Enhancing Credit Risk Management: Machine Learning Approaches for Credit Card Default Prediction

Vinay Narendra Gurrap
Faculty of Engineering, Environment and Computing,
Coventry University MSc Data Science ()
Coventry, United Kingdom
gurrapv@uni.coventry.ac.uk

Abstract—Accurate credit card default prediction is critical for financial organisations to manage risks and make informed decisions. This study looks into the usefulness of several machine learning algorithms in forecasting credit card default. The research focuses on well-known methods such as logistic regression, decision trees, random forests, and gradient boosting, taking into account their ability to handle imbalanced datasets, identify complicated patterns, and produce interpretable predictions. Accuracy, precision, recall, and area under the receiver operating characteristic curve (AUC-ROC) are among the evaluation measures used to ensure a thorough assessment of model performance. Several data preparation approaches were used, including class imbalance concerns, categorical data encoding, and feature extraction. Prediction findings and algorithm performance measures were acquired and visualized for comparison and analysis using the Python programming language and various machine learning libraries and frameworks.

Keywords— credit risk, default payment, machine learning, python, class imbalance, data mining, lightgbm, random forest, xgboost, extremely randomized trees, logistic regression, decision trees, credit card fraud.

GitHub Link: ***https://github.com/DragonVG/Credit-Card-Default.git***

SOCIAL, ETHICAL, LEGAL AND PROFESSIONAL CONSIDERATIONS RELATED TO DATA

All data utilised in this study were anonymized and processed in accordance with the institution's Privacy and Personal Data Processing Policy and the guidelines of the General Data Protection Regulation (GDPR) to ensure the privacy and protection of sensitive student information [20]. The dataset is also intended to follow the FAIR (Findability, Accessibility, Interoperability, and Reusability) guidelines for managing scientific data, encouraging openness, usability, and reproducibility in research [21].

I. INTRODUCTION

According to report mentioned on the Nilson website, $28.6 billion losses occurred due to fraud transactions worldwide in 2020 [1]. This number is increasing day by day as fraudsters are using new ways of conducting fraudulent activities. Credit card default is a serious concern for financial institutions around the world, affecting both users and lenders. The increasing accessibility and utilization of credit cards have increased the requirement for precise credit risk assessment and prediction in order to mitigate potential losses and ensure responsible lending practices [2].
A new algorithm known as Genetic Programming was introduced to predict customer default from Taiwan Dataset and said to have results of 86% of precision, recall and accuracy by constructing a model using IF-THEN rules [3]. Reference [4] applied Synthetic Minority Oversampling Technique to tackle the class imbalance and to improve the performance of the model along with decision tree and artificial neural networks and found out that decision tree performed relatively better as compared to artificial neural networks for predicting the default. Reference [5] used One-Way Analysis hypothesis-testing technique on the models to analyse if it improves the performance of the model and found out it significantly improved the accuracy of the models and moreover, the models performed better on balanced dataset as compared to imbalanced dataset. Reference [6] discovered that by applying XGBoost to bond default risk prediction, it outperformed the standard traditional algorithms when dealing with imbalanced dataset.

This paper discusses and compares the results of different machine learning algorithms to predict the probability of customer defaulting the payments. In Section II, problem statement and the objectives of this paper is established. In Section III, the data described used in this study. In Section IV, the experimental setup is discussed in detail which includes tackling of class imbalance issue and feature analysis. Section V includes the description of the machine learning employed. Section VI presents the results and analysis showcasing the performance of the models and comparing the results which is concluded with Section VII by summarizing the paper.

II. PROBLEM STATEMENT & OBJECTIVES

Financial loss has dramatically increased due to fraud, which is causing a significant increase in credit card default and rather than dividing customers into defaulters and non-defaulters categories, it will be more important to estimate the likelihood of default payment.

In order to tackle these problems, my objectives is to:

1. To predict customers who will default.
2. To discover the best algorithms for this type of analysis.

III. DATASET DESCRIPTION

The Dataset was obtained from the UCI Machine Learning repository and is based on credit card customer data from April to September 2005 in Taiwan[7]. It has 30000 instances and 23 attributes (X1 – X23) and 1 response variable (Default) with no missing values. The attributes include demographic information (such as age, gender, and marital status), credit limit, payment history (previous repayment status from April to September), amount of bill statement

(from April to September), and the amount of previous payment (from April to September). The dataset consists of 10 categorical features from X2-X11 which are converted into numerical values, 13 numerical variables from X12-X23 along with X1 and the last feature X24 which represents the class, if the customer has defaulted it is assigned as 1 otherwise it is assigned as 0. Fig. 1 displays the dataset.

Attribute ID	Attribute Name	Description
X1	LIMIT_BAL	Amount of credit given (NT Dollar)
X2	SEX	(1 = Male, 2 = Female)
X3	Education	(1 = Graduate school, 2 = University, 3 = High school, 4 = Others)
X4	Marriage	Marital status (1 = Married, 2 = Single, 3 = Others)
X5	Age	Age(Year)
X6-X11	Pay_1 to Pay_6	History of past payment (From April to September, 2005): X6 = The repayment status in September, 2005... X11 = The repayment status in April, 2005. The measurement scale for the repayment status is: -1 = Pay on time, 1 = Payment delay for one month, 2 = Payment delay for two months,. . ., 8 = Payment delay for eight months, 9 = Payment delay for nine months and above.
X12-X17	Bill_Amt1 to Bill_Amt6	Amount of bill statements (NT dollar). X12 = Amount of bill statement in September, 2005, . . ., X17 = Amount of bill statement in April, 2005.
X18-X23	Pay_Amt1 to Pay_Amt6	Amount of previous payment (NT dollar). X18 = Amount paid in September, 2005, . . ., X23 = Amount paid in April, 2005.
X24	default_payment_next_month	Default = 1 and Not-Default = 0

Fig 1. Credit Card Default Dataset

IV. EXPERIMENTAL SETUP

The dataset has been initially checked and studied for features description by looking at minimum, maximum, standard deviation and percentiles for better analysis. As part of data cleaning process, the deletion operation is carried out because the variable ID has no association with the response variable and the response variable is renamed for better understanding. The dataset has no missing values and class imbalance analysis was applied using Python on the response variable which shows that 23,364 (77.9%) in the negative category (no default) and 6,636 (22.1%) are in the positive category (default). The results depicted in Fig. 2 and Fig. 3 reveal a class imbalance on the dataset.

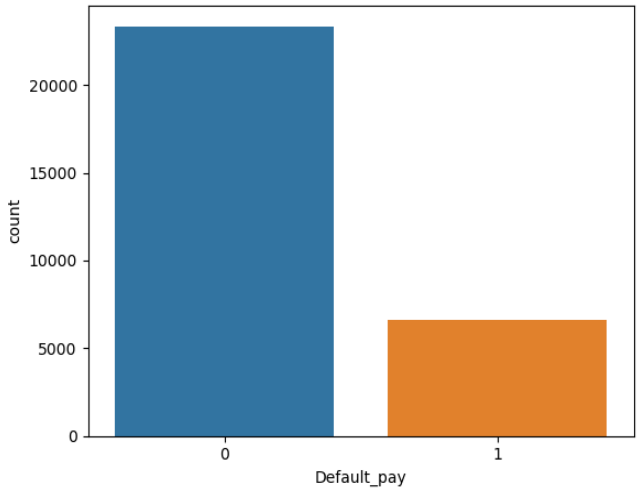

Fig 2. Class Imbalance

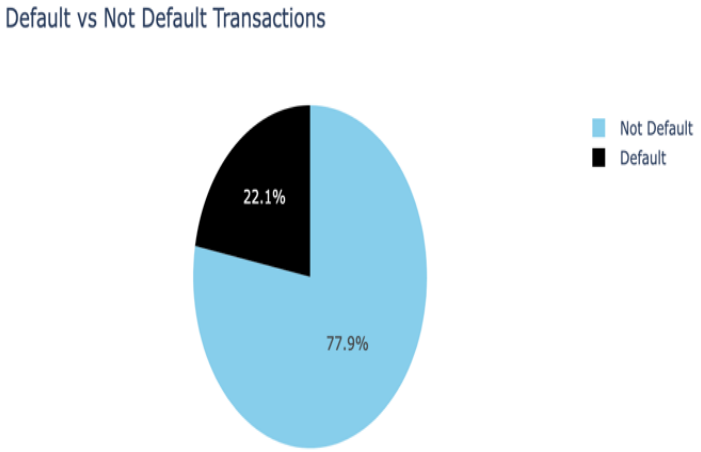

Fig 3. Class Imbalance

Imbalance class techniques such as Random Under-Sampling, Random Over-Sampling and SMOTE Over-sampling was carried out to tackle the class imbalance so as to balance the dataset otherwise it can lead to the underperformance of machine learning methods and the issue must be addressed before implementing algorithms [8].

