Medicare Provider Fraud Detection

Milestone: Project Report

Group 18

Dishita Jain
Kishan Patel

+1(857)-693-8718 (Dishita Jain)
+1(478)-663-1131(Kishan Patel)

jain.dis@northeastern.edu
patel.kisha@northeastern.edu

Percentage of Effort Contributed by Student 1:___50_____

Percentage of Effort Contributed by Student 2:_____50_____

Signature of Student 1: __Dishita Jain___

Signature of Student 2: __Kishan Patel____

Submission Date: __04/23/2022_____

PROBLEM STATEMENT

Medicare is a public healthcare program designed to offer coverage for people over the age of 65 and individuals with particular disabilities. Although it plays a crucial role in providing healthcare services for millions of Americans, the program has faced significant challenges due to fraudulent activities. Some healthcare providers have been found to engage in fraudulent practices such as billing for services that were not rendered, exaggerating the costs of services provided, or billing for unnecessary services to increase their reimbursement. These fraudulent activities not only compromise the integrity of the program but also cost taxpayers billions of dollars each year. Therefore, there is a need for strict measures to prevent fraud and abuse in Medicare.

PROBLEM DEFINITION

The issue of fraudulent practices among Medicare providers has caused substantial losses running into billions of dollars yearly. This not only strains the financial resources of the program but also results in higher healthcare costs for all Americans. Additionally, such fraudulent practices may lead to the provision of inadequate care to patients, jeopardizing their health and well-being.

To combat provider fraud in the Medicare program, the Centers for Medicare & Medicaid Services (CMS) has put in place various measures. These measures include the use of data analytics to detect patterns of fraud, the establishment of a specialized team to investigate and prosecute fraudulent providers, and the implementation of a national provider identifier system to make the identification and tracking of providers more accessible. These measures aim to prevent and reduce fraudulent activities, thereby protecting the program's finances and the health of beneficiaries.

DATA SOURCE

The Medicare Provider dataset was taken from Kaggle Datasets (Medicare Dataset) , an online free open repository.

DATA DESCRIPTION

The dataset comprises of 138555 rows and 27 columns.

1. BeneID: It is beneficiary ID which is unique to all customers.
2. DOB: Date of birth of the customer
3. Gender: Customer’s Gender
4. Race: Race of customers

5. State
6. County The dataset also contains customer’s existing medical conditions
7. RenalDiseaseIndicator
8. ChronicCond_Alzheimer
9. ChronicCond_Heartfailure
10. ChronicCond_KidneyDisease
11. ChronicCond_Cancer
12. ChronicCond_ObstrPulmonary
13. ChronicCond_Depression
14. ChronicCond_Diabetes
15. ChronicCond_IschemicHeart
16. ChronicCond_Osteoporasis
17. ChronicCond_rheumatoidarthritis
18. ChronicCond_stroke

It also contains other information of the customers such as:

19. IPAnnualReimbursementAmt
20. IPAnnualDeductibleAmt
21. OPAnnualReimbursementAmt
22. OPAnnualDeductibleAmt
23. Patient_Age_Year
24. Patient_Age_Month
25. Dead_or_Alive
26. AGE
27. AGE_groups

Data Processing:

In order to conduct an analysis of the Medicare program's fraud and abuse, it is crucial to ensure that the data used is clean and accurate. Data cleaning involves identifying and correcting errors, inconsistencies, and missing values in the data set.

The first step in data cleaning would be to examine the data set for any duplicate or irrelevant entries, such as incomplete or outdated information. Once identified, these entries can be removed to ensure that the analysis is based on accurate and relevant data.

Next, it is important to address any missing values in the data set. This can be done through various methods such as imputation, where missing values are replaced with estimated values based on other data points in the set.

Another crucial step in data cleaning is to check for outliers and anomalies in the data set. This involves identifying values that are significantly different from other values in the set and determining whether they are legitimate or erroneous.

Finally, the cleaned data set can be further prepared for analysis by standardizing and normalizing the data to ensure consistency and comparability. By conducting thorough data cleaning, the analysis of the Medicare program's fraud and abuse will be based on accurate and reliable information, leading to more meaningful insights and conclusions.

Shape of the dataset

```
from google.colab import files

train_bene_df = pd.read_csv("Train_Beneficiarydata-1542865627584.csv")
train_ip_df = pd.read_csv("Train_Inpatientdata-1542865627584.csv")
train_op_df = pd.read_csv("Train_Outpatientdata-1542865627584.csv")
```

```
train_bene_df.shape
```

```
(138556, 25)
```

```
train_bene_df.columns
```

```
Index(['BeneID', 'DOB', 'DOD', 'Gender', 'Race', 'RenalDiseaseIndicator',
       'State', 'County', 'NoOfMonths_PartACov', 'NoOfMonths_PartBCov',
       'ChronicCond_Alzheimer', 'ChronicCond_Heartfailure',
       'ChronicCond_KidneyDisease', 'ChronicCond_Cancer',
       'ChronicCond_ObstrPulmonary', 'ChronicCond_Depression',
       'ChronicCond_Diabetes', 'ChronicCond_IschemicHeart',
       'ChronicCond_Osteoporasis', 'ChronicCond_rheumatoidarthritis',
       'ChronicCond_stroke', 'IPAnnualReimbursementAmt',
       'IPAnnualDeductibleAmt', 'OPAnnualReimbursementAmt',
       'OPAnnualDeductibleAmt'],
      dtype='object')
```

Datatypes of the dataset columns

```
train_bene_df.dtypes
```

```
BeneID                          object
DOB                             object
DOD                             object
Gender                           int64
Race                             int64
RenalDiseaseIndicator           object
State                            int64
County                           int64
NoOfMonths_PartACov              int64
NoOfMonths_PartBCov              int64
ChronicCond_Alzheimer            int64
ChronicCond_Heartfailure         int64
ChronicCond_KidneyDisease        int64
ChronicCond_Cancer               int64
ChronicCond_ObstrPulmonary       int64
ChronicCond_Depression           int64
ChronicCond_Diabetes             int64
ChronicCond_IschemicHeart        int64
ChronicCond_Osteoporasis         int64
ChronicCond_rheumatoidarthritis  int64
ChronicCond_stroke               int64
IPAnnualReimbursementAmt         int64
IPAnnualDeductibleAmt            int64
OPAnnualReimbursementAmt         int64
OPAnnualDeductibleAmt            int64
dtype: object
```

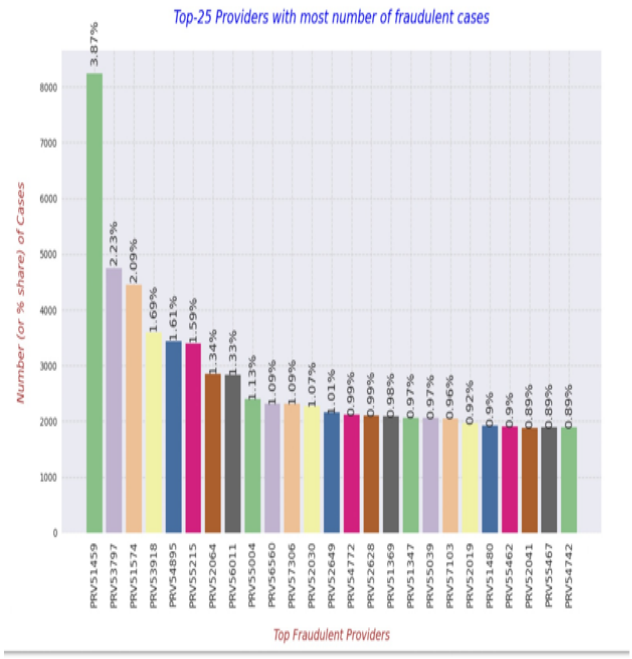
Data Cleaning:

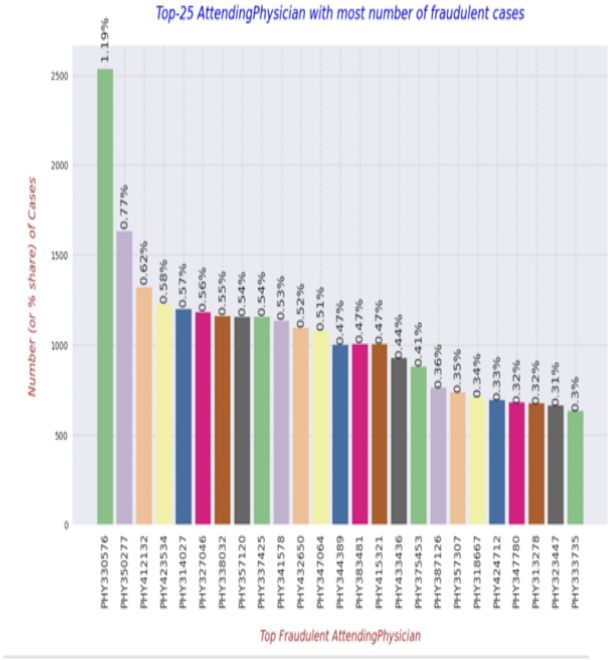
```
[ ] train_bene_df['DOB'] = pd.to_datetime(train_bene_df['DOB'], format="%Y-%m-%d")
```

```
[ ] train_bene_df['Patient_Age_Year'] = train_bene_df['DOB'].dt.year
    train_bene_df['Patient_Age_Month'] = train_bene_df['DOB'].dt.month
```

```
[ ] bene_age_year_df = pd.DataFrame(train_bene_df['Patient_Age_Year'].value_counts()).reset_index(drop=False)
    bene_age_year_df.columns= ['year','num_of_beneficiaries']
    bene_age_year_df = bene_age_year_df.sort_values(by='year')
```

