Authenticate to Kaggle

```
!pip install kaggle --quiet
```

```
%env KAGGLE_USERNAME=jiabaozhuang
%env KAGGLE_KEY=5a9074c902ea26301cb9c242222a49
```

```
env: KAGGLE_USERNAME=jiabaozhuang
env: KAGGLE_KEY=5a9074c902ea26301cb9c24222
```

```
!kaggle datasets download -d sakshigoyal7/cred
```

```
Dataset URL: https://www.kaggle.com/datase
License(s): CC0-1.0
Downloading credit-card-customers.zip to /
100% 379k/379k [00:00<00:00, 764kB/s]
100% 379k/379k [00:00<00:00, 764kB/s]
```

```
!unzip credit-card-customers.zip
```

```
Archive:  credit-card-customers.zip
  inflating: BankChurners.csv
```

Data Loading

```
# Import all of the packages we will need
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objs as go
from plotly.subplots import make_subplots
import seaborn as sns
```

```
# Load data
df = pd.read_csv("BankChurners.csv")

# Asked to ignore last two columns
df = df[df.columns[:-2]]
df = df.drop('CLIENTNUM', axis=1)
df.head()
```

	Attrition_Flag	Customer_Age	Gender	De
0	Existing Customer	45	M	
1	Existing Customer	49	F	
2	Existing Customer	51	M	
3	Existing Customer	40	F	
4	Existing Customer	40	M	

```
df.shape
```

```
(10127, 20)
```

```
numerical_features = df.select_dtypes(include=
numerical_features.describe().T.round(4)
```

	count	mean	
Customer_Age	10127.0	46.3260	
Dependent_count	10127.0	2.3462	
Months_on_book	10127.0	35.9284	
Total_Relationship_Count	10127.0	3.8126	
Months_Inactive_12_mon	10127.0	2.3412	
Contacts_Count_12_mon	10127.0	2.4553	
Credit_Limit	10127.0	8631.9537	90
Total_Revolving_Bal	10127.0	1162.8141	8
Avg_Open_To_Buy	10127.0	7469.1396	90
Total_Amt_Chng_Q4_Q1	10127.0	0.7599	
Total_Trans_Amt	10127.0	4404.0863	33
Total_Trans_Ct	10127.0	64.8587	
Total_Ct_Chng_Q4_Q1	10127.0	0.7122	
Avg_Utilization_Ratio	10127.0	0.2749	

Yannaphol Kaewbaidhoon
Apr 27, 2024
(edited Apr 27, 2024)

Please round to 4 decimals.

Yannaphol Kaewbaidhoon
Apr 27, 2024

@jz5863@nyu.edu

Yannaphol Kaewbaidhoon
Apr 27, 2024

Re-opened

```
categorical_features = df.select_dtypes(includ
categorical_features.describe().T
```

	count	unique	top	fre
Attrition_Flag	10127	2	Existing Customer	85(
Gender	10127	2	F	53!
Education_Level	10127	7	Graduate	31;
Marital_Status	10127	4	Married	46{
Income_Category	10127	6	Less than $40K	35(

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 20 columns):
 #   Column                    Non-Null Co
---  ------                    -----------
 0   Attrition_Flag            10127 non-n
 1   Customer_Age              10127 non-n
 2   Gender                    10127 non-n
 3   Dependent_count           10127 non-n
 4   Education_Level           10127 non-n
 5   Marital_Status            10127 non-n
 6   Income_Category           10127 non-n
 7   Card_Category             10127 non-n
 8   Months_on_book            10127 non-n
 9   Total_Relationship_Count  10127 non-n
 10  Months_Inactive_12_mon    10127 non-n
 11  Contacts_Count_12_mon     10127 non-n
 12  Credit_Limit              10127 non-n
 13  Total_Revolving_Bal       10127 non-n
 14  Avg_Open_To_Buy           10127 non-n
 15  Total_Amt_Chng_Q4_Q1      10127 non-n
 16  Total_Trans_Amt           10127 non-n
 17  Total_Trans_Ct            10127 non-n
 18  Total_Ct_Chng_Q4_Q1       10127 non-n
 19  Avg_Utilization_Ratio     10127 non-n
dtypes: float64(5), int64(9), object(6)
memory usage: 1.5+ MB
```

```
na_count = df.isna().sum()
```

```
na_count
```

```
Attrition_Flag             0
Customer_Age               0
Gender                     0
Dependent_count            0
Education_Level            0
Marital_Status             0
```

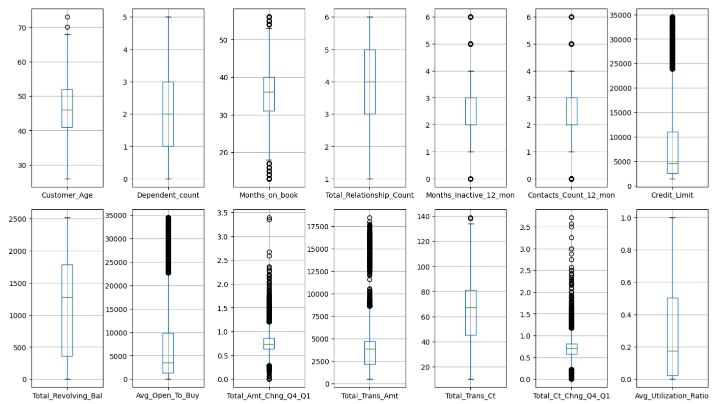
```
Income_Category             0
Card_Category               0
Months_on_book              0
Total_Relationship_Count    0
Months_Inactive_12_mon      0
Contacts_Count_12_mon       0
Credit_Limit                0
Total_Revolving_Bal         0
Avg_Open_To_Buy             0
Total_Amt_Chng_Q4_Q1        0
Total_Trans_Amt             0
Total_Trans_Ct              0
Total_Ct_Chng_Q4_Q1         0
Avg_Utilization_Ratio       0
dtype: int64
```