- *Random Under-sampling*

Random under-sampling is accomplished by picking a random number of samples from the majority class to equal the number of samples from the minority class. In this paper, 4676 instances with default payments were selected and merged with 4676 instances with non-default payments to form a new dataset of 9352 instances.

- *Random Over-sampling*

Random over-sampling is done by randomly replicating instances of the minority class to equal the number of instances of the majority class. In this paper, 16324 instances from the minority class were chosen at random with replacement and combined with 16324 instances from the majority class to form a new dataset having 32648 instances.

- *SMOTE Over-sampling*

The SMOTE over-sampling technique generates synthetic instances by taking a default payment instance and finding the nearest neighbour who is also a default payment instance, then building an instance between those two records.

In this research, SMOTE was built using the technique from the Python imbalanced learning module. To match the number of default payments with non-default payments, synthetic instances of default payments were constructed, and a new dataset with 20844 instances was created.

The feature engineering step was applied on ‘Education’ and ‘Marriage’ variable as both the variables has unknown values within them for which there is no description provided and hence they were grouped with the ‘Others’ category in the respective variable columns. The output depicted in TABLE. 1 and TABLE. 2 shows the new ‘Others’ category numbers.

Education
1: 10585
2: 14030
3: 4917
4: 123
5: 280
6: 51
0: 14

→

Education
1: 10585
2: 14030
3: 4917
4: 468

TABLE 1. Education Variable Feature Engineering

Marriage
1: 13659
2: 15964
3: 323
0: 54

→

Marriage
1: 13659
2: 15964
3: 377

TABLE 2. Marriage Variable Feature Engineering

Since most of the machine learning methods work with numbers, categorical data needs be converted into numeric values and then further be converted into binary numbers. In this dataset, categorical variables were represented in the form of numbers as shown in the data description and 'pd.get_dummies' function from the pandas library was applied, which performs one-hot encoding, on the 9 categorical features by creating dummy variables for each unique value in those columns and 'drop_first = True' parameter was used to drop the dummy variables in order to avoid multicollinearity issues [9]. The results are shown in TABLE. 3 and TABLE. 4.

Numerical		Binary		
Education	→	Education_2	Education_3	Education_4
2		1	0	0
3		0	1	0
4		0	0	1

TABLE 3. Encoding categorical variable to binary

Educatio n_2	Educatio n_3	Educatio n_4		Educatio n_2	Educatio n_3
1	0	0	→	1	0
0	1	0		0	1
0	0	1		0	0

TABLE 4. Removal of Dummy Variables

The z-score normalization which is a part of data standardization technique was used on the numerical attributes as part of feature scaling process to make sure that all the attributes values fall under the same scale. It is performed by subtracting the mean of the data and dividing by the standard deviation [10].

$$x' = \frac{x - \mu_X}{\sigma_X}$$

The correlation matrix in the form of heatmap was executed to analyse the relationship amongst the features and with the response variable. The attributes were dropped from the dataset which showed high correlation (>0.9) amongst each other and had lesser correlation with the response variable [11] [12]. Fig. 4 and Fig. 5 shows the correlation with respect to the response variable and the final heatmap after removing high correlated attributes.

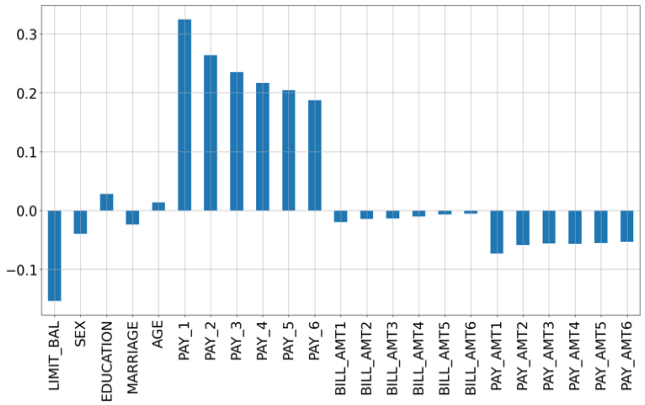

Fig 4. Correlation with Default Variable

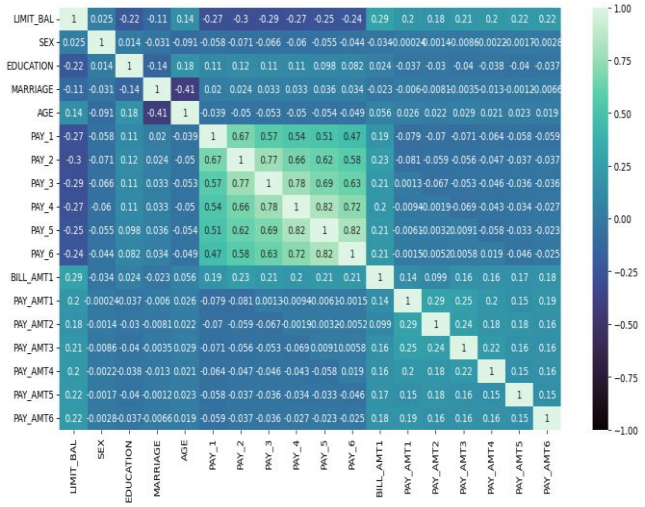

Fig.5 Features Heatmap Correlation

The Python programming language along with its libraries were used to perform the algorithms in this project. The code is executed using Jupyter Notebook which runs Python 3.6 and is available by downloading Anaconda software. The project requires a number of libraries to be imported initially that included predefined functions for carrying out the relevant operations. As the dataset contains binary values in its dependent variable, classification algorithms are performed using scikit-learn package which provides the necessary functionalities. The pandas package is used to load the dataset in the excel format and print function was executed to verify if the dataset has been loaded properly.

After the data pre-processing, data cleaning, feature analysis, feature scaling, feature encoding and data standardization, the data is then split into training set (70%) and testing set (30%) by importing train_test_split function from the scikit-learn library which will allow the program to execute classification algorithms. The class imbalance methods like Random Under-sampling, Random Over-sampling and SMOTE Over-

sampling was executed using training set to balance the dataset. The GridSearchCV function was carried out on the individual training balanced set for hyperparameter tuning to determine the optimal values for each model.

V. MACHINE LEARNING CLASSIFICATION METHODS

In this experiment, a total of 18 models were developed which is a combination of six classification methods along with three class imbalance techniques to carry out the supervised learning process. Models were trained with parameters that produced the best outcomes, which were acquired through GridSearchCV.

Metrics such as Confusion matrices, ROC curves, Accuracy, F1 Score and other model metrics were derived for all models.

A. Decision Tree

A machine learning approach called decision tree builds a flowchart-like structure comprising decision nodes and leaf nodes. To divide the data into subsets that are as pure as feasible in terms of the target variable, the algorithm chooses the best feature at each decision node such as Gini Impurity, Information Gain or Entropy and the process is continued until parameter such as maximum depth or minimum number of instances is used to terminate the process. Each decision node is a feature or an attribute The final predictions or results are located in the leaf nodes, sometimes referred to as terminal nodes. Decision Trees have the flexibility to handle both numerical and categorical data and ease of interpretability but it is prone to suffer from overfitting if the data is overly complicated [13].

B. Random Forest

Random Forest builds a number of trees and each tree is trained on random subset of training data and a random subset of features which is known as bagging. It then combines the results of each tree and provides the output through majority voting as final prediction. Due to this, the algorithm is considered to be one of the most efficient algorithm for higher accuracy. Ease of handling missing data and outliers, estimation of importance of features and less prone to overfitting are some of the advantages of Random forest [14].

C. Logistic Regression

Logistic regression is a classification problem-solving statistical machine learning technique. It performs a binary classification by modelling the likelihood of an event occurring based on the input features. Classification is done by assessing the likelihood of an observation belonging to a given class. If the response variable has 0 and 1 or heads and tails then the problem is consider as Binomial Logistic Regression but if there are more than two possible values then it is a Multinomial Logistic Regression [15].

D. XGBoost

XGBoost, an abbreviation for "Extreme Gradient Boosting", is a sophisticated machine learning technique that works with decision trees and gradient boosting collectively to generate accurate models. It works by adding new trees to rectify previous trees' faults, hence boosting the model's performance. XGBoost is well-known for its capacity to handle different types of data while preventing overfitting via regularisation techniques. It provides flexibility by allowing you to tweak multiple hyperparameters for individual purposes [16].

E. LightGBM

It is a gradient boosting framework that employs tree-based learning algorithms, which are thought to be extremely strong in terms of computation. It is thought to be a fast processing algorithm. While other algorithms' trees develop horizontally, the LightGBM algorithm grows vertically, which means it grows leaf-wise while other algorithms grow level-wise. LightGBM can be trained in parallel and on GPUs, making it considerably quicker and it can handle overfitting due to its sensitivity [17].