Data Visualization

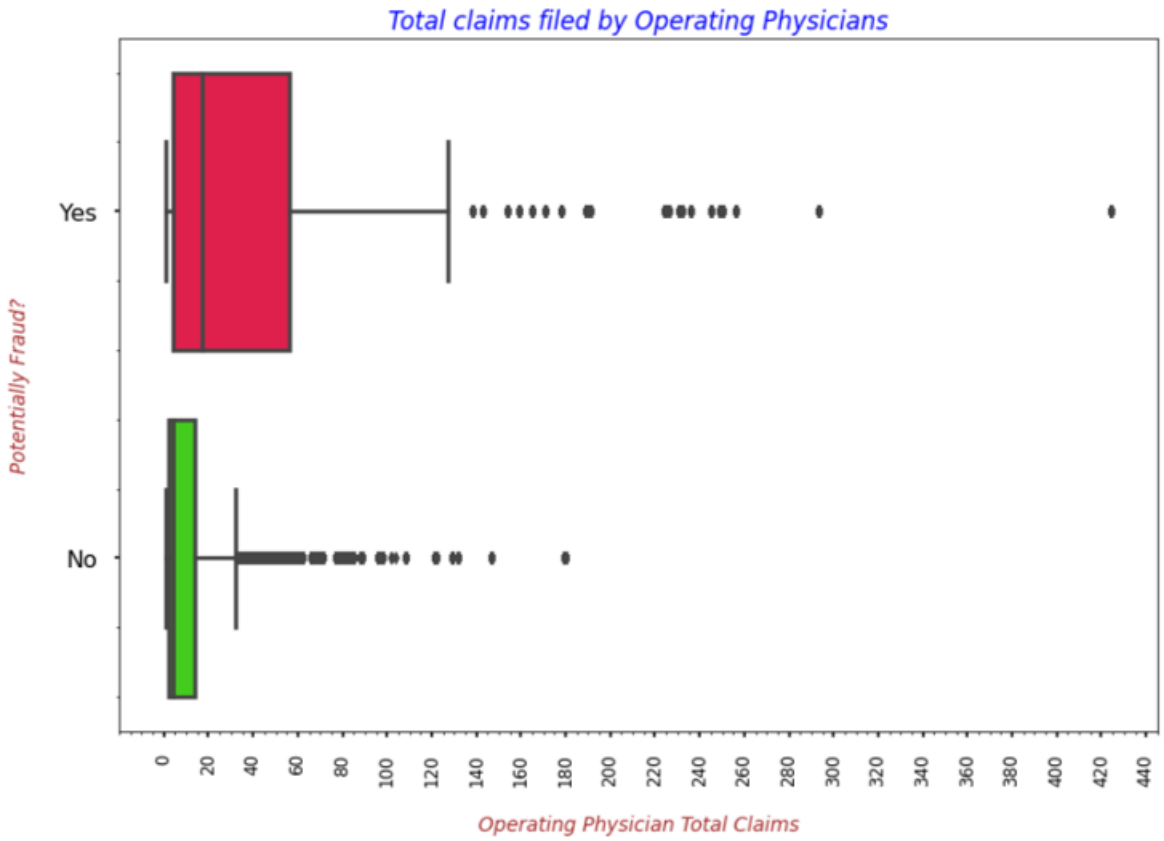




A pair plot is a type of plot that shows pairwise relationships between different variables in a dataset. In this case, the two columns being compared are "Number of cases" and "Physician Id".

The "Number of cases" column represents the number of cases that a particular physician has treated, while the "Physician Id" column represents the unique identifier for each physician in the dataset. By plotting a pair plot between these two columns, we can gain insights into the relationship between the number of cases treated by each physician and their unique identifier.

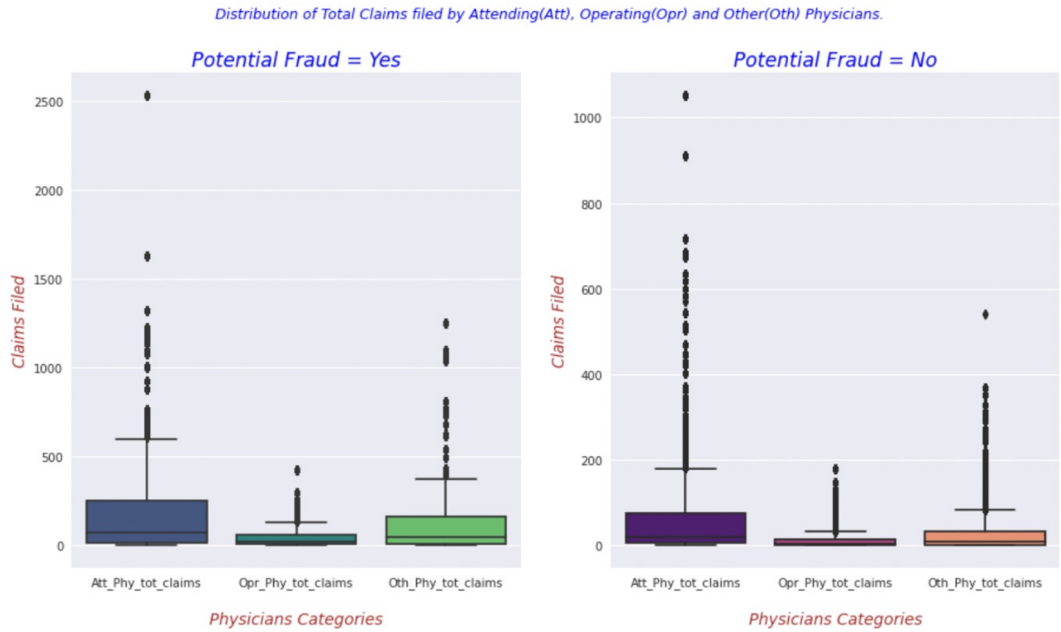
Box plot



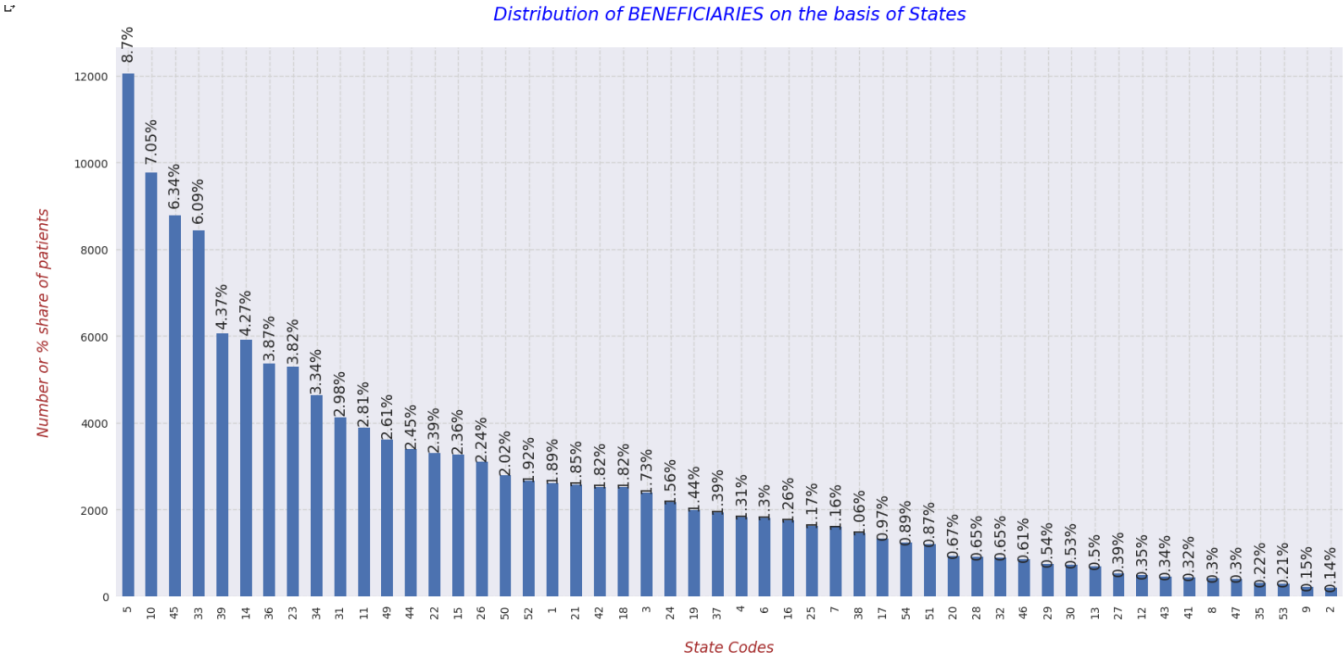
"Att_Phy_tot_claims" may be useful in distinguishing potentially fraudulent and non-fraudulent cases. This conclusion is supported by the KDE and box plots, which suggest that if an Attending Physician has filed more than 100 total claims, there may be a higher likelihood of fraud.

The KDE plot estimates the probability density function of the "Att_Phy_tot_claims" variable, helping to identify any patterns or clusters in the data that may be indicative of subgroups.

Similarly, the box plot is used to visualize the distribution of the "Att_Phy_tot_claims" variable for fraudulent and non-fraudulent cases, showing the median value, quartiles, and any outliers in the data.

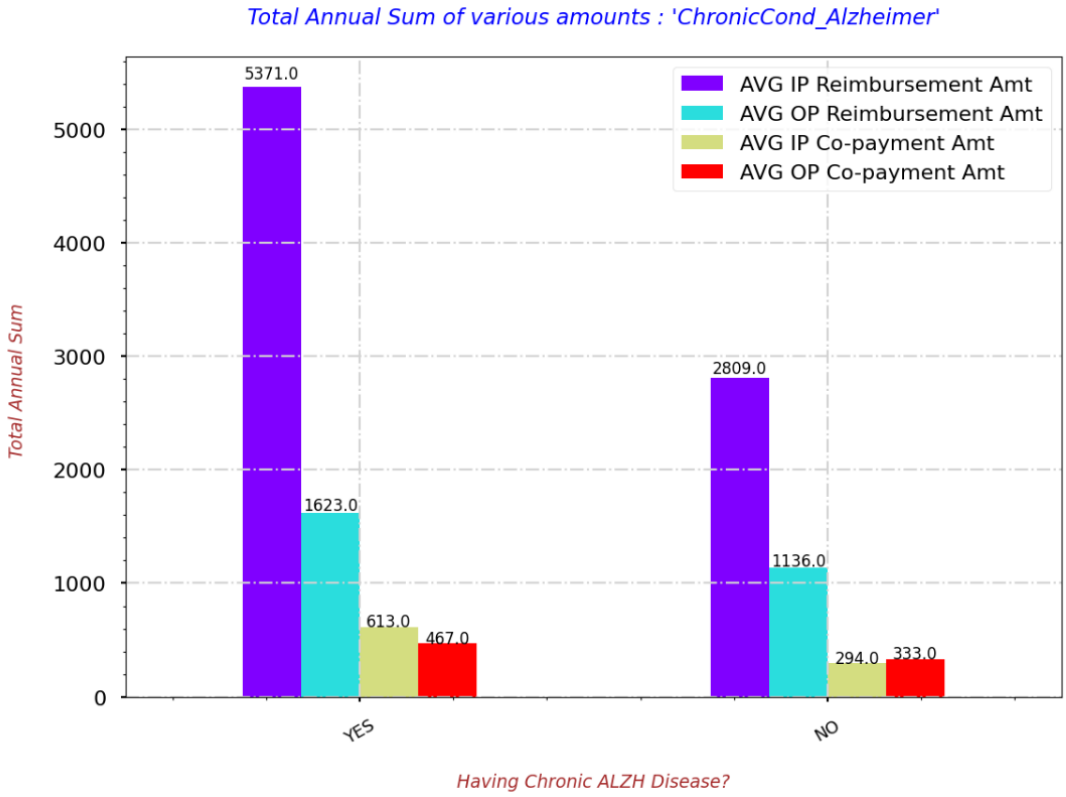


The above Box plots suggests that theses newly added features may be slightly useful in segregating the potentially fraud and non-fraudulent cases. As we can see that the total false claims filed by physicians are slightly more than the non-false claims filed by them.



