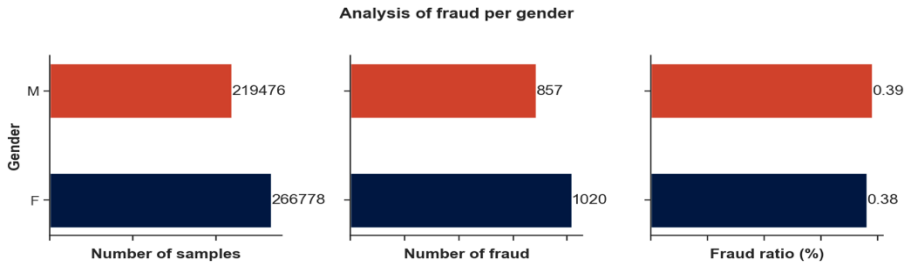
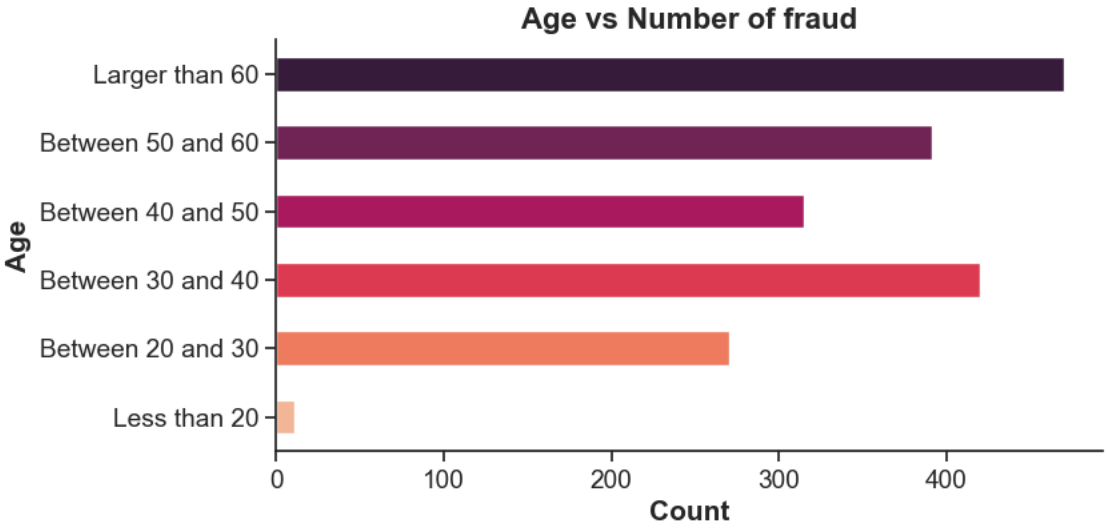
Before attempting to create a machine learning algorithm to detect credit card fraud, I did research online to see which features modern algorithms look for. This led me to a deep look into various research papers and techniques that researchers have been using. There were two GitHub repositories that caught my eye. The first one, Master-Fraud-Detection-for-Credit-Card-Transaction was very useful for providing various examples to visualize the data set. Fortunately, the column names were identical to that of the data set used to train our model. Therefore, I used the same techniques used in the eda-visualization.ipynb file to initiate a statistical analysis of the data. The second repository, fraud-detection-handbook, was extremely helpful as it outlined different methods of creating a machine learning algorithm to detect credit card fraud and outlined all of their key results. After reading through the summaries of each chapter, it became clear that XGBoost was the most suitable model found.

With these insights, I started by creating the necessary visualizations to extract hidden relationships and information from a combination of features. Following the visualizations outlined in the Master-Fraud-Detection-for-Credit-Card-Transaction repo, I looked into the top ten cities with cases of fraud. Each city had a respective subset for each gender. The visualization created looked like this:

This immediately indicated the gender played a large role in fraudulent charges within these cities. However, taking a step back from this visualization I realized that if I were to include gender in the training set for the model, this could create one that discriminates transactions based off of the gender of the individual. Thus, I delved deeper to see if there truly was a relationship of fraudulent charges per gender.



By looking at the fraud ratio between each gender, it became clear that whilst there were more cases of fraud with female consumers, the reality was that both genders had similar fraud ratios. This meant that gender was a feature to be disregarded as it didn't provide much information as to whether a transaction was fraudulent or not. Following the next visualizations located in the file, I looked into whether age was an important factor. I created this visualization to see if certain age intervals yielded higher correlations with fraudulent charges.



This clearly showed that there was a correlation with older individuals and fraudulent charges.

With all of these visualizations in mind, I then focused my attention on the second GitHub repository and did research into how to get started with extracting the appropriate features. I began by creating simple features such as the mean, std, max, min buying habits of each cardholder. Then I created temporal features outlining whether these transactions happened in they day or night, during the week or the weekend. Then I calculated the age of the cardholder at the time of the transaction and added this as another feature. Using the library LabelEncoder() from sklearn, I turned each category, job, and merchant into a numerical feature. Then, I removed all other non-numerical data including location, names, and gender. This gave me the first iteration of my algorithm which initially was a decision tree. My f1 score grew from the initial 0.42253 to 0.72622 on Kaggle. Clearly these features were very influential on the model's flagging accuracy. Thus, I switched model from DecisionTree to XGBoost based on the second GitHub repository. I then implemented RandomSearch to tune the model parameters. However, this only raised the score

marginally and was computationally intensive. After reading through more of the repository, the researchers noted that the XGBoost model was highly robust against imbalanced datasets and tuning the parameters only lead to marginal improvements. Thus, I decided to not include random search and instead use the model with its default parameters.

One of the most informative aspects of XGBoost is its feature_importances_ attribute. This allowed me to see which features the model found most informative. I also used the shap library to see which features were the most influential. Both of these functions focused my attention to the category_encoded feature. Thus, I tried to extract as much information as possible. I created functions that calculated the odds ratio of fraudulent charges for categories and merchants, the standard deviation of spending within each category versus each cardholder, the percentage of transactions in the most frequent category for each customer, the number of transactions in the most frequent category per customer, the category change frequency for each customer, and the category-based spend proportions for each customer. With all of these new features, I trained the model again and found a substantial improvement. To ensure the model was indeed improving I was validating it by finding its f1 score and comparing it to its previous one. If there was substantial improvement, I'd upload the submission to Kaggle to ensure the algorithm did truly improve.

However, I then started to think to myself, was I overfitting this algorithm. Did I add too many features, were there features that were redundant? So, I used recursive feature elimination with cross-validation to find redundant features. From the sklearn library, I managed to find that there were indeed features that weren't as influential. I found that the model stopped improving after 36 features remained of the initial 40. I located, removed, re-trained the model and saw that its score marginally improved. However, it become very clear that I made a critical error earlier on and I left the transaction ids in the training set. Instead of finding relationships, the model also learned which transactions were fraudulent within the training set and thus it couldn't truly be used on more generalized data sets. Thus, I removed IDs from the training set, only to include them in the submission set for validation.

This transformed my model to reach an f1 score of 0.89 on Kaggle. I tried to include more features focusing on location, job, and age but my score would only marginally improve. I felt that I couldn't use any more features or exclude others without switching to a different model. With time of the essence, I tried to squeeze the remaining score improvement by implementing precision recall curve from the sklearn library to calculate precision, recall, and thresholds for the probabilities of the positive class predicted by my model. This information was then used to calculate F1 scores for each threshold and find the best threshold for making predictions.

With all of this tied togethers, I managed to increase my score to be above 0.9, an achievement that seemed unobtainable at the start of the competition.