```
df.columns
```

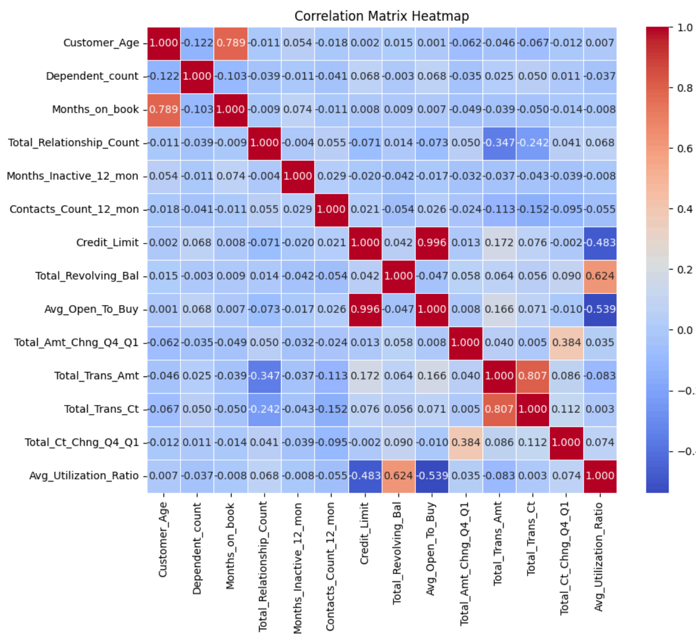
```
Index(['Attrition_Flag', 'Customer_Age',
'Gender', 'Dependent_count',
       'Education_Level',
'Marital_Status', 'Income_Category',
'Card_Category',
       'Months_on_book',
'Total_Relationship_Count',
'Months_Inactive_12_mon',
       'Contacts_Count_12_mon',
'Credit_Limit', 'Total_Revolving_Bal',
       'Avg_Open_To_Buy',
'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt',
       'Total_Trans_Ct',
'Total_Ct_Chng_Q4_Q1',
'Avg_Utilization_Ratio'],
      dtype='object')
```

Exploratory Data Analysis

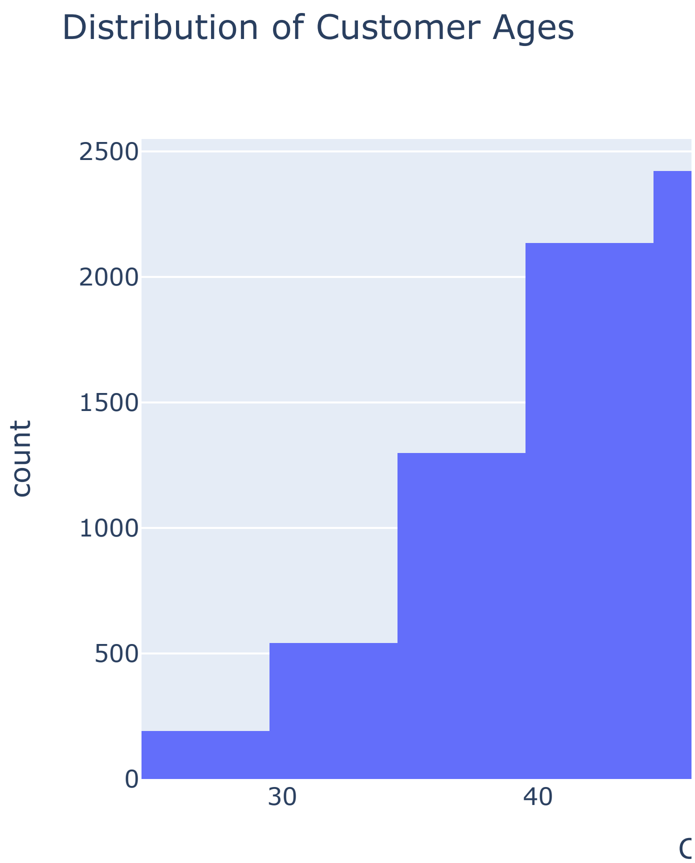
```
# Box plot to determine outliers
fig, axes = plt.subplots(nrows=2,ncols=7)
fig.set_figheight(8)
fig.set_figwidth(14)
fig.tight_layout()

i = 0
j = 0
for cols in numerical_features.columns:
  numerical_features.boxplot(column=cols, ax=a
  j = j + 1
  if j == 7:
    j = 0
    i = i + 1