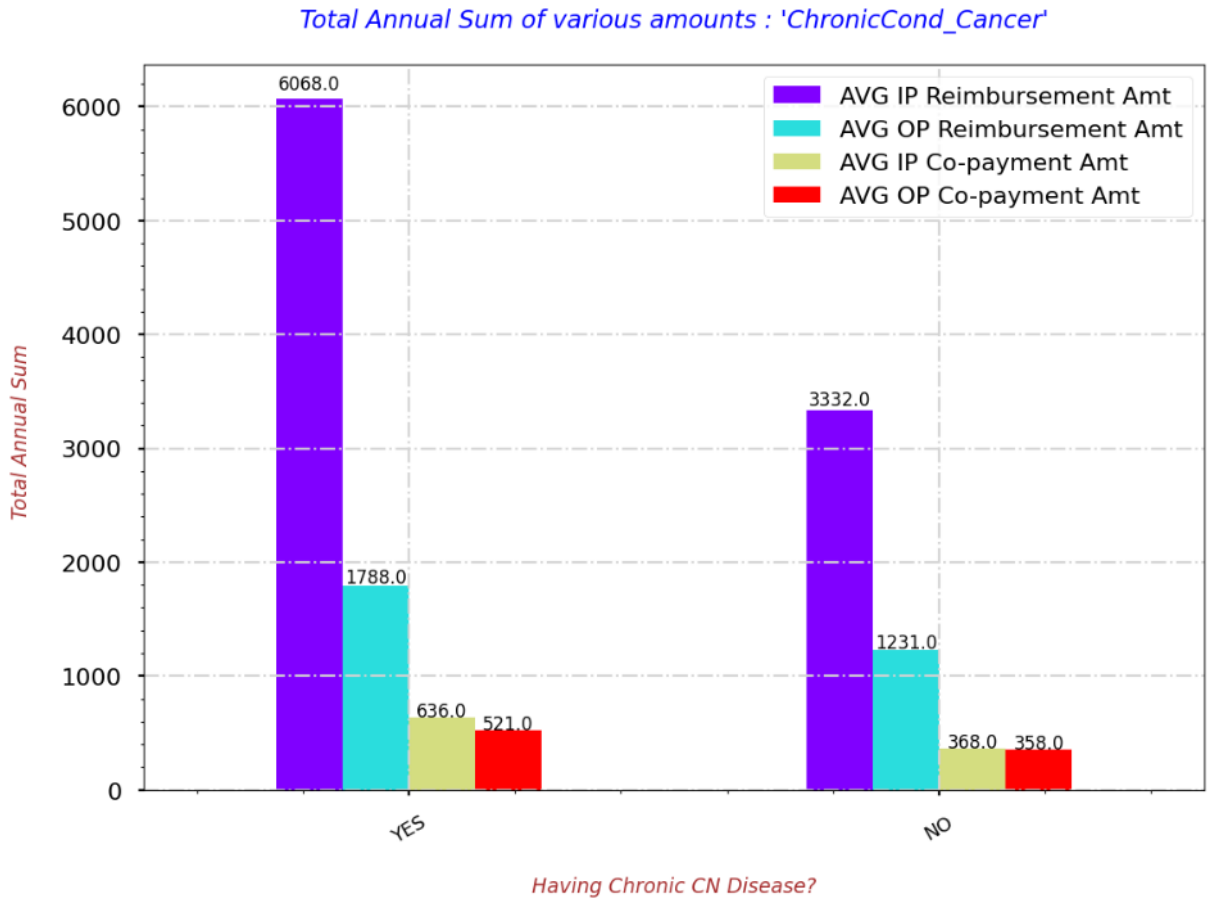
The output of this code is a bar chart that displays the distribution of beneficiaries based on their state codes. The x-axis of the chart shows the state codes, while the y-axis shows the number or percentage share of patients in each state. Each bar in the chart represents a state code, and the height of the bar represents the number of patients in that state.

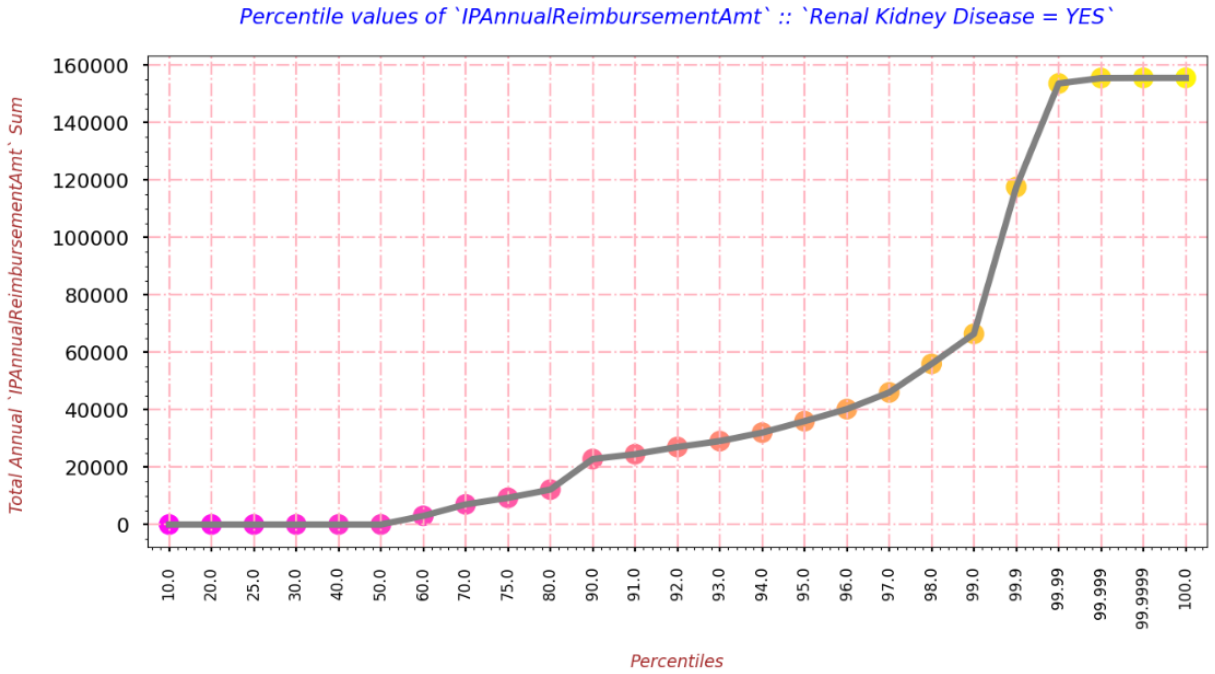
The code uses the matplotlib library to create the bar chart and the seaborn style context to set the visual style of the chart. The figsize parameter sets the size of the chart, while the kind parameter in the plot function specifies the type of chart as a bar chart.



The given code generates a bar graph that shows the Total Annual Sum of IP Co-payment for patients having and not having Chronic Alzheimer's Disease. The data for this graph is obtained from the CC_ALZH_all_amts DataFrame.

The x-axis of the bar graph represents the presence or absence of Chronic Alzheimer's Disease, denoted by 'YES' and 'NO' labels, respectively. The y-axis represents the Total Annual Sum of IP Co-payment for the respective groups.

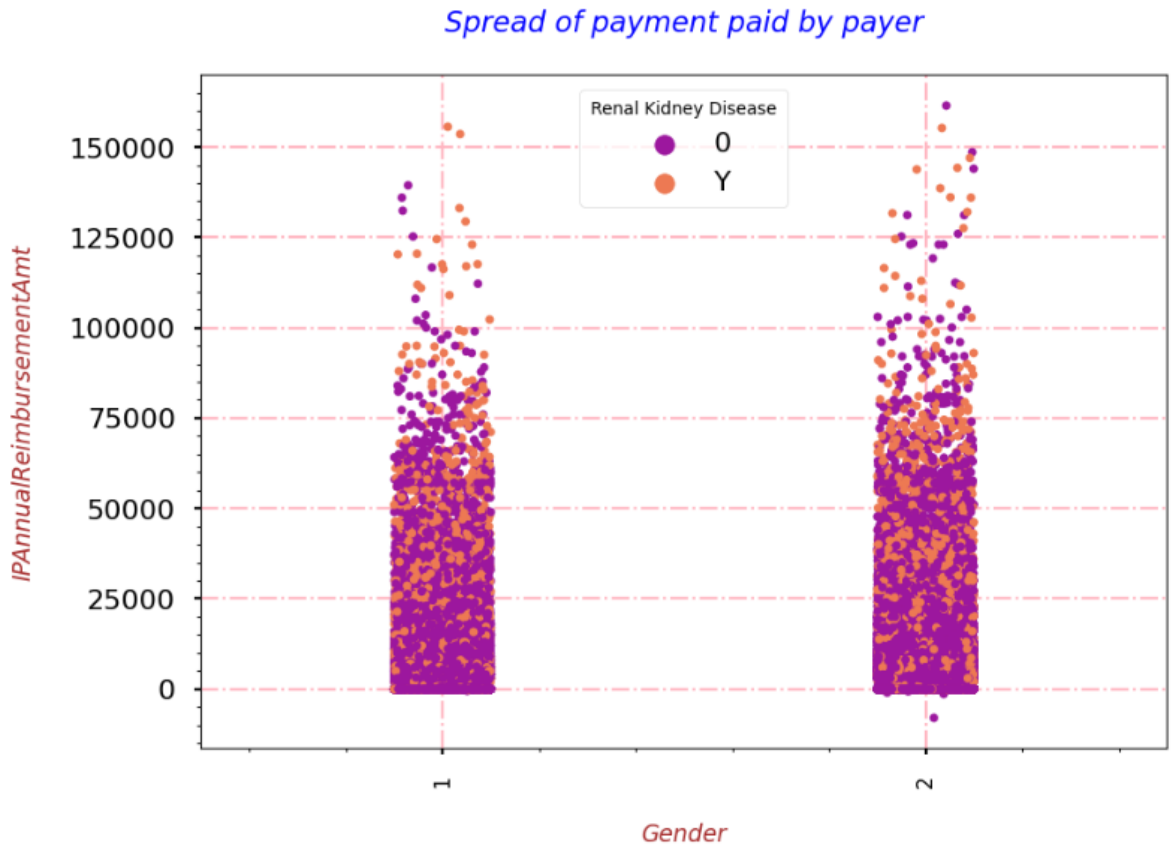

Total Annual Sum of various amounts : 'ChronicCond_Cancer'
AVG IP Reimbursement Amt
AVG OP Reimbursement Amt
AVG IP Co-payment Amt
AVG OP Co-payment Amt
Total Annual Sum
6000
5000
4000
3000
2000
1000
0
6068.0
1788.0
636.0
521.0
3332.0
1231.0
368.0
358.0
YES
NO
Having Chronic CN Disease?


