.

```
def FSunbalanced_run(X, y, FeatureSet, Classifier):
    AllX_train, AllX_test, Ally_train, Ally_test = train_test_split(X,
y, test_size=0.30)
    X_train = AllX_train[FeatureSet]
    X_test = AllX_test[FeatureSet]
```

Experiment specific hyperparameter tuning - Needed for parameters

```
depth=[2, 8, 16]
n_estimators = [64, 128, 256]
RFParams = dict(max_depth=depth, n_estimators=n_estimators)
LRParams = {"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}
kernel = ['linear']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale']
SVMparams_grid = dict(kernel=kernel,C=C,gamma=gamma)
NBParams = {'var_smoothing': np.logspace(0,-9, num=100)}
```

have to move the GridSearch insive the loop, because SVC needs the kernel defined as linear

```
if Classifier == LR:
    params = LRParams
    grid_search = GridSearchCV(Classifier, params, cv=5,
scoring='accuracy')
    ClassifierParams = grid_search.fit(AllX_train, Ally_train)
```

```
if Classifier == RF:
    params = RFParams
    grid_search = GridSearchCV(Classifier, params, cv=5,
scoring='accuracy')
    ClassifierParams = grid_search.fit(AllX_train, Ally_train)
```

```
#if Classifier == SVM:
    # SVC has to have a linear kernel for RFE to work
    #grid_search = GridSearchCV(estimator=SVC(),
param_grid=SVMparams_grid, n_jobs=-1, cv=5, scoring='accuracy',
error_score=0)
```

```
if Classifier == NB:
    grid_search = GridSearchCV(estimator=GaussianNB(),
param_grid=NBParams, n_jobs=-1, cv=5, scoring='accuracy')
    ClassifierParams = grid_search.fit(AllX_train, Ally_train)
```

Grid search results for all giving best parameters

```
if Classifier == LR:
```



```
    #Modell = LogisticRegression(**ClassifierParams.best_params_)
    # Correction here to add in Feature Selection
    Modell =
RFE(LogisticRegression(**ClassifierParams.best_params_),
n_features_to_select=5, verbose=5)

    if Classifier == SVM:
        #Modell = SVC(**ClassifierParams.best_params_)
        Modell = PreppedModell

    if Classifier == RF:
        #Modell =
RandomForestClassifier(**ClassifierParams.best_params_)
        Modell =
RFE(RandomForestClassifier(**ClassifierParams.best_params_),
n_features_to_select=5, verbose=5)

    if Classifier == NB:
        #Modell = GaussianNB(**ClassifierParams.best_params_)
        Modell = RFE(GaussianNB(**ClassifierParams.best_params_),
n_features_to_select=250, verbose=5)

    Model = Modell.fit(AllX_train, Ally_train)
    Ally_pred = Model.predict(AllX_test)

    # Printing out results - same for each function
    Accuracy = (accuracy_score(Ally_test,Ally_pred))*100
    FScore = (f1_score(Ally_test, Ally_pred))*100
    ROC_AUC = roc_auc_score(Ally_test, Ally_pred, multi_class='ovo')
    Precision = precision_score(Ally_test, Ally_pred)
    Recall = recall_score(Ally_test, Ally_pred)

    Table = AllX_test
    Table['By_pred']=Ally_pred
    Table['By_true']=Ally_test
    Table['Accuracy']=Accuracy
    Table['FScore']=FScore
    Table['ROC_AUC']=ROC_AUC
    Table['Precision']=Precision
    Table['Recall']=Recall

    # Calculating true positives and negatives
    conditions = [
        Table['By_pred'].eq(0) & Table['By_true'].eq(0),
        Table['By_pred'].eq(0) & Table['By_true'].eq(1),
        Table['By_pred'].eq(1) & Table['By_true'].eq(0),
        Table['By_pred'].eq(1) & Table['By_true'].eq(1)
    ]
    choices = ['TN', 'FN', 'FP', 'TP']
    Table['Result'] = np.select(conditions, choices, default=0)
```

```

FemaleDF = Table[Table['Gender']==0.0]
F_Accuracy = (accuracy_score(FemaleDF['By_true'],
FemaleDF['By_pred']))*100
F_FScore = (f1_score(FemaleDF['By_true'],
FemaleDF['By_pred']))*100
F_ROC_AUC = roc_auc_score(FemaleDF['By_true'],
FemaleDF['By_pred'], multi_class='ovo')
F_Precision = precision_score(FemaleDF['By_true'],
FemaleDF['By_pred'])
F_Recall = recall_score(FemaleDF['By_true'], FemaleDF['By_pred'])

F_FNs = len(FemaleDF[FemaleDF['Result']=='FN'])
F_TNs = len(FemaleDF[FemaleDF['Result']=='TN'])
F_FPs = len(FemaleDF[FemaleDF['Result']=='FP'])
F_TPs = len(FemaleDF[FemaleDF['Result']=='TP'])
F_All = FemaleDF['Result'].sum()

# ** EDITING IN HERE, TRUE VALUES FOR FALSE POSITIVE ETC
F_FalseNegativeRate = (F_FNs/(F_FNs+F_TPs))*100
F_TrueNegativeRate = (F_TNs/(F_TNs+F_FPs))*100
F_FalsePositiveRate = (F_FPs/(F_FPs+F_TNs))*100
F_TruePositiveRate = (F_TPs/(F_TPs+F_FNs))*100

# Males
MaleDF = Table[Table['Gender']==1.0]
M_Accuracy = (accuracy_score(MaleDF['By_true'],
MaleDF['By_pred']))*100
M_FScore = (f1_score(MaleDF['By_true'], MaleDF['By_pred']))*100
M_ROC_AUC = roc_auc_score(MaleDF['By_true'], MaleDF['By_pred'],
multi_class='ovo')
M_Precision = precision_score(MaleDF['By_true'],
MaleDF['By_pred'])
M_Recall = recall_score(MaleDF['By_true'], MaleDF['By_pred'])

M_FNs = len(MaleDF[MaleDF['Result']=='FN'])
M_TNs = len(MaleDF[MaleDF['Result']=='TN'])
M_FPs = len(MaleDF[MaleDF['Result']=='FP'])
M_TPs = len(MaleDF[MaleDF['Result']=='TP'])
M_All = MaleDF['Result'].sum()

M_FalseNegativeRate = (M_FNs/(M_FNs+M_TPs))*100
M_TrueNegativeRate = (M_TNs/(M_TNs+M_FPs))*100
M_FalsePositiveRate = (M_FPs/(M_FPs+M_TNs))*100
M_TruePositiveRate = (M_TPs/(M_TPs+M_FNs))*100
return Accuracy, FScore, ROC_AUC, Precision, Recall, F_Accuracy,
F_FScore, F_ROC_AUC, F_Precision, F_Recall, F_FNs, F_TNs, F_FPs,
F_TPs, F_FalseNegativeRate, F_TrueNegativeRate, F_FalsePositiveRate,
F_TruePositiveRate, M_Accuracy, M_FScore, M_ROC_AUC, M_Precision,
M_Recall, M_FNs, M_TNs, M_FPs, M_TPs, M_FalseNegativeRate,

```

M_TrueNegativeRate, M_FalsePositiveRate, M_TruePositiveRate

Example - Can swap in the classifiers here
 FSunbalanced_run(AllX, Ally, AllFeatures, RF)

Analysis of differences in feature selection

Feature Selection Experiment

1.

```
def Analyse_FSunbalanced_run(X, y, FeatureSet, Classifier):
    AllX_train, AllX_test, Ally_train, Ally_test = train_test_split(X,
y, test_size=0.30)
    X_train = AllX_train[FeatureSet]
    X_test = AllX_test[FeatureSet]

    # Experiment specific hyperparameter tuning - Needed for
parameters
    depth=[2, 8, 16]
    n_estimators = [64, 128, 256]
    RFParams = dict(max_depth=depth, n_estimators=n_estimators)

    if Classifier == RF:
        params = RFParams
        grid_search = GridSearchCV(Classifier, params, cv=5,
scoring='accuracy')
        ClassifierParams = grid_search.fit(AllX_train, Ally_train)

    if Classifier == RF:
        #Model1 =
RandomForestClassifier(**ClassifierParams.best_params_)
        Model1 =
RFE(RandomForestClassifier(**ClassifierParams.best_params_),
n_features_to_select=5, verbose=5)

        Model = Model1.fit(AllX_train, Ally_train)
        Ally_pred = Model.predict(AllX_test)

    # Printing out results - same for each function
    Accuracy = (accuracy_score(Ally_test,Ally_pred))*100
    FScore = (f1_score(Ally_test, Ally_pred))*100
    ROC_AUC = roc_auc_score(Ally_test, Ally_pred, multi_class='ovo')
    Precision = precision_score(Ally_test, Ally_pred)
    Recall = recall_score(Ally_test, Ally_pred)

    Table = AllX_test
    Table['By_pred']=Ally_pred
    Table['By_true']=Ally_test
    Table['Accuracy']=Accuracy
```

```

Table['FScore']=FScore
Table['ROC_AUC']=ROC_AUC
Table['Precision']=Precision
Table['Recall']=Recall

# Calculating true positives and negatives
conditions = [
    Table['By_pred'].eq(0) & Table['By_true'].eq(0),
    Table['By_pred'].eq(0) & Table['By_true'].eq(1),
    Table['By_pred'].eq(1) & Table['By_true'].eq(0),
    Table['By_pred'].eq(1) & Table['By_true'].eq(1)
]
choices = ['TN', 'FN', 'FP', 'TP']
Table['Result'] = np.select(conditions, choices, default=0)

FemaleDF = Table[Table['Gender']==0.0]
F_Accuracy = (accuracy_score(FemaleDF['By_true'],
FemaleDF['By_pred']))*100
F_FScore = (f1_score(FemaleDF['By_true'],
FemaleDF['By_pred']))*100
F_ROC_AUC = roc_auc_score(FemaleDF['By_true'],
FemaleDF['By_pred'], multi_class='ovo')
F_Precision = precision_score(FemaleDF['By_true'],
FemaleDF['By_pred'])
F_Recall = recall_score(FemaleDF['By_true'], FemaleDF['By_pred'])

F_FNs = len(FemaleDF[FemaleDF['Result']=='FN'])
F_TNs = len(FemaleDF[FemaleDF['Result']=='TN'])
F_FPs = len(FemaleDF[FemaleDF['Result']=='FP'])
F_TPs = len(FemaleDF[FemaleDF['Result']=='TP'])
F_All = FemaleDF['Result'].sum()

# ** EDITING IN HERE, TRUE VALUES FOR FALSE POSITIVE ETC
F_FalseNegativeRate = (F_FNs/(F_FNs+F_TPs))*100
F_TrueNegativeRate = (F_TNs/(F_TNs+F_FPs))*100
F_FalsePositiveRate = (F_FPs/(F_FPs+F_TNs))*100
F_TruePositiveRate = (F_TPs/(F_TPs+F_FNs))*100

# Males
MaleDF = Table[Table['Gender']==1.0]
M_Accuracy = (accuracy_score(MaleDF['By_true'],
MaleDF['By_pred']))*100
M_FScore = (f1_score(MaleDF['By_true'], MaleDF['By_pred']))*100
M_ROC_AUC = roc_auc_score(MaleDF['By_true'], MaleDF['By_pred'],
multi_class='ovo')
M_Precision = precision_score(MaleDF['By_true'],
MaleDF['By_pred'])
M_Recall = recall_score(MaleDF['By_true'], MaleDF['By_pred'])

M_FNs = len(MaleDF[MaleDF['Result']=='FN'])