F. Extremely Randomized Trees

ExtraTrees, which stands for "Extremely Randomised Trees," is an ensemble learning method that creates many decision trees and then combines their decisions to generate correct predictions. It splits tree nodes randomly by using the entire training data set [18].

VI. EXPERIMENTAL RESULTS

The performance of all the models after using the best optimal values for GridSearchCV is compared in Table. 5. In the training and testing phases of each classification model, the 5-fold cross validation technique is applied. The validation method divides the instances into 5 partition. Every partition has a set of randomly chosen instances. A single partition is utilised each time as the training data for the model, and the other 4 partitions are used as the testing data. This cross-validation process is performed five times. The results were evaluated using the accuracy metric to compare the performance of the model.

It can be clearly seen that Random Forest Oversampling is the most accurate classification model and is predicted to be 81% accurate when categorizing new data and achieving a 94% Average 5-Fold Cross Validation Score. It is followed closely by Decision tree Over-sampling and SMOTE and XGBoost SMOTE with 80% accurate of predicting new data. The model that performs the worst is Decision Tree Under-sample which gives 69% accuracy for prediction. Fig. 6. and Fig. 7. below shows that Confusion Matrix of Random Forest Over-sample and Decision Tree Under sample which classifies the test data most and which misclassifies the data most respectively.

It is also noticeable that the over-sampling and SMOTE over-sampling models perform better and have similar results as compared to under-sampling models which produced the worst results. Given that the training set is considerably less when compared to Random oversampling and SMOTE over-sampling, under-sampling was predicted to yield the worst results.

	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	Decision Tree - Undersampling	0.694556	0.380491	0.640816	0.477476	0.675167
1	Decision Tree - Overersampling	0.803444	0.564483	0.426531	0.485905	0.667456
2	Decision Tree - SMOTE	0.803444	0.564483	0.426531	0.485905	0.667456
3	Random Forest - Undersampling	0.735444	0.427389	0.632143	0.509981	0.698174
4	Random Forest - Oversampling	0.808889	0.581191	0.438265	0.499709	0.675170
5	Random Forest - SMOTE	0.786667	0.510593	0.491837	0.501040	0.680293
6	Logistic Regression - Undersampling	0.762111	0.463581	0.587755	0.518335	0.699204
7	Logistic Regression - Oversampling	0.767667	0.472697	0.578571	0.520303	0.699442
8	Logisitic Regression - SMOTE	0.737000	0.426667	0.604082	0.500106	0.689044
9	XGBoost - Undersampling	0.703889	0.391237	0.646939	0.487599	0.683342
10	XGBoost - Oversampling	0.785778	0.509650	0.431122	0.467109	0.657820
11	XGBoost - SMOTE	0.796333	0.545651	0.387245	0.452999	0.648736
12	LightGBM - Undersampling	0.693556	0.381250	0.653571	0.481579	0.679129
13	LightGBM - Oversampling	0.759667	0.455619	0.531633	0.490699	0.677393
14	LightGBM - SMOTE	0.794222	0.537552	0.394388	0.454974	0.649964
15	ExtraTrees - Undersampling	0.786444	0.509369	0.527041	0.518054	0.692853
16	ExtraTrees - Oversampling	0.786111	0.508557	0.530612	0.519351	0.693928
17	ExtraTrees - SMOTE	0.787000	0.510681	0.524490	0.517493	0.692288

TABLE 5. Comparison of Different Metrics for Different Models

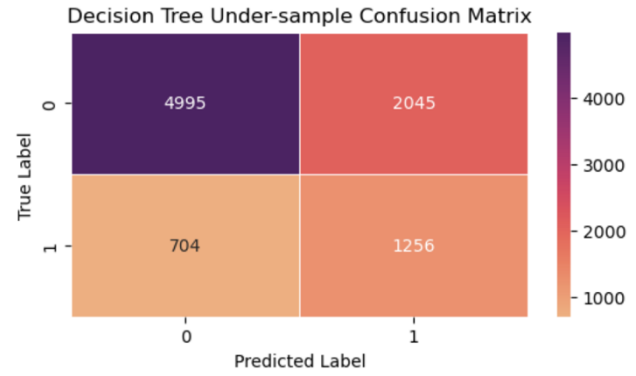

Fig. 6. Decision Tree Confusion Matrix

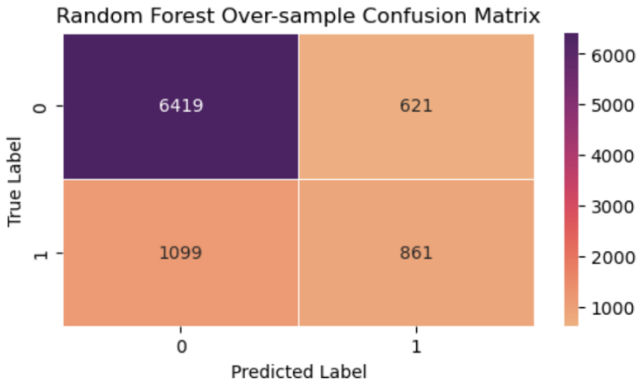

Fig. 7. Random Forest Confusion Matrix

VII. DISCUSSIONS AND CONCLUSIONS

In the previous research for predicting default of credit card, as stated in [4], it was discovered that Decision tree technique performs better in terms of generating accuracy after applying SMOTE technique to balance the dataset.

Whereas this research in contrary to the previous research proves that Random Forest after applying Over-sampling technique method performs even better than Decision tree algorithm. It is however not surprising that the ensembles models perform much better as compared to the standard traditional classification models especially Decision Tree algorithm as seen in Fig. 8 [19], as they are more prone to overfitting due to the complex nature of the data. Random Forest Over-sampling might be the best predictive model but it requires considerable amount of time to train it as compared to Decision Tree method having worse accurate classification rate but requires less time to train it.

Therefore, we can consider this research as a supporting research to the previous ones, to assist future researchers working on data sets that can be either balanced/ imbalanced.

To summarize lending institutions can use machine learning as a potent tool to forecast and identify patterns in consumer data, adding a high level of precision. This model may be used to determine credit card defaults in a better and more advantageous way. The overfitting of trees in memory as data volume grows is the only issue with random forests. Future work on this problem can focus on eliminating the decision tree's overfitting issue and detecting real-time fraud transactions. It can also apply advanced machine learning algorithms such as neural networks and a more thorough hyperparameter tuning values in trying to achieve higher accuracy and better optimal results. Suggestions from a domain expert is also advisable to analyse the model according to the problem's objectives.

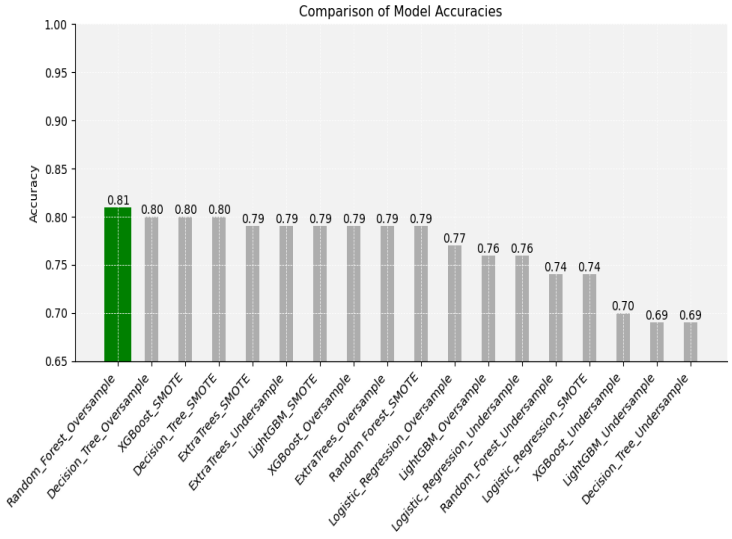

Fig. 8. Comparison of Accuracy for Different Models

VIII. REFERENCES

[1] The Nilson Report . (n.d.). News and Statistics for Card and Mobile Payment Executives. (2019). Nilsonreport.com. https://nilsonreport.com/

[2] Bardiya, A. A., & Tijare, D. P. A. Prediction of credit card defaulters using machine .https://www.ijcrt.org/papers/IJCRT2205920.pdf

[3] Makram Soui, Salima Smiti, Salma Bribech, & Gasmi, I. (2018). Credit Card Default Prediction as a Classification Problem. 88–100. https://doi.org/10.1007/978-3-319-92058-0_9

[4] Khemakhem, S., & Boujelbene, Y. (2018). Predicting credit risk on the basis of financial and non-financial variables and data mining. Review of Accounting and Finance, 17(3), 316–340. https://doi.org/10.1108/raf-07-2017-0143

