Mini Project

On

TERM DEPOSIT PREDICTION

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE & ENGINEERING

BY

20WH1A05D8	A. TEJASWINI
20WH1A05D9	V. SATHWIKA
20WH1A05E0	P. SHINEE
20WH1A05E1	G. CHANDHANA
20WH1A05E2	P. PRANATHI

Under the esteemed guidance of

Dr. Reya Sharma
Assistant Professor

Department of Computer Science & Engineering

BVRIT HYDERABAD
COLLEGE OF ENGINEERING FOR WOMEN

(NBA Accredited EEE.ECE.CSE.IT B. Tech Courses)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090

Dec, 2023

Department of Computer Science & Engineering

CERTIFICATE

This is to certify that the mini project entitled "**Term Deposit Prediction**" is a bonafide work carried out by **Ms. A. Tejaswini (20WH1A05D8), Ms. V. Sathwika (20WH1A05D9), Ms. P. Shinee (20WH1A05E0), Ms. G. Chandhana (20WH1A05E1), Ms. P. Pranathi (20WH1A05E2)** in partial fulfillment for the award of B. Tech degree in **Computer Science & Engineering** , **BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision. The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Guide	**Head of the Department**
Dr. Reya Sharma	**Dr. R. Suneetha Rani**
Assistant Professor, CSE	**Professor, CSE**

External Examiner

DECLARATION

We hereby declare that the work presented in this project entitled **“Term Deposit Prediction”** submitted towards completion of Project work in IV Year of B. Tech of CSE at **BVRIT HYDERABAD College of Engineering for Women,** Hyderabad is an authentic record of our original work carried out under the guidance of **Dr. Reya Sharma, Assistant Professor, Department of CSE.**

Sign with Date:
A. Tejaswini
(20WH1A05D8)

Sign with Date:
V.Sathwika
(20WH1A05D9)

Sign with Date:
P. Shinee
(20WH1A05E0)

Sign with Date:
G. Chandhana
(20WH1A0E1)

Sign with Date:
P. Pranathi
(20WH1A05E2)

ACKNOWLEDGEMENT

We would like to express our sincere thanks to **Dr. K. V. N. Sunitha**, **Principal, BVRIT HYDERABAD College of Engineering for Women,** for her support by providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr. R. Suneetha Rani, Head, Department of CSE, BVRIT HYDERABAD College of Engineering for Women,** for all timely support and valuable suggestions during the period of our project.

We are extremely thankful to our Internal Guide, **Dr. Reya Sharma, Assistant Professor, CSE, BVRIT HYDERABAD College of Engineering for Women,** for his constant guidance and encouragement throughout the project.

Finally, we would like to thank our Mini Project Coordinator, all Faculty and Staff of CSE department who helped us directly or indirectly. Last but not least, we wish to acknowledge our **Parents** and **Friends** for giving moral strength and constant encouragement.

A. Tejaswini (20WH1A05D8)

V. Sathwika (20WH1A05D9)

P. Shinee (20WH1A05E0)

G. Chandhana (20WH1A05E1)

P. Pranathi (20WH1A05E2)

ABSTRACT

Term Deposit Prediction addresses a critical challenge for a retail banking institution, where term deposits play a pivotal role in revenue generation. With a focus on telephonic marketing campaigns, this project develops a predictive system leveraging client and call data. By exploring, preprocessing, and visualizing relevant variables, a classification model is implemented using the sklearn library to predict client subscriptions to term deposits. Trained on the provided dataset, the model's performance is assessed with appropriate metrics. The objective is to provide the bank with an effective tool for identifying potential subscribers, optimizing telephonic marketing efforts, and achieving a more targeted and resource-efficient approach in promoting term deposits.

LIST OF FIGURES

CONTENTS

1. INTRODUCTION

1.1 Problem Statement:

The retail banking institution, heavily reliant on term deposits for income, seeks to optimize its outreach strategies, including telephonic marketing campaigns, which prove to be cost-intensive due to the operation of large call centers. The objective is to develop a predictive model leveraging client data (such as age, job type, marital status) and call information (including call duration, day, and month) to identify customers most likely to subscribe to term deposits. This predictive capability aims to enhance the efficiency of telephonic marketing efforts, allowing for targeted engagement with high-conversion potential clients, ultimately reducing costs and maximizing the return on investment in term deposit sales.