```

```

M_TNs = len(MaleDF[MaleDF['Result']=='TN'])
M_FPs = len(MaleDF[MaleDF['Result']=='FP'])
M_TPs = len(MaleDF[MaleDF['Result']=='TP'])
M_All = MaleDF['Result'].sum()

M_FalseNegativeRate = (M_FNs/(M_FNs+M_TPs))*100
M_TrueNegativeRate = (M_TNs/(M_TNs+M_FPs))*100
M_FalsePositiveRate = (M_FPs/(M_FPs+M_TNs))*100
M_TruePositiveRate = (M_TPs/(M_TPs+M_FNs))*100

rfe_features = AllX_train.columns[Model.support_]
print(rfe_features.sort_values())
return Accuracy, FScore, ROC_AUC, Precision, Recall, F_Accuracy,
F_FScore, F_ROC_AUC, F_Precision, F_Recall, F_FNs, F_TNs, F_FPs,
F_TPs, F_FalseNegativeRate, F_TrueNegativeRate, F_FalsePositiveRate,
F_TruePositiveRate, M_Accuracy, M_FScore, M_ROC_AUC, M_Precision,
M_Recall, M_FNs, M_TNs, M_FPs, M_TPs, M_FalseNegativeRate,
M_TrueNegativeRate, M_FalsePositiveRate, M_TruePositiveRate

## Example for feature Selection - RF as this has largest sex
disparity
Analyse_FSunbalanced_run(AllX, Ally, AllFeatures, RF)

Experiment 3.3.1 Random Forest (Unbalanced Training Data with Feature Selection)
experiment_number = np.arange(100)

## RF Results
cols = ['UB_Accuracy', 'UB_FScore', 'UB_ROC_AUC', 'UB_Precision',
'UB_Recall',

        'UB_F_Accuracy', 'UB_F_FScore', 'UB_F_ROC_AUC',
'UB_F_Precision', 'UB_F_Recall',
        'UB_F_FNs', 'UB_F_TNs', 'UB_F_FPs', 'UB_F_TPs',
        'UB_F_FNR', 'UB_F_TNR', 'UB_F_FPR', 'UB_F_TPR',

        'UB_M_Accuracy', 'UB_M_FScore', 'UB_M_ROC_AUC',
'UB_M_Precision', 'UB_M_Recall',
        'UB_M_FNs', 'UB_M_TNs', 'UB_M_FPs', 'UB_M_TPs',
        'UB_M_FNR', 'UB_M_TNR', 'UB_M_FPR', 'UB_M_TPR',
    ]

Unbal_W_FS_RFdf = pd.DataFrame(columns=cols)

# Scroll left here to see the function that defines it as a balanced
experiment

# NEEDED TO ENSURE THERE WAS AN INDENT

for i in tqdm(experiment_number):
    UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall,

```

```

UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall,
UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR, UB_F_TNR, UB_F_FPR,
UB_F_TPR, UB_M_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision,
UB_M_Recall, UB_M_FNs, UB_M_TNs, UB_M_FPs, UB_M_TPs, UB_M_FNR,
UB_M_TNR, UB_M_FPR, UB_M_TPR = FSunbalanced_run(AllX, Ally,
AllFeatures, RF)

```

```

list = UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision,
UB_Recall, UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision,
UB_F_Recall, UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR,
UB_F_TNR, UB_F_FPR, UB_F_TPR, UB_M_Accuracy, UB_M_FScore,
UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNs, UB_M_TNs,
UB_M_FPs, UB_M_TPs, UB_M_FNR, UB_M_TNR, UB_M_FPR, UB_M_TPR
Unbal_W_FS_RFdf.loc[len(Unbal_W_FS_RFdf)] = list

```

```

# reset the name of the df so can make multiple outputs with same code

```

```

df = Unbal_W_FS_RFdf
print('All Accuracy (Mean)', df['UB_Accuracy'].mean(), 'SD',
df['UB_Accuracy'].std())
print('All FScore (Mean)', df['UB_FScore'].mean(), 'SD',
df['UB_FScore'].std())
print('All ROC_AUC (Mean)', df['UB_ROC_AUC'].mean(), 'SD',
df['UB_ROC_AUC'].std())
print('All Precision (Mean)', df['UB_Precision'].mean(), 'SD',
df['UB_Precision'].std())
print('All Recall (Mean)', df['UB_Recall'].mean(), 'SD',
df['UB_Recall'].std())
print('')

print('F Accuracy (Mean)', df['UB_F_Accuracy'].mean(), 'SD',
df['UB_F_Accuracy'].std())
print('F FScore (Mean)', df['UB_F_FScore'].mean(), 'SD',
df['UB_F_FScore'].std())
print('F ROC_AUC (Mean)', df['UB_F_ROC_AUC'].mean(),
df['UB_F_ROC_AUC'].std())
print('F Precision (Mean)', df['UB_F_Precision'].mean(),
df['UB_F_Precision'].std())
print('F Recall (Mean)', df['UB_F_Recall'].mean(),
df['UB_F_Recall'].std())

print('F FNR (Mean)', df['UB_F_FNR'].mean(), df['UB_F_FNR'].std())
print('F TNR (Mean)', df['UB_F_TNR'].mean(), df['UB_F_TNR'].std())
print('F FPR (Mean)', df['UB_F_FPR'].mean(), df['UB_F_FPR'].std())
print('F TPR (Mean)', df['UB_F_TPR'].mean(), df['UB_F_TPR'].std())
print('')

```

```

print('M Accuracy (Mean)', df['UB_M_Accuracy'].mean(),
df['UB_M_Accuracy'].std())
print('M FScore (Mean)', df['UB_M_FScore'].mean(),

```



```

df['UB_M_FScore'].std()
print('M ROC_AUC (Mean)', df['UB_M_ROC_AUC'].mean(),
df['UB_M_ROC_AUC'].std())
print('M Precision (Mean)', df['UB_M_Precision'].mean(),
df['UB_M_Precision'].std())
print('M Recall (Mean)', df['UB_M_Recall'].mean(),
df['UB_M_Recall'].std())

print('M FNR (Mean)', df['UB_M_FNR'].mean(), df['UB_M_FNR'].std())
print('M TNR (Mean)', df['UB_M_TNR'].mean(), df['UB_M_TNR'].std())
print('M FPR (Mean)', df['UB_M_FPR'].mean(), df['UB_M_FPR'].std())
print('M TPR (Mean)', df['UB_M_TPR'].mean(), df['UB_M_TPR'].std())
print(' ')

print("Female/Male T Tests:")

# This gives us each run
Unbal_W_FS_RFdf.to_csv('Unbal_W_FS_RFdf.csv')

Experiment 3.1.3 Logistic Regression (Unbalanced Training Data with Feature Selection)
experiment_number = np.arange(100)

## RF Results
cols = ['UB_Accuracy', 'UB_FScore', 'UB_ROC_AUC', 'UB_Precision',
'UB_Recall',

        'UB_F_Accuracy', 'UB_F_FScore', 'UB_F_ROC_AUC',
'UB_F_Precision', 'UB_F_Recall',
        'UB_F_FNs', 'UB_F_TNs', 'UB_F_FPs', 'UB_F_TPs',
        'UB_F_FNR', 'UB_F_TNR', 'UB_F_FPR', 'UB_F_TPR',

        'UB_M_Accuracy', 'UB_M_FScore', 'UB_M_ROC_AUC',
'UB_M_Precision', 'UB_M_Recall',
        'UB_M_FNs', 'UB_M_TNs', 'UB_M_FPs', 'UB_M_TPs',
        'UB_M_FNR', 'UB_M_TNR', 'UB_M_FPR', 'UB_M_TPR',
]

Unbal_W_FS_LRdf = pd.DataFrame(columns=cols)

# Scroll left here to see the function that defines it as a balanced
experiment

# NEEDED TO ENSURE THERE WAS AN INDENT

for i in tqdm(experiment_number):
    UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall,
    UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall,
    UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR, UB_F_TNR, UB_F_FPR,

```

```

UB_F_TPR, UB_M_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision,
UB_M_Recall, UB_M_FNs, UB_M_TNs, UB_M_FPs, UB_M_TPs, UB_M_FNR,
UB_M_TNR, UB_M_FPR, UB_M_TPR = FSunbalanced_run(AllX, Ally,
AllFeatures, LR)

list = UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision,
UB_Recall, UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision,
UB_F_Recall, UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR,
UB_F_TNR, UB_F_FPR, UB_F_TPR, UB_M_Accuracy, UB_M_FScore,
UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNs, UB_M_TNs,
UB_M_FPs, UB_M_TPs, UB_M_TNR, UB_M_FPR, UB_M_TPR
Unbal_W_FS_LRdf.loc[len(Unbal_W_FS_LRdf)] = list

# reset the name of the df so can make multiple outputs with same code
df = Unbal_W_FS_LRdf
print('All Accuracy (Mean)', df['UB_Accuracy'].mean(), 'SD',
df['UB_Accuracy'].std())
print('All FScore (Mean)', df['UB_FScore'].mean(), 'SD',
df['UB_FScore'].std())
print('All ROC_AUC (Mean)', df['UB_ROC_AUC'].mean(), 'SD',
df['UB_ROC_AUC'].std())
print('All Precision (Mean)', df['UB_Precision'].mean(), 'SD',
df['UB_Precision'].std())
print('All Recall (Mean)', df['UB_Recall'].mean(), 'SD',
df['UB_Recall'].std())
print(' ')

print('F Accuracy (Mean)', df['UB_F_Accuracy'].mean(), 'SD',
df['UB_F_Accuracy'].std())
print('F FScore (Mean)', df['UB_F_FScore'].mean(), 'SD',
df['UB_F_FScore'].std())
print('F ROC_AUC (Mean)', df['UB_F_ROC_AUC'].mean(),
df['UB_F_ROC_AUC'].std())
print('F Precision (Mean)', df['UB_F_Precision'].mean(),
df['UB_F_Precision'].std())
print('F Recall (Mean)', df['UB_F_Recall'].mean(),
df['UB_F_Recall'].std())

print('F FNR (Mean)', df['UB_F_FNR'].mean(), df['UB_F_FNR'].std())
print('F TNR (Mean)', df['UB_F_TNR'].mean(), df['UB_F_TNR'].std())
print('F FPR (Mean)', df['UB_F_FPR'].mean(), df['UB_F_FPR'].std())
print('F TPR (Mean)', df['UB_F_TPR'].mean(), df['UB_F_TPR'].std())
print(' ')

print('M Accuracy (Mean)', df['UB_M_Accuracy'].mean(),
df['UB_M_Accuracy'].std())
print('M FScore (Mean)', df['UB_M_FScore'].mean(),
df['UB_M_FScore'].std())
print('M ROC_AUC (Mean)', df['UB_M_ROC_AUC'].mean(),

```

```

df['UB_M_ROC_AUC'].std())
print('M Precision (Mean)', df['UB_M_Precision'].mean(),
df['UB_M_Precision'].std())
print('M Recall (Mean)', df['UB_M_Recall'].mean(),
df['UB_M_Recall'].std())

print('M FNR (Mean)', df['UB_M_FNR'].mean(), df['UB_M_FNR'].std())
print('M TNR (Mean)', df['UB_M_TNR'].mean(), df['UB_M_TNR'].std())
print('M FPR (Mean)', df['UB_M_FPR'].mean(), df['UB_M_FPR'].std())
print('M TPR (Mean)', df['UB_M_TPR'].mean(), df['UB_M_TPR'].std())
print('          ')

print("Female/Male T Tests:")

# This gives us each run
Unbal_W_FS_LRdf.to_csv('Unbal_W_FS_LRdf.csv')

Experiment 3.3.3 Support Vector Machine (Unbalanced training data, with
Feature Selection)
# Prepping parameters for SVM

# SVM Parameters are tuned here due to the duration of the function
when done inside
ExMX_train, ExMX_test, ExMy_train, ExMy_test = train_test_split(MX,
My, test_size=0.30)
ExFX_train, ExFX_test, ExFy_train, ExFy_test = train_test_split(FX,
Fy, test_size=0.30)
EBX_train = ExMX_train.append(ExFX_train)
EBX_test = ExMX_test.append(ExFX_test)
EBy_train = ExMy_train.append(ExFy_train)
EBy_test = ExMy_test.append(ExFy_test)
EX_train = EBX_train[AllFeatures]
EX_test = EBX_test[AllFeatures]
kernel = ['linear']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale']
SVMparams_grid = dict(kernel=kernel,C=C,gamma=gamma)
grid_search = GridSearchCV(estimator=SVC(), param_grid=SVMparams_grid,
n_jobs=-1, cv=5, scoring='accuracy', error_score=0)
grid_search.fit(EBX_train, EBy_train)
SVCParams = grid_search.fit(EBX_train, EBy_train)
print(SVCParams)
SVCParams.best_params_
PreppedModel1 = RFE(SVC(C=50, gamma='scale', kernel='linear'),
n_features_to_select=5, verbose=5)
PreppedModel1.fit(EBX_train, EBy_train)
predictions = PreppedModel1.predict(EX_test)

