```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 27 21:59:47 2024

@author: ananyaashahi
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, classification_report




#loading the dataset
df=pd.read_csv("Application_Data 2.csv")

#%%

#printing the information of the dataset

print(df.info())

#   Column                Non-Null Count  Dtype
# ---  ------                --------------  -----
# 0   Applicant_ID          25128 non-null  int64
# 1   Applicant_Gender      25128 non-null  object
# 2   Owned_Car             25128 non-null  int64
# 3   Owned_Realty          25128 non-null  int64
# 4   Total_Children        25128 non-null  int64
# 5   Total_Income          25128 non-null  int64
# 6   Income_Type           25128 non-null  object
# 7   Education_Type        25128 non-null  object
# 8   Family_Status         25128 non-null  object
# 9   Housing_Type          25128 non-null  object
# 10  Owned_Mobile_Phone    25128 non-null  int64
# 11  Owned_Work_Phone      25128 non-null  int64
# 12  Owned_Phone           25128 non-null  int64
# 13  Owned_Email           25128 non-null  int64
# 14  Job_Title             25128 non-null  object
# 15  Total_Family_Members  25128 non-null  int64
# 16  Applicant_Age         25128 non-null  int64
# 17  Years_of_Working      25128 non-null  int64
# 18  Total_Bad_Debt        25128 non-null  int64
# 19  Total_Good_Debt       25128 non-null  int64
# 20  Status                25128 non-null  int64
# dtypes: int64(15), object(6)

# Categorical features are - ['Applicant_Gender', 'Income_Type', 'Education_Type',
#                               'Family_Status', 'Housing_Type', 'Job_Title']
#%%
print(df.shape)
```

```
#Our datatset has 25128 rows and 21 columns


#%%

#counting missing values per row

row_nan_count = df.isna().sum(axis=1)
print(row_nan_count)


#  NO missing values
#%%

#checking for duplicate values
print("Number of duplicate rows", df.duplicated().sum())

#   NO duplicate values
#%%

print(df.head())


#%%
#Checking the uniques values in the Job_Title
unique_job_titles = df['Job_Title'].unique()

#Counting occurrences of each unique value in the 'Job_Title' column
job_titles_counts = df['Job_Title'].value_counts()

# Display unique values
print("Unique Job Titles:")
print(unique_job_titles)


# Display counts of each employment type
print("\nJob Title Counts:")
print(job_titles_counts)



#Job Title Counts:
#Laborers                                                  6211
#Core staff                                                3591
#Sales staff                                               3485
#Managers                                                  3012
#Drivers                                                   2135
#High skill tech staff                                     1383
#Accountants                                               1241
#Medicine staff                                            1207
#Cooking staff                                              655
#Security staff                                             592
#Cleaning staff                                             549
#Private service staff                                      344
#Low-skill Laborers                                         175
#Waiters/barmen staff                                       173
#Secretaries                                                151
#HR staff                                                    85
#Realty agents                                               79
```

```
#IT staff                                                         60

#%%

# Removing the trailing space in the column Job_Title
# Noticed we had trailing space in that column, which was preventing the conversion
of grouping of Job_Titles

df['Job_Title'] = df['Job_Title'].str.strip()


#%%
job_title_mapping = {
    'Professional Roles': ['Managers', 'High skill tech staff', 'Accountants',
'Medicine staff'],
    'Service and Support Roles': ['Sales staff', 'Cooking staff', 'Secretaries',
'HR staff',
                                  'Realty agents', 'Private service staff',
'Waiters/barmen staff'],
    'Skilled Labor Roles': ['Drivers', 'Security staff'],
    'Unskilled Labor Roles': ['Laborers', 'Core staff', 'Cleaning staff', 'Low-
skill Laborers']
}


# Function to map job titles to broader categories
def map_job_title_to_category(job_title):
    for category, titles in job_title_mapping.items():
        if job_title in titles:
            return category
    return 'Other'  # Default category for job titles not in the mapping

# Applying the mapping function to create a new column 'Job_Category'
df['Job_Category'] = df['Job_Title'].apply(map_job_title_to_category)

print(df.head())

# Validating if the Job_Category column contains the expected categories and their
respective counts.