1.2 Objective:

To develop a precise predictive model that leverages client demographic data and call-related information to identify individuals with a higher propensity to subscribe to term deposits. By doing so, the bank aims to optimize telephonic marketing campaigns, strategically targeting specific customer segments, and thereby increasing conversion rates while minimizing costs. The overarching goal is to enhance the efficiency of marketing efforts, provide a more personalized experience for clients interested in term deposits, and ultimately maximize the return on investment by refining the overall marketing strategy based on the insights gained from the predictive model.

1.3 Proposed System:

In the proposed system, we introduce a technique called Stacking to enhance the accuracy of predicting whether a customer will subscribe to a term deposit. The existing system relies on training and testing data using various algorithms, but to further refine predictions, we implement Stacking, a model ensemble technique.

1.3.1 Ensemble Learning with Stacking:

We leverage the power of ensemble learning by combining predictions from multiple algorithms in a meta-model, known as a Stacker. This Stacker then produces the final prediction, enhancing the overall accuracy by capturing diverse insights from different base models.

1.3.2 Model Diversity:

To ensure the effectiveness of Stacking, we introduce diversity in the choice of base models. Different algorithms contribute unique perspectives, enabling the Stacker to learn and adapt to various patterns within the data, ultimately improving the predictive accuracy.

1.3.3 Improved Accuracy Rate:

The primary objective of introducing Stacking is to elevate the accuracy of predicting term deposit subscriptions beyond what is achievable with individual algorithms. By combining diverse models, we aim to capture a more nuanced understanding of customer behavior.

1.3.4 Robustness and Generalization:

Stacking enhances the robustness of the predictive model, making it more adaptable to different datasets and scenarios. This ensures that the system maintains high accuracy levels across varied customer profiles and market conditions.