```

```
experiment_number = np.arange(20)

## RF Results
cols = ['UB_Accuracy', 'UB_FScore', 'UB_ROC_AUC', 'UB_Precision',
        'UB_Recall',

        'UB_F_Accuracy', 'UB_F_FScore', 'UB_F_ROC_AUC',
        'UB_F_Precision', 'UB_F_Recall',
        'UB_F_FNs', 'UB_F_TNs', 'UB_F_FPs', 'UB_F_TPs',
        'UB_F_FNR', 'UB_F_TNR', 'UB_F_FPR', 'UB_F_TPR',

        'UB_M_Accuracy', 'UB_M_FScore', 'UB_M_ROC_AUC',
        'UB_M_Precision', 'UB_M_Recall',
        'UB_M_FNs', 'UB_M_TNs', 'UB_M_FPs', 'UB_M_TPs',
        'UB_M_FNR', 'UB_M_TNR', 'UB_M_FPR', 'UB_M_TPR',
        ]

Unbal_W_FS_SVMdf1 = pd.DataFrame(columns=cols)

# Scroll left here to see the function that defines it as a balanced
experiment

# NEEDED TO ENSURE THERE WAS AN INDENT

for i in tqdm(experiment_number):
    UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall,
    UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall,
    UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR, UB_F_TNR, UB_F_FPR,
    UB_F_TPR, UB_M_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision,
    UB_M_Recall, UB_M_FNs, UB_M_TNs, UB_M_FPs, UB_M_TPs, UB_M_FNR,
    UB_M_TNR, UB_M_FPR, UB_M_TPR = FSunbalanced_run(AllX, Ally,
    AllFeatures, SVM)

    list = UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision,
    UB_Recall, UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision,
    UB_F_Recall, UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR,
    UB_F_TNR, UB_F_FPR, UB_F_TPR, UB_M_Accuracy, UB_M_FScore,
    UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNs, UB_M_TNs,
    UB_M_FPs, UB_M_TPs, UB_M_FNR, UB_M_TNR, UB_M_FPR, UB_M_TPR
    Unbal_W_FS_SVMdf1.loc[len(Unbal_W_FS_SVMdf1)] = list

# reset the name of the df so can make multiple outputs with same code
df = Unbal_W_FS_SVMdf1
print('All Accuracy (Mean)', df['UB_Accuracy'].mean(), 'SD',
df['UB_Accuracy'].std())
print('All FScore (Mean)', df['UB_FScore'].mean(), 'SD',
df['UB_FScore'].std())
print('All ROC_AUC (Mean)', df['UB_ROC_AUC'].mean(), 'SD',
df['UB_ROC_AUC'].std())
```

```
print('All Precision (Mean)', df['UB_Precision'].mean(), 'SD',
df['UB_Precision'].std())
print('All Recall (Mean)', df['UB_Recall'].mean(), 'SD',
df['UB_Recall'].std())
print('

')

print('F Accuracy (Mean)', df['UB_F_Accuracy'].mean(), 'SD',
df['UB_F_Accuracy'].std())
print('F FScore (Mean)', df['UB_F_FScore'].mean(), 'SD',
df['UB_F_FScore'].std())
print('F ROC_AUC (Mean)', df['UB_F_ROC_AUC'].mean(),
df['UB_F_ROC_AUC'].std())
print('F Precision (Mean)', df['UB_F_Precision'].mean(),
df['UB_F_Precision'].std())
print('F Recall (Mean)', df['UB_F_Recall'].mean(),
df['UB_F_Recall'].std())

print('F FNR (Mean)', df['UB_F_FNR'].mean(), df['UB_F_FNR'].std())
print('F TNR (Mean)', df['UB_F_TNR'].mean(), df['UB_F_TNR'].std())
print('F FPR (Mean)', df['UB_F_FPR'].mean(), df['UB_F_FPR'].std())
print('F TPR (Mean)', df['UB_F_TPR'].mean(), df['UB_F_TPR'].std())
print('

')

print('M Accuracy (Mean)', df['UB_M_Accuracy'].mean(),
df['UB_M_Accuracy'].std())
print('M FScore (Mean)', df['UB_M_FScore'].mean(),
df['UB_M_FScore'].std())
print('M ROC_AUC (Mean)', df['UB_M_ROC_AUC'].mean(),
df['UB_M_ROC_AUC'].std())
print('M Precision (Mean)', df['UB_M_Precision'].mean(),
df['UB_M_Precision'].std())
print('M Recall (Mean)', df['UB_M_Recall'].mean(),
df['UB_M_Recall'].std())

print('M FNR (Mean)', df['UB_M_FNR'].mean(), df['UB_M_FNR'].std())
print('M TNR (Mean)', df['UB_M_TNR'].mean(), df['UB_M_TNR'].std())
print('M FPR (Mean)', df['UB_M_FPR'].mean(), df['UB_M_FPR'].std())
print('M TPR (Mean)', df['UB_M_TPR'].mean(), df['UB_M_TPR'].std())
print('

')

# This gives us each run
Unbal_W_FS_SVMdf1.to_csv('Unbal_W_FS_SVMdf1.csv')
```

Experiment 3.3.4 Gaussian Naive Bayes (Unbalanced training data, with Feature Selection)

```
experiment_number = np.arange(100)
```

```
## RF Results
```

```

cols = ['UB_Accuracy', 'UB_FScore', 'UB_ROC_AUC', 'UB_Precision',
'UB_Recall',

        'UB_F_Accuracy', 'UB_F_FScore', 'UB_F_ROC_AUC',
'UB_F_Precision', 'UB_F_Recall',
        'UB_F_FNs', 'UB_F_TNs', 'UB_F_FPs', 'UB_F_TPs',
        'UB_F_FNR', 'UB_F_TNR', 'UB_F_FPR', 'UB_F_TPR',

        'UB_M_Accuracy', 'UB_M_FScore', 'UB_M_ROC_AUC',
'UB_M_Precision', 'UB_M_Recall',
        'UB_M_FNs', 'UB_M_TNs', 'UB_M_FPs', 'UB_M_TPs',
        'UB_M_FNR', 'UB_M_TNR', 'UB_M_FPR', 'UB_M_TPR',
    ]

Unbal_W_FS_GNBdf = pd.DataFrame(columns=cols)

# Scroll left here to see the function that defines it as a balanced
# experiment

# NEEDED TO ENSURE THERE WAS AN INDENT

for i in tqdm(experiment_number):
    UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall,
    UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall,
    UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR, UB_F_TNR, UB_F_FPR,
    UB_F_TPR, UB_M_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision,
    UB_M_Recall, UB_M_FNs, UB_M_TNs, UB_M_FPs, UB_M_TPs, UB_M_FNR,
    UB_M_TNR, UB_M_FPR, UB_M_TPR = FSunbalanced_run(AllX, Ally,
    AllFeatures, NB)

    list = UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision,
    UB_Recall, UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision,
    UB_F_Recall, UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR,
    UB_F_TNR, UB_F_FPR, UB_F_TPR, UB_M_Accuracy, UB_M_FScore,
    UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNs, UB_M_TNs,
    UB_M_FPs, UB_M_TPs, UB_M_FNR, UB_M_TNR, UB_M_FPR, UB_M_TPR
    Unbal_W_FS_GNBdf.loc[len(Unbal_W_FS_GNBdf)] = list

# reset the name of the df so can make multiple outputs with same code
df = Unbal_W_FS_GNBdf
print('All Accuracy (Mean)', df['UB_Accuracy'].mean(), 'SD',
df['UB_Accuracy'].std())
print('All FScore (Mean)', df['UB_FScore'].mean(), 'SD',
df['UB_FScore'].std())
print('All ROC_AUC (Mean)', df['UB_ROC_AUC'].mean(), 'SD',
df['UB_ROC_AUC'].std())
print('All Precision (Mean)', df['UB_Precision'].mean(), 'SD',
df['UB_Precision'].std())
print('All Recall (Mean)', df['UB_Recall'].mean(), 'SD',
df['UB_Recall'].std())

```



```

print('          ')

print('F Accuracy (Mean)', df['UB_F_Accuracy'].mean(), 'SD',
df['UB_F_Accuracy'].std())
print('F FScore (Mean)', df['UB_F_FScore'].mean(), 'SD',
df['UB_F_FScore'].std())
print('F ROC_AUC (Mean)', df['UB_F_ROC_AUC'].mean(),
df['UB_F_ROC_AUC'].std())
print('F Precision (Mean)', df['UB_F_Precision'].mean(),
df['UB_F_Precision'].std())
print('F Recall (Mean)', df['UB_F_Recall'].mean(),
df['UB_F_Recall'].std())

print('F FNR (Mean)', df['UB_F_FNR'].mean(), df['UB_F_FNR'].std())
print('F TNR (Mean)', df['UB_F_TNR'].mean(), df['UB_F_TNR'].std())
print('F FPR (Mean)', df['UB_F_FPR'].mean(), df['UB_F_FPR'].std())
print('F TPR (Mean)', df['UB_F_TPR'].mean(), df['UB_F_TPR'].std())
print('          ')

print('M Accuracy (Mean)', df['UB_M_Accuracy'].mean(),
df['UB_M_Accuracy'].std())
print('M FScore (Mean)', df['UB_M_FScore'].mean(),
df['UB_M_FScore'].std())
print('M ROC_AUC (Mean)', df['UB_M_ROC_AUC'].mean(),
df['UB_M_ROC_AUC'].std())
print('M Precision (Mean)', df['UB_M_Precision'].mean(),
df['UB_M_Precision'].std())
print('M Recall (Mean)', df['UB_M_Recall'].mean(),
df['UB_M_Recall'].std())

print('M FNR (Mean)', df['UB_M_FNR'].mean(), df['UB_M_FNR'].std())
print('M TNR (Mean)', df['UB_M_TNR'].mean(), df['UB_M_TNR'].std())
print('M FPR (Mean)', df['UB_M_FPR'].mean(), df['UB_M_FPR'].std())
print('M TPR (Mean)', df['UB_M_TPR'].mean(), df['UB_M_TPR'].std())
print('          ')

# This gives us each run
Unbal_W_FS_GNBdf.to_csv('Unbal_W_FS_GNBdf.csv')

```

Experiment 3.4.1 Balanced Training Data with Feature Selection

First - Preparing SVM model due to speed of RFE and Grid Search when inside the function - tends to crash the operation

```

# SVM Parameters are tuned here due to the duration of the function
when done inside
ExMX_train, ExMX_test, ExMy_train, ExMy_test = train_test_split(MX,
My, test_size=0.30)
ExFX_train, ExFX_test, ExFy_train, ExFy_test = train_test_split(FX,