job_category_counts = df['Job_Category'].value_counts()
print(job_category_counts)


#Dropping the Job-Title feature
df.drop('Job_Title', axis=1, inplace=True)

print(df.head())


#   Applicant_ID Applicant_Gender  ...  Status               Job_Category
#0       5008806           M        ...       1        Skilled Labor Roles
#1       5008808           F        ...       1  Service and Support Roles
#2       5008809           F        ...       1  Service and Support Roles
#3       5008810           F        ...       1  Service and Support Roles
#4       5008811           F        ...       1  Service and Support Roles

#[5 rows x 21 columns]

#%%
```

```
# Checking the unique values of
'Owned_Mobile_Phone',Owned_Mobile_Phone,'Owned_Phone','Owned_Email'

# Check unique values and counts for 'Owned_Mobile_Phone'
print("Unique values and counts for 'Owned_Mobile_Phone':")
print(df['Owned_Mobile_Phone'].value_counts())
print("\n")

# Check unique values and counts for 'Owned_Work_Phone'
print("Unique values and counts for 'Owned_Work_Phone':")
print(df['Owned_Work_Phone'].value_counts())
print("\n")

# Check unique values and counts for 'Owned_Phone'
print("Unique values and counts for 'Owned_Phone':")
print(df['Owned_Phone'].value_counts())
print("\n")

# Check unique values and counts for 'Owned_Email'
print("Unique values and counts for 'Owned_Email':")
print(df['Owned_Email'].value_counts())



#Unique values and counts for 'Owned_Mobile_Phone':
#1    25128
#Name: Owned_Mobile_Phone, dtype: int64


#Unique values and counts for 'Owned_Work_Phone':
#0    18249
#1     6879
#Name: Owned_Work_Phone, dtype: int64


#Unique values and counts for 'Owned_Phone':
#0    17772
#1     7356
#Name: Owned_Phone, dtype: int64


#Unique values and counts for 'Owned_Email':
#0    22598
#1     2530
#Name: Owned_Email, dtype: int64


#It seems like for the attribute 'Owned_Mobile_Phone', all the values are '1'.
#The column is constant, there are no varied values, so, we will drop it.

df.drop('Owned_Mobile_Phone', axis=1, inplace=True)

#%%
# Also, the 'Applicant_ID' should be having separate values for all rows. So, it
wouldn't be adding much
#values to the analysis, so we will drop it.

df.drop('Applicant_ID', axis=1, inplace=True)
```

```
print(df.head())

print(df.info())

#%%

## Creating a boxplot for all numerical attributes to check for outliers

# Define numerical columns to plot
numerical_columns = ['Total_Income', 'Total_Family_Members', 'Applicant_Age',
                     'Years_of_Working', 'Total_Bad_Debt', 'Total_Good_Debt']

# Create separate box plots for each numerical column
sns.set(style="whitegrid")
for col in numerical_columns:
    plt.figure(figsize=(8, 5))
    sns.boxplot(x=df[col], color='skyblue')
    plt.title(f'Box Plot for {col}')
    plt.xlabel('')
    plt.show()


# Most of pur plots still have decent amount of values over the 75th percentile.
# Using 1th & 99th percentile as our benchmark for removing our outliers

#%%
# Removing outliers

# Defining numerical columns to filter
numerical_columns = ['Total_Income', 'Total_Family_Members', 'Years_of_Working',
'Total_Bad_Debt']

# Calculating 1th and 99th percentiles for each column
percentiles = df[numerical_columns].quantile([0.01, 0.99])

# Filtering DataFrame to remove outliers
filtered_df = df.copy()  # Create a copy of the DataFrame
for col in numerical_columns:
    lower_bound = percentiles.loc[0.01, col]
    upper_bound = percentiles.loc[0.99, col]
    filtered_df = filtered_df[(filtered_df[col] >= lower_bound) & (filtered_df[col]
<= upper_bound)]

# Display
print("Summary Statistics Before Outlier Removal:")
print(df[numerical_columns].describe())
print("\nSummary Statistics After Outlier Removal:")
print(filtered_df[numerical_columns].describe())




#Summary Statistics After Outlier Removal:
#        Total_Income  Total_Family_Members  Years_of_Working  Total_Bad_Debt
#count   24130.000000          24130.000000      24130.000000    24130.000000
#mean   190360.741442              2.284252          7.421508        0.213717
#std     85019.559173              0.904108          5.856716        0.735050
#min     63000.000000              1.000000          1.000000        0.000000
```

```
#25%    135000.000000              2.000000          3.000000        0.000000
#50%    180000.000000              2.000000          6.000000        0.000000
#75%    225000.000000              3.000000         10.000000        0.000000
#max    585000.000000              5.000000         30.000000        6.000000

#%%

print(filtered_df.info())
print(filtered_df.shape)


#%%

#Converting the categorical features into numeric

categorical_features = ['Applicant_Gender',
'Owned_Car','Owned_Realty','Income_Type', 'Education_Type',

'Family_Status','Owned_Work_Phone','Owned_Phone','Owned_Email', 'Housing_Type',
'Job_Category']

# converting object-type categorical features to dummy variables
filtered_df = pd.get_dummies(filtered_df, columns=categorical_features,
drop_first=True)

print(filtered_df.head())
print(filtered_df.info())

#%%

#checking for under/over sampling

status_distribution = filtered_df['Status'].value_counts()
print("Class Distribution:")
print(status_distribution)

# plotting distribution of status
status_distribution.plot(kind='bar', rot=0)
plt.title("Class Distribution of Status")
plt.xlabel("Status")
plt.ylabel("Count")
plt.show()

# calculating percentages of status distribution (%)
class_proportions = status_distribution / len(filtered_df) * 100
print("\nClass Proportions (%):")
print(class_proportions)

# observation - the data of people whose credit got approved is clearly more,
# meanwhile the folks who didn't get their credit approved have quite less
representation in the dataset.