2. REQUIREMENTS

2.1 Software Requirements:

- Windows 10
- Python 3.8
- Tkinter

2.2 Hardware Requirements:

- Intel Core i5 Processor
- RAM 8GB
- Hard Disk 256GB

3. DESIGN

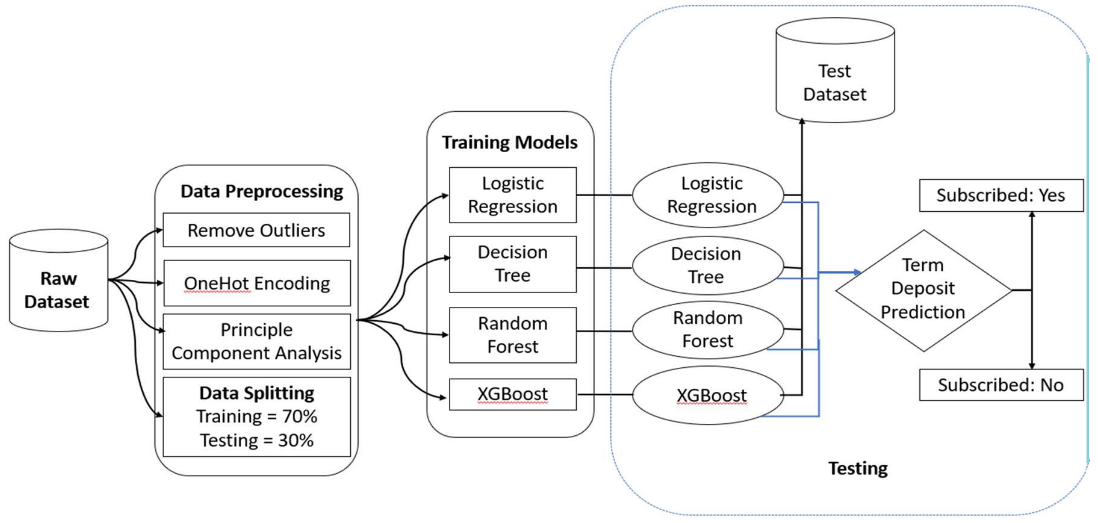


Fig 1: Workflow of Principle Component Analysis

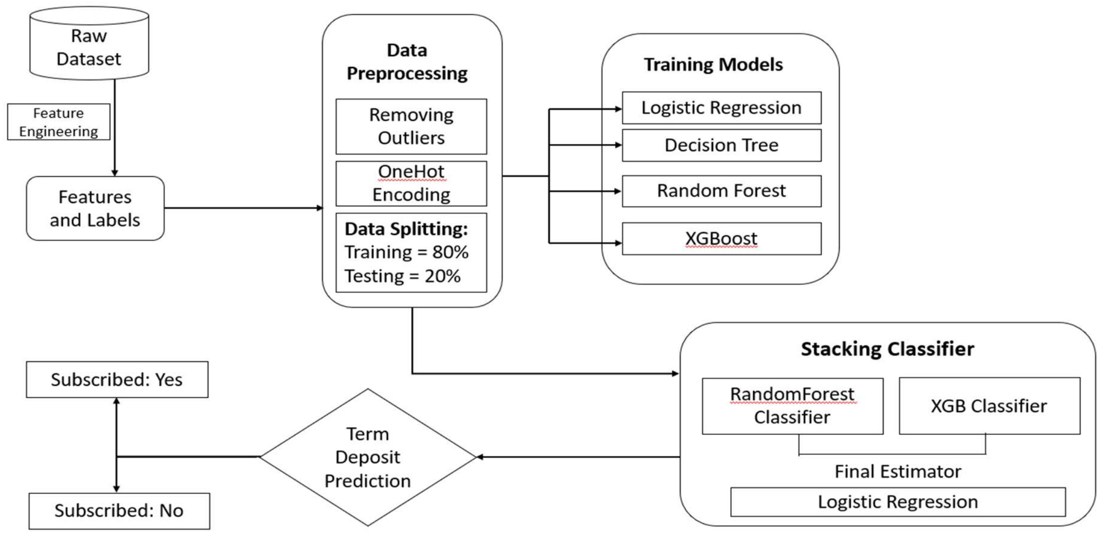


Fig 2: Workflow of Stacking Classifier

3.1 Data Exploration & Data Visualization:

Data exploration is the initial phase of the data analysis process, where analysts or data scientists examine and investigate a dataset to understand its characteristics, patterns, and underlying structure. The primary goal of data exploration is to gain insights into the data, identify trends, patterns, anomalies, and potential relationships between variables.

Data visualization is the graphical representation of data to uncover insights, patterns, and trends that may be less apparent in raw, tabular form. Visualization transforms data into visual elements such as charts, graphs, maps, and dashboards, allowing users to grasp concepts and detect patterns more easily than through textual or numerical representations.

3.1.1CatPlot:

Catplot, short for "categorical plot," is a versatile and powerful function in the Seaborn library for Python, commonly used for creating a variety of categorical plots. It is particularly useful for visualizing relationships between variables in categorical data.

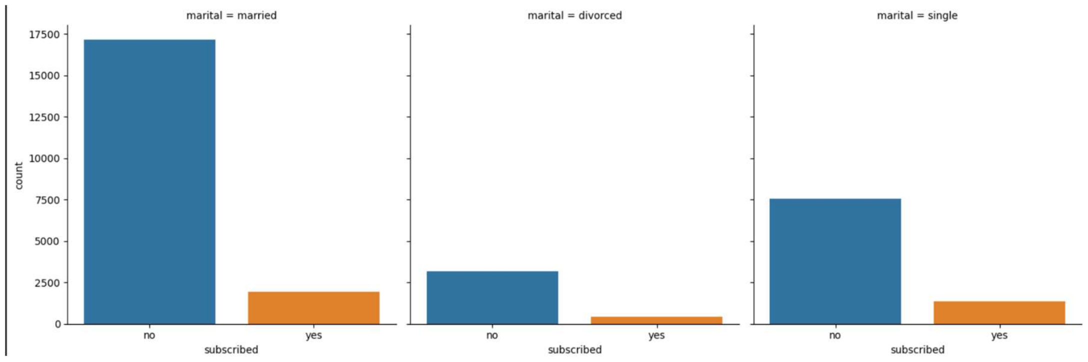


Fig 3: Relationship between categorical variable (Marital) and dependent variable (Subscription)

3.1.2 DistPlot:

The distplot function in the Seaborn library for Python is a powerful tool for visualizing the distribution of univariate data. It is specifically designed for examining the distribution of a single variable, making it a valuable tool for understanding the underlying characteristics of the data.