```

```
Fy, test_size=0.30)
EBX_train = ExMX_train.append(ExFX_train)
EBX_test = ExMX_test.append(ExFX_test)
EBy_train = ExMy_train.append(ExFy_train)
EBy_test = ExMy_test.append(ExFy_test)
EX_train = EBX_train[AllFeatures]
EX_test = EBX_test[AllFeatures]

kernel = ['linear']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale']
SVMparams_grid = dict(kernel=kernel,C=C,gamma=gamma)
grid_search = GridSearchCV(estimator=SVC(), param_grid=SVMparams_grid,
n_jobs=-1, cv=5, scoring='accuracy', error_score=0)
grid_search.fit(EBX_train, EBy_train)
SVCParams = grid_search.fit(EBX_train, EBy_train)
print(SVCParams)

SVCParams.best_params_

PreppedModel1 = RFE(SVC(C=1, gamma='scale', kernel='linear'),
n_features_to_select=5, verbose=5)

PreppedModel1.fit(EBX_train, EBy_train)
predictions = PreppedModel1.predict(EX_test)

predictions

# 4. Defining Function, Balanced data with feature selection

def FSbalanced_run(MX, My, FX, Fy, FeatureSet, Classifier):
    MX_train, MX_test, My_train, My_test = train_test_split(MX, My,
test_size=0.30)
    FX_train, FX_test, Fy_train, Fy_test = train_test_split(FX, Fy,
test_size=0.30)

    # Redefine X train so its just desired features (i.e. No Sex)
    BX_train = MX_train.append(FX_train)
    BX_test = MX_test.append(FX_test)
    By_train = My_train.append(Fy_train)
    By_test = My_test.append(Fy_test)

    X_train = BX_train[FeatureSet]
    X_test = BX_test[FeatureSet]

    # Experiment specific hyperparameter tuning - Needed for
parameters
    depth=[2, 8, 16]
    n_estimators = [64, 128, 256]
    RFParams = dict(max_depth=depth, n_estimators=n_estimators)
    LRParams = {"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}
```

```
NBParams = {'var_smoothing': np.logspace(0,-9, num=100)}

# Name of Parameters
# have to move the GridSearch insive the loop, because SVC needs
the kernel defined as linear
if Classifier == LR:
    params = LRParams
    grid_search = GridSearchCV(Classifier, params, cv=5,
scoring='accuracy')
    ClassifierParams = grid_search.fit(X_train, By_train)

if Classifier == RF:
    params = RFParams
    grid_search = GridSearchCV(Classifier, params, cv=5,
scoring='accuracy')
    ClassifierParams = grid_search.fit(X_train, By_train)

#if Classifier == SVM:
# SVC has to have a linear kernel for RFE to work
#grid_search = GridSearchCV(estimator=SVC(),
param_grid=SVMparams_grid, n_jobs=-1, cv=5, scoring='accuracy',
error_score=0)

if Classifier == NB:
    grid_search = GridSearchCV(estimator=GaussianNB(),
param_grid=NBParams, n_jobs=-1, cv=5, scoring='accuracy')
    ClassifierParams = grid_search.fit(X_train, By_train)
# Grid search results for all giving best parameters

if Classifier == LR:
    #Modell = LogisticRegression(**ClassifierParams.best_params_)
    # Correction here to add in Feature Selection
    Modell =
RFE(LogisticRegression(**ClassifierParams.best_params_),
n_features_to_select=5, verbose=5)

if Classifier == SVM:
    #Modell = SVC(**ClassifierParams.best_params_)
    Modell = PreppedModell

if Classifier == RF:
    #Modell =
RandomForestClassifier(**ClassifierParams.best_params_)
    Modell =
RFE(RandomForestClassifier(**ClassifierParams.best_params_),
n_features_to_select=5, verbose=5)

if Classifier == NB:
    #Modell = GaussianNB(**ClassifierParams.best_params_)
    Modell = RFE(GaussianNB(**ClassifierParams.best_params_),
```

```
n_features_to_select=250, verbose=5)

Model = Model1.fit(X_train, By_train)
Ally_pred = Model.predict(X_test)

# Printing out results - same for each function
Accuracy = (accuracy_score(By_test,Ally_pred))*100
FScore = (f1_score(By_test, Ally_pred))*100
ROC_AUC = roc_auc_score(By_test, Ally_pred, multi_class='ovo')
Precision = precision_score(By_test, Ally_pred)
Recall = recall_score(By_test, Ally_pred)

Table = X_test
Table['By_pred']=Ally_pred
Table['By_true']=By_test
Table['Accuracy']=Accuracy
Table['FScore']=FScore
Table['ROC_AUC']=ROC_AUC
Table['Precision']=Precision
Table['Recall']=Recall

# Calculating true positives and negatives
conditions = [
    Table['By_pred'].eq(0) & Table['By_true'].eq(0),
    Table['By_pred'].eq(0) & Table['By_true'].eq(1),
    Table['By_pred'].eq(1) & Table['By_true'].eq(0),
    Table['By_pred'].eq(1) & Table['By_true'].eq(1)
]
choices = ['TN', 'FN', 'FP', 'TP']
Table['Result'] = np.select(conditions, choices, default=0)

FemaleDF = Table[Table['Gender']==0.0]
F_Accuracy = (accuracy_score(FemaleDF['By_true'],
FemaleDF['By_pred']))*100
F_FScore = (f1_score(FemaleDF['By_true'],
FemaleDF['By_pred']))*100
F_ROC_AUC = roc_auc_score(FemaleDF['By_true'],
FemaleDF['By_pred'], multi_class='ovo')
F_Precision = precision_score(FemaleDF['By_true'],
FemaleDF['By_pred'])
F_Recall = recall_score(FemaleDF['By_true'], FemaleDF['By_pred'])

F_FNs = len(FemaleDF[FemaleDF['Result']=='FN'])
F_TNs = len(FemaleDF[FemaleDF['Result']=='TN'])
F_FPs = len(FemaleDF[FemaleDF['Result']=='FP'])
F_TPs = len(FemaleDF[FemaleDF['Result']=='TP'])
F_All = FemaleDF['Result'].sum()

# ** EDITING IN HERE, TRUE VALUES FOR FALSE POSITIVE ETC
F_FalseNegativeRate = (F_FNs/(F_FNs+F_TPs))*100
```

```

F_TrueNegativeRate = (F_TNs/(F_TNs+F_FPs))*100
F_FalsePositiveRate = (F_FPs/(F_FPs+F_TNs))*100
F_TruePositiveRate = (F_TPs/(F_TPs+F_FNs))*100

# Males
MaleDF = Table[Table['Gender']==1.0]
M_Accuracy = (accuracy_score(MaleDF['By_true'],
MaleDF['By_pred']))*100
M_FScore = (f1_score(MaleDF['By_true'], MaleDF['By_pred']))*100
M_ROC_AUC = roc_auc_score(MaleDF['By_true'], MaleDF['By_pred'],
multi_class='ovo')
M_Precision = precision_score(MaleDF['By_true'],
MaleDF['By_pred'])
M_Recall = recall_score(MaleDF['By_true'], MaleDF['By_pred'])

M_FNs = len(MaleDF[MaleDF['Result']=='FN'])
M_TNs = len(MaleDF[MaleDF['Result']=='TN'])
M_FPs = len(MaleDF[MaleDF['Result']=='FP'])
M_TPs = len(MaleDF[MaleDF['Result']=='TP'])
M_All = MaleDF['Result'].sum()

M_FalseNegativeRate = (M_FNs/(M_FNs+M_TPs))*100
M_TrueNegativeRate = (M_TNs/(M_TNs+M_FPs))*100
M_FalsePositiveRate = (M_FPs/(M_FPs+M_TNs))*100
M_TruePositiveRate = (M_TPs/(M_TPs+M_FNs))*100
return Accuracy, FScore, ROC_AUC, Precision, Recall, F_Accuracy,
F_FScore, F_ROC_AUC, F_Precision, F_Recall, F_FNs, F_TNs, F_FPs,
F_TPs, F_FalseNegativeRate, F_TrueNegativeRate, F_FalsePositiveRate,
F_TruePositiveRate, M_Accuracy, M_FScore, M_ROC_AUC, M_Precision,
M_Recall, M_FNs, M_TNs, M_FPs, M_TPs, M_FalseNegativeRate,
M_TrueNegativeRate, M_FalsePositiveRate, M_TruePositiveRate

# testing experiment
FSbalanced_run(MX, My, FX, Fy, AllFeatures, SVM)

```

Analysing Feature Selection with RF

Example for feature Selection - RF as this has largest sex disparity

Feature Selection Experiment

1.

```

def Analyse_FSbalanced_run(MX, My, FX, Fy, FeatureSet, Classifier):
    MX_train, MX_test, My_train, My_test = train_test_split(MX, My,
test_size=0.30)
    FX_train, FX_test, Fy_train, Fy_test = train_test_split(FX, Fy,
test_size=0.30)

```

Redefine X train so its just desired features (i.e. No Sex)

```

BX_train = MX_train.append(FX_train)
BX_test = MX_test.append(FX_test)

```

```
By_train = My_train.append(Fy_train)
By_test = My_test.append(Fy_test)

X_train = BX_train[FeatureSet]
X_test = BX_test[FeatureSet]

# Experiment specific hyperparameter tuning - Needed for
parameters
depth=[2, 8, 16]
n_estimators = [64, 128, 256]
RFParams = dict(max_depth=depth, n_estimators=n_estimators)
if Classifier == RF:
    params = RFParams
    grid_search = GridSearchCV(Classifier, params, cv=5,
scoring='accuracy')
    ClassifierParams = grid_search.fit(X_train, By_train)

#if Classifier == SVM:
# SVC has to have a linear kernel for RFE to work
#grid_search = GridSearchCV(estimator=SVC(),
param_grid=SVMparams_grid, n_jobs=-1, cv=5, scoring='accuracy',
error_score=0)

if Classifier == RF:
    #Modell =
RandomForestClassifier(**ClassifierParams.best_params_)
    Modell =
RFE(RandomForestClassifier(**ClassifierParams.best_params_),
n_features_to_select=5, verbose=5)

Model = Modell.fit(X_train, By_train)
Ally_pred = Model.predict(X_test)

# Printing out results - same for each function
Accuracy = (accuracy_score(By_test,Ally_pred))*100
FScore = (f1_score(By_test, Ally_pred))*100
ROC_AUC = roc_auc_score(By_test, Ally_pred, multi_class='ovo')
Precision = precision_score(By_test, Ally_pred)
Recall = recall_score(By_test, Ally_pred)

Table = X_test
Table['By_pred']=Ally_pred
Table['By_true']=By_test
Table['Accuracy']=Accuracy
Table['FScore']=FScore
Table['ROC_AUC']=ROC_AUC
Table['Precision']=Precision
Table['Recall']=Recall

# Calculating true positives and negatives
```

```

conditions = [
    Table['By_pred'].eq(0) & Table['By_true'].eq(0),
    Table['By_pred'].eq(0) & Table['By_true'].eq(1),
    Table['By_pred'].eq(1) & Table['By_true'].eq(0),
    Table['By_pred'].eq(1) & Table['By_true'].eq(1)
]
choices = ['TN', 'FN', 'FP', 'TP']
Table['Result'] = np.select(conditions, choices, default=0)

FemaleDF = Table[Table['Gender']==0.0]
F_Accuracy = (accuracy_score(FemaleDF['By_true'],
FemaleDF['By_pred']))*100
F_FScore = (f1_score(FemaleDF['By_true'],
FemaleDF['By_pred']))*100
F_ROC_AUC = roc_auc_score(FemaleDF['By_true'],
FemaleDF['By_pred'], multi_class='ovo')
F_Precision = precision_score(FemaleDF['By_true'],
FemaleDF['By_pred'])
F_Recall = recall_score(FemaleDF['By_true'], FemaleDF['By_pred'])

F_FNs = len(FemaleDF[FemaleDF['Result']=='FN'])
F_TNs = len(FemaleDF[FemaleDF['Result']=='TN'])
F_FPs = len(FemaleDF[FemaleDF['Result']=='FP'])
F_TPs = len(FemaleDF[FemaleDF['Result']=='TP'])
F_All = FemaleDF['Result'].sum()

# ** EDITING IN HERE, TRUE VALUES FOR FALSE POSITIVE ETC
F_FalseNegativeRate = (F_FNs/(F_FNs+F_TPs))*100
F_TrueNegativeRate = (F_TNs/(F_TNs+F_FPs))*100
F_FalsePositiveRate = (F_FPs/(F_FPs+F_TNs))*100
F_TruePositiveRate = (F_TPs/(F_TPs+F_FNs))*100

# Males
MaleDF = Table[Table['Gender']==1.0]
M_Accuracy = (accuracy_score(MaleDF['By_true'],
MaleDF['By_pred']))*100
M_FScore = (f1_score(MaleDF['By_true'], MaleDF['By_pred']))*100
M_ROC_AUC = roc_auc_score(MaleDF['By_true'], MaleDF['By_pred'],
multi_class='ovo')
M_Precision = precision_score(MaleDF['By_true'],
MaleDF['By_pred'])
M_Recall = recall_score(MaleDF['By_true'], MaleDF['By_pred'])

M_FNs = len(MaleDF[MaleDF['Result']=='FN'])
M_TNs = len(MaleDF[MaleDF['Result']=='TN'])
M_FPs = len(MaleDF[MaleDF['Result']=='FP'])
M_TPs = len(MaleDF[MaleDF['Result']=='TP'])
M_All = MaleDF['Result'].sum()

M_FalseNegativeRate = (M_FNs/(M_FNs+M_TPs))*100

```



```

M_TrueNegativeRate = (M_TNs/(M_TNs+M_FPs))*100
M_FalsePositiveRate = (M_FPs/(M_FPs+M_TNs))*100
M_TruePositiveRate = (M_TPs/(M_TPs+M_FNs))*100

rfe_features = BX_train.columns[Model.support_]
print(rfe_features.sort_values())
return Accuracy, FScore, ROC_AUC, Precision, Recall, F_Accuracy,
F_FScore, F_ROC_AUC, F_Precision, F_Recall, F_FNs, F_TNs, F_FPs,
F_TPs, F_FalseNegativeRate, F_TrueNegativeRate, F_FalsePositiveRate,
F_TruePositiveRate, M_Accuracy, M_FScore, M_ROC_AUC, M_Precision,
M_Recall, M_FNs, M_TNs, M_FPs, M_TPs, M_FalseNegativeRate,
M_TrueNegativeRate, M_FalsePositiveRate, M_TruePositiveRate

Analyse_FSbalanced_run(MX, My, FX, Fy, AllFeatures, RF)

```

Experiment 3.4.1 Random Forest (Balanced training data, with Feature Selection)