[5] Alam, T.M., Shaukat, K., Hameed, I.A., Luo, S., Sarwar, M.U., Shabbir, S., Li, J. and Khushi, M. (2020). An Investigation of Credit Card Default Prediction in the Imbalanced Datasets. IEEE Access, 8, pp.201173–201198. https://doi.org/10.1109/access.2020.3033784

[6] Zhang, Y., & Chen, L. (2021). A Study on Forecasting the Default Risk of Bond Based on XGboost Algorithm and Over-Sampling Method. Theoretical Economics Letters, 11(02), 258–267. https://doi.org/10.4236/tel.2021.112019

[7] UCI Machine Learning Repository. (n.d.). Archive.ics.uci.edu. Retrieved June 17, 2023, from https://archive.ics.uci.edu/dataset/350/default+of+credit+card+clients

[8] guest_blog. (2023, April 26). 10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2023). Analytics Vidhya https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/

[9] Sandhyakrishnan02. (2021). Multicollinearity, how to handle it to avoid dummy variable trap?. Kaggle. Retrieved June 17, 2023, from https://www.kaggle.com/general/294096

[10] Zohaib Ahmed. (2021). How to Calculate Z-Scores in Python (scipy.stats as stats). Kaggle. Retrieved June 17, 2023, from https://www.kaggle.com/general/210726

[11] Datai. (2021). Feature Selection and Data Visualization. Kaggle. Retrieved June 17, 2023, from https://www.kaggle.com/code/kanncaa1/feature-selection-and-data-visualization/notebook

[12] Gupta, A. (2023, April 26). Features Selection Techniques in Machine Learning (Updated 2023). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/

[13] Sayad, Dr. Saed. (n.d.). Decision Tree Regression .Saed Sayad. http://www.saedsayad.com/decision_tree_reg.htm

[14] Breiman, L. (2001). Random Forests. https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

[15] Agarwal, A. (2017, March 31). Logistic Regression. Simplified. Medium. https://medium.com/data-science-group-iitr/logistic-regression-simplified-9b4efe801389

[16] Brownlee, J. (2021, February 17). A Gentle Introduction to XGBoost for Applied Machine Learning. Machine Learning Mastery. https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

[17] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (n.d.). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Retrieved June 17, 2023, from https://www.microsoft.com/en-us/research/wp-content/uploads/2017/11/lightgbm.pdf

[18] Browlee, J. (2021, April 17). How to Develop an Extra Trees Ensemble with Python. Machine Learning Mastery. https://machinelearningmastery.com/extra-trees-ensemble-with-python/

[19] Zach. (2021, October 24). How to Create a Pareto Chart in Python (Step-by-Step). Statology. https://www.statology.org/pareto-chart-python/

[20] *A guide to the data protection principles*. ICO. https://ico.org.uk/media/for-organisations/guide-to-the-general-data-protection-regulation-gdpr-1-0.pdf

[21] *FAIR Principles.* GO FAIR. June 17, 2023. https://www.go-fair.org/fair-principles/

IX. APPENDIX

IMPORTING LIBRARIES

```
# Importing relevant libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import confusion_matrix, classification_report, recall_score, precision_recall_curve, accuracy
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from collections import Counter
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, ExtraTreesClassifier
from sklearn.model_selection import StratifiedKFold
from xgboost.sklearn import XGBClassifier
from lightgbm import LGBMClassifier
import warnings
warnings.filterwarnings('ignore')
```

DATA PRE-PROCESSING

```
# Loading the dataset
dataset = pd.read_excel('Credit Card Default Dataset.xls')
```

```
# Printing the dataset
print(dataset)
```

```
# Distribution of data
dataset.shape
```

```
(30000, 25)
```

```
# Return the first 5 values
dataset.head()
```

```
# Description of the dataset
dataset.describe().T
```

```
# Dropping the ID column as it is irrelavant
dataset = dataset.drop('ID', axis=1)
```

```
# Renaming the columns to ensure quality across the dataset
dataset.rename(columns={'PAY_0':'PAY_1'},inplace=True)
dataset.rename(columns={'default payment next month':'Default_pay'},inplace=True)
```

```
# Rechecking the dataset after renaming columns
dataset.head()
```

```
# Checking of Missing values
dataset.isna().sum()
```

```
# Check the count of unique values w.r.t SEX variable
dataset['SEX'].value_counts(dropna=False)
```

```
2    18112
1    11888
Name: SEX, dtype: int64
```

```
# Check the count of unique values w.r.t MARRIAGE variable
dataset['MARRIAGE'].value_counts(dropna=False)
```

```
2    15964
1    13659
3      323
0       54
Name: MARRIAGE, dtype: int64
```

```
# Check the count of unique values w.r.t EDUCATION variable
dataset['EDUCATION'].value_counts(dropna=False)
```

```
# Create a countplot to check the unique values in the Default variable
sns.countplot(x="Default_pay", data = dataset)
```

```
# Calculates the number of unique instances in the dataset where the column "Default_pay" has a value of 0 and 1
Count_No_Default = len(dataset[dataset["Default_pay"]==0])
Count_Yes_Default = len(dataset[dataset["Default_pay"]==1])
```

```
# Calculates the count of Non-Default customers in the Default Pay column
Count_No_Default
```

```
23364
```

```
# Calculates the count of Default customers in the Default Pay column
Count_Yes_Default
```

```
6636
```

```
# Visualization of Default Pay column in the form of pie-chart
labels=["Not Default","Default"]

fraud_or_not = dataset["Default_pay"].value_counts().tolist()
values = [fraud_or_not[0], fraud_or_not[1]]

fig = px.pie(values=dataset['Default_pay'].value_counts(), names=labels , width=700, height=400, color_discrete_seq
             ,title="Default vs Not Default Transactions")
fig.show()
```

FEATURE ANALYSIS

```
# Heatmap Correlation between features of the Dataset
fig, ax = plt.subplots(figsize = (14,7))
corr = dataset.corr().round(2)
sns.heatmap(corr, annot = True, ax=ax, vmin=-1, vmax=1, center=0, cmap='mako')
```

```
# Observing Correlation between features of the Dataset
X = dataset.drop(['Default_pay'],axis=1)
X.corrwith(dataset['Default_pay']).plot.bar(figsize = (20, 10),
                                    fontsize = 25,rot = 90, grid = True)
```

DATA CLEANING

```
# Assigning the values of 0,5,6 categories to 4 (others)categort since we do not have description for 0,5,6
#EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others)

fil = (X_1['EDUCATION'] == 5) | (X_1['EDUCATION'] == 6) | (X_1['EDUCATION'] == 0)
X_1.loc[fil, 'EDUCATION'] = 4
X_1['EDUCATION'].value_counts()
```

```
2    14030
1    10585
3     4917
4      468
Name: EDUCATION, dtype: int64
```

```
 #Assigning the values of 0 category to 3 (others)category since we do not have description for 0
#MARRIAGE: Marital status (1=married, 2=single, 3=others)

X_1.loc[X_1['MARRIAGE']==0, 'MARRIAGE'] = 3
X_1['MARRIAGE'].value_counts()
```

```
2    15964
1    13659
3      377
Name: MARRIAGE, dtype: int64
```

```
# Check the udpated dataset
X_1.head()
```

FEATURE ENGINEERING

```
# One Hot encoding for categorical variable using pandas library
cat_columns = ['SEX','EDUCATION','MARRIAGE','PAY_1','PAY_2','PAY_3','PAY_4',
                    'PAY_5','PAY_6']
X_1[cat_columns] = X_1[cat_columns].astype(str)
X_1 = pd.get_dummies(X_1,columns=cat_columns,drop_first=True)
pd.set_option('display.max_columns', None)
X_1.head()
```

```
# Convert the columns to lowercase
X_1.columns = X_1.columns.map(str.lower)
```

```
# Feature Scaling of Numerical Attributes
columns_to_normalization = ['limit_bal', 'age', 'bill_amt1', 'pay_amt1', 'pay_amt2', 'pay_amt3',
       'pay_amt4', 'pay_amt5', 'pay_amt6']
X_1[columns_to_normalization] = X_1[columns_to_normalization].apply(lambda x : (x-np.mean(x))/np.std(x))
```

```
X_1.head()
```

	limit_bal	age	bill_amt1	pay_amt1	pay_amt2	pay_amt3	pay_amt4	pay_amt5	pay_amt6	sex_2	education_2	education_3	education_4	marriage_2
0	-1.136720	-1.246020	-0.642501	-0.341942	-0.227086	-0.296801	-0.308063	-0.314136	-0.293382	1	1	0	0	0
1	-0.365981	-1.029047	-0.659219	-0.341942	-0.213588	-0.240005	-0.244230	-0.314136	-0.180878	1	1	0	0	1
2	-0.597202	-0.161156	-0.298560	-0.250292	-0.191887	-0.240005	-0.244230	-0.248683	-0.012122	1	1	0	0	1
3	-0.905498	0.164303	-0.057491	-0.221191	-0.169361	-0.228645	-0.237846	-0.244166	-0.237130	1	1	0	0	0
4	-0.905498	2.334029	-0.578618	-0.221191	1.335034	0.271165	0.266434	-0.269039	-0.255187	0	1	0	0	0