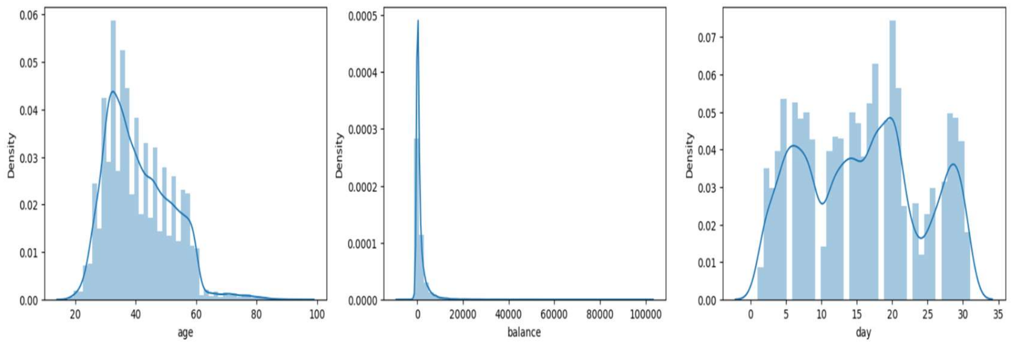


Fig 4: A univariate distribution of continues observations

3.1.3 BoxPlot:

Boxplots are effective at highlighting potential outliers in the data. Observations that fall beyond the whiskers may be considered as outliers, providing a quick and intuitive way to identify data points that deviate significantly from the overall pattern.

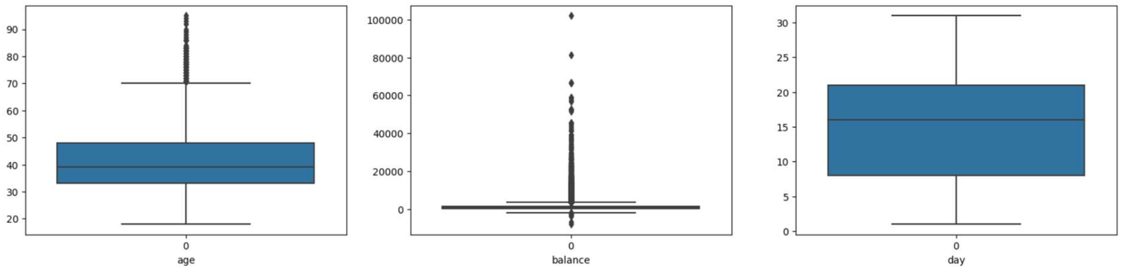


Fig 5: Boxplot on numerical features to find outliers

3.2 Data Preprocessing:

Data preprocessing is a crucial phase in the data analysis pipeline that involves cleaning, transforming, and organizing raw data into a format suitable for analysis. In the initial steps, copy of the original dataset, *train_bank_data*, was duplicated into *df2*.

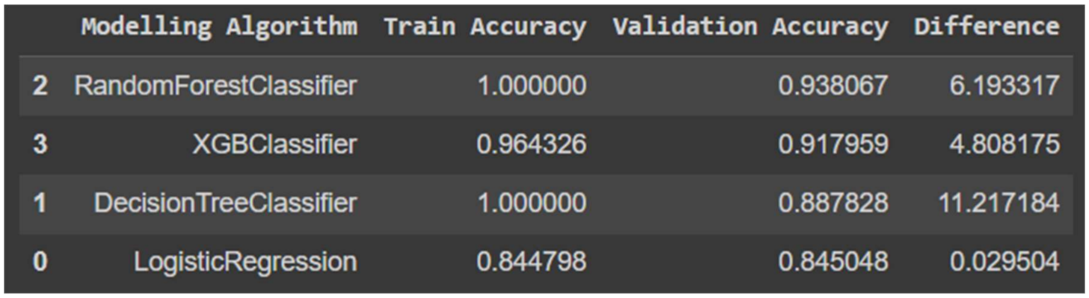
Irrelevant features, 'default' and 'pdays', were dropped, as their impact on the analysis was deemed negligible. Outliers in 'age', 'balance', 'duration', and 'campaign' were assessed and addressed by retaining values within reasonable ranges.