3.4.1 Random Forest

```
experiment_number = np.arange(100)
```

```
## RF Results
```

```
cols = ['UB_Accuracy', 'UB_FScore', 'UB_ROC_AUC', 'UB_Precision',
'UB_Recall',
```

```

'UB_F_Accuracy', 'UB_F_FScore', 'UB_F_ROC_AUC',
'UB_F_Precision', 'UB_F_Recall',
'UB_F_FNs', 'UB_F_TNs', 'UB_F_FPs', 'UB_F_TPs',
'UB_F_FNR', 'UB_F_TNR', 'UB_F_FPR', 'UB_F_TPR',
```

```

'UB_M_Accuracy', 'UB_M_FScore', 'UB_M_ROC_AUC',
'UB_M_Precision', 'UB_M_Recall',
'UB_M_FNs', 'UB_M_TNs', 'UB_M_FPs', 'UB_M_TPs',
'UB_M_FNR', 'UB_M_TNR', 'UB_M_FPR', 'UB_M_TPR',
]

```

```
Bal_W_FS_RFdf = pd.DataFrame(columns=cols)
```

```
# Scroll left here to see the function that defines it as a balanced experiment
```

```
# NEEDED TO ENSURE THERE WAS AN INDENT
```

```

for i in tqdm(experiment_number):
    UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall,
    UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall,
    UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR, UB_F_TNR, UB_F_FPR,
    UB_F_TPR, UB_M_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision,
    UB_M_Recall, UB_M_FNs, UB_M_TNs, UB_M_FPs, UB_M_TPs, UB_M_FNR,

```

```
UB_M_TNR, UB_M_FPR, UB_M_TPR = FSbalanced_run(MX, My, FX, Fy,
AllFeatures, RF)
```

```
list = UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision,
UB_Recall, UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision,
UB_F_Recall, UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR,
UB_F_TNR, UB_F_FPR, UB_F_TPR, UB_M_Accuracy, UB_M_FScore,
UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNs, UB_M_TNs,
UB_M_FPs, UB_M_TPs, UB_M_FNR, UB_M_TNR, UB_M_FPR, UB_M_TPR
Bal_W_FS_RFdf.loc[len(Bal_W_FS_RFdf)] = list
```

```
# reset the name of the df so can make multiple outputs with same code
```

```
df = Bal_W_FS_RFdf
print('All Accuracy (Mean)', df['UB_Accuracy'].mean(), 'SD',
df['UB_Accuracy'].std())
print('All FScore (Mean)', df['UB_FScore'].mean(), 'SD',
df['UB_FScore'].std())
print('All ROC_AUC (Mean)', df['UB_ROC_AUC'].mean(), 'SD',
df['UB_ROC_AUC'].std())
print('All Precision (Mean)', df['UB_Precision'].mean(), 'SD',
df['UB_Precision'].std())
print('All Recall (Mean)', df['UB_Recall'].mean(), 'SD',
df['UB_Recall'].std())
print(' ')

print('F Accuracy (Mean)', df['UB_F_Accuracy'].mean(), 'SD',
df['UB_F_Accuracy'].std())
print('F FScore (Mean)', df['UB_F_FScore'].mean(), 'SD',
df['UB_F_FScore'].std())
print('F ROC_AUC (Mean)', df['UB_F_ROC_AUC'].mean(),
df['UB_F_ROC_AUC'].std())
print('F Precision (Mean)', df['UB_F_Precision'].mean(),
df['UB_F_Precision'].std())
print('F Recall (Mean)', df['UB_F_Recall'].mean(),
df['UB_F_Recall'].std())

print('F FNR (Mean)', df['UB_F_FNR'].mean(), df['UB_F_FNR'].std())
print('F TNR (Mean)', df['UB_F_TNR'].mean(), df['UB_F_TNR'].std())
print('F FPR (Mean)', df['UB_F_FPR'].mean(), df['UB_F_FPR'].std())
print('F TPR (Mean)', df['UB_F_TPR'].mean(), df['UB_F_TPR'].std())
print(' ')

print('M Accuracy (Mean)', df['UB_M_Accuracy'].mean(),
df['UB_M_Accuracy'].std())
print('M FScore (Mean)', df['UB_M_FScore'].mean(),
df['UB_M_FScore'].std())
print('M ROC_AUC (Mean)', df['UB_M_ROC_AUC'].mean(),
df['UB_M_ROC_AUC'].std())
print('M Precision (Mean)', df['UB_M_Precision'].mean(),
```

```
df['UB_M_Precision'].std())
print('M Recall (Mean)', df['UB_M_Recall'].mean(),
df['UB_M_Recall'].std())

print('M FNR (Mean)', df['UB_M_FNR'].mean(), df['UB_M_FNR'].std())
print('M TNR (Mean)', df['UB_M_TNR'].mean(), df['UB_M_TNR'].std())
print('M FPR (Mean)', df['UB_M_FPR'].mean(), df['UB_M_FPR'].std())
print('M TPR (Mean)', df['UB_M_TPR'].mean(), df['UB_M_TPR'].std())
print('          ')

# This gives us each run
Bal_W_FS_RFdf.to_csv('Bal_W_FS_RFdf.csv')
```

3.4.2 Logistic Regression (Balanced training data with Feature Selection)

```
experiment_number = np.arange(100)
```

```
## RF Results
```

```
cols = ['UB_Accuracy', 'UB_FScore', 'UB_ROC_AUC', 'UB_Precision',
'UB_Recall',
```

```
        'UB_F_Accuracy', 'UB_F_FScore', 'UB_F_ROC_AUC',
'UB_F_Precision', 'UB_F_Recall',
        'UB_F_FNs', 'UB_F_TNs', 'UB_F_FPs', 'UB_F_TPs',
        'UB_F_FNR', 'UB_F_TNR', 'UB_F_FPR', 'UB_F_TPR',
```

```
        'UB_M_Accuracy', 'UB_M_FScore', 'UB_M_ROC_AUC',
'UB_M_Precision', 'UB_M_Recall',
        'UB_M_FNs', 'UB_M_TNs', 'UB_M_FPs', 'UB_M_TPs',
        'UB_M_FNR', 'UB_M_TNR', 'UB_M_FPR', 'UB_M_TPR',
    ]
```

```
Bal_W_FS_LRdf = pd.DataFrame(columns=cols)
```

```
# Scroll left here to see the function that defines it as a balanced
experiment
```

```
# NEEDED TO ENSURE THERE WAS AN INDENT
```

```
for i in tqdm(experiment_number):
```

```
    UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall,
    UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall,
    UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR, UB_F_TNR, UB_F_FPR,
    UB_F_TPR, UB_M_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision,
    UB_M_Recall, UB_M_FNs, UB_M_TNs, UB_M_FPs, UB_M_TPs, UB_M_FNR,
    UB_M_TNR, UB_M_FPR, UB_M_TPR = FSbalanced_run(MX, My, FX, Fy,
    AllFeatures, LR)
```

```
    list = UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision,
    UB_Recall, UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision,
```

```
UB_F_Recall, UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR,
UB_F_TNR, UB_F_FPR, UB_F_TPR, UB_M_Accuracy, UB_M_FScore,
UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNs, UB_M_TNs,
UB_M_FPs, UB_M_TPs, UB_M_FNR, UB_M_TNR, UB_M_FPR, UB_M_TPR
    Bal_W_FS_LRdf.loc[len(Bal_W_FS_LRdf)] = list

# reset the name of the df so can make multiple outputs with same code
df = Bal_W_FS_LRdf
print('All Accuracy (Mean)', df['UB_Accuracy'].mean(), 'SD',
df['UB_Accuracy'].std())
print('All FScore (Mean)', df['UB_FScore'].mean(), 'SD',
df['UB_FScore'].std())
print('All ROC_AUC (Mean)', df['UB_ROC_AUC'].mean(), 'SD',
df['UB_ROC_AUC'].std())
print('All Precision (Mean)', df['UB_Precision'].mean(), 'SD',
df['UB_Precision'].std())
print('All Recall (Mean)', df['UB_Recall'].mean(), 'SD',
df['UB_Recall'].std())
print('

')

print('F Accuracy (Mean)', df['UB_F_Accuracy'].mean(), 'SD',
df['UB_F_Accuracy'].std())
print('F FScore (Mean)', df['UB_F_FScore'].mean(), 'SD',
df['UB_F_FScore'].std())
print('F ROC_AUC (Mean)', df['UB_F_ROC_AUC'].mean(),
df['UB_F_ROC_AUC'].std())
print('F Precision (Mean)', df['UB_F_Precision'].mean(),
df['UB_F_Precision'].std())
print('F Recall (Mean)', df['UB_F_Recall'].mean(),
df['UB_F_Recall'].std())

print('F FNR (Mean)', df['UB_F_FNR'].mean(), df['UB_F_FNR'].std())
print('F TNR (Mean)', df['UB_F_TNR'].mean(), df['UB_F_TNR'].std())
print('F FPR (Mean)', df['UB_F_FPR'].mean(), df['UB_F_FPR'].std())
print('F TPR (Mean)', df['UB_F_TPR'].mean(), df['UB_F_TPR'].std())
print('

')

print('M Accuracy (Mean)', df['UB_M_Accuracy'].mean(),
df['UB_M_Accuracy'].std())
print('M FScore (Mean)', df['UB_M_FScore'].mean(),
df['UB_M_FScore'].std())
print('M ROC_AUC (Mean)', df['UB_M_ROC_AUC'].mean(),
df['UB_M_ROC_AUC'].std())
print('M Precision (Mean)', df['UB_M_Precision'].mean(),
df['UB_M_Precision'].std())
print('M Recall (Mean)', df['UB_M_Recall'].mean(),
df['UB_M_Recall'].std())

print('M FNR (Mean)', df['UB_M_FNR'].mean(), df['UB_M_FNR'].std())
```

```
print('M TNR (Mean)', df['UB_M_TNR'].mean(), df['UB_M_TNR'].std())
print('M FPR (Mean)', df['UB_M_FPR'].mean(), df['UB_M_FPR'].std())
print('M TPR (Mean)', df['UB_M_TPR'].mean(), df['UB_M_TPR'].std())
print('')
```

```
# This gives us each run
```

```
Bal_W_FS_LRdf.to_csv('Bal_W_FS_LRdf.csv')
```

3.1.4 Support Vector Machine (Balanced Training Data with Feature Selection)

```
experiment_number = np.arange(10)
```

```
## RF Results
```

```
cols = ['UB_Accuracy', 'UB_FScore', 'UB_ROC_AUC', 'UB_Precision',
        'UB_Recall',
```

```
        'UB_F_Accuracy', 'UB_F_FScore', 'UB_F_ROC_AUC',
        'UB_F_Precision', 'UB_F_Recall',
        'UB_F_FNs', 'UB_F_TNs', 'UB_F_FPs', 'UB_F_TPs',
        'UB_F_FNR', 'UB_F_TNR', 'UB_F_FPR', 'UB_F_TPR',
```

```
        'UB_M_Accuracy', 'UB_M_FScore', 'UB_M_ROC_AUC',
        'UB_M_Precision', 'UB_M_Recall',
        'UB_M_FNs', 'UB_M_TNs', 'UB_M_FPs', 'UB_M_TPs',
        'UB_M_FNR', 'UB_M_TNR', 'UB_M_FPR', 'UB_M_TPR',
        ]
```

```
Bal_W_FS_SVMdf1 = pd.DataFrame(columns=cols)
```

```
# Scroll left here to see the function that defines it as a balanced
experiment
```

```
# NEEDED TO ENSURE THERE WAS AN INDENT
```

```
for i in tqdm(experiment_number):
```

```
    UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall,
    UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall,
    UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR, UB_F_TNR, UB_F_FPR,
    UB_F_TPR, UB_M_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision,
    UB_M_Recall, UB_M_FNs, UB_M_TNs, UB_M_FPs, UB_M_TPs, UB_M_FNR,
    UB_M_TNR, UB_M_FPR, UB_M_TPR = FSbalanced_run(MX, My, FX, Fy,
    AllFeatures, SVM)
```

```
    list = UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision,
    UB_Recall, UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision,
    UB_F_Recall, UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR,
    UB_F_TNR, UB_F_FPR, UB_F_TPR, UB_M_Accuracy, UB_M_FScore,
    UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNs, UB_M_TNs,
    UB_M_FPs, UB_M_TPs, UB_M_FNR, UB_M_TNR, UB_M_FPR, UB_M_TPR
    Bal_W_FS_SVMdf1.loc[len(Bal_W_FS_SVMdf1)] = list
```

```
# reset the name of the df so can make multiple outputs with same code
df = Bal_W_FS_SVMdf1
print('All Accuracy (Mean)', df['UB_Accuracy'].mean(), 'SD',
df['UB_Accuracy'].std())
print('All FScore (Mean)', df['UB_FScore'].mean(), 'SD',
df['UB_FScore'].std())
print('All ROC_AUC (Mean)', df['UB_ROC_AUC'].mean(), 'SD',
df['UB_ROC_AUC'].std())
print('All Precision (Mean)', df['UB_Precision'].mean(), 'SD',
df['UB_Precision'].std())
print('All Recall (Mean)', df['UB_Recall'].mean(), 'SD',
df['UB_Recall'].std())
print('
    ')

print('F Accuracy (Mean)', df['UB_F_Accuracy'].mean(), 'SD',
df['UB_F_Accuracy'].std())
print('F FScore (Mean)', df['UB_F_FScore'].mean(), 'SD',
df['UB_F_FScore'].std())
print('F ROC_AUC (Mean)', df['UB_F_ROC_AUC'].mean(),
df['UB_F_ROC_AUC'].std())
print('F Precision (Mean)', df['UB_F_Precision'].mean(),
df['UB_F_Precision'].std())
print('F Recall (Mean)', df['UB_F_Recall'].mean(),
df['UB_F_Recall'].std())

print('F FNR (Mean)', df['UB_F_FNR'].mean(), df['UB_F_FNR'].std())
print('F TNR (Mean)', df['UB_F_TNR'].mean(), df['UB_F_TNR'].std())
print('F FPR (Mean)', df['UB_F_FPR'].mean(), df['UB_F_FPR'].std())
print('F TPR (Mean)', df['UB_F_TPR'].mean(), df['UB_F_TPR'].std())
print('
    ')

print('M Accuracy (Mean)', df['UB_M_Accuracy'].mean(),
df['UB_M_Accuracy'].std())
print('M FScore (Mean)', df['UB_M_FScore'].mean(),
df['UB_M_FScore'].std())
print('M ROC_AUC (Mean)', df['UB_M_ROC_AUC'].mean(),
df['UB_M_ROC_AUC'].std())
print('M Precision (Mean)', df['UB_M_Precision'].mean(),
df['UB_M_Precision'].std())
print('M Recall (Mean)', df['UB_M_Recall'].mean(),
df['UB_M_Recall'].std())

print('M FNR (Mean)', df['UB_M_FNR'].mean(), df['UB_M_FNR'].std())
print('M TNR (Mean)', df['UB_M_TNR'].mean(), df['UB_M_TNR'].std())
print('M FPR (Mean)', df['UB_M_FPR'].mean(), df['UB_M_FPR'].std())
print('M TPR (Mean)', df['UB_M_TPR'].mean(), df['UB_M_TPR'].std())
print('
    ')