MODEL TRAINING AND TESTING

```
# Dataset split into dependent and independent feature
X = X_1
y = dataset['Default_pay']
```

```
# Dataset split into train and test set, allocate 70% to train set and 30% to test set
X_train, X_test, y_train, y_test = train_test_split(X_1, y, test_size=0.3, random_state=42)
```

```
# Return of Independent Train set Data
print(X_train)
```

CROSS-VALIDATION

```
# Cross Validation using 5 Folds
skfold = StratifiedKFold(n_splits=5)

```

** CLASS IMBALANCE TECHNIQUES TO BALANCE THE DATASET

```
SMOTE
```

```
print("Before oversampling: ",Counter(y_train))
SMOTE= SMOTE()

X_smote,y_smote= SMOTE.fit_resample(X_train,y_train)

# summarize class distribution
print("After oversampling: ",Counter(y_smote))
```

```
Before oversampling:  Counter({0: 16324, 1: 4676})
After oversampling:  Counter({1: 16324, 0: 16324})
```

RANDOM UNDERSAMPLING

```
print("Before Undersampling: ",Counter(y_train))

rus = RandomUnderSampler(random_state=42)
X_under, y_under = rus.fit_resample(X_train, y_train)

print("After Undersampling: ",Counter(y_under))
```

```
Before Undersampling:  Counter({0: 16324, 1: 4676})
After Undersampling:  Counter({0: 4676, 1: 4676})
```

RANDOM OVERSAMPLING

```
print("Before Oversampling: ",Counter(y_train))

ros = RandomOverSampler(random_state=42)
X_over, y_over = ros.fit_resample(X_train, y_train)

print("After Oversampling: ",Counter(y_over))
```

```
Before Oversampling:  Counter({0: 16324, 1: 4676})
After Oversampling:  Counter({1: 16324, 0: 16324})
```

** GRIDSEARCHCV W.R.T EACH CLASS IMBALANCE METHOD

GRIDSEARCHCV (UNDERSAMPLING)

```
# DecisionTree Classifier
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
               "min_samples_leaf": list(range(5,7,1))}
grid_tree = GridSearchCV(DecisionTreeClassifier(), tree_params, cv=skfold, scoring='recall')
grid_tree = grid_tree.fit(X_under, y_under.ravel())
tree_accuracy = grid_tree.best_score_
tree_clf = grid_tree.best_params_
print('DecisionTree GridSearchCV Accuracy: ', tree_accuracy)
print('DecisionTree Best Parameters: ', tree_clf)
print("-------------------------------------------------------------------------------------------")

# Random Forest
random_params = {"criterion": ["gini", "entropy"], 'n_estimators':[100, 123, 145]}
grid_random = GridSearchCV(RandomForestClassifier(), random_params, cv=skfold, scoring='recall')
grid_random = grid_random.fit(X_under, y_under.ravel())
random_accuracy = grid_random.best_score_
random = grid_random.best_params_
print('Random Forest GridSearchCV Accuracy: ', random_accuracy)
print('Random Forest Best Parameters: ', random)
print("-------------------------------------------------------------------------------------------")

# Logistic Regression
log_reg_params = {'solver': ['saga'], 'C': [0.01, 0.1, 1, 10, 100]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params, cv=skfold, scoring='recall')
grid_log_reg = grid_log_reg.fit(X_under, y_under.ravel())
log_accuracy = grid_log_reg.best_score_
log_reg = grid_log_reg.best_params_
print('Logistic Regression GridSearchCV Accuracy: ', log_accuracy)
print('Logistic Regression Best Parameters: ', log_reg)
print("-------------------------------------------------------------------------------------------")

#XGBoost
xb_params = {'n_estimators': [50,100,150,200], 'max_depth': [3,5,7,10], 'min_child_weight': [2,3,4,5]}
grid_xb = GridSearchCV(XGBClassifier(), xb_params, cv=skfold, scoring='recall', n_jobs=-1)
grid_xb = grid_xb.fit(X_under, y_under.ravel())
xb_accuracy = grid_xb.best_score_
xb = grid_xb.best_params_
print('XGBoost GridSearchCV Accuracy: ', xb_accuracy)
print('XGBoost Best Parameters: ', xb)
print("-------------------------------------------------------------------------------------------")

```

```
#LightGBM
lgb_params = {'learning_rate': [0.4, 0.2, 0.1],
              'max_depth': [15, 10, 20],
              'num_leaves': [32,25,40],
              'feature_fraction': [0.8, 0.5, 0.2],
              'subsample': [0.2, 0.5, 0.8, 1]}
grid_lgb = GridSearchCV(LGBMClassifier(), lgb_params, cv=skfold, scoring='recall', n_jobs=-1)
grid_lgb = grid_lgb.fit(X_under, y_under.ravel())
lgb_accuracy = grid_lgb.best_score_
lgb = grid_lgb.best_params_
print('LightGBM GridSearchCV Accuracy: ', lgb_accuracy)
print('LightGBM Best Parameters: ', lgb)
print("-----------------------------------------------------------------------------------------------")

#Adaptive Boost
ab_params = {'n_estimators':[50, 110, 133],'learning_rate': [1], 'algorithm': ['SAMME', 'SAMME.R']}
grid_ab = GridSearchCV(AdaBoostClassifier(), ab_params, cv=skfold, scoring='recall')
grid_ab = grid_ab.fit(X_under, y_under.ravel())
ab_accuracy = grid_ab.best_score_
ab = grid_ab.best_params_
print('Adaptive Boost GridSearchCV Accuracy: ', ab_accuracy)
print('Adaptive Boost Best Parameters: ', ab)
print("-----------------------------------------------------------------------------------------------")

#ExtraTrees
et_params = {"criterion": ["gini", "entropy"], 'n_estimators':[100, 300, 1500], "max_depth": list(range(2,6,1)),
             "min_samples_leaf": list(range(3,7,1))}
grid_et = GridSearchCV(ExtraTreesClassifier(), et_params, cv=skfold, scoring='recall')
grid_et = grid_et.fit(X_under, y_under.ravel())
et_accuracy = grid_et.best_score_
et = grid_et.best_params_
print('ExtraTrees GridSearchCV Accuracy: ', et_accuracy)
print('ExtraTrees Best Parameters: ', et)
```

GRIDSEARCHCV (OVERSAMPLING)

```
# DecisionTree Classifier
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
               "min_samples_leaf": list(range(5,7,1))}
grid_tree = GridSearchCV(DecisionTreeClassifier(), tree_params, cv=skfold, scoring='recall')
grid_tree = grid_tree.fit(X_over, y_over.ravel())
tree_accuracy = grid_tree.best_score_
tree_clf = grid_tree.best_params_
print('DecisionTree GridSearchCV Accuracy: ', tree_accuracy)
print('DecisionTree Best Parameters: ', tree_clf)
print("-----------------------------------------------------------------------------------------------")

# Random Forest
random_params = {"criterion": ["gini", "entropy"], 'n_estimators':[100, 123, 145]}
grid_random = GridSearchCV(RandomForestClassifier(), random_params, cv=skfold, scoring='recall')
grid_random = grid_random.fit(X_over, y_over.ravel())
random_accuracy = grid_random.best_score_
random = grid_random.best_params_
print('Random Forest GridSearchCV Accuracy: ', random_accuracy)
print('Random Forest Best Parameters: ', random)
print("-----------------------------------------------------------------------------------------------")

# Logistic Regression
log_reg_params = {'solver': ['saga'], 'C': [0.01, 0.1, 1, 10, 100]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params, cv=skfold, scoring='recall')
grid_log_reg = grid_log_reg.fit(X_over, y_over.ravel())
log_accuracy = grid_log_reg.best_score_
log_reg = grid_log_reg.best_params_
print('Logistic Regression GridSearchCV Accuracy: ', log_accuracy)
print('Logistic Regression Best Parameters: ', log_reg)
print("-----------------------------------------------------------------------------------------------")

#XGBoost
xb_params = {'n_estimators': [50,100,150,200], 'max_depth': [3,5,7,10], 'min_child_weight': [2,3,4,5]}
grid_xb = GridSearchCV(XGBClassifier(), xb_params, cv=skfold, scoring='recall', n_jobs=-1)
grid_xb = grid_xb.fit(X_over, y_over.ravel())
xb_accuracy = grid_xb.best_score_
xb = grid_xb.best_params_
print('XGBoost GridSearchCV Accuracy: ', xb_accuracy)
print('XGBoost Best Parameters: ', xb)
print("-----------------------------------------------------------------------------------------------")