The dataset was refined to df4, incorporating one-hot encoding for categorical variables ('job', 'marital', 'education', 'contact', 'month', 'poutcome') and transforming Boolean variables ('housing', 'loan', 'subscribed'). After creating binary representations for boolean variables, unnecessary columns were dropped.

The dataset was split into training and testing sets using a ratio of 80:20. To enhance accuracy, we incorporated Principal Component Analysis (PCA) where categorical features were encoded, and numerical features were standardized.

Principal Component Analysis (PCA) reduced dimensionality to the top 25 components.

We have implemented Logistic Regression, Decision Tree, Random Forest, and XGBoost algorithms by using these selected 25 components and these models were trained and ranked based on validation accuracy.



	Modelling Algorithm	Train Accuracy	Validation Accuracy	Difference
2	RandomForestClassifier	1.000000	0.938067	6.193317
3	XGBClassifier	0.964326	0.917959	4.808175
1	DecisionTreeClassifier	1.000000	0.887828	11.217184
0	LogisticRegression	0.844798	0.845048	0.029504

Fig 6: Validation accuracies of trained models after implementing PCA

3.3 Building Machine Learning Models:

We implemented training for several models, including Logistic Regression, Decision Tree, Random Forest, and XGBoost.

3.3.1 Logistic Regression:

Logistic Regression, a fundamental binary classification algorithm, operates on the principle of linear modeling, predicting the probability of an instance belonging to a particular class. Known for its simplicity, efficiency, and scalability, Logistic Regression is widely employed across diverse domains, especially when the decision boundary is presumed to be linear.

We achieved an accuracy of 89% in Logistic Regression.

3.3.2 Decision Tree:

Decision Trees are powerful and interpretable models widely used for both classification and regression tasks. Operating on a tree-like structure, they recursively partition the dataset based on the most significant features, resulting in a clear decision-making process. The visual representation of a Decision Tree allows for intuitive understanding and communication of complex decision rules.

Our Decision Tree model exhibited a commendable accuracy of 87%.

3.3.3 Random Forest:

Random Forest is an ensemble learning method that excels in both classification and regression tasks, renowned for its robustness and high predictive accuracy. This algorithm operates by constructing a multitude of decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. Random Forest is capable of handling large datasets with high dimensionality and exhibits resilience to noisy data.

Random Forest yielded an accuracy rate of 90%.

3.3.4 XGBoost:

XGBoost, short for eXtreme Gradient Boosting, is a powerful and widely adopted ensemble learning algorithm known for its efficiency and exceptional performance in structured/tabular data scenarios. Based on the gradient boosting framework, XGBoost sequentially builds a series of decision trees, with each subsequent tree correcting errors made by the previous ones. Its key strengths include high predictive accuracy, scalability, and the ability to handle missing data effectively.

In the XGBoost model, we attained an accuracy level of 90%.