Percentile values of `IPAnnualReimbursementAmt` :: `Renal Kidney Disease = YES`
Total Annual `IPAnnualReimbursementAmt` Sum
160000
140000
120000
100000
80000
60000
40000
20000
0
10.0
20.0
25.0
30.0
40.0
50.0
60.0
70.0
75.0
80.0
90.0
91.0
92.0
93.0
94.0
95.0
96.0
97.0
98.0
99.0
99.9
99.99
99.999
99.9999
100.0
Percentiles

The code is calculating and displaying the percentiles of the 'IPAnnualReimbursementAmt' column for the subset of data where the 'RenalDiseaseIndicator' column is equal to 'Y'.

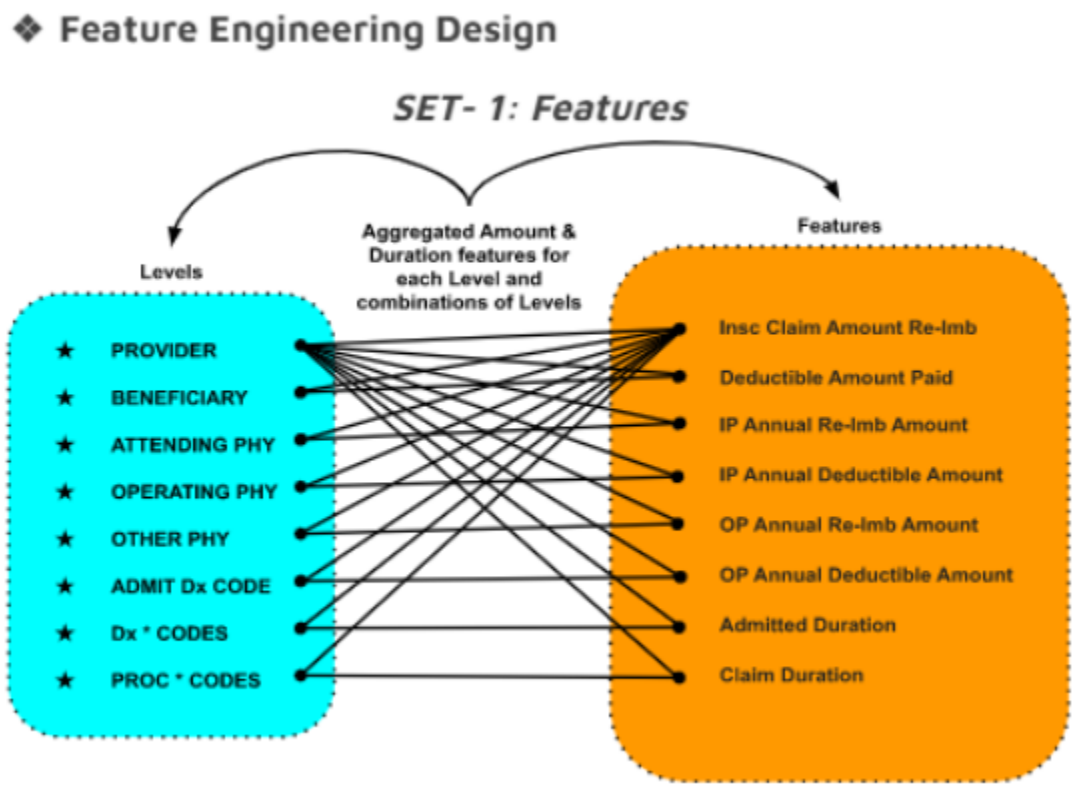
The function 'cal_display_percentiles' takes in four arguments:

x_col: The column name for the independent variable 'RenalDiseaseIndicator'
y_col: The column name for the dependent variable 'IPAnnualReimbursementAmt'
title_lbl: The title for the graph
x_filter_code: The value to filter the 'x_col' column by, in this case 'Y'
The output is a bar chart showing the percentiles of the 'IPAnnualReimbursementAmt' column for patients with Renal Kidney Disease. The x-axis displays the percentiles (from 0% to 100%), while the y-axis displays the 'IPAnnualReimbursementAmt' values. The chart also includes labels for the 25th, 50th (median), and 75th percentiles.



The plot_strip_plots() function is used to plot the spread of data points of pre-disease indicators for the Annual IP and OP expenditures across males and females using a seaborn stripplot. The function takes in four parameters, including the gender feature, pre-disease indicator, feature such as reimbursement or deductible amount whose percentiles need to be generated, and category code for which the data spread needs to be generated. The function sets the style to 'seaborn-poster', sets the figure size, and provides labels and titles to the graph. It also adds a legend with the specified title.

Feature Engineering



MODEL IMPLEMENTATION

We have implemented the following model:

- Logistic Regression
- Random Forest
- Decision Tree
- K-Nearest Neighbour
- Gradient Booster Regressor
- Neural Networks

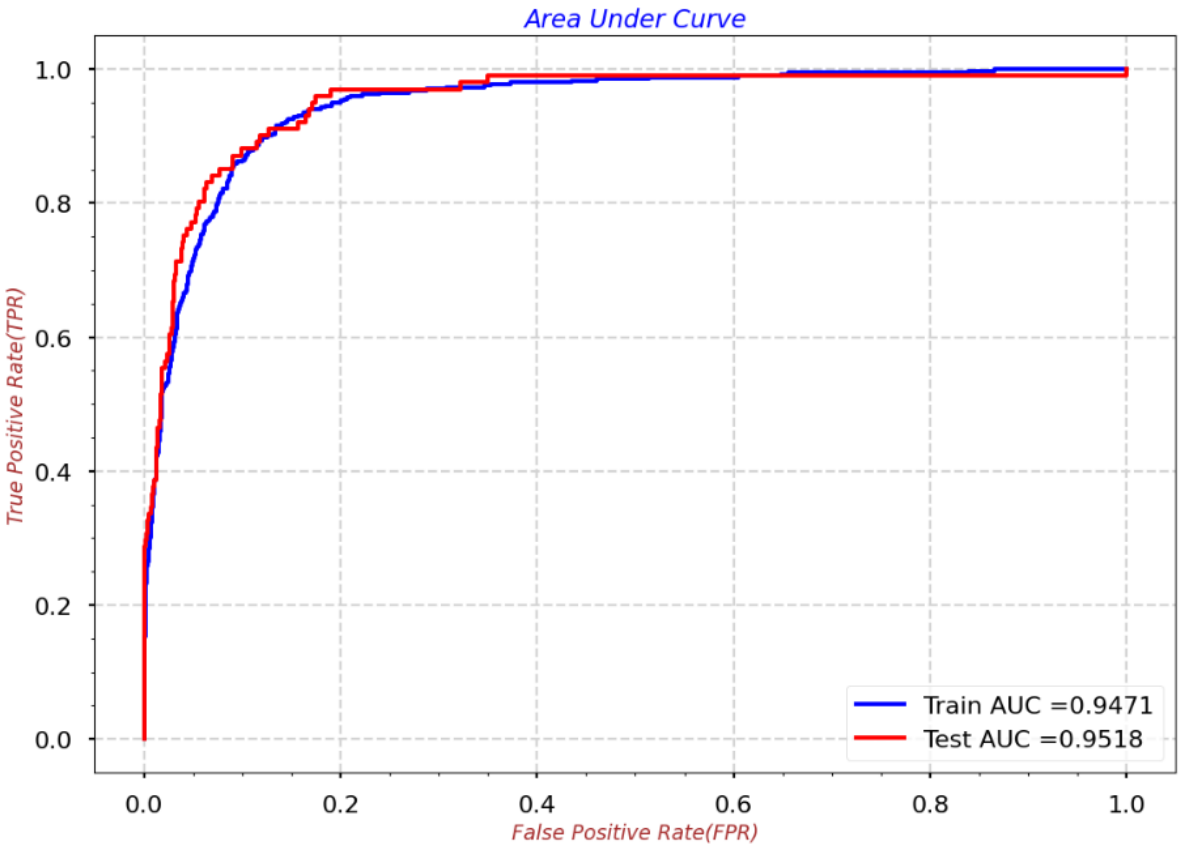
Logistic Regression

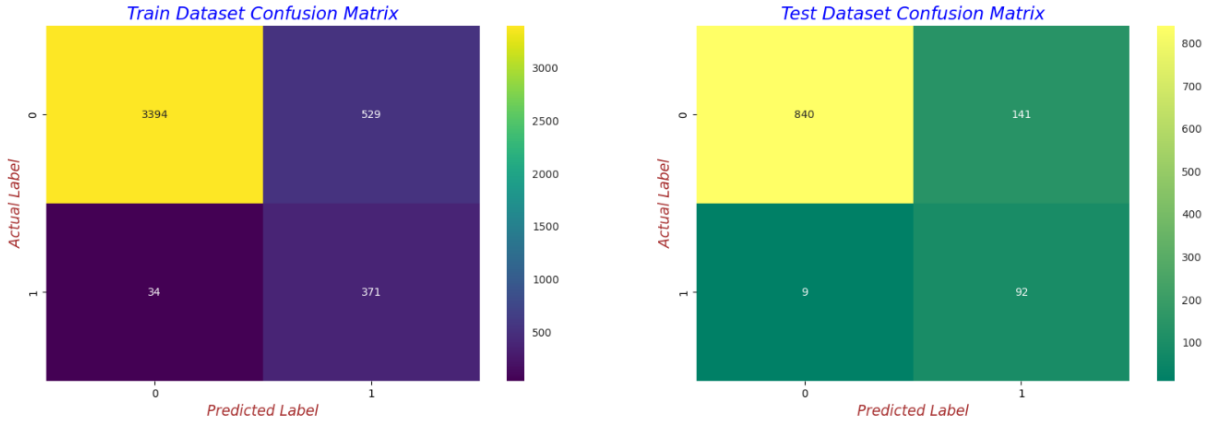
```
[ ] from sklearn.linear_model import LogisticRegressionCV
    from sklearn import metrics
    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, roc_curve, auc
    from sklearn.linear_model import LogisticRegression
    from sklearn.model_selection import GridSearchCV
    from sklearn.model_selection import RandomizedSearchCV
    from sklearn.calibration import CalibratedClassifierCV
```

```
# Training the model with all features and hyper-parameterized values
log_reg_1 = LogisticRegression(C=0.0316228, penalty='l1',
                               fit_intercept=True, solver='liblinear', tol=0.0001, max_iter=500,
                               class_weight='balanced',
                               verbose=0,
                               intercept_scaling=1.0,
                               multi_class='auto',
                               random_state=49)

log_reg_1.fit(X_train_std, y_train)
```

```
LogisticRegression(C=0.0316228, class_weight='balanced', intercept_scaling=1.0,
                   max_iter=500, penalty='l1', random_state=49,
                   solver='liblinear')
```

```
def pred_prob(clf, data):
    y_pred = clf.predict_proba(data)[:,1]
    return y_pred

def draw_roc(train_fpr, train_tpr, test_fpr, test_tpr):

    # calculate auc for train and test
    train_auc = auc(train_fpr, train_tpr)
    test_auc = auc(test_fpr, test_tpr)
    with plt.style.context('seaborn-poster'):
      plt.plot(train_fpr, train_tpr, label="Train AUC ="+"{:.4f}".format(train_auc), color='blue')
      plt.plot(test_fpr, test_tpr, label="Test AUC ="+"{:.4f}".format(test_auc), color='red')
      plt.legend()
      plt.xlabel("False Positive Rate(FPR)", fontdict=label_font_dict)
      plt.ylabel("True Positive Rate(TPR)", fontdict=label_font_dict)
      plt.title("Area Under Curve", fontdict=title_font_dict)
      plt.grid(visible=True, which='major', color='lightgrey', linestyle='--')
      plt.minorticks_on()
      plt.show()

def find_best_threshold(threshold, fpr, tpr):

    t = threshold[np.argmax(tpr * (1-fpr))]
    return t

def predict_with_best_t(proba, threshold):

    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

def draw_confusion_matrix(best_t, x_train, x_test, y_train, y_test, y_train_pred, y_test_pred):

    fig, ax = plt.subplots(1,2, figsize=(20,6))

    train_prediction = predict_with_best_t(y_train_pred, best_t)
    cm = confusion_matrix(y_train, train_prediction)
```






```
### Best Threshold = 0.3544
### Model AUC is : 0.9518
### Model Train F1 Score is : 0.5686
### Model Test F1 Score is : 0.5509
```