#Class Proportions (%):
#1    99.772068
#0     0.227932

#%%

### have to use a way to prevent oversampling while splitting/training/testing
```

```
### Might have to check the collinearity of attributes


#%%

# Separating features (X) and target variable (y)
X = filtered_df.drop('Status', axis=1)  # Features
y = filtered_df['Status']               # Target variable



#%%



from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, roc_curve, roc_auc_score, confusion_matrix
import matplotlib.pyplot as plt

# Splitting the data into training and test sets (stratified sampling for
imbalance)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
stratify=y, random_state=42)

# Appling SMOTE to the training data only
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)


#%%

############# MODEL - I    ######### Logistic Regression

# Printing value counts of target variables after oversampling
print("After oversampling:")
print("Train set - Class 1:", sum(y_train_resampled == 1), "Class 0:",
sum(y_train_resampled == 0))


# Defining the parameter grid for logistic regression
log_grid = {
    'clf__C': np.logspace(-2, 5, 21),  # Regularization parameter
    'clf__penalty': ['l1', 'l2'],       # Type of penalty compatible with LBFGS
solver
    'clf__solver': ['liblinear'],  # Solver algorithm
    'clf__class_weight': ['balanced', None],  # Class weight for imbalance
    'clf__max_iter': [100, 200, 500]  # Maximum number of iterations
}

# Defining the pipeline with standard scaling and logistic regression
estimator = ImbPipeline([
    ('scale', StandardScaler()),
    ('clf', LogisticRegression(random_state=10))
])
```

```
# Creating a GridSearchCV object with the pipeline
logistic = GridSearchCV(estimator=estimator,
                        param_grid=log_grid,
                        cv=5,
                        scoring='f1',
                        n_jobs=-1)

# Fitting the GridSearchCV object on the resampled training data
logistic.fit(X_train_resampled, y_train_resampled)

# Printing the best hyperparameters found
print("Best Hyperparameters:", logistic.best_params_)

# Performing evaluation on test set
y_pred = logistic.predict(X_test)
f1 = f1_score(y_test, y_pred)
print("F1-score on test set:", f1)

# Calculating other metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score

precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print("Precision on test set:", precision)
print("Recall on test set:", recall)
print("Accuracy on test set:", accuracy)

# Plotting ROC curve
fpr, tpr, _ = roc_curve(y_test, logistic.predict_proba(X_test)[:, 1])
roc_auc = roc_auc_score(y_test, logistic.predict_proba(X_test)[:, 1])

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()


# confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# finding feature importance
coef = logistic.best_estimator_.named_steps['clf'].coef_[0]
feature_names = X.columns
feature_importance = pd.DataFrame({'Feature': feature_names, 'Importance': coef})
top_10_features = feature_importance.sort_values(by='Importance',
ascending=False).head(10)
print("Top 10 Important Features:")
print(top_10_features)
```

```
import seaborn as sns

# plotting feature importance
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=top_10_features)
plt.title('Top 10 Important Features')
plt.show()



# ==============================================================================
# After oversampling:
# Train set - Class 1: 19260 Class 0: 19260
# Best Hyperparameters: {'clf__C': 44668.35921509626, 'clf__class_weight':
'balanced', 'clf__max_iter': 100, 'clf__penalty': 'l2', 'clf__solver': 'liblinear'}
# F1-score on test set: 1.0
# Precision on test set: 1.0
# Recall on test set: 1.0
# Accuracy on test set: 1.0
# Confusion Matrix:
# [[  11    0]
#  [   0 4815]]
# Top 10 Important Features:
#                                              Feature  Importance
# 6                                    Total_Good_Debt  113.980978
# 17  Education_Type_Secondary / secondary special  ...   12.361044
# 14  Education_Type_Higher education               ...   10.073297
# 33               Job_Category_Unskilled Labor Roles    7.066006
# 30                  Job_Category_Professional Roles    5.504682
# 15  Education_Type_Incomplete higher              ...    5.132137
# 31           Job_Category_Service and Support Roles    5.082376
# 32                 Job_Category_Skilled Labor Roles    4.380168
# 2                               Total_Family_Members    3.470471
# 21  Family_Status_Widow                           ...    2.522999
#
# ==============================================================================


#%%