```

```

# This gives us each run
Bal_W_FS_SVMdf1.to_csv('Bal_W_FS_SVMdf1.csv')

3.1.4 Gaussian Naive Bayes (Balanced Training Data with Feature Selection)
experiment_number = np.arange(100)

## RF Results
cols = ['UB_Accuracy', 'UB_FScore', 'UB_ROC_AUC', 'UB_Precision',
'UB_Recall',

        'UB_F_Accuracy', 'UB_F_FScore', 'UB_F_ROC_AUC',
'UB_F_Precision', 'UB_F_Recall',
        'UB_F_FNs', 'UB_F_TNs', 'UB_F_FPs', 'UB_F_TPs',
        'UB_F_FNR', 'UB_F_TNR', 'UB_F_FPR', 'UB_F_TPR',

        'UB_M_Accuracy', 'UB_M_FScore', 'UB_M_ROC_AUC',
'UB_M_Precision', 'UB_M_Recall',
        'UB_M_FNs', 'UB_M_TNs', 'UB_M_FPs', 'UB_M_TPs',
        'UB_M_FNR', 'UB_M_TNR', 'UB_M_FPR', 'UB_M_TPR',
    ]

Bal_W_FS_NBdf = pd.DataFrame(columns=cols)

# Scroll left here to see the function that defines it as a balanced
experiment

# NEEDED TO ENSURE THERE WAS AN INDENT
for i in tqdm(experiment_number):
    UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall,
    UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall,
    UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR, UB_F_TNR, UB_F_FPR,
    UB_F_TPR, UB_M_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision,
    UB_M_Recall, UB_M_FNs, UB_M_TNs, UB_M_FPs, UB_M_TPs, UB_M_FNR,
    UB_M_TNR, UB_M_FPR, UB_M_TPR = FSbalanced_run(MX, My, FX, Fy,
    AllFeatures, NB)

    list = UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision,
    UB_Recall, UB_F_Accuracy, UB_F_FScore, UB_F_ROC_AUC, UB_F_Precision,
    UB_F_Recall, UB_F_FNs, UB_F_TNs, UB_F_FPs, UB_F_TPs, UB_F_FNR,
    UB_F_TNR, UB_F_FPR, UB_F_TPR, UB_M_Accuracy, UB_M_FScore,
    UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNs, UB_M_TNs,
    UB_M_FPs, UB_M_TPs, UB_M_FNR, UB_M_TNR, UB_M_FPR, UB_M_TPR
    Bal_W_FS_NBdf.loc[len(Bal_W_FS_NBdf)] = list

# reset the name of the df so can make multiple outputs with same code
df = Bal_W_FS_NBdf
print('All Accuracy (Mean)', df['UB_Accuracy'].mean(), 'SD',

```



```
df['UB_Accuracy'].std())
print('All FScore (Mean)', df['UB_FScore'].mean(), 'SD',
df['UB_FScore'].std())
print('All ROC_AUC (Mean)', df['UB_ROC_AUC'].mean(), 'SD',
df['UB_ROC_AUC'].std())
print('All Precision (Mean)', df['UB_Precision'].mean(), 'SD',
df['UB_Precision'].std())
print('All Recall (Mean)', df['UB_Recall'].mean(), 'SD',
df['UB_Recall'].std())
print('
    ')

print('F Accuracy (Mean)', df['UB_F_Accuracy'].mean(), 'SD',
df['UB_F_Accuracy'].std())
print('F FScore (Mean)', df['UB_F_FScore'].mean(), 'SD',
df['UB_F_FScore'].std())
print('F ROC_AUC (Mean)', df['UB_F_ROC_AUC'].mean(),
df['UB_F_ROC_AUC'].std())
print('F Precision (Mean)', df['UB_F_Precision'].mean(),
df['UB_F_Precision'].std())
print('F Recall (Mean)', df['UB_F_Recall'].mean(),
df['UB_F_Recall'].std())

print('F FNR (Mean)', df['UB_F_FNR'].mean(), df['UB_F_FNR'].std())
print('F TNR (Mean)', df['UB_F_TNR'].mean(), df['UB_F_TNR'].std())
print('F FPR (Mean)', df['UB_F_FPR'].mean(), df['UB_F_FPR'].std())
print('F TPR (Mean)', df['UB_F_TPR'].mean(), df['UB_F_TPR'].std())
print('
    ')

print('M Accuracy (Mean)', df['UB_M_Accuracy'].mean(),
df['UB_M_Accuracy'].std())
print('M FScore (Mean)', df['UB_M_FScore'].mean(),
df['UB_M_FScore'].std())
print('M ROC_AUC (Mean)', df['UB_M_ROC_AUC'].mean(),
df['UB_M_ROC_AUC'].std())
print('M Precision (Mean)', df['UB_M_Precision'].mean(),
df['UB_M_Precision'].std())
print('M Recall (Mean)', df['UB_M_Recall'].mean(),
df['UB_M_Recall'].std())

print('M FNR (Mean)', df['UB_M_FNR'].mean(), df['UB_M_FNR'].std())
print('M TNR (Mean)', df['UB_M_TNR'].mean(), df['UB_M_TNR'].std())
print('M FPR (Mean)', df['UB_M_FPR'].mean(), df['UB_M_FPR'].std())
print('M TPR (Mean)', df['UB_M_TPR'].mean(), df['UB_M_TPR'].std())
print('
    ')

# This gives us each run
df = Bal_W_FS_NBdf.to_csv('df = Bal_W_FS_NBdf.csv')
```

Supplementary Material (A), 'Tables in Text'

Manuscript – “Investigating for bias in healthcare algorithms: A sex stratified analysis of supervised machine learning models in liver disease prediction”

Section 2.1 Supplementary: Data Exploration and Initial Analysis

Supplementary Table 1.0 gives the variables included in our dataset and their initial datatypes (gender is later converted to binary numerical form).

Supplementary Table 1.0 Features of the Indian Liver Patient Dataset

Column	Datatype	Description
Age	Int64	Age of patient
Gender	Object	Sex of patient
Total Bilirubin (TB)	Float64	Bilirubin is synthesised by the liver and acts as a marker of liver function.
Direct Bilirubin (DB)	Float64	Bilirubin is synthesised by the liver and acts as a marker of function.
ALP (Alkphos Alkaline Phosphatase)	Int64	An enzyme produced by the liver acting as a marker of function.
SGPT Alamine Aminotransferase	Int64	An enzyme produced by the liver acting as a marker of function.
SGOT Aspartate Aminotransferase	Int64	An enzyme produced by the liver acting as a marker of function.
Total Proteins (TP)	Float64	The liver synthesis proteins, this count reduces with disease.
Albumin (ALB)	Float64	The liver synthesis proteins, this count reduces with disease.
Selector Field	Float64	Column that indicates whether a patient has liver disease or not, this has been previously labelled by experts.

2.2 Supplementary; Feature Exploration

Supplementary Table 2.0 presents sex-stratified feature importance ranked by Pearson's correlation coefficient.

Supplementary Table 2.0 Feature importance ranked by Pearson Correlation Coefficient

Feature Rankings: Pearson Correlation with Target Variable (Diagnosis of Liver Disease)					
All Patients		Female Patients		Male Patients	
Feature	Pearson Correlation Coefficient	Feature	Pearson Correlation Coefficient	Feature	Pearson Correlation Coefficient
1. Direct Bilirubin (DB)	0.25	1. Alkaline Phosphotase (ALP)	0.24	1. Direct Bilirubin (DB)	0.25
2. Total Bilirubin (TB)	0.22	2. Direct Bilirubin (DB)	0.22	2. Total Bilirubin (TB)	0.22
3. Alkaline Phosphotase (ALP)	0.18	3. Total Bilirubin (TB)	0.22	3. Albumin (ALB)	0.19
4. SGPT Alamine Aminotransferase	0.16	4. SGPT Aspartate Aminotransferase (AST)	0.22	4. Age	0.17
5. A/G Ratio	0.16	5. SGPT Alamine Aminotransferase	0.19	5. A/G Ratio	0.17
6. Albumin (ALB)	0.16	6. A/G Ratio	0.15	6. Alkaline Phosphotase (ALP)	0.16
7. SGPT Aspartate Aminotransferase (AST)	0.15	7. Albumin (ALB)	0.07	7. SGPT Alamine Aminotransferase	0.16
8. Age	0.14	8. Total Proteins (TP)	0.05	8. SGPT Aspartate Aminotransferase (AST)	0.14
9. Gender	0.08	9. Age	0.02	9. Total Proteins (TP)	0.06
10. Total Proteins (TP)	0.04	10. Gender	N/A	Gender	N/A
Sum of correlation			1.38		1.52
Q1			0.07		0.16
Q3			0.22		0.19
Mean IQR			0.145		0.175

Supplementary Table 3.0: Summary Experiment 1 (Unbalanced Training Data, No Feature Selection)

	Random Forest Classifier			Logistic Regression Classifier			Support Vector Machine			Gaussian Naive Bayes		
	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)
All Accuracy	78.17	2.36		71.31	2.37		79.40	2.50		71.53	2.61	
All FScore	76.57	2.86		68.18	2.88		79.29	2.91		64.18	3.65	
All ROC_AUC	78.25	2.30		71.42	2.28		79.44	2.47		71.49%	2.33%	
All Precision	82.65	3.69		77.04	3.98		80.46	3.95		85.98%	4.12%	
All Recall	71.58	4.69		61.37	4.19		78.59	6.01		51.37%	4.36%	
Females Accuracy	76.06	4.40	0.00	73.33	3.95	0.01	81.55	4.80	0.02	73.45	4.57	0.02
Females FScore	64.09	7.50	0.00	55.24	7.28	0.00	76.03	7.18	0.00	51.11	7.93	0.00
Females ROC_AUC	72.55	4.76	0.00	68.04	3.91	0.00	80.91	5.46	0.08	66.66%	3.91%	0.00
Females Precision	78.52	10.63	0.00	81.31	11.31	0.30	78.01	9.48	0.00	88.66%	8.11%	0.05
Females Recall	55.27	9.49	0.00	42.70	7.85	0.00	76.68	13.47	0.04	36.37%	7.58%	0.00
Females FNR	44.73	9.49	0.00	57.30	7.85	0.00	23.32	13.47	0.08	63.63	7.58	0.00
Females TNR	89.84	5.69	0.00	93.38	4.64	0.00	85.13	7.89	0.00	96.95	2.26	0.00
Females FPR	10.16	5.69	0.00	6.62	4.64	0.00	14.87	7.89	0.00	3.05	2.26	0.00
Females TPR	55.27	9.49	0.00	42.70	7.85	0.00	76.68	13.47	0.04	36.37	7.58	0.00
Males Accuracy	79.02	2.81		70.49	2.74		78.57	2.85		70.74	3.08	
Males FScore	79.72	2.98		71.11	3.14		80.17	2.94		67.30	3.90	
Males ROC_AUC	79.36	2.71		70.97	2.71		78.50	2.92		72.20%	2.79%	
Males Precision	83.76	3.75		76.44	4.52		81.42	4.09		85.53%	4.25%	
Males Recall	76.29	4.82		66.76	4.80		79.26	5.04		55.68%	4.83%	
Males FNR	23.71	4.82		33.24	4.80		20.74	5.04		44.32	4.83	
Males TNR	82.42	3.99		75.18	4.77		77.73	5.85		88.71	3.66	
Males FPR	17.58	3.99		24.82	4.77		22.27	5.85		11.29	3.66	
Males TPR	76.29	4.82		66.76	4.80		79.26	5.04		55.68	4.83	