plt.show()
```



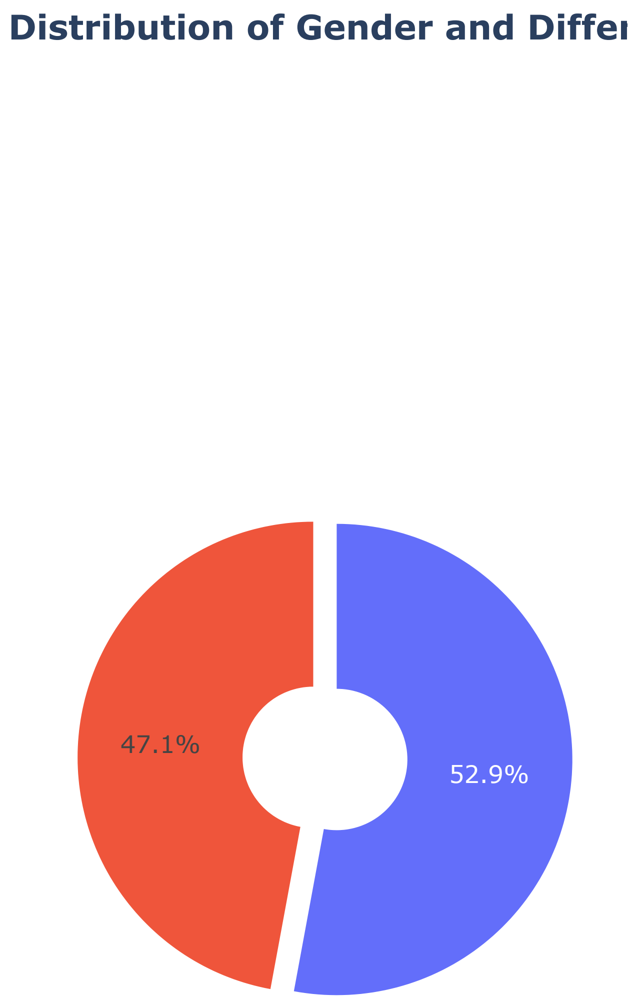
```
corr_matrix = numerical_features.corr()

# Generate a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".3f"
plt.title('Correlation Matrix Heatmap')
plt.show()
```



```
fig = px.histogram(df, x='Customer_Age', nbins
fig.update_layout(height=500, width=800)
fig.show()
```



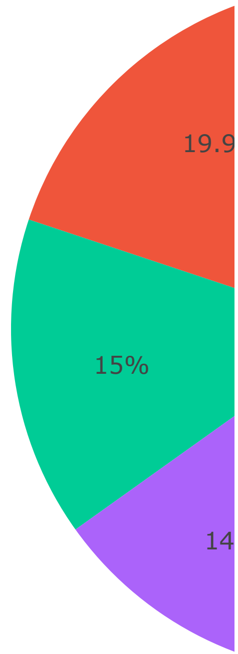
```
fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=('', '<b>Platinum Card Hold
    vertical_spacing=0.09,
    specs=[
        [{"type": "pie", "rowspan": 2}, {"type
        [None, {"type": "pie"}]
    ]
)
fig.add_trace(
    go.Pie(
        values=df['Gender'].value_counts().val
        labels=['Female', 'Male'],
        hole=0.3,
        pull=[0, 0.1]  # Adjust the pull to ha
    ),
    row=1, col=1
)
fig.add_trace(
    go.Pie(
        labels=['Female Platinum Card Holders'
        values=df[df['Card_Category'] == "Plat
        hole=0.3,
        pull=[0, 0.1]  # Adjust the pull to ha
    ),
    row=1, col=2
)
fig.add_trace(
    go.Pie(
        labels=['Female Blue Card Holders', 'M
        values=df[df['Card_Category'] == "Blue
        hole=0.3,
        pull=[0, 0.1]  # Adjust the pull to ha
    ),
    row=2, col=2
)
fig.update_layout(
    height=800,
    showlegend=True,
    title_text="<b>Distribution of Gender and
)
fig.show()
"""The dataset shows a balanced gender distrib
```