```

```
#LightGBM
lgb_params = {'learning_rate': [0.4, 0.2, 0.1],
              'max_depth': [15, 10, 20],
              'num_leaves': [32,25,40],
              'feature_fraction': [0.8, 0.5, 0.2],
              'subsample': [0.2, 0.5, 0.8, 1]}
grid_lgb = GridSearchCV(LGBMClassifier(), lgb_params, cv=skfold, scoring='recall', n_jobs=-1)
grid_lgb = grid_lgb.fit(X_over, y_over.ravel())
lgb_accuracy = grid_lgb.best_score_
lgb = grid_lgb.best_params_
print('LightGBM GridSearchCV Accuracy: ', lgb_accuracy)
print('LightGBM Best Parameters: ', lgb)
print("-----------------------------------------------------------------------------------------")

#Adaptive Boost
ab_params = {'n_estimators':[50, 110, 133],'learning_rate': [1], 'algorithm': ['SAMME', 'SAMME.R']}
grid_ab = GridSearchCV(AdaBoostClassifier(), ab_params, cv=skfold, scoring='recall')
grid_ab = grid_ab.fit(X_over, y_over.ravel())
ab_accuracy = grid_ab.best_score_
ab = grid_ab.best_params_
print('Adaptive Boost GridSearchCV Accuracy: ', ab_accuracy)
print('Adaptive Boost Best Parameters: ', ab)
print("-----------------------------------------------------------------------------------------")

#ExtraTrees
et_params = {"criterion": ["gini", "entropy"], 'n_estimators':[100, 300, 1500], "max_depth": list(range(2,6,1)),
             "min_samples_leaf": list(range(3,7,1))}
grid_et = GridSearchCV(ExtraTreesClassifier(), et_params, cv=skfold, scoring='recall')
grid_et = grid_et.fit(X_over, y_over.ravel())
et_accuracy = grid_et.best_score_
et = grid_et.best_params_
print('ExtraTrees GridSearchCV Accuracy: ', et_accuracy)
print('ExtraTrees Best Parameters: ', et)
```

GRIDSEARCHCV(SMOTE)

```
# DecisionTree Classifier
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
               "min_samples_leaf": list(range(5,7,1))}
grid_tree = GridSearchCV(DecisionTreeClassifier(), tree_params, cv=skfold, scoring='recall')
grid_tree = grid_tree.fit(X_smote, y_smote.ravel())
tree_accuracy = grid_tree.best_score_
tree_clf = grid_tree.best_params_
print('DecisionTree GridSearchCV Accuracy: ', tree_accuracy)
print('DecisionTree Best Parameters: ', tree_clf)
print("-----------------------------------------------------------------------------------------")

# Random Forest
random_params = {"criterion": ["gini", "entropy"], 'n_estimators':[100, 123, 145]}
grid_random = GridSearchCV(RandomForestClassifier(), random_params, cv=skfold, scoring='recall')
grid_random = grid_random.fit(X_smote, y_smote.ravel())
random_accuracy = grid_random.best_score_
random = grid_random.best_params_
print('Random Forest GridSearchCV Accuracy: ', random_accuracy)
print('Random Forest Best Parameters: ', random)
print("-----------------------------------------------------------------------------------------")

# Logistic Regression
log_reg_params = {'solver': ['saga'], 'C': [0.01, 0.1, 1, 10, 100]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params, cv=skfold, scoring='recall')
grid_log_reg = grid_log_reg.fit(X_smote, y_smote.ravel())
log_accuracy = grid_log_reg.best_score_
log_reg = grid_log_reg.best_params_
print('Logistic Regression GridSearchCV Accuracy: ', log_accuracy)
print('Logistic Regression Best Parameters: ', log_reg)
print("-----------------------------------------------------------------------------------------")

#XGBoost
xb_params = {'n_estimators': [50,100,150,200], 'max_depth': [3,5,7,10], 'min_child_weight': [2,3,4,5]}
grid_xb = GridSearchCV(XGBClassifier(), xb_params, cv=skfold, scoring='recall', n_jobs=-1)
grid_xb = grid_xb.fit(X_smote, y_smote.ravel())
xb_accuracy = grid_xb.best_score_
xb = grid_xb.best_params_
print('XGBoost GridSearchCV Accuracy: ', xb_accuracy)
print('XGBoost Best Parameters: ', xb)
print("-----------------------------------------------------------------------------------------")

```

```
#LightGBM
lgb_params = {'learning_rate': [0.4, 0.2, 0.1],
                'max_depth': [15, 10, 20],
                'num_leaves': [32,25,40],
                'feature_fraction': [0.8, 0.5, 0.2],
                'subsample': [0.2, 0.5, 0.8, 1]}
grid_lgb = GridSearchCV(LGBMClassifier(), lgb_params, cv=skfold, scoring='recall', n_jobs=-1)
grid_lgb = grid_lgb.fit(X_smote, y_smote.ravel())
lgb_accuracy = grid_lgb.best_score_
lgb = grid_lgb.best_params_
print('LightGBM GridSearchCV Accuracy: ', lgb_accuracy)
print('LightGBM Best Parameters: ', lgb)
print("-----------------------------------------------------------------------------------------------")

#Adaptive Boost
ab_params = {'n_estimators':[50, 110, 133],'learning_rate': [1], 'algorithm': ['SAMME', 'SAMME.R']}
grid_ab = GridSearchCV(AdaBoostClassifier(), ab_params, cv=skfold, scoring='recall')
grid_ab = grid_ab.fit(X_smote, y_smote.ravel())
ab_accuracy = grid_ab.best_score_
ab = grid_ab.best_params_
print('Adaptive Boost GridSearchCV Accuracy: ', ab_accuracy)
print('Adaptive Boost Best Parameters: ', ab)
print("-----------------------------------------------------------------------------------------------")

#ExtraTrees
et_params = {"criterion": ["gini", "entropy"], 'n_estimators':[100, 300, 1500], "max_depth": list(range(2,6,1)),
              "min_samples_leaf": list(range(3,7,1))}
grid_et = GridSearchCV(ExtraTreesClassifier(), et_params, cv=skfold, scoring='recall')
grid_et = grid_et.fit(X_smote, y_smote.ravel())
et_accuracy = grid_et.best_score_
et = grid_et.best_params_
print('ExtraTrees GridSearchCV Accuracy: ', et_accuracy)
print('ExtraTrees Best Parameters: ', et)
```

MODEL TRAINING AND TESTING

DECISION TREE

```
7]:
# DecisionTree - Undersampling
dt_under = DecisionTreeClassifier(criterion = 'gini', max_depth = 3, min_samples_leaf = 5)

#Fit the classifier to the training data
dt_under.fit(X_under,y_under)

#Generate predictions for test data
pred_dt_under= dt_under.predict(X_test)

# Generate the confusion matrix
cnf_matrix = confusion_matrix(y_test, pred_dt_under)

# Calculate recall
recall = cnf_matrix[1, 1] / (cnf_matrix[1, 1] + cnf_matrix[1, 0])

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(6, 3))
sns.heatmap(cnf_matrix, cmap="flare", annot=True, linewidths=0.5, fmt='g')
plt.title("Decision Tree Under-sample Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# Perform Cross Validation of the model and returns average 5 Fold Cross Validation Score
cv_scores = cross_val_score(dt_under, X_under, y_under, cv=skfold)
print("Average 5-Fold CV Score: {}".format(round(np.mean(cv_scores),4)))

# Print the classification report
print("\n---------- Classification Report ----------")
print(classification_report(y_test, pred_dt_under))

# Calculates various evaluation metrics to assess the performance of the model
roc=roc_auc_score(y_test, pred_dt_under)
acc = accuracy_score(y_test, pred_dt_under)
prec = precision_score(y_test, pred_dt_under)
rec = recall_score(y_test, pred_dt_under)
f1 = f1_score(y_test, pred_dt_under)

# Returns the results of various evaluation metrics of the model in the tabular form
results = pd.DataFrame([['Decision Tree - Undersampling', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
results

```

```
# DecisionTree - Oversampling
dt_over = DecisionTreeClassifier(criterion = 'entropy', max_depth = 2, min_samples_leaf = 5)

dt_over.fit(X_over,y_over)

pred_dt_over= dt_over.predict(X_test)

# Generate the confusion matrix
cnf_matrix = confusion_matrix(y_test, pred_dt_over)

# Calculate recall
recall = cnf_matrix[1, 1] / (cnf_matrix[1, 1] + cnf_matrix[1, 0])

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(6, 3))
sns.heatmap(cnf_matrix, cmap='flare', annot=True, linewidths=0.5, fmt='g')
plt.title("Decision Tree Over-sample Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

cv_scores = cross_val_score(dt_over, X_over, y_over, cv=skfold)
print("Average 5-Fold CV Score: {}".format(round(np.mean(cv_scores),4)))

# Print the classification report
print("\n---------- Classification Report ----------")
print(classification_report(y_test, pred_dt_over))

roc=roc_auc_score(y_test, pred_dt_over)
acc = accuracy_score(y_test, pred_dt_over)
prec = precision_score(y_test, pred_dt_over)
rec = recall_score(y_test, pred_dt_over)
f1 = f1_score(y_test, pred_dt_over)

#
model_results = pd.DataFrame([['Decision Tree - Overersampling', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
results = results.append(model_results,ignore_index =True)
results