Supplementary Table 4.0 Summary Experiment 2 (Balanced Training Data, No Feature Selection)

	Random Forest Classifier			Logistic Regression Classifier			Support Vector Machine			Gaussian Naive Bayes		
	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)
All Accuracy	81.66	2.33		74.53	1.96		83.30	1.75		74.75	1.97	
All FScore	76.52	3.09		65.63	2.88		80.07	2.25		62.80	3.26	
All ROC_AUC	80.22	2.41		72.40	1.97		82.70	1.85		71.67	1.97	
All Precision	84.80	3.38		78.52	4.30		82.71	3.84		85.28	3.80	
All Recall	69.87	4.42		56.57	3.85		77.91	4.71		49.83	3.77	
Females Accuracy	84.75	3.03	0.00	77.71	2.42	0.00	89.03	2.23	0.00	78.47	2.35	0.00
Females FScore	71.41	5.74	0.00	51.69	5.87	0.00	82.18	3.72	0.00	51.55	5.38	0.00
Females ROC_AUC	78.47	3.76	0.13	67.10	2.86	0.00	86.87	3.01	0.00	67.12	2.59	0.00
Females Precision	85.44	6.07	0.88	82.22	8.04	0.00	84.22	5.64	0.14	85.13	7.00	0.37
Females Recall	61.70	7.03	0.00	38.12	6.09	0.00	80.88	6.90	0.00	37.19	5.02	0.00
Females FNR	38.30	7.03	0.00	61.88	6.09	0.00	19.12	6.90	0.00	62.81	5.02	0.00
Females TNR	95.24	2.03	0.00	96.09	2.04	0.00	92.86	3.00	0.00	97.05	1.48	0.00
Females FPR	4.76	2.03	0.00	3.91	2.04	0.00	7.14	3.00	0.00	2.95	1.48	0.00
Females TPR	61.70	7.03	0.00	38.12	6.09	0.00	80.88	6.90	0.00	37.19	5.02	0.00
Males Accuracy	78.58	3.05		71.35	3.22		77.56	2.93		71.04	3.22	
Males FScore	79.10	3.21		71.86	3.41		78.78	3.10		68.20	3.96	
Males ROC_AUC	79.07	2.93		71.89	3.07		77.80	2.95		72.57	2.82	
Males Precision	84.49	3.92		77.47	4.44		81.90	4.62		85.37	4.05	
Males Recall	74.59	4.93		67.34	5.33		76.24	5.26		57.01	5.09	
Males FNR	25.41	4.93		32.66	5.33		23.76	5.26		42.99	5.09	
Males TNR	83.55	4.16		76.44	4.82		79.36	5.69		88.12	3.48	
Males FPR	16.45	4.16		23.56	4.82		20.64	5.69		11.88	3.48	
Males TPR	74.59	4.93		67.34	5.33		76.24	5.26		57.01	5.09	

Supplementary Table 5.0 - Experiment 3.1.2 – Comparison of evaluation metrics before and after balancing the training data

	Random Forest Classifier				Logistic Regression Classifier				Support Vector Machine				Gaussian Naive Bayes			
	Unbalanced Training Data		Balanced Training Data		Unbalanced Training Data		Balanced Training Data		Unbalanced Training Data		Balanced Training Data		Unbalanced Training Data		Balanced Trained Data	
	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD
All Accuracy	78.17	2.36	81.66	2.33	71.31	2.37	74.53	1.96	79.40	2.50	83.30	1.75	71.53	2.61	74.75	1.97
All ROC_AUC	78.25	2.30	80.22	2.41	71.42	2.28	72.40	1.97	79.44	2.47	82.70	1.85	71.49	2.33	71.67	1.97
All Precision	82.65	3.69	84.80	3.38	77.04	3.98	78.52	4.30	80.46	3.95	82.71	3.84	85.98	4.12	85.28	3.80
All Recall	71.58	4.69	69.87	4.42	61.37	4.19	56.57	3.85	78.59	6.01	77.91	4.71	51.37	4.36	49.83	3.77
Females Accuracy	76.06	4.40	84.75	3.03	73.33	3.95	77.71	2.42	81.55	4.80	89.03	2.23	73.45	4.57	78.47	2.35
Females ROC_AUC	72.55	4.76	78.47	3.76	68.04	3.91	67.10	2.86	80.91	5.46	86.87	3.01	66.66	3.91	67.12	2.59
Females Precision	78.52	10.63	85.44	6.07	81.31	11.31	82.22	8.04	78.01	9.48	84.22	5.64	88.66	8.11	85.13	7.00
Females Recall	55.27	9.49	61.70	7.03	42.70	7.85	38.12	6.09	76.68	13.47	80.88	6.90	36.37	7.58	37.19	5.02
Males Accuracy	79.02	2.81	78.58	3.05	70.49	2.74	71.35	3.22	78.57	2.85	77.56	2.93	70.74	3.08	71.04	3.22
Males ROC_AUC	79.36	2.71	79.07	2.93	70.97	2.71	71.89	3.07	78.50	2.92	77.80	2.95	72.20	2.79	72.57	2.82
Males Precision	83.76	3.75	84.49	3.92	76.44	4.52	77.47	4.44	81.42	4.09	81.90	4.62	85.53	4.25	85.37	4.05
Males Recall	76.29	4.82	74.59	4.93	66.76	4.80	67.34	5.33	79.26	5.04	76.24	5.26	55.68	4.83	57.01	5.09

Supplementary Table 6.0: Summary Experiment 3 (Unbalanced Training Data, With Feature Selection)

	Random Forest Classifier			Logistic Regression Classifier			Support Vector Machine			Gaussian Naive Bayes		
	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)
All Accuracy	76.88	2.68		70.34	2.31		70.24	2.36		71.56	2.60	
All FScore	75.65	2.94		65.88	2.86		64.00	3.24		64.57	3.62	
All ROC_AUC	76.92	2.65		70.35	2.21		70.38	2.25		71.67	2.24	
All Precision	80.32	4.24		77.38	4.23		81.75	4.61		86.07	3.68	
All Recall	71.72	4.10		57.56	3.84		52.77	3.98		51.86	4.50	
Females Accuracy	74.45	4.79	0.00	72.40	4.26	0.01	72.22	4.03	0.01	73.93	4.45	0.00
Females FScore	63.50	7.71	0.00	53.13	7.06	0.00	50.66	7.56	0.00	52.19	8.13	0.00
Females ROC_AUC	71.49	5.08	0.00	66.65	3.92	0.00	65.92	3.96	0.00	67.24	4.06	0.00
Females Precision	72.51	9.58	0.00	77.21	8.66	0.44	82.53	9.80	0.90	88.92	7.98	0.03
Females Recall	57.62	10.12	0.00	41.05	7.62	0.00	36.98	7.22	0.00	37.43	7.92	0.00
Females FNR	42.38	10.12	0.00	58.95	7.62	0.00	63.02	7.22	0.00	62.57	7.92	0.00
Females TNR	85.37	6.39	0.00	92.26	3.47	0.00	94.86	3.28	0.00	97.06	2.18	0.00
Females FPR	14.63	6.39	0.00	7.74	3.47	0.00	5.14	3.28	0.00	2.94	2.18	0.00
Females TPR	57.62	10.12	0.00	41.05	7.62	0.00	36.98	7.22	0.00	37.43	7.92	0.00
Males Accuracy	77.87	3.12		69.51	3.19		69.47	2.85		70.62	3.04	
Males FScore	78.87	3.14		68.92	3.34		67.16	3.36		67.48	3.98	
Males ROC_AUC	78.10	3.19		70.25	3.15		70.82	2.67		72.23	2.73	
Males Precision	82.37	4.43		77.45	4.66		81.66	4.77		85.51	4.30	
Males Recall	75.83	3.85		62.29	4.25		57.27	4.47		55.97	5.15	
Males FNR	24.17	3.85		37.71	4.25		42.73	4.47		44.03	5.15	
Males TNR	80.38	5.24		78.22	5.08		84.36	4.50		88.49	3.54	
Males FPR	19.62	5.24		21.78	5.08		15.64	4.50		11.51	3.54	
Males TPR	75.83	3.85		62.29	4.25		57.27	4.47		55.97	5.15	

Supplementary Table 7.0: Summary Experiment 4 (Balanced Training Data, With Feature Selection)

	Random Forest Classifier			Logistic Regression Classifier			Support Vector Machin			Gaussian Naive Bayes		
	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)
All Accuracy	80.37	2.83		73.28	2.03		71.60	2.20		73.04	1.77	
All FScore	75.34	3.77		62.61	3.27		57.44	3.28		60.22	3.19	
All ROC_AUC	79.14	2.95		70.72	2.11		68.43	1.92		69.99	1.75	
All Precision	81.66	4.30		78.15	4.40		81.91	4.45		83.13	4.49	
All Recall	70.14	4.99		52.42	4.14		44.37	3.66		47.44	4.10	
Females Accuracy	83.18	3.55	0.00	76.69	2.55	0.00	74.70	3.06	0.00	75.36	2.89	0.00
Females FScore	70.21	6.99	0.00	53.18	5.80	0.00	46.78	5.96	0.00	45.78	6.45	0.00
Females ROC_AUC	78.02	4.71	0.46	67.48	3.15	0.00	64.32	2.97	0.00	64.09	3.10	0.00
Females Precision	78.65	6.76	0.00	71.91	7.85	0.00	73.34	8.33	0.00	76.37	9.99	0.00
Females Recall	63.99	9.10	0.00	42.59	6.27	0.00	34.74	5.85	0.00	33.05	5.84	0.00
Females FNR	36.01	9.10	0.00	57.41	6.27	0.00	65.26	5.85	0.00	66.95	5.84	0.00
Females TNR	92.05	2.72	0.00	92.37	2.27	0.00	93.90	2.23	0.01	95.12	2.35	0.00
Females FPR	7.95	2.72	0.00	7.63	2.27	0.00	6.10	2.23	0.00	4.88	2.35	0.00
Females TPR	63.99	9.10	0.00	42.59	6.27	0.00	34.74	5.85	0.00	33.05	5.84	0.00
Males Accuracy	77.56	3.23		69.88	2.92		68.50	2.93		70.73	2.73	
Males FScore	78.07	3.31		67.57	3.45		63.24	3.93		67.41	3.87	
Males ROC_AUC	77.98	3.24		71.05	2.79		70.27	2.67		72.25	2.33	
Males Precision	83.25	4.48		81.19	4.86		86.16	4.79		85.72	3.83	
Males Recall	73.69	4.33		58.10	4.50		50.11	4.35		55.83	5.34	
Males FNR	26.31	4.33		41.90	4.50		49.89	4.35		44.17	5.34	
Males TNR	82.26	5.02		84.00	4.40		90.43	3.39		88.68	3.56	
Males FPR	17.74	5.02		16.00	4.40		9.57	3.39		11.32	3.56	
Males TPR	73.69	4.33		58.10	4.50		50.11	4.35		55.83	5.34	

3.2 Supplementary, Analysis of Feature Selection

Supplementary Table 8.0 gives the feature rankings assigned by the RFE model when fitted to unbalanced data (Experiment 3) and balanced data (Experiment 4), focusing on RF classifiers.