#%%

####     MODEL - II   ##### RANDOM FOREST


from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score, roc_curve, roc_auc_score, confusion_matrix,
classification_report, precision_recall_curve

#Defining the parameter grid for random forest
rf_grid = {'n_estimators': np.linspace(100, 1000, 10, dtype = int),
           'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'max_features': ['sqrt', 'log2', None]}
```

```
RF = GridSearchCV(RandomForestClassifier(min_samples_leaf = 10, random_state = 10,
max_features = 'sqrt'),
                    param_grid = rf_grid, cv = 5, n_jobs = -1, scoring = 'f1')
RF.fit(X_train_resampled, y_train_resampled)

#%%

# Printing the best hyperparameters found
print("Best Hyperparameters:", RF.best_params_)



# Getting the best estimator from GridSearchCV
best_rf_model = RF.best_estimator_

# Extracting feature importances from the best model
importances = best_rf_model.feature_importances_

# Getting the names of the features/columns
feature_names = X_train.columns

# Create a DataFrame to store feature names and their importances
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance':
importances})

# Sorting the DataFrame by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance',
ascending=False)

# Printing the top 10 important attributes
print("Top 10 Important Attributes:")
print(feature_importance_df.head(10))

# Plotting the top 10 important features
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df.head(10))
plt.title('Top 10 Important Features')
plt.show()


# ===============================================================================
# #Best Hyperparameters: {'max_depth': 20, 'max_features': 'sqrt',
'min_samples_split': 2, 'n_estimators': 200}
# #Top 10 Important Attributes:
#                                                 Feature  Importance
# 6                                        Total_Good_Debt    0.355194
# 5                                         Total_Bad_Debt    0.341899
# 9                                         Owned_Realty_1    0.029581
# 13  Income_Type_Working                                ...    0.027199
# 8                                            Owned_Car_1    0.026915
# 31            Job_Category_Service and Support Roles    0.024955
# 33                Job_Category_Unskilled Labor Roles    0.021827
# 23                                        Owned_Phone_1    0.020966
# 30                   Job_Category_Professional Roles    0.020873
# 22                                   Owned_Work_Phone_1    0.014933
# ===============================================================================
```

```
#%%
# Performing evaluation on holdout set
from sklearn.metrics import f1_score
print(f1_score(y_test, RF.predict(X_test)))


#%%
# ROC Curves
from sklearn.metrics import roc_curve, roc_auc_score
fpr_rf, tpr_rf, _ = roc_curve(y_train, RF.predict_proba(X_train)[:,1])


#%%
plt.figure(figsize=(10, 6))
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

# Generating Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, RF.predict_proba(X_test)[:,
1])
plt.figure(figsize=(10, 6))
plt.plot(recall, precision, marker='.', label='Random Forest')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.show()

# Printing classification report
print(classification_report(y_test, y_pred))

# Printing AUC values
print("AUC Score on test set:", roc_auc)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=[1, 0])
print("Confusion Matrix:")
print(cm)
# ===============================================================================
#
#                precision    recall  f1-score   support
#
#            0       1.00      1.00      1.00        11
#            1       1.00      1.00      1.00      4815
#
#     accuracy                           1.00      4826
#    macro avg       1.00      1.00      1.00      4826
# weighted avg       1.00      1.00      1.00      4826
#
# AUC Score on test set: 1.0
# Confusion Matrix:
# [[4815    0]
```

```
#  [    0    11]]
#
# ============================================================================
```