```
'The dataset shows a balanced gender distr
ibution: 52.9% female and 47.1% male. This
balance is maintained across Platinum and
```

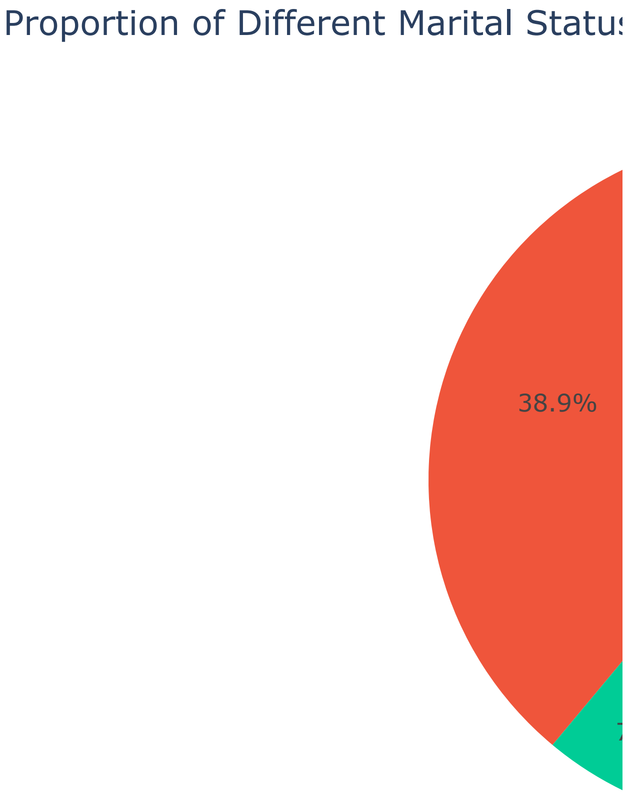
```
fig = px.pie(df, names='Education_Level', titl
fig.show()
"""Assuming 'Unknown' does not significantly r
```

Proportion Of Education Levels



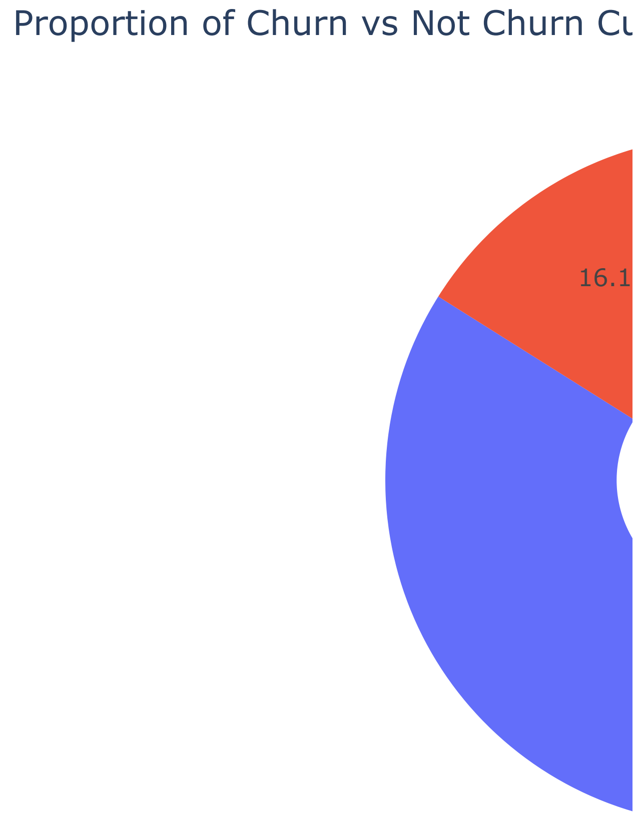
'Assuming 'Unknown' does not significantly
represent formal education, the data shows
that over 70% of our customers have formal
education. Among these, nearly 10% have at

```
fig = px.pie(df, names='Marital_Status', title
fig.show()
"""Close to half of the bank's customers are m
```




```
'Close to half of the bank's customers are
married (46.3%), while a significant propo
rtion are single (38.9%). A smaller fracti
```

```
fig = px.pie(df, names='Attrition_Flag', title
fig.show()
"""In our dataset with 16% churn customers, I
```




```
'In our dataset with 16% churn customers,
I will use SMOTE to balance the class dist
ribution, supplemented by adjusted class w
eights in the modeling process. This combi
ned method will help the model capture chu
```

Create a Data Pipeline

```
from sklearn.compose import ColumnTransformer
from imblearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScal

# Manual encoding
df['Attrition_Flag'] = df['Attrition_Flag'].re
df.Gender = df.Gender.replace({'F':1,'M':0})

# Define transformers for numerical and catego
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='consta
    ('onehot', OneHotEncoder(handle_unknown='i
])



# Define numerical and categorical columns
categorical_columns = df.select_dtypes(include
numerical_columns = df.select_dtypes(include=[

# Remove target variable from numerical column
numerical_columns = numerical_columns.drop('At

# Combine transformers using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numeric
        ('cat', categorical_transformer, categ
    ], remainder='passthrough')

# Create a pipeline with the preprocessor
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor)])

# Prepare target and feature sets
X = df.drop(['Attrition_Flag'], axis=1)
y = df['Attrition_Flag']
X_preprocessed = pipeline.fit_transform(X)
```

ROC curves

```
from sklearn import metrics

def get_model_roc(models, Xs_test, names, Y_te
    plt.figure(figsize=(10, 8))  # Set the fig
    plt.rcParams['figure.dpi'] = 100  # Set th
    plt.plot([0, 1], [0, 1], linestyle='--', c

    # Iterate over the models to plot each ROC
    for model, X_test, name in zip(models, Xs_
        probs = model.predict_proba(X_test)[:,
        fpr, tpr, thresholds = metrics.roc_cur
        plt.plot(fpr, tpr, label=f"{name} (AUC

    # Adding labels, title and legend
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve Comparison")
    plt.legend(title="Models", loc="lower righ
    plt.grid(True)
    plt.show()
```

Fit and Parameter Tune models

```
from sklearn.linear_model import LogisticRegre
from sklearn.ensemble import RandomForestClass
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassi
from sklearn.model_selection import GridSearch
from sklearn.metrics import accuracy_score, f1
from imblearn.over_sampling import SMOTE

# Split the data into training and testing set
X_train, X_test, y_train, y_test = train_test_

# Apply SMOTE to the preprocessed training dat
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resam

# Define the models for classification
models = {
    'NaiveBayes': GaussianNB(), # Baseline Mod
    'LogisticRegression': LogisticRegression(r
    'KNN': KNeighborsClassifier(),
    'RandomForest': RandomForestClassifier(ran
    'GradientBoosting': GradientBoostingClassi
}