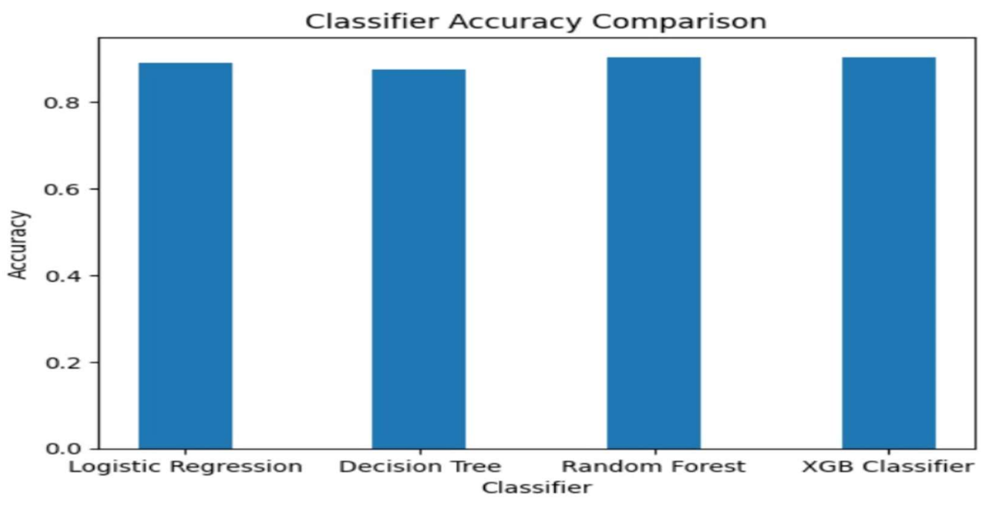


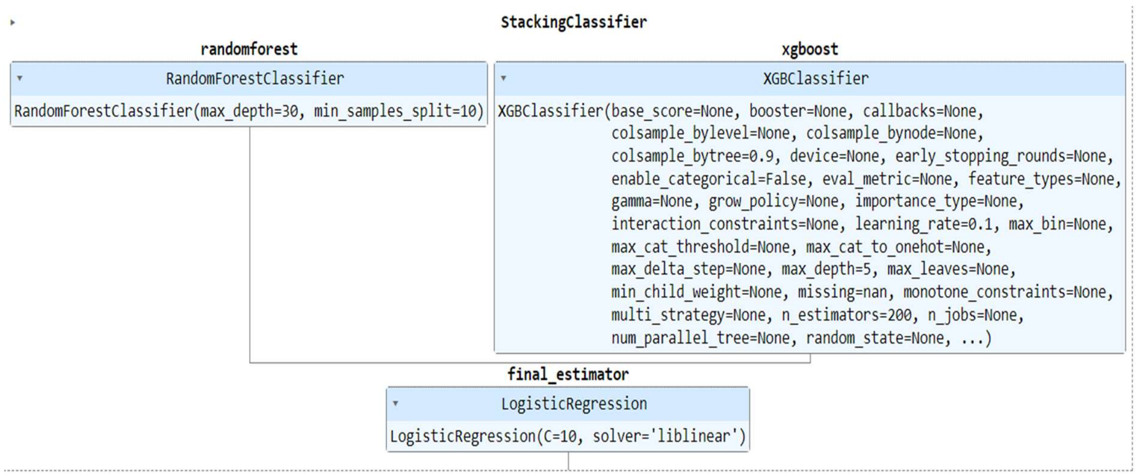
Fig 7: Accuracies of Trained models

3.3.5 Stacking

Improving accuracy was our goal, and we achieved it by incorporating the stacking technique.
Stacking, an ensemble learning technique, involves combining multiple individual models to create a meta-model that often outperforms its constituent models. The process consists of training a set of diverse base models on the same dataset and then using a higher-level model, known as the meta-model, to make predictions based on the outputs of these base models.

As part of the stacking classifier, we have gone through multiple Stacking combinations and at the combination of Random Forest and XGBoost were considered as base models, complemented by Logistic Regression as the meta-classifier.

Stacking contributed to an accuracy improvement, reaching 92%.



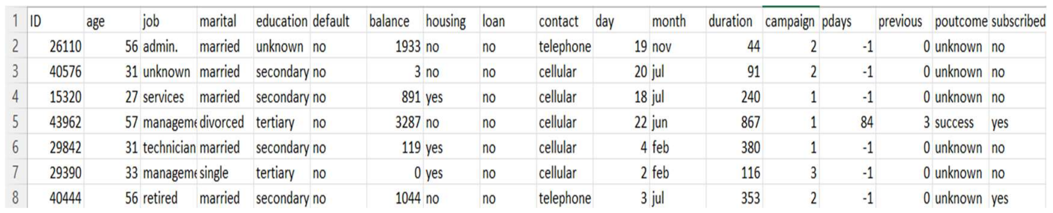
```
StackingClassifier
randomforest
RandomForestClassifier
RandomForestClassifier(max_depth=30, min_samples_split=10)
xgboost
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.9, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=200, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
final_estimator
LogisticRegression
LogisticRegression(C=10, solver='liblinear')
```

Fig 8: Stacking Model

4. IMPLEMENTATION

4.1 Dataset:

The dataset is taken from Kaggle which consists of 31647 rows and 18 columns.