Supplementary Table 8.0 – Comparison of RFE Feature Rankings when trained on unbalanced and balanced training data

Top 5 feature selected by Recursive Feature Elimination (RFE) on unbalanced and balanced training data			
Experiment 3 (Unbalanced Training Data)		Experiment 4 (Balanced Training Data)	
Rank	Feature	Rank	Feature
1.	A/G Ratio	1.	ALP (Alkphos Alkaline Phosphotase)
2.	ALP (Alkphos Alkaline Phosphotase)	2.	Gender
3.	SGOT Aspartate Aminotransferase	3.	SGOT Aspartate Aminotransferase
4.	SGPT Alamine Aminotransferate	4.	SGPT Alamine Aminotransferate
5.	Total Bilirubin (TB)	5.	Total Protiens (TP)

Table with columns: LB Accuracy, LB F1 Score, LB ROC AUC, LB Precision, LB Recall, LB F Accuracy, Experiment 3.1.1 Unbalanced training data, no feature selection - Random Forest Classifier, LB F Score, LB F ROC AUC, LB F Precision, LB F Recall, LB F FNR, LB F FPR, LB F FPP, LB F TNR, LB F TPR, LB AUC Accuracy, LB AUC F Score, LB AUC Precision, LB AUC Recall, LB AUC MRR, LB AUC TNR, LB AUC F1 Score, LB AUC F2 Score, LB AUC F3 Score, LB AUC F4 Score, LB AUC F5 Score, LB AUC F6 Score, LB AUC F7 Score, LB AUC F8 Score, LB AUC F9 Score, LB AUC F10 Score, LB AUC F11 Score, LB AUC F12 Score, LB AUC F13 Score, LB AUC F14 Score, LB AUC F15 Score, LB AUC F16 Score, LB AUC F17 Score, LB AUC F18 Score, LB AUC F19 Score, LB AUC F20 Score, LB AUC F21 Score, LB AUC F22 Score, LB AUC F23 Score, LB AUC F24 Score, LB AUC F25 Score, LB AUC F26 Score, LB AUC F27 Score, LB AUC F28 Score, LB AUC F29 Score, LB AUC F30 Score, LB AUC F31 Score, LB AUC F32 Score, LB AUC F33 Score, LB AUC F34 Score, LB AUC F35 Score, LB AUC F36 Score, LB AUC F37 Score, LB AUC F38 Score, LB AUC F39 Score, LB AUC F40 Score, LB AUC F41 Score, LB AUC F42 Score, LB AUC F43 Score, LB AUC F44 Score, LB AUC F45 Score, LB AUC F46 Score, LB AUC F47 Score, LB AUC F48 Score, LB AUC F49 Score, LB AUC F50 Score, LB AUC F51 Score, LB AUC F52 Score, LB AUC F53 Score, LB AUC F54 Score, LB AUC F55 Score, LB AUC F56 Score, LB AUC F57 Score, LB AUC F58 Score, LB AUC F59 Score, LB AUC F60 Score, LB AUC F61 Score, LB AUC F62 Score, LB AUC F63 Score, LB AUC F64 Score, LB AUC F65 Score, LB AUC F66 Score, LB AUC F67 Score, LB AUC F68 Score, LB AUC F69 Score, LB AUC F70 Score, LB AUC F71 Score, LB AUC F72 Score, LB AUC F73 Score, LB AUC F74 Score, LB AUC F75 Score, LB AUC F76 Score, LB AUC F77 Score, LB AUC F78 Score, LB AUC F79 Score, LB AUC F80 Score, LB AUC F81 Score, LB AUC F82 Score, LB AUC F83 Score, LB AUC F84 Score, LB AUC F85 Score, LB AUC F86 Score, LB AUC F87 Score, LB AUC F88 Score, LB AUC F89 Score, LB AUC F90 Score, LB AUC F91 Score, LB AUC F92 Score, LB AUC F93 Score, LB AUC F94 Score, LB AUC F95 Score, LB AUC F96 Score, LB AUC F97 Score, LB AUC F98 Score, LB AUC F99 Score, LB AUC F100 Score.

Mean Standard Deviation T Test Significance (Female vs Male)

Experiment 3.1.1 Unbalanced training data, no feature selection - Logistic Regression Classifier

Table with 20 columns: UB Accuracy, UB FScore, UB ROC AUC, UB Precision, UB Recall, UB Accuracy, UB FScore, UB ROC AUC, UB Precision, UB Recall, UB Accuracy, UB FScore, UB ROC AUC, UB Precision, UB Recall, UB Accuracy, UB FScore, UB ROC AUC, UB Precision, UB Recall. Rows 1-1000 contain numerical data for each metric.

Summary statistics table with 20 columns: Mean, Standard Deviation, 1 Test, 2 Test, 3 Test, 4 Test, 5 Test, 6 Test, 7 Test, 8 Test, 9 Test, 10 Test, 11 Test, 12 Test, 13 Test, 14 Test, 15 Test, 16 Test, 17 Test, 18 Test, 19 Test. Rows 1-1000 contain numerical data for each test.

Table with columns: UB_Accuracy, UB_FScore, UB_ROC_AUC, UB_Precision, UB_Recall, UB_F_Accuracy, UB_F_PScore, UB_F_ROC_AUC, UB_F_Precision, UB_F_Recall, UB_F_TNR, UB_F_FPR, UB_F_FNR, UB_Accuracy, UB_M_FScore, UB_M_ROC_AUC, UB_M_Precision, UB_M_Recall, UB_M_FNR, UB_M_TNR, UB_M_FPR. Rows 0-812. Includes Mean, Standard Deviation, T test, and P value at the bottom.

Summary of Results for Experiment 3.1.2

	Random Forest Classifier			Logistic Regression Classifier			Support Vector Machine			Gaussian Naive Bayes		
	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)	Mean	Standard Deviation	T Test Significance (Female to Male)
All Accuracy	81.66	2.33		74.53	1.96		83.30	1.75		74.75	1.97	
All FScore	76.52	3.09		65.63	2.88		80.07	2.25		62.80	3.26	
All ROC_AUC	80.22%	2.41%		72.40%	1.97%		82.70%	1.85%		71.67%	1.97%	
All Precision	84.80%	3.38%		78.52%	4.30%		82.71%	3.84%		85.28%	3.80%	
All Recall	69.87%	4.42%		56.57%	3.85%		77.91%	4.71%		49.83%	3.77%	
Females Accuracy	84.75	3.03	0.00	77.71	2.42	0.00	89.03	2.23	0.00	78.47	2.35	0.00
Females FScore	71.41	5.74	0.00	51.69	5.87	0.00	82.18	3.72	0.00	51.55	5.38	0.00
Females ROC_AUC	78.47%	3.76%	0.13	67.10%	2.86%	0.00	86.87%	3.01%	0.00	67.12%	2.59%	0.00
Females Precision	85.44%	6.07%	0.88	82.22%	8.04%	0.00	84.22%	5.64%	0.14	85.13%	7.00%	0.37
Females Recall	61.70%	7.03%	0.00	38.12%	6.09%	0.00	80.88%	6.90%	0.00	37.19%	5.02%	0.00
Females FNR	38.30	7.03	0.00	61.88	6.09	0.00	19.12	6.90	0.00	62.81	5.02	0.00
Females TNR	95.24	2.03	0.00	96.09	2.04	0.00	92.86	3.00	0.00	97.05	1.48	0.00
Females FPR	4.76	2.03	0.00	3.91	2.04	0.00	7.14	3.00	0.00	2.95	1.48	0.00
Females TPR	61.70	7.03	0.00	38.12	6.09	0.00	80.88	6.90	0.00	37.19	5.02	0.00
Males Accuracy	78.58	3.05		71.35	3.22		77.56	2.93		71.04	3.22	
Males FScore	79.10	3.21		71.86	3.41		78.78	3.10		68.20	3.96	
Males ROC_AUC	79.07%	2.93%		71.89%	3.07%		77.80%	2.95%		72.57%	2.82%	
Males Precision	84.49%	3.92%		77.47%	4.44%		81.90%	4.62%		85.37%	4.05%	
Males Recall	74.59%	4.93%		67.34%	5.33%		76.24%	5.26%		57.01%	5.09%	
Males FNR	25.41	4.93		32.66	5.33		23.76	5.26		42.99	5.09	
Males TNR	83.55	4.16		76.44	4.82		79.36	5.69		88.12	3.48	
Males FPR	16.45	4.16		23.56	4.82		20.64	5.69		11.88	3.48	
Males TPR	74.59	4.93		67.34	5.33		76.24	5.26		57.01	5.09	
	Random Forest Results			Logistic Regression Results			Support Vector Machine Results			Gaussian Naive Bayes Results		
Averaged over 100 experiments	Sex Disparity (Male Mean - Female Mean)		T Test P Value	Sex Disparity (Male Mean - Female Mean)		T Test P Value	Sex Disparity (Male Mean - Female Mean)		T Test P Value	Sex Disparity (Male Mean - Female Mean)		T Test P Value
Accuracy	-6.17		0.00	-6.36		0.00	-11.47		0.00	-7.43		0.00
FScore	7.69		0.00	20.17		0.00	-3.40		0.00	16.65		0.00
ROC_AUC	0.60%		0.13	4.79%		0.00	-9.06%		0.00	5.45%		0.00
Precision	-0.94%		0.88	-4.75%		0.00	-2.32%		0.14	0.24%		0.37
Recall	12.88%		0.00	29.22%		0.00	-4.64%		0.00	19.82%		0.00
False Negative Rate	-12.88		0.00	-29.22		0.00	4.64		0.00	-19.82		0.00
True Negative Rate	-11.69		0.00	-19.65		0.00	-13.49		0.00	-8.93		0.00
False Positive Rate	11.69		0.00	19.65		0.00	13.49		0.00	8.93		0.00
True Positive Rate	12.88		0.00	29.22		0.00	-4.64		0.00	19.82		0.00

Table with 28 columns: B Accuracy, B FScore, B ROC AUC, B Precision, B Recall, B F Accuracy, B F Score, B F ROC AUC, B Precision, B Recall, B F PR, B F F1, B F F2, B F F3, B F F4, B F F5, B F F6, B F F7, B F F8, B F F9, B F F10, B F F11, B F F12, B F F13, B F F14, B F F15, B F F16, B F F17, B F F18, B F F19, B F F20, B F F21, B F F22, B F F23, B F F24, B F F25, B F F26, B F F27, B F F28, B F F29, B F F30, B F F31, B F F32, B F F33, B F F34, B F F35, B F F36, B F F37, B F F38, B F F39, B F F40, B F F41, B F F42, B F F43, B F F44, B F F45, B F F46, B F F47, B F F48, B F F49, B F F50, B F F51, B F F52, B F F53, B F F54, B F F55, B F F56, B F F57, B F F58, B F F59, B F F60, B F F61, B F F62, B F F63, B F F64, B F F65, B F F66, B F F67, B F F68, B F F69, B F F70, B F F71, B F F72, B F F73, B F F74, B F F75, B F F76, B F F77, B F F78, B F F79, B F F80, B F F81, B F F82, B F F83, B F F84, B F F85, B F F86, B F F87, B F F88, B F F89, B F F90, B F F91, B F F92, B F F93, B F F94, B F F95, B F F96, B F F97, B F F98, B F F99, B F F100. Rows 0-1000 contain numerical data for each metric. Row 1001 contains summary statistics: Mean, Standard Deviation, Skewness, Kurtosis, Min, Max, Range, IQR, and Percentiles.

3.1.2 Balanced training data, no feature selection - Gaussian Naïve Bayes Classifier

Table with 20 columns: B.Accuracy, B.FScore, B.ROC.AUC, B.Precision, B.Recall, B.F.Accuracy, B.F.SFscore, B.F.ROC.AUC, B.F.Precision, B.F.Recall, B.F.NTR, B.F.TPR, B.M.Accuracy, B.M.FScore, B.M.ROC.AUC, B.M.Precision, B.M.Recall, B.M.FPR, B.M.TNR, B.M.FNR, B.M.FPP, B.M.TPR. Rows 1-99 contain numerical data for each metric. Row 100 is a summary row with labels like 'Mean', 'Std Deviation', 'T Test', etc.

Table with 28 columns: UB_Accracy, UB_Floors, UB_ROC_AUC, UB_Precision, UB_Recall, UB_F1_Accracy, UB_F1_Precision, UB_F1_Recall, UB_F1_FNR, UB_F1_FPR, UB_F1_TPR, UB_Accracy, UB_Floors, UB_ROC_AUC, UB_Precision, UB_Recall, UB_F1_FNR, UB_F1_FPR, UB_F1_TPR, UB_Accracy, UB_Floors, UB_ROC_AUC, UB_Precision, UB_Recall, UB_F1_FNR, UB_F1_FPR, UB_F1_TPR. Rows 1-99 and summary rows 100-105.

Mean Standard Deviation T Test (Female to Male)