# Define the hyperparameter grids for each mod
param_grids = {
    'NaiveBayes': {},
    'LogisticRegression': {
        'C': [0.1, 1, 10]
    },
    'KNN': {
    'n_neighbors': [3, 5, 7, 10],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
    },
    'RandomForest': {
        'n_estimators': [100, 200, 500],
        'max_depth': [None, 10, 30],
        'min_samples_split': [2, 5, 10],
    },
    'GradientBoosting': {
        'n_estimators': [100, 200],
        'learning_rate': [0.01, 0.1],
        'max_depth': [3, 5]
    }
}

# 3-fold cross-validation
cv = KFold(n_splits=3, shuffle=True, random_st

# Train and tune the models using GridSearchCV
```

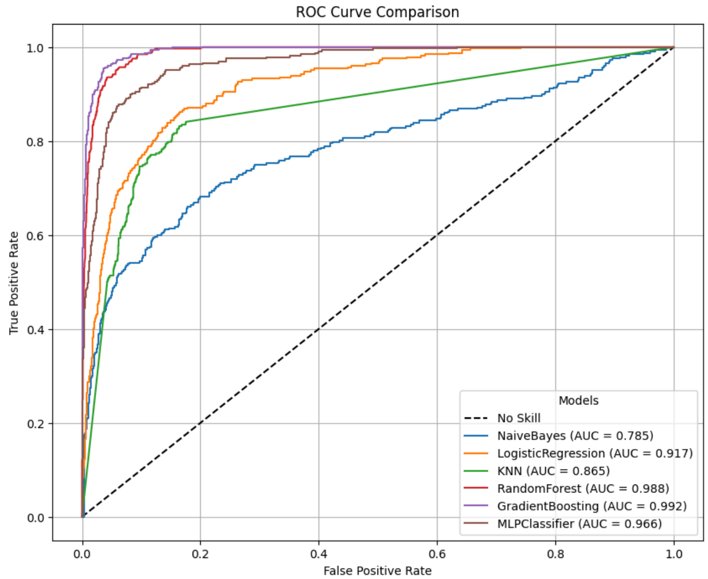
```
grids = {}
best_models = []
model_names = []
Xs_test = []

for model_name, model in models.items():
    grid_search = GridSearchCV(estimator=model
    grid_search.fit(X_train_smote, y_train_smo
    best_model = grid_search.best_estimator_
    best_params = grid_search.best_params_
    best_score = grid_search.best_score_

    best_models.append(best_model)
    model_names.append(model_name)
    Xs_test.append(X_test)

    print(f'Best parameters for {model_name}:
    print(f'Best accuracy for {model_name}: {b
```

```
Fitting 3 folds for each of 1 candidates,
Best parameters for NaiveBayes: {}
Best accuracy for NaiveBayes: 0.8022349654

Fitting 3 folds for each of 3 candidates,
Best parameters for LogisticRegression: {'
Best accuracy for LogisticRegression: 0.85

Fitting 3 folds for each of 16 candidates,
Best parameters for KNN: {'metric': 'manha
Best accuracy for KNN: 0.9481693868548743

Fitting 3 folds for each of 27 candidates,
Best parameters for RandomForest: {'max_de
Best accuracy for RandomForest: 0.97904719

Fitting 3 folds for each of 8 candidates,
Best parameters for GradientBoosting: {'le
Best accuracy for GradientBoosting: 0.9828
```

```
from sklearn.neural_network import MLPClassifi

X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

# Create an MLPClassifier instance
mlp = MLPClassifier(random_state=42, max_iter=

# Define the parameter grid for tuning
param_grid = {
    'hidden_layer_sizes': [(20,), (25,), (30,)
    'activation': ['relu'],
    'solver': ['adam'],
    'alpha': [0.005, 0.01, 0.015],
    'learning_rate': ['constant'],
    'learning_rate_init': [0.001, 0.01, 0.1]
}

# Create the GridSearchCV object
grid_search_mlp = GridSearchCV(mlp, param_grid

# Fit the model on the training data
grid_search_mlp.fit(X_train_scaled, y_train)

# Best MLP model
best_mlp = grid_search_mlp.best_estimator_

# Print the best parameters found during the s
print("Best parameters found: ", grid_search_m

# Evaluate the model on the test data
y_pred = grid_search_mlp.predict(X_test_scaled
test_accuracy = accuracy_score(y_test, y_pred)
print("Test accuracy: ", test_accuracy)
```

```
Fitting 3 folds for each of 27 candidates,
Best parameters found:  {'activation': 're
Test accuracy:  0.9318854886475815
```

```
best_models.append(best_mlp)
model_names.append('MLPClassifier')
Xs_test.append(X_test_scaled)

get_model_roc(best_models, Xs_test, model_name
```



Feature Engineering

```
X_preprocessed.shape
```

```
(10127, 36)
```

```
from sklearn.preprocessing import FunctionTran

# feature engineering functions
def custom_features(df):
    df_out = df.copy()
    # Interaction features
    df_out['Avg_Transaction_Amt'] = df_out['To
    df_out['Utilized_Credit_Limit'] = df_out['
    # Aggregated features
    df_out['Inactive_Months_Ratio'] = df_out['
    # Rate of change features
    df_out['Amt_Chng_Rate'] = df_out['Total_Am
    df_out['Ct_Chng_Rate'] = df_out['Total_Ct_
    # Composite features
    df_out['Actual_Credit_Utilization'] = df_o
    # Flag features
    df_out['Zero_Revolving_Balance'] = (df_out

    return df_out

# Apply the custom feature engineering functio
feature_engineering_transformer = FunctionTran



# Define numerical and categorical columns
categorical_columns = df.select_dtypes(include
# Update numerical column
numerical_columns = df.select_dtypes(include=[