```

```
# DecisionTree - SMOTE
dt_smote = DecisionTreeClassifier(criterion = 'gini', max_depth = 2, min_samples_leaf = 5)

dt_smote.fit(X_smote,y_smote)

pred_dt_smote= dt_smote.predict(X_test)

# Generate the confusion matrix
cnf_matrix = confusion_matrix(y_test, pred_dt_smote)

# Calculate recall
recall = cnf_matrix[1, 1] / (cnf_matrix[1, 1] + cnf_matrix[1, 0])

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(6, 3))
sns.heatmap(cnf_matrix, cmap="flare", annot=True, linewidths=0.5, fmt='g')
plt.title("Decision Tree SMOTE Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

cv_scores = cross_val_score(dt_smote, X_smote, y_smote, cv=skfold)
print("Average 5-Fold CV Score: {}".format(round(np.mean(cv_scores),4)))

# Print the classification report
print("\n---------- Classification Report ----------")
print(classification_report(y_test, pred_dt_smote))

roc=roc_auc_score(y_test, pred_dt_smote)
acc = accuracy_score(y_test, pred_dt_smote)
prec = precision_score(y_test, pred_dt_smote)
rec = recall_score(y_test, pred_dt_smote)
f1 = f1_score(y_test, pred_dt_smote)


model_results = pd.DataFrame([['Decision Tree - SMOTE', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
results = results.append(model_results,ignore_index=True)
results
```

```
# Decision Trees - ROC-AUC Curve

y_pred_dt_smote = dt_smote.predict_proba(X_test)[:, 1]
fpr_dt_smote, tpr_dt_smote, _ = roc_curve(y_test, y_pred_dt_smote)
auc_dt_smote = roc_auc_score(y_test, y_pred_dt_smote)

y_pred_dt_over = dt_over.predict_proba(X_test)[:, 1]
fpr_dt_over, tpr_dt_over, _ = roc_curve(y_test, y_pred_dt_over)
auc_dt_over = roc_auc_score(y_test, y_pred_dt_over)

y_pred_dt_under = dt_under.predict_proba(X_test)[:, 1]
fpr_dt_under, tpr_dt_under, _ = roc_curve(y_test, y_pred_dt_under)
auc_dt_under = roc_auc_score(y_test, y_pred_dt_under)

# Plot ROC curves
plt.plot(fpr_dt_smote, tpr_dt_smote, label='Decision Trees (SMOTE), AUC = {:.2f}'.format(auc_dt_smote))
plt.plot(fpr_dt_over, tpr_dt_over, label='Decision Trees (Oversample), AUC = {:.2f}'.format(auc_dt_over))
plt.plot(fpr_dt_under, tpr_dt_under, label='Decision Trees (Undersample), AUC = {:.2f}'.format(auc_dt_under))
plt.xlim([-0.01, 1.01])
plt.ylim([0, 1])
plt.plot([0, 1], [0, 1], 'k--')  # Baseline ROC curve
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves - Decision Trees with Sampling Techniques')
plt.legend(loc='lower right')
plt.show()
```

RANDOM FOREST

```
# Random Forest - Undersampling
rf_under = RandomForestClassifier(criterion = 'gini', n_estimators = 145)

rf_under.fit(X_under,y_under)

pred_rf_under= rf_under.predict(X_test)

# Generate the confusion matrix
cnf_matrix = confusion_matrix(y_test, pred_rf_under)

# Calculate recall
recall = cnf_matrix[1, 1] / (cnf_matrix[1, 1] + cnf_matrix[1, 0])

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(6, 3))
sns.heatmap(cnf_matrix, cmap="flare", annot=True, linewidths=0.5, fmt='g')
plt.title("Random Forest Under-sample Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

cv_scores = cross_val_score(rf_under, X_under, y_under, cv=skfold)
print("Average 5-Fold CV Score: {}".format(round(np.mean(cv_scores),4)))

# Print the classification report
print("\n---------- Classification Report ----------")
print(classification_report(y_test, pred_rf_under))

roc=roc_auc_score(y_test, pred_rf_under)
acc = accuracy_score(y_test, pred_rf_under)
prec = precision_score(y_test, pred_rf_under)
rec = recall_score(y_test, pred_rf_under)
f1 = f1_score(y_test, pred_rf_under)


model_results = pd.DataFrame([['Random Forest - Undersampling', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
results = results.append(model_results,ignore_index =True)
results
```

```python
# Random Forest Oversampling
rf_over = RandomForestClassifier(criterion='entropy', n_estimators=123)

rf_over.fit(X_over,y_over)

pred_rf_over= rf_over.predict(X_test)

# Generate the confusion matrix
cnf_matrix = confusion_matrix(y_test, pred_rf_over)

# Calculate recall
recall = cnf_matrix[1, 1] / (cnf_matrix[1, 1] + cnf_matrix[1, 0])

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(6, 3))
sns.heatmap(cnf_matrix, cmap="flare", annot=True, linewidths=0.5, fmt='g')
plt.title("Random Forest Over-sample Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

cv_scores = cross_val_score(rf_over, X_over, y_over, cv=skfold)
print("Average 5-Fold CV Score: {}".format(round(np.mean(cv_scores),4)))

# Print the classification report
print("\n---------- Classification Report ----------")
print(classification_report(y_test, pred_rf_over))

roc=roc_auc_score(y_test, pred_rf_over)
acc = accuracy_score(y_test, pred_rf_over)
prec = precision_score(y_test, pred_rf_over)
rec = recall_score(y_test, pred_rf_over)
f1 = f1_score(y_test, pred_rf_over)


model_results = pd.DataFrame([['Random Forest - Oversampling', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
results = results.append(model_results,ignore_index =True)
results
```

```python
# Random Forest - SMOTE
rf_smote = RandomForestClassifier(criterion='entropy', n_estimators=145)

rf_smote.fit(X_smote,y_smote)

pred_rf_smote= rf_smote.predict(X_test)

# Generate the confusion matrix
cnf_matrix = confusion_matrix(y_test, pred_rf_smote)

# Calculate recall
recall = cnf_matrix[1, 1] / (cnf_matrix[1, 1] + cnf_matrix[1, 0])

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(6, 3))
sns.heatmap(cnf_matrix, cmap="flare", annot=True, linewidths=0.5, fmt='g')
plt.title("Random Forest SMOTE Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

cv_scores = cross_val_score(rf_smote, X_smote, y_smote, cv=skfold)
print("Average 5-Fold CV Score: {}".format(round(np.mean(cv_scores),4)))

# Print the classification report
print("\n---------- Classification Report ----------")
print(classification_report(y_test, pred_rf_smote))

roc=roc_auc_score(y_test, pred_rf_smote)
acc = accuracy_score(y_test, pred_rf_smote)
prec = precision_score(y_test, pred_rf_smote)
rec = recall_score(y_test, pred_rf_smote)
f1 = f1_score(y_test, pred_rf_smote)


model_results = pd.DataFrame([['Random Forest - SMOTE', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
results = results.append(model_results,ignore_index =True)
results
```

]:
```python
# Random Forest - ROC-AUC Curve
y_pred_rf_smote = rf_smote.predict_proba(X_test)[:, 1]
fpr_rf_smote, tpr_rf_smote, _ = roc_curve(y_test, y_pred_rf_smote)
auc_rf_smote = roc_auc_score(y_test, y_pred_rf_smote)

y_pred_rf_over = rf_over.predict_proba(X_test)[:, 1]
fpr_rf_over, tpr_rf_over, _ = roc_curve(y_test, y_pred_rf_over)
auc_rf_over = roc_auc_score(y_test, y_pred_rf_over)

y_pred_rf_under = rf_under.predict_proba(X_test)[:, 1]
fpr_rf_under, tpr_rf_under, _ = roc_curve(y_test, y_pred_rf_under)
auc_rf_under = roc_auc_score(y_test, y_pred_rf_under)

# Plot ROC curves
plt.plot(fpr_rf_smote, tpr_rf_smote, label='Random Forest (SMOTE), AUC = {:.2f}'.format(auc_rf_smote))
plt.plot(fpr_rf_over, tpr_rf_over, label='Random Forest (Oversample), AUC = {:.2f}'.format(auc_rf_over))
plt.plot(fpr_rf_under, tpr_rf_under, label='Random Forest (Undersample), AUC = {:.2f}'.format(auc_rf_under))
plt.xlim([-0.01, 1.01])
plt.ylim([0, 1])
plt.plot([0, 1], [0, 1], 'k--')  # Baseline ROC curve
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves - Random Forest with Sampling Techniques')
plt.legend(loc='lower right')
plt.show()
```