Decision Tree

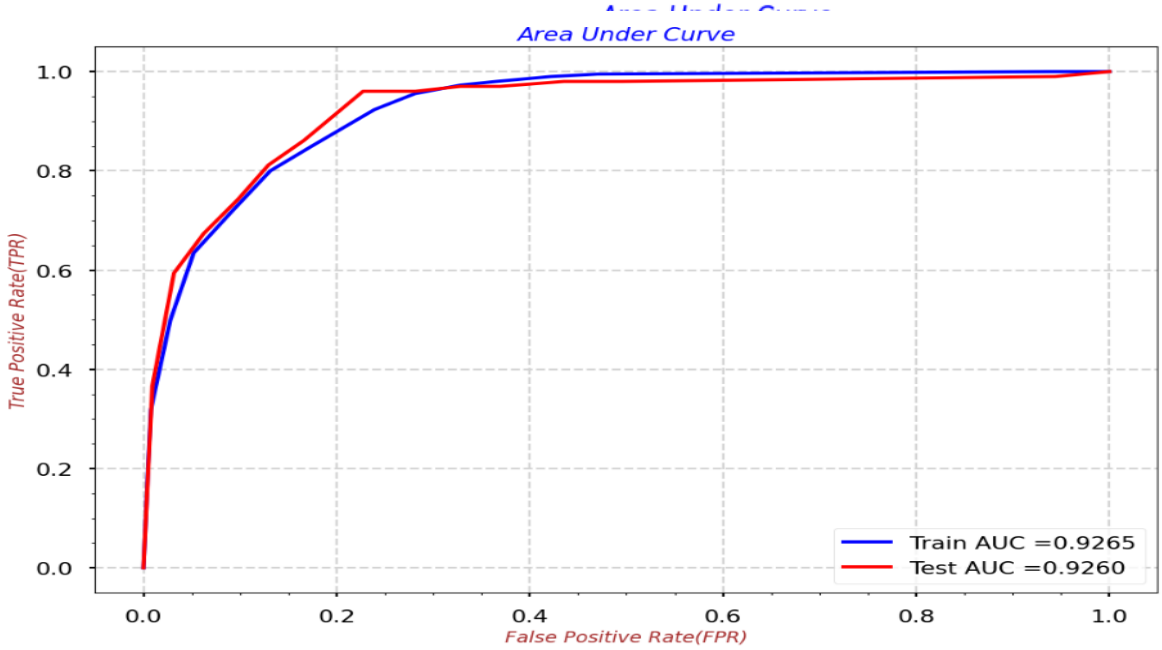
```
[ ] from sklearn.tree import DecisionTreeClassifier
```

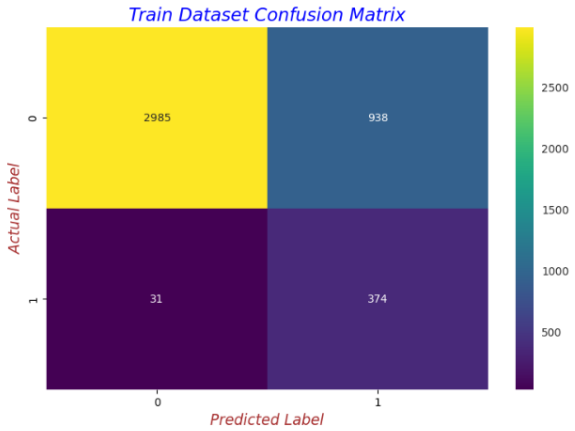
```
# Training the model with all features and hyper-parameterized values
dec_tree_2 = DecisionTreeClassifier(criterion='gini',
                                    max_depth= 6,
                                    max_features='log2',
                                    min_samples_leaf=150,
                                    min_samples_split=150,
                                    class_weight='balanced',
                                    random_state=49,
                                    splitter='best',
                                    min_weight_fraction_leaf=0.0,
                                    max_leaf_nodes=None,
                                    min_impurity_decrease=0.0,
                                    ccp_alpha=0.0,)

dec_tree_2.fit(X_train_std, y_train)
```

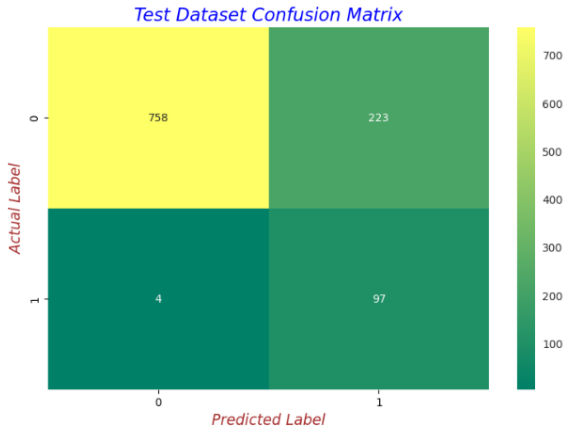
```
DecisionTreeClassifier(class_weight='balanced', max_depth=6,
                       max_features='log2', min_samples_leaf=150,
                       min_samples_split=150, random_state=49)
```

```
[ ] # Validate model
test_auc, train_f1_score, test_f1_score, best_t = validate_model(dec_tree_2, X_train_std, X_test_std, y_train, y_test)

print("\n")
print("### Best Threshold = {:.4f}".format(best_t))
print("### Model AUC is : {:.4f}".format(test_auc))
print("### Model Train F1 Score is : {:.4f}".format(train_f1_score))
print("### Model Test F1 Score is : {:.4f}".format(test_f1_score))
```

```
### Train AUC = 0.9264832595361951
### Test AUC = 0.9259948930672883
```