# Remove target variable from numerical column
numerical_columns = numerical_columns.drop('At

# Combine transformers using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numeric
        ('cat', categorical_transformer, categ
    ],remainder = 'passthrough')

# Create a pipeline with the preprocessor
pipeline_fe = Pipeline(steps=[
    ('fe', feature_engineering_transformer),
    ('preprocessor', preprocessor)])

# Apply the pipeline to your dataset
X = df.drop('Attrition_Flag', axis=1)
y = df['Attrition_Flag']
X_preprocessed_fe = pipeline_fe.fit_transform(
```

```
X_preprocessed_fe.shape
```

```
(10127, 43)
```

```
# Split the data into training and testing set
X_train_fe, X_test_fe, y_train, y_test = train

# Apply SMOTE to the preprocessed training dat
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resam

# Define the models for classification
models = {
    'NaiveBayes': GaussianNB(), # Baseline Mod
    'LogisticRegression': LogisticRegression(r
    'KNN': KNeighborsClassifier(),
    'RandomForest': RandomForestClassifier(ran
    'GradientBoosting': GradientBoostingClassi
}

# Define the hyperparameter grids for each mod
param_grids = {
    'NaiveBayes': {},
    'LogisticRegression': {
        'C': [0.1, 1, 10]
    },
    'KNN': {
    'n_neighbors': [3, 5, 7, 10],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
    },
    'RandomForest': {
        'n_estimators': [100, 200, 500],
        'max_depth': [None, 10, 30],
        'min_samples_split': [2, 5, 10],
    },
    'GradientBoosting': {
        'n_estimators': [100, 200],
        'learning_rate': [0.01, 0.1],
        'max_depth': [3, 5]
    }
}

# 3-fold cross-validation
cv = KFold(n_splits=3, shuffle=True, random_st

# Train and tune the models using GridSearchCV
grids = {}
best_models = []
model_names = []
Xs_test = []

for model_name, model in models.items():
    grid_search = GridSearchCV(estimator=model
    grid_search.fit(X_train_smote, y_train_smo
```

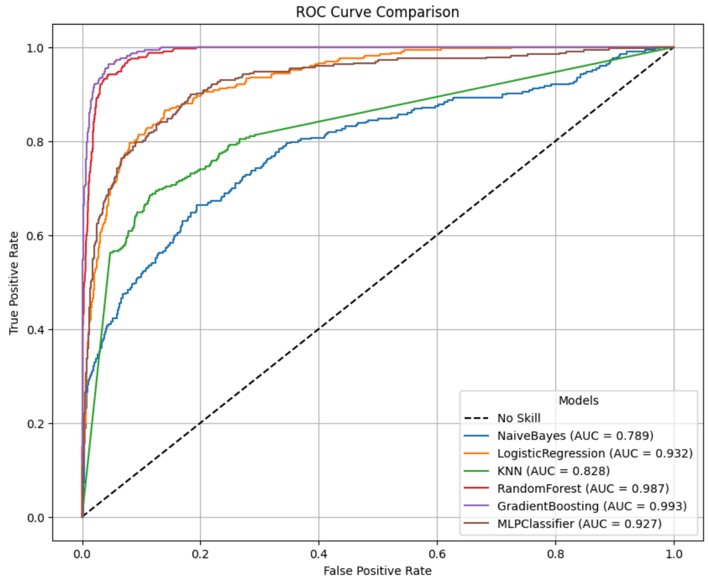
```
    best_model = grid_search.best_estimator_
    best_params = grid_search.best_params_
    best_score = grid_search.best_score_

    best_models.append(best_model)
    model_names.append(model_name)
    Xs_test.append(X_test_fe)

    print(f'Best parameters for {model_name}:
    print(f'Best accuracy for {model_name}: {b
```

```
Fitting 3 folds for each of 1 candidates,
Best parameters for NaiveBayes: {}
Best accuracy for NaiveBayes: 0.8040729304

Fitting 3 folds for each of 3 candidates,
Best parameters for LogisticRegression: {'
Best accuracy for LogisticRegression: 0.88

Fitting 3 folds for each of 16 candidates,
Best parameters for KNN: {'metric': 'manha
Best accuracy for KNN: 0.9039111895309513

Fitting 3 folds for each of 27 candidates,
Best parameters for RandomForest: {'max_de
Best accuracy for RandomForest: 0.97720923

Fitting 3 folds for each of 8 candidates,
Best parameters for GradientBoosting: {'le
Best accuracy for GradientBoosting: 0.9836
```

```
from sklearn.neural_network import MLPClassifi

X_train_scaled = X_train_fe.copy()
X_test_scaled = X_test_fe.copy()