LOGISTIC REGRESSION

```python
# Logistic Regrssion - Undersampling
lr_under = LogisticRegression(C = 0.01, solver = 'saga')

lr_under.fit(X_under,y_under)

pred_lr_under= lr_under.predict(X_test)

# Generate the confusion matrix
cnf_matrix = confusion_matrix(y_test, pred_lr_under)

# Calculate recall
recall = cnf_matrix[1, 1] / (cnf_matrix[1, 1] + cnf_matrix[1, 0])

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(6, 3))
sns.heatmap(cnf_matrix, cmap="flare", annot=True, linewidths=0.5, fmt='g')
plt.title("Logistic Regression Under-sample Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

cv_scores = cross_val_score(lr_under, X_under, y_under, cv=skfold)
print("Average 5-Fold CV Score: {}".format(round(np.mean(cv_scores),4)))

# Print the classification report
print("\n---------- Classification Report ----------")
print(classification_report(y_test, pred_lr_under))

roc=roc_auc_score(y_test, pred_lr_under)
acc = accuracy_score(y_test, pred_lr_under)
prec = precision_score(y_test, pred_lr_under)
rec = recall_score(y_test, pred_lr_under)
f1 = f1_score(y_test, pred_lr_under)


model_results = pd.DataFrame([['Logistic Regression - Undersampling', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
results = results.append(model_results,ignore_index =True)
results
```

```python
# Logistic Regression - Oversampling
lr_over = LogisticRegression(C = 0.01, solver = 'saga')

lr_over.fit(X_over,y_over)

pred_lr_over= lr_over.predict(X_test)

# Generate the confusion matrix
cnf_matrix = confusion_matrix(y_test, pred_lr_over)

# Calculate recall
recall = cnf_matrix[1, 1] / (cnf_matrix[1, 1] + cnf_matrix[1, 0])

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(6, 3))
sns.heatmap(cnf_matrix, cmap="flare", annot=True, linewidths=0.5, fmt='g')
plt.title("Logistic Regression Over-sample Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

cv_scores = cross_val_score(lr_over, X_over, y_over, cv=skfold)
print("Average 5-Fold CV Score: {}".format(round(np.mean(cv_scores),4)))

# Print the classification report
print("\n---------- Classification Report ----------")
print(classification_report(y_test, pred_lr_over))

roc=roc_auc_score(y_test, pred_lr_over)
acc = accuracy_score(y_test, pred_lr_over)
prec = precision_score(y_test, pred_lr_over)
rec = recall_score(y_test, pred_lr_over)
f1 = f1_score(y_test, pred_lr_over)


model_results = pd.DataFrame([['Logistic Regression - Oversampling', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
results = results.append(model_results,ignore_index =True)
results
```

```python
# Logistic Regression - SMOTE
lr_smote = LogisticRegression(C = 100, solver = 'saga')

lr_smote.fit(X_smote,y_smote)

pred_lr_smote= lr_smote.predict(X_test)

# Generate the confusion matrix
cnf_matrix = confusion_matrix(y_test, pred_lr_smote)

# Calculate recall
recall = cnf_matrix[1, 1] / (cnf_matrix[1, 1] + cnf_matrix[1, 0])

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(6, 3))
sns.heatmap(cnf_matrix, cmap="flare", annot=True, linewidths=0.5, fmt='g')
plt.title("Logistic Regression SMOTE Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

cv_scores = cross_val_score(lr_smote, X_smote, y_smote, cv=skfold)
print("Average 5-Fold CV Score: {}".format(round(np.mean(cv_scores),4)))

# Print the classification report
print("\n---------- Classification Report ----------")
print(classification_report(y_test, pred_lr_smote))

roc=roc_auc_score(y_test, pred_lr_smote)
acc = accuracy_score(y_test, pred_lr_smote)
prec = precision_score(y_test, pred_lr_smote)
rec = recall_score(y_test, pred_lr_smote)
f1 = f1_score(y_test, pred_lr_smote)


model_results = pd.DataFrame([['Logisitic Regression - SMOTE', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
results = results.append(model_results,ignore_index =True)
results
```

```
# Logistic Regression - ROC-AUC Curve
y_pred_lr_smote = lr_smote.predict_proba(X_test)[:, 1]
fpr_lr_smote, tpr_lr_smote, _ = roc_curve(y_test, y_pred_lr_smote)
auc_lr_smote = roc_auc_score(y_test, y_pred_lr_smote)

y_pred_lr_over = lr_over.predict_proba(X_test)[:, 1]
fpr_lr_over, tpr_lr_over, _ = roc_curve(y_test, y_pred_lr_over)
auc_lr_over = roc_auc_score(y_test, y_pred_lr_over)

y_pred_lr_under = lr_under.predict_proba(X_test)[:, 1]
fpr_lr_under, tpr_lr_under, _ = roc_curve(y_test, y_pred_lr_under)
auc_lr_under = roc_auc_score(y_test, y_pred_lr_under)

# Plot ROC curves
plt.plot(fpr_lr_smote, tpr_lr_smote, label='Logistic Regression (SMOTE), AUC = {:.2f}'.format(auc_lr_smote))
plt.plot(fpr_lr_over, tpr_lr_over, label='Logistic Regression (Oversample), AUC = {:.2f}'.format(auc_lr_over))
plt.plot(fpr_lr_under, tpr_lr_under, label='Logistic Regression (Undersample), AUC = {:.2f}'.format(auc_lr_under))
plt.xlim([-0.01, 1.01])
plt.ylim([0, 1])
plt.plot([0, 1], [0, 1], 'k--')  # Baseline ROC curve
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves - Logistic Regression with Sampling Techniques')
plt.legend(loc='lower right')
plt.show()
```

** COMPARISION OF MODEL ACCURACIES USING PARETO CHART**

```
# List of model names
model_names = ['Decision_Tree_Undersample', 'Decision_Tree_Oversample','Decision_Tree_SMOTE',
               'Random_Forest_Undersample', 'Random_Forest_Oversample', 'Random Forest_SMOTE',
               'Logistic_Regression_Undersample', 'Logistic_Regression_Oversample','Logistic_Regression_SMOTE',
               'XGBoost_Undersample', 'XGBoost_Oversample', 'XGBoost_SMOTE',
               'LightGBM_Undersample', 'LightGBM_Oversample', 'LightGBM_SMOTE',
               'ExtraTrees_Undersample','ExtraTrees_Oversample', 'ExtraTrees_SMOTE']

# List of accuracy values for each model
accuracies = [0.69, 0.80, 0.80, 0.74, 0.81, 0.79, 0.76, 0.77,
              0.74, 0.70, 0.79, 0.80, 0.69, 0.76, 0.79, 0.79, 0.79, 0.79]

# Sort the model names and accuracies in descending order based on accuracies
sorted_indices = np.argsort(accuracies)[::-1]
sorted_model_names = [model_names[i] for i in sorted_indices]
sorted_accuracies = [accuracies[i] for i in sorted_indices]

# Set colors for the bars
bar_colors = ['grey' if i != np.argmax(sorted_accuracies) else 'red' for i in range(len(sorted_model_names))]

# Create a Pareto chart
fig, ax1 = plt.subplots(figsize=(10, 6))

# Set the background color
ax1.set_facecolor('#F2F2F2')

# Plot the bars for accuracies
ax1.bar(sorted_model_names, sorted_accuracies, color=bar_colors, alpha=0.6, width=0.4)

# Set the y-axis label for accuracies
ax1.set_ylabel('Accuracy')

# Set the y-axis limits
ax1.set_ylim([0.65, 1.0])

# Find the index and value of the model with the highest accuracy
best_model_idx = np.argmax(sorted_accuracies)
best_accuracy = sorted_accuracies[best_model_idx]

# Highlight the model with the highest accuracy using a red marker
ax1.bar(sorted_model_names[best_model_idx], best_accuracy, color='green', label='Highest Accuracy')

# Display the accuracies of all models on top of the bars
for i, acc in enumerate(sorted_accuracies):
    ax1.text(i, acc, f'{acc:.2f}', ha='center', va='bottom')

# Set the chart title
plt.title('Comparison of Model Accuracies')

# Customize tick labels
plt.xticks(rotation=45, fontsize=10, ha='right')

# Add a grid
plt.grid(color='white', linestyle='--', linewidth=0.5)

# Remove top and right spines
ax1.spines['top'].set_visible(False)
ax1.spines['right'].set_visible(False)

# Show the plot
plt.tight_layout()
plt.show()

```