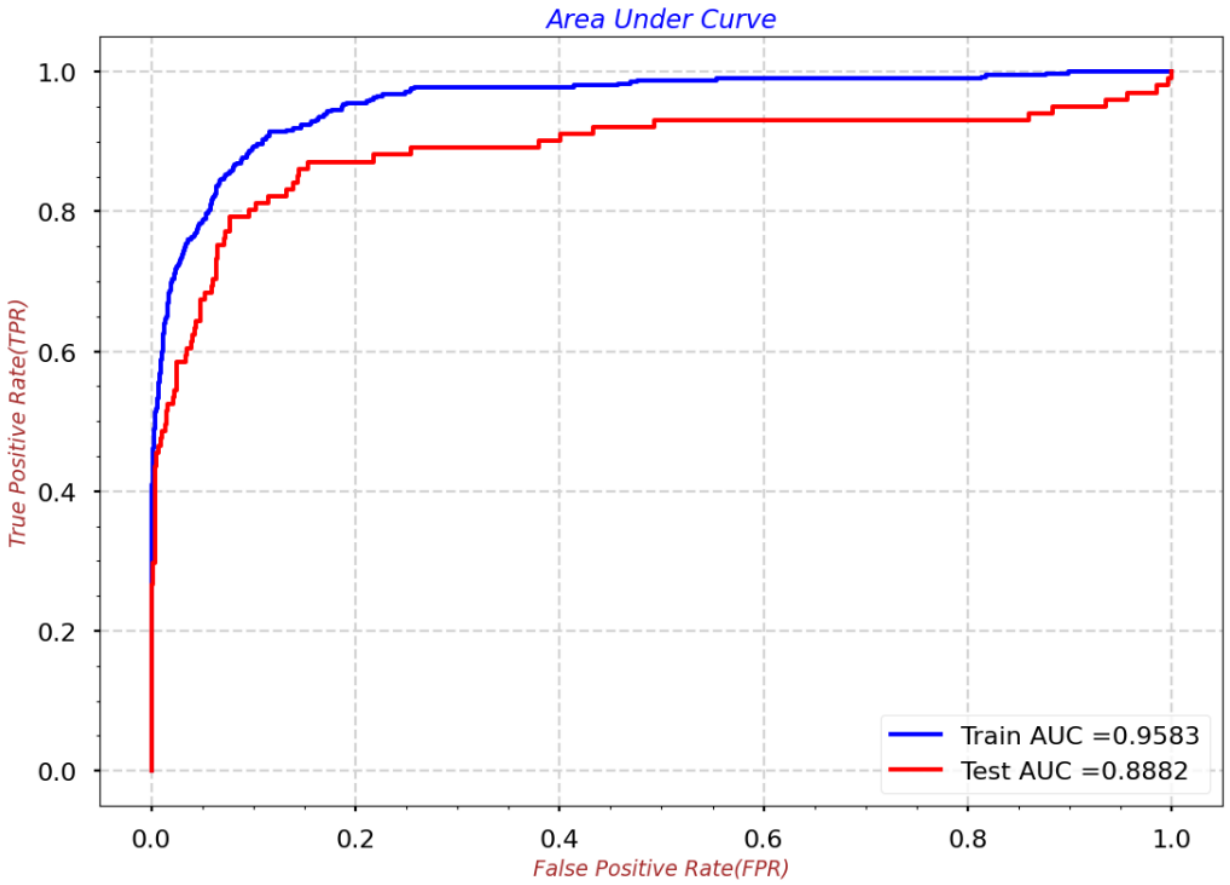


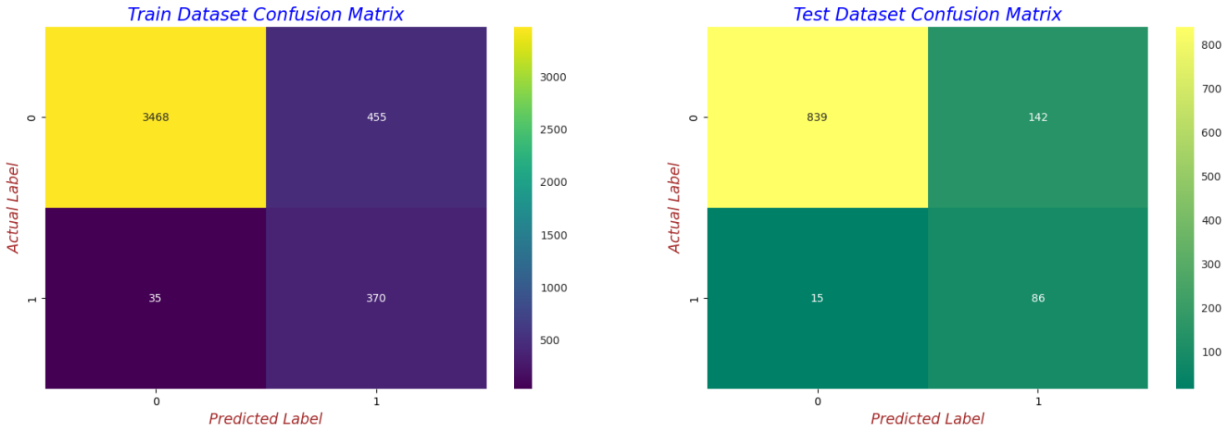
Linear Discrimant Analysis

```
[ ]  # Import the necessary libraries
     from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

     # Create an LDA classifier
     lda = LinearDiscriminantAnalysis()

     # Train the LDA classifier on the standardized training data
     lda.fit(X_train_std, y_train)

     # Now you can use the trained LDA classifier for making predictions on new data
     y_pred = lda.predict(X_test_std)
```

```
[ ]  # Validate model
     test_auc, train_f1_score, test_f1_score, best_t = validate_model(lda, X_train_std, X_test_std, y_train, y_test)

     print("\n")
     print("### Best Threshold = {:.4f}".format(best_t))
     print("### Model AUC is : {:.4f}".format(test_auc))
     print("### Model Train F1 Score is : {:.4f}".format(train_f1_score))
     print("### Model Test F1 Score is : {:.4f}".format(test_f1_score))
```

```
### Train AUC = 0.9582915569150595
### Test AUC = 0.888172303468879
```






```
### Best Threshold = 0.0012
### Model AUC is : 0.8882
### Model Train F1 Score is : 0.6016
### Model Test F1 Score is : 0.5228
```

Random Forest Classifier

```
[ ] from sklearn.ensemble import RandomForestClassifier
```

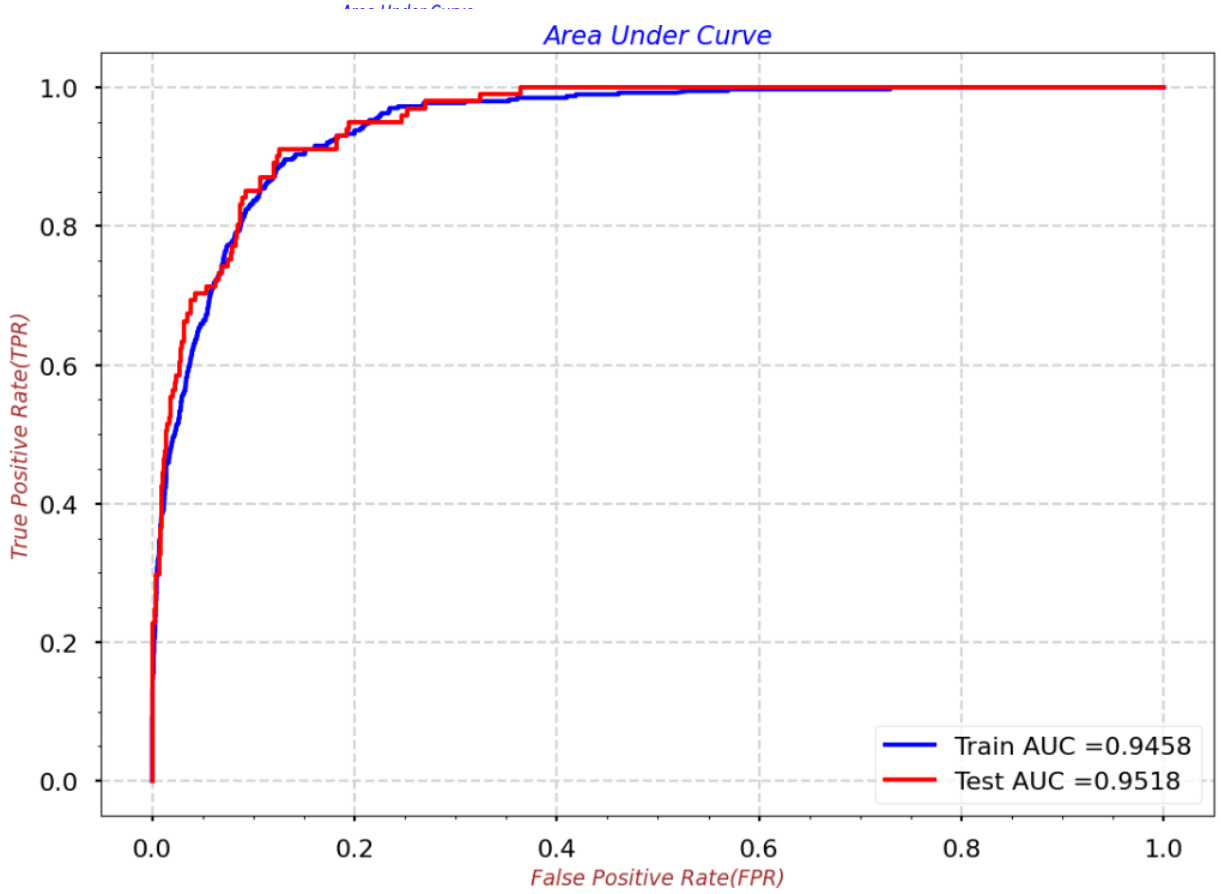
```
# Training the model with all features and hyper-parameterized values
rfc_3 = RandomForestClassifier(n_estimators=30,criterion='gini',
                                    max_depth= 4,
                                    max_features='auto',
                                    min_samples_leaf=50,
                                    min_samples_split=50,
                                    class_weight='balanced',
                                    random_state=49,
                                    min_weight_fraction_leaf=0.0,
                                    max_leaf_nodes=None,
                                    min_impurity_decrease=0.0,
                                    ccp_alpha=0.0,)

rfc_3.fit(X_train_std, y_train)
```

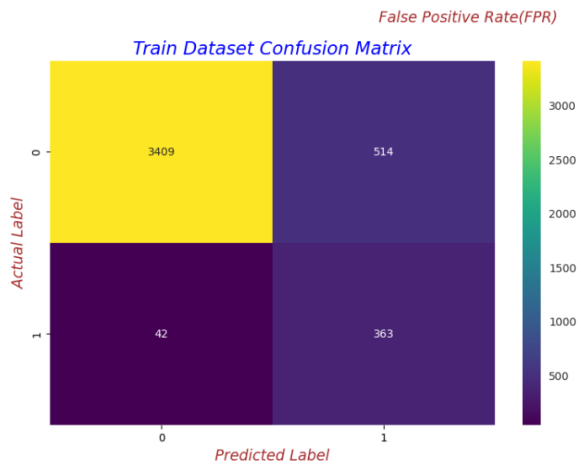
```
/usr/local/lib/python3.9/dist-packages/sklearn/ensemble/_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove thi
  warn(
```

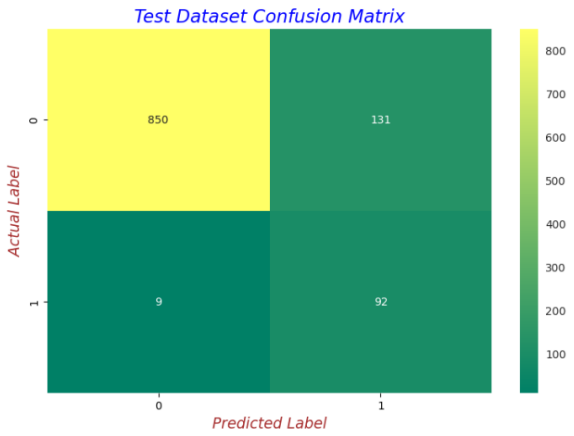
```
RandomForestClassifier(class_weight='balanced', max_depth=4,
                       max_features='auto', min_samples_leaf=50,
                       min_samples_split=50, n_estimators=30, random_state=49)
```

```
[ ] # Validate model
test_auc, train_f1_score, test_f1_score, best_t = validate_model(rfc_3, X_train_std, X_test_std, y_train, y_test)

print("\n")
print("### Best Threshold = {:.4f}".format(best_t))
print("### Model AUC is : {:.4f}".format(test_auc))
print("### Model Train F1 Score is : {:.4f}".format(train_f1_score))
print("### Model Test F1 Score is : {:.4f}".format(test_f1_score))
```

```
### Train AUC = 0.9457812898292124
### Test AUC = 0.9517970145638417
```