# Create an MLPClassifier instance
mlp = MLPClassifier(random_state=42, max_iter=

# Define the parameter grid for tuning
param_grid = {
    'hidden_layer_sizes': [(20,), (25,), (30,)
    'activation': ['relu'],
    'solver': ['adam'],
    'alpha': [0.005, 0.01, 0.015],
    'learning_rate': ['constant'],
    'learning_rate_init': [0.001, 0.01, 0.1]
}

# Create the GridSearchCV object
grid_search_mlp = GridSearchCV(mlp, param_grid

# Fit the model on the training data
grid_search_mlp.fit(X_train_scaled, y_train)

# Best MLP model
best_mlp = grid_search_mlp.best_estimator_

# Print the best parameters found during the s
print("Best parameters found: ", grid_search_m

# Evaluate the model on the test data
y_pred = grid_search_mlp.predict(X_test_scaled
test_accuracy = accuracy_score(y_test, y_pred)
print("Test accuracy: ", test_accuracy)
```

```
Fitting 3 folds for each of 27 candidates,
Best parameters found:  {'activation': 're
Test accuracy:  0.9155972359328727
```

```
best_models.append(best_mlp)
model_names.append('MLPClassifier')
Xs_test.append(X_test_scaled)

get_model_roc(best_models, Xs_test, model_name
```



Finding Important Features

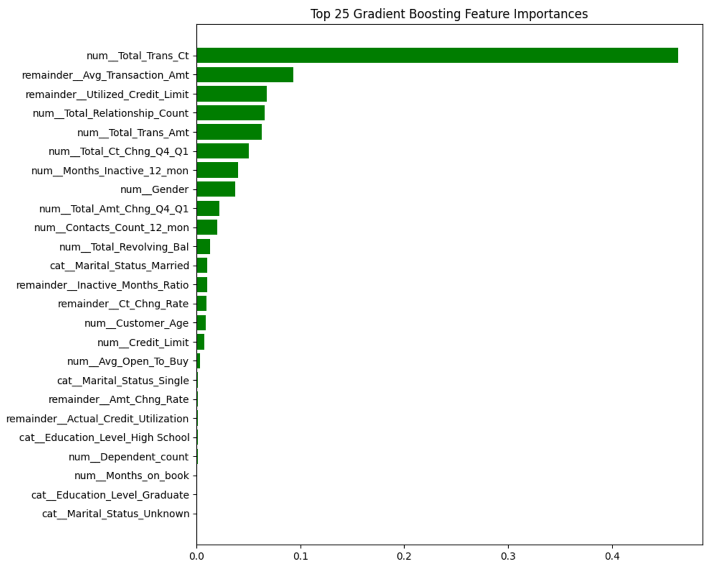
```
# Access the trained Gradient Boosting model f
gradient_boosting_model = [model for name, mod

# Get the feature importances from the trained
importances = gradient_boosting_model.feature_
```

```
# Get the feature names from the preprocessed
feature_names = [f for f in pipeline_fe.named_

# Get the indices of the sorted importances
indices = np.argsort(importances)[-25:]

# Prepare the data for plotting
sorted_features = np.array(feature_names)[indi
sorted_importances = importances[indices]

sorted_features
```

```
array(['cat__Marital_Status_Unknown',
'cat__Education_Level_Graduate',
       'num__Months_on_book',
'num__Dependent_count',
       'cat__Education_Level_High
School',

'remainder__Actual_Credit_Utilization',
'remainder__Amt_Chng_Rate',
       'cat__Marital_Status_Single',
'num__Avg_Open_To_Buy',
       'num__Credit_Limit',
'num__Customer_Age',
       'remainder__Ct_Chng_Rate',
'remainder__Inactive_Months_Ratio',
       'cat__Marital_Status_Married',
'num__Total_Revolving_Bal',
       'num__Contacts_Count_12_mon',
'num__Total_Amt_Chng_Q4_Q1',
       'num__Gender',
'num__Months_Inactive_12_mon',
       'num__Total_Ct_Chng_Q4_Q1',
'num__Total_Trans_Amt',
       'num__Total_Relationship_Count',

'remainder__Utilized_Credit_Limit',
       'remainder__Avg_Transaction_Amt',
'num__Total_Trans_Ct'],
      dtype='<U36')
```

```
# Create the plot
fig, ax = plt.subplots(figsize=(10, 8))
y_ticks = np.arange(0, len(sorted_features))
ax.barh(y_ticks, sorted_importances, color='gr
ax.set_yticklabels(sorted_features)
ax.set_yticks(y_ticks)
ax.set_title("Top 25 Gradient Boosting Feature
fig.tight_layout()
plt.show()
```

```
<ipython-input-35-095d73a73971>:5: UserWar

FixedFormatter should only be used togethe
```




```
top_indices = np.argsort(importances)[-17:]  #
top_features = np.array(feature_names)[top_ind
print("Top 17 features:", top_features)
```

```
Top 17 features: ['num__Avg_Open_To_Buy' '
 'remainder__Ct_Chng_Rate' 'remainder__Ina
 'cat__Marital_Status_Married' 'num__Total
 'num__Contacts_Count_12_mon' 'num__Total_
```

```
     'num__Months_Inactive_12_mon' 'num__Total
     'num__Total_Trans_Amt' 'num__Total_Relati
     'remainder__Utilized_Credit_Limit' 'remai
     'num__Total_Trans_Ct']
```

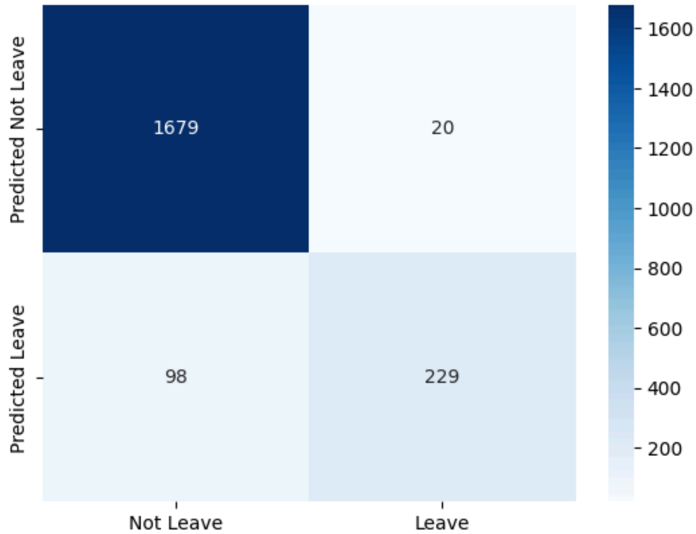
Ensemble

```
from sklearn.ensemble import StackingClassifie

mlp = MLPClassifier(
    activation='relu', alpha=0.005, hidden_lay
    learning_rate='constant', learning_rate_in
    random_state=42, max_iter=10000, n_iter_no
)

# Define base models with their best tuned par
base_models = [
    ('NaiveBayes', GaussianNB()),
    ('LogisticRegression', LogisticRegression(
    ('KNN', KNeighborsClassifier(n_neighbors=3
    ('RandomForest', RandomForestClassifier(ma
    ('GradientBoosting', GradientBoostingClass
    ('MLP', mlp)
]

# Define the meta-model
meta_model = GradientBoostingClassifier(n_esti

# Create the stacking classifier
stacking_classifier = StackingClassifier(estim

# Train the stacking classifier on the SMOTE-a
stacking_classifier.fit(X_train_smote, y_train

# Predict on the test data and evaluate accura
y_pred = stacking_classifier.predict(X_test_fe
test_accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy for Stacking Classifier:
```

```
    Test Accuracy for Stacking Classifier: 0.9
```

```
from sklearn.metrics import confusion_matrix,
from tabulate import tabulate

# Predict on the test data using the stacking
y_pred = stacking_classifier.predict(X_test_fe
cm = confusion_matrix(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)
clf_report = classification_report(y_test, y_p

# Print formatted outputs
print(f"{'='*30}\nModel: Stacking Classifier\n
print("Confusion Matrix:")
sns.heatmap(cm, annot=True, fmt="d", cmap='Blu
plt.show()

print(f"Accuracy: {acc:.2f}")
print("Classification Report:")
print(tabulate(pd.DataFrame(clf_report).T, hea
```

```
==============================
Model: Stacking Classifier
==============================
Confusion Matrix:
```




```
Accuracy: 0.94
Classification Report:
+--------------+-------------+----------+-
|              |   precision |   recall |
|--------------+-------------+----------+-
| 0            |    0.944851 | 0.988228 |
| 1            |    0.919679 | 0.700306 |
| accuracy     |    0.941757 | 0.941757 |
| macro avg    |    0.932265 | 0.844267 |
| weighted avg |    0.940788 | 0.941757 |
+--------------+-------------+----------+-
```

Model Evaluation

```
from sklearn.metrics import accuracy_score, pr


# Initialize a list to store the performance o
model_performance = []

# Loop through each model to get predictions a
for model, X_test_fe, model_name in zip(best_m
    # Predict the responses for the test datas
    y_pred = model.predict(X_test_fe)

    # Predict the probabilities on the test se
    y_probs = model.predict_proba(X_test_fe)[:

    # Calculate AUC
    auc_score = roc_auc_score(y_test, y_probs)

    # Compute confusion matrix and other perfo
    cm = confusion_matrix(y_test, y_pred)
    acc = accuracy_score(y_test, y_pred)

    # Get precision, recall, fscore, support
    precision, recall, fscore, support = preci

    # Add model metrics to the dictionary
    performance_dict = {
        'Model': model_name,
        'Accuracy': acc,
        'Precision': precision[1],  # Index 1
        'Recall': recall[1],        # Index 1
        'F1-Score': fscore[1],      # Index 1
        'AUC': auc_score
    }

    # Append the performance metrics of the cu
    model_performance.append(performance_dict)

# Convert model_performance into a DataFrame
performance_df = pd.DataFrame(model_performanc

# Display the performance of all models
print(performance_df)
```

```
                 Model  Accuracy  Precision
0           NaiveBayes  0.769003   0.377391
1   LogisticRegression  0.865252   0.554656
2                  KNN  0.836130   0.494600
3         RandomForest  0.958045   0.892857
4     GradientBoosting  0.967917   0.904321
5        MLPClassifier  0.915597   0.825000
```

```
# Find the best model based on the highest F1-
best_model_row = performance_df['F1-Score'].id
best_model = performance_df.iloc[best_model_ro

print("Best Model Based on F1-Score:")
print(best_model)

# Retrieve the confusion matrix for the best m
best_model_name = best_model['Model']

# Plot the confusion matrix for the best model
```