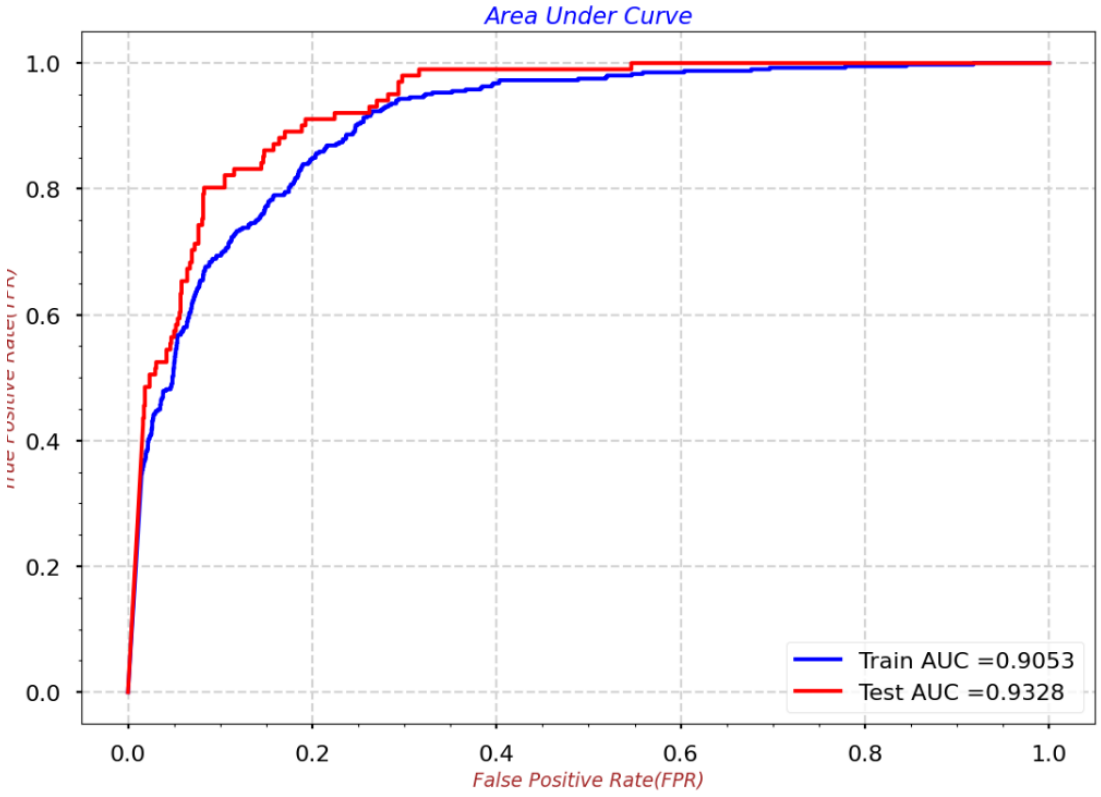


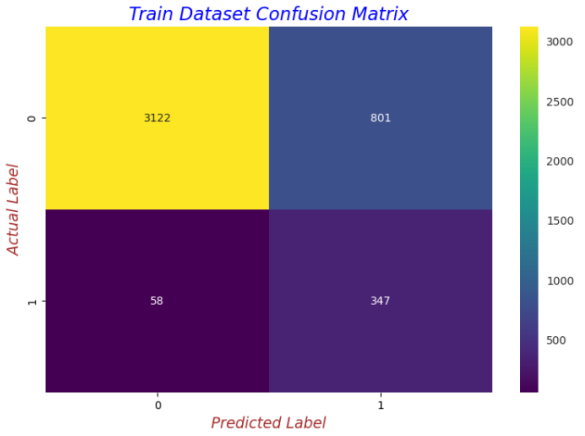
Naive Bayes

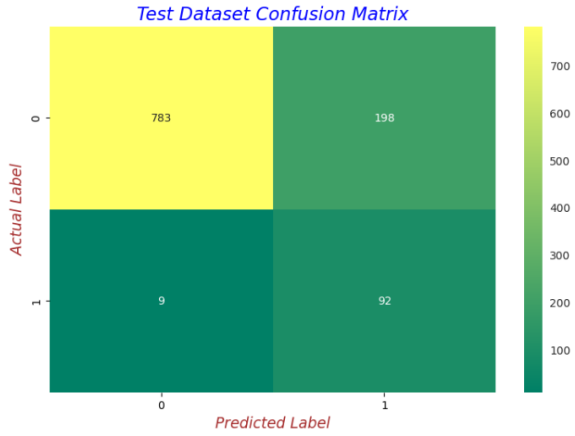
```
[ ] from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
    from sklearn.model_selection import train_test_split

    # Import the necessary libraries
    from sklearn.naive_bayes import GaussianNB

    # Create a Gaussian Naive Bayes classifier
    gnb = GaussianNB()

    # Train the Naive Bayes classifier on the standardized training data
    gnb.fit(X_train_std, y_train)

    # Now you can use the trained Naive Bayes classifier for making predictions on new data
    y_pred = gnb.predict(X_test_std)
```

```
[ ] test_auc, train_f1_score, test_f1_score, best_t = validate_model(gnb, X_train_std, X_test_std, y_train, y_test)

    print("\n")
    print("### Model AUC is : {:.4f}".format(test_auc))
    print("### Model Train F1 Score is : {:.4f}".format(train_f1_score))
    print("### Model Test F1 Score is : {:.4f}".format(test_f1_score))
```

```
### Train AUC = 0.9052624125527516
### Test AUC = 0.9327822690525933
```


Area Under Curve
True Positive Rate(TPR)
False Positive Rate(FPR)
Train AUC =0.9053
Test AUC =0.9328


Train Dataset Confusion Matrix
Actual Label
Predicted Label
3122
801
58
347

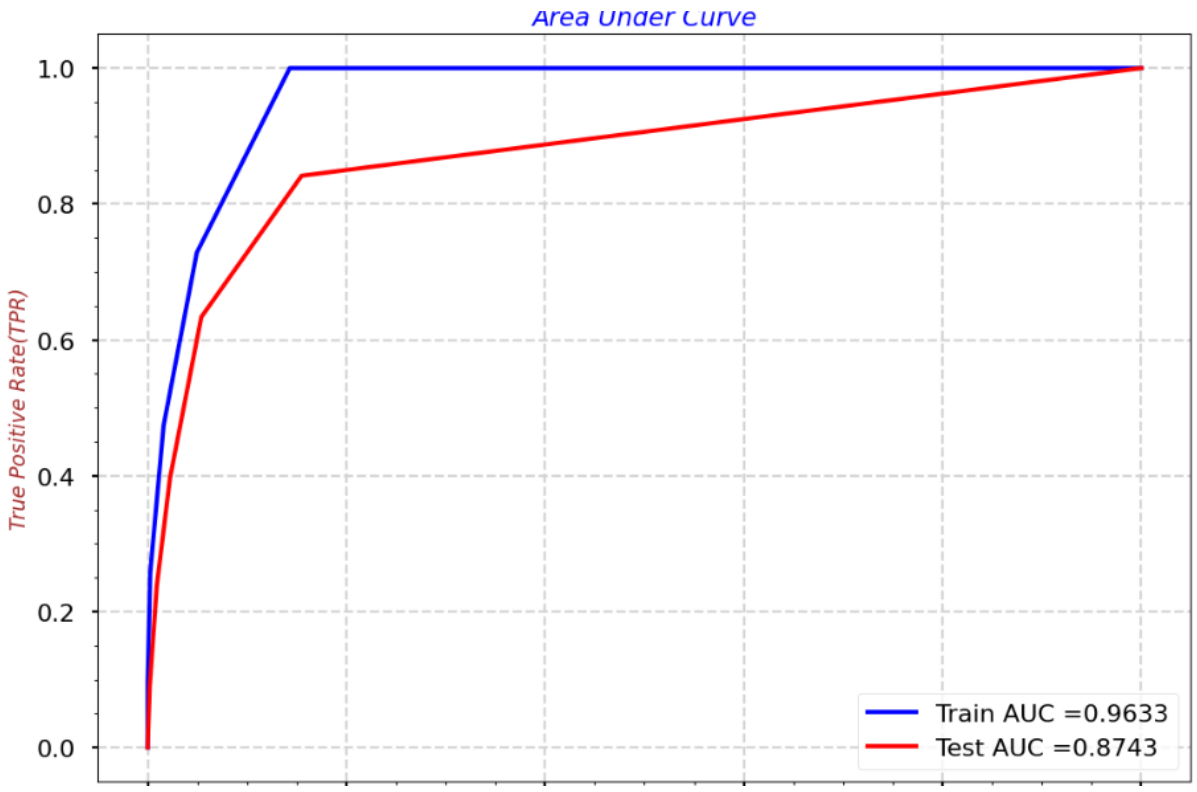

Test Dataset Confusion Matrix
Actual Label
Predicted Label
783
198
9
92

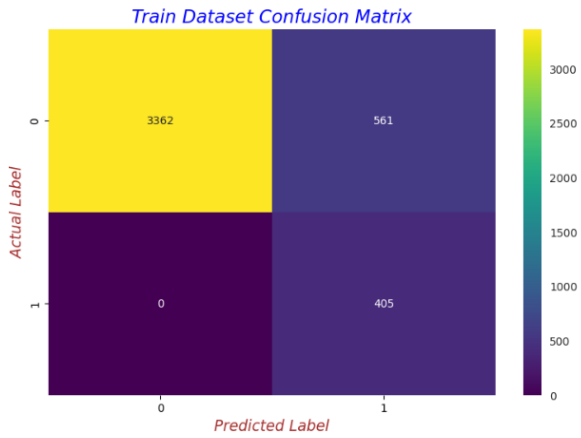
KNN

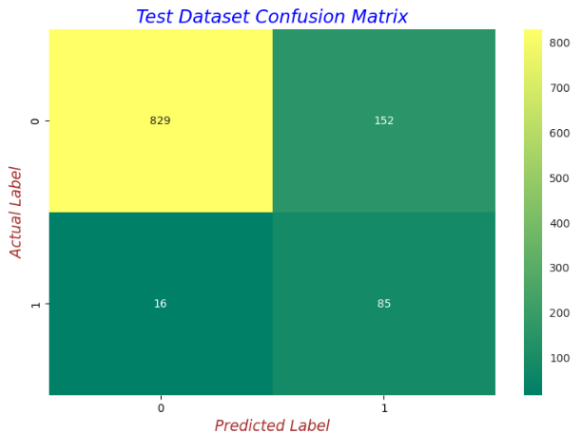
```
[ ]  # Import the necessary libraries
     from sklearn.neighbors import KNeighborsClassifier

     # Create a KNN classifier
     knn = KNeighborsClassifier(n_neighbors=5)

     # Train the KNN classifier on the standardized training data
     knn.fit(X_train_std, y_train)

     # Now you can use the trained KNN classifier for making predictions on new data
     y_pred = knn.predict(X_test_std)
```

```
[ ]  # Validate model
     test_auc, train_f1_score, test_f1_score, best_t = validate_model(knn, X_train_std, X_test_std, y_train, y_test)

     print("\n")
     print("### Best Threshold = {:.4f}".format(best_t))
     print("### Model AUC is : {:.4f}".format(test_auc))
     print("### Model Train F1 Score is : {:.4f}".format(train_f1_score))
     print("### Model Test F1 Score is : {:.4f}".format(test_f1_score))
```

```
### Train AUC = 0.9633075594074829
### Test AUC = 0.8742846761740394
```



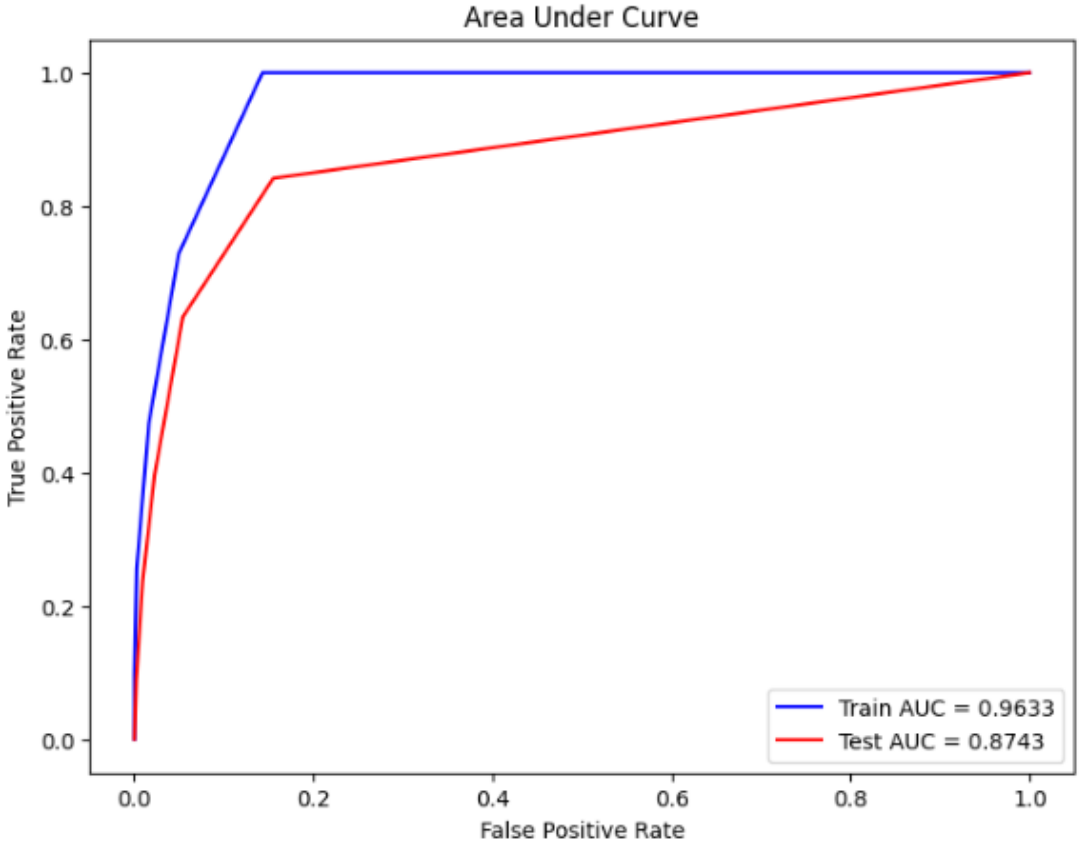


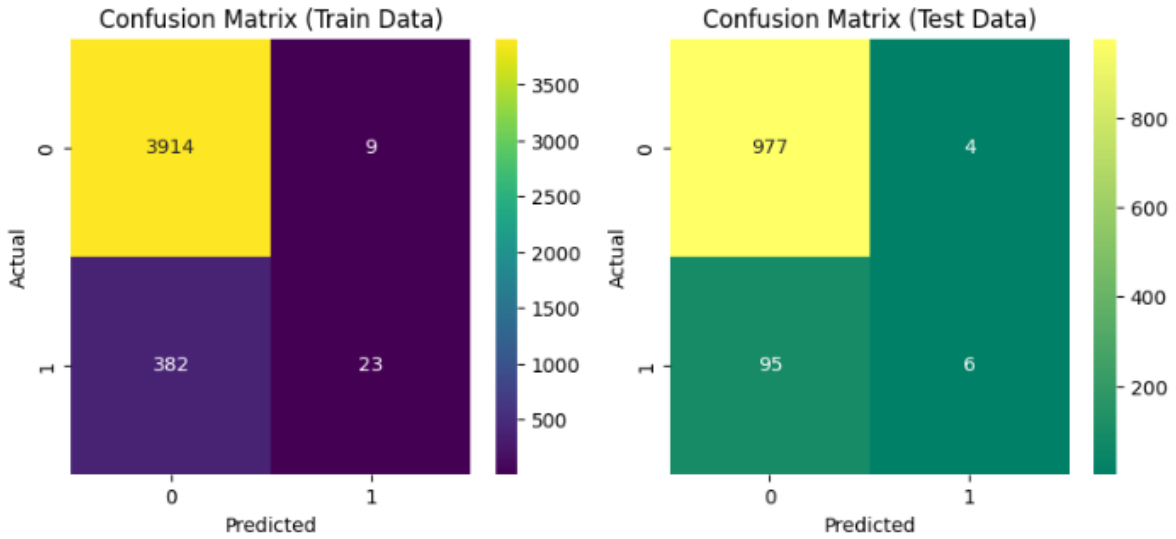


Neural Network

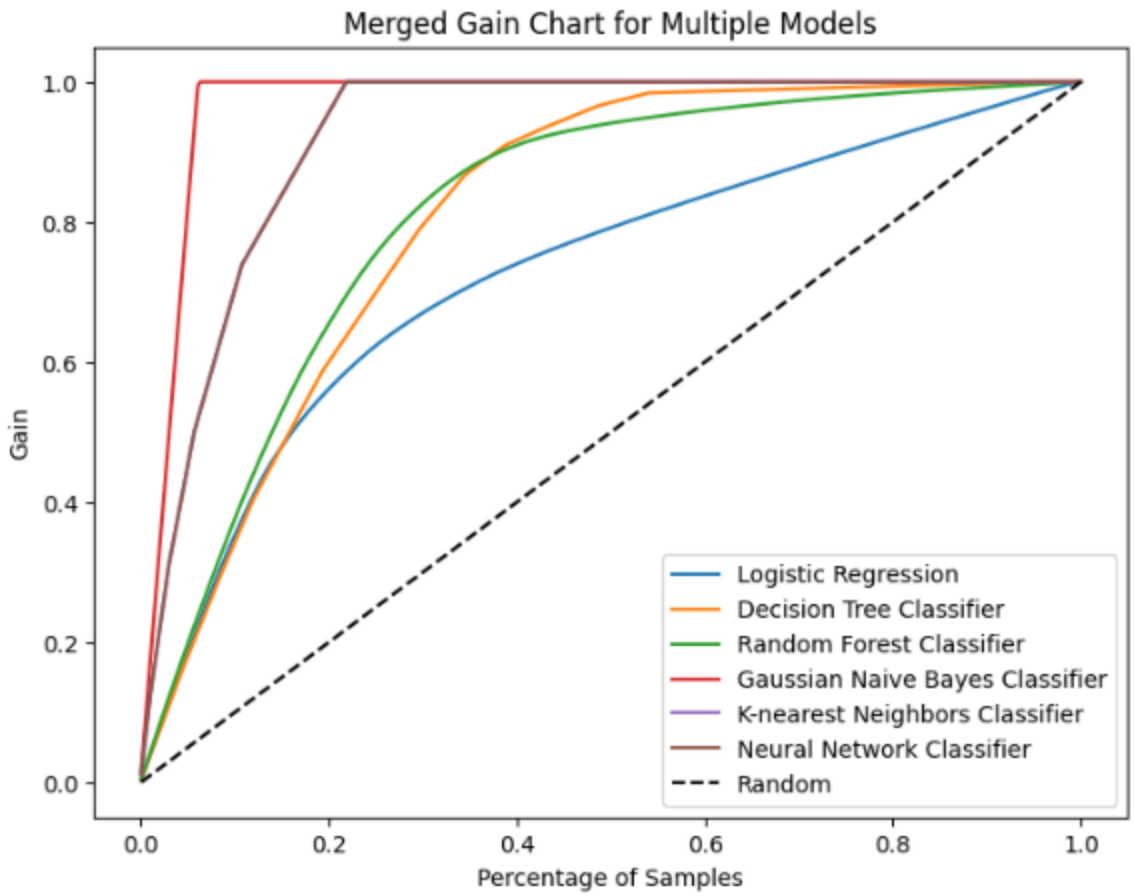
```
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import f1_score, roc_auc_score, roc_curve, confusion_matrix
from sklearn.metrics import classification_report

# Define the neural network model
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(X_train_std.shape[1],)),  # Input layer with input shape same as X_train_std
    tf.keras.layers.Dense(units=5, activation='relu'),  # Hidden layer with 5 units and ReLU activation
    tf.keras.layers.Dense(units=len(np.unique(y_train)), activation='softmax')  # Output layer with softmax activation for multiclass classification
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model on the standardized training data
model.fit(X_train_std, y_train, epochs=10, batch_size=32)

# Predict probabilities for train and test data
y_train_pred_probs = model.predict(X_train_std)
y_test_pred_probs = model.predict(X_test_std)

# Convert predicted probabilities to class labels using best threshold
best_t = 0.5  # Example threshold value
y_train_pred = np.argmax(y_train_pred_probs, axis=1)
y_test_pred = np.argmax(y_test_pred_probs, axis=1)

# Calculate AUC for test data
test_auc = roc_auc_score(y_test, y_test_pred_probs[:, 1])  # Assuming binary classification, use probabilities of class 1

# Calculate F1 score for train and test data
train_f1_score = f1_score(y_train, y_train_pred, average='weighted')
test_f1_score = f1_score(y_test, y_test_pred, average='weighted')

# Plot ROC curve
from sklearn.metrics import roc_auc_score, roc_curve

# Calculate predicted probabilities for train and test data
y_train_pred_prob = knn.predict_proba(X_train_std)[:, 1]
y_test_pred_prob = knn.predict_proba(X_test_std)[:, 1]

# Calculate ROC AUC scores for train and test data
train_auc = roc_auc_score(y_train, y_train_pred_prob)
test_auc = roc_auc_score(y_test, y_test_pred_prob)

# Plot ROC curves for train and test data
fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_train_pred_prob)
```

```
Epoch 10/10
136/136 [==============================] - 0s 2ms/step - loss: 0.7518 - accuracy: 0.9048
136/136 [==============================] - 0s 1ms/step
34/34 [==============================] - 0s 1ms/step
```






```
### Best Threshold = 0.5000
### Model AUC is : 0.8743
### Model Train F1 Score is : 0.8732
### Model Test F1 Score is : 0.8730
```



	F1 Score		Accuracy	AUC
Models	**TRAIN**	**TEST**		
Logistic Regression	0.56	0.55	90.39%	0.95
Decision Tree	0.43	0.46	77.89%	0.92
Linear Discrimant Analysis	0.6	0.52	92.32%	0.88
Random Forest	0.57	0.57	85.49%	0.95
Naïve Bayes	0.44	0.47	93.44%	0.93
KNN	0.59	0.5	92.32%	0.87
Neural Network	0.86	0.86	90.25%	0.87

Conclusion and Recommendation

•Incorporating natural language processing (NLP) capabilities: Currently, the AI-based system may primarily focus on analyzing structured data, such as claims data. However, incorporating NLP capabilities would enable the system to analyze unstructured data, such as medical notes and doctor's comments. This could potentially help to identify fraudulent claims that are disguised as legitimate ones.

•Implementing real-time feedback: Currently, the system may only flag suspicious claims for further investigation. However, implementing real-time feedback would enable the system to immediately alert providers when a potentially fraudulent claim is submitted. This could help to deter fraudulent behavior in real-time, as providers may be less likely to submit false claims if they know they are being closely monitored.

•Leveraging blockchain technology: Blockchain technology could be leveraged to create a secure and transparent system for storing and sharing Medicare data. This could help to reduce the risk of data breaches or tampering and provide greater visibility into the healthcare ecosystem. Additionally, blockchain technology could potentially enable patients to control their own healthcare data, which could help to increase trust and engagement in the healthcare system.

•Scaling the system to other healthcare programs: The AI-based fraud detection system could be scaled to other government-funded healthcare programs, such as Medicaid or Veterans Affairs (VA) healthcare. This would enable the system to identify patterns of fraud across multiple programs and potentially reduce overall healthcare costs. Additionally, the system could be adapted to work in other countries with government-funded healthcare systems, which could help to reduce fraud on a global scale.