1	ID	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	subscribed
2	26110	56	admin.	married	unknown	no	1933	no	no	telephone	19	nov	44	2	-1	0	unknown	no
3	40576	31	unknown	married	secondary	no	3	no	no	cellular	20	jul	91	2	-1	0	unknown	no
4	15320	27	services	married	secondary	no	891	yes	no	cellular	18	jul	240	1	-1	0	unknown	no
5	43962	57	manageme	divorced	tertiary	no	3287	no	no	cellular	22	jun	867	1	84	3	success	yes
6	29842	31	technician	married	secondary	no	119	yes	no	cellular	4	feb	380	1	-1	0	unknown	no
7	29390	33	manageme	single	tertiary	no	0	yes	no	cellular	2	feb	116	3	-1	0	unknown	no
8	40444	56	retired	married	secondary	no	1044	no	no	telephone	3	jul	353	2	-1	0	unknown	yes

Fig 9: Dataset Screenshot

4.2 ROC Curves:

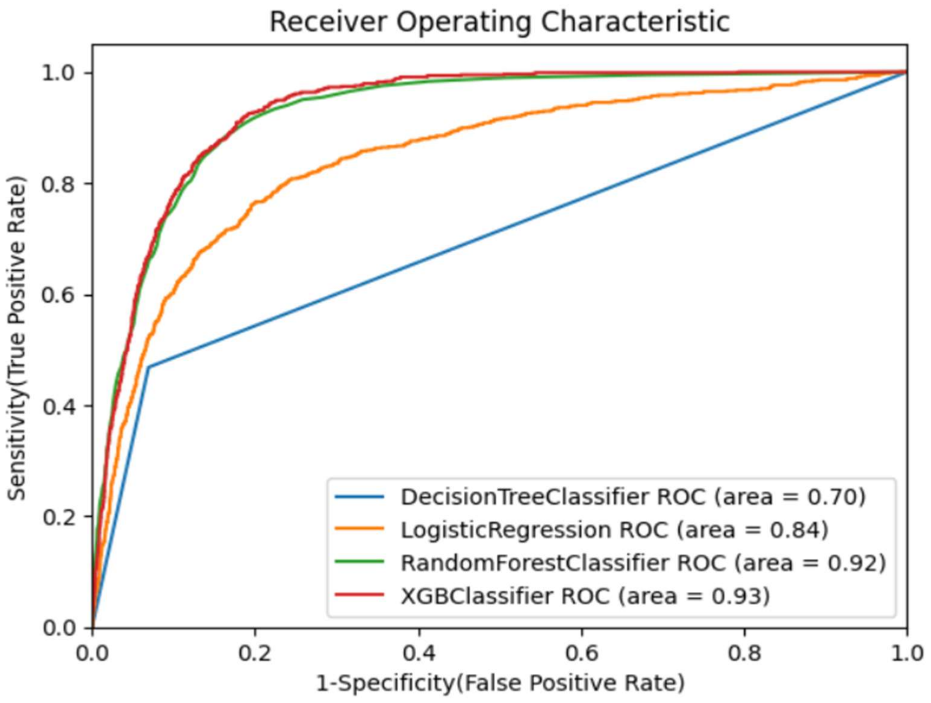
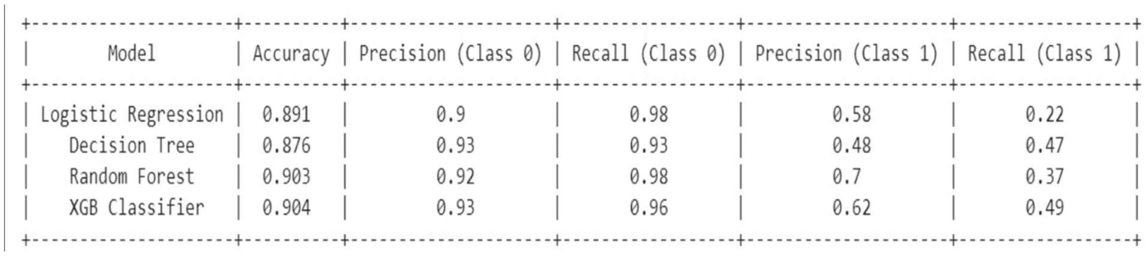


Fig 10: Roc Curves of Training models

4.3 Evaluation Metrices Comparison:



```
+--------------------+----------+--------------------+-----------------+--------------------+-----------------+
|       Model        | Accuracy | Precision (Class 0) | Recall (Class 0) | Precision (Class 1) | Recall (Class 1) |
+--------------------+----------+--------------------+-----------------+--------------------+-----------------+
| Logistic Regression | 0.891  |        0.9         |      0.98       |        0.58        |      0.22       |
|    Decision Tree   | 0.876   |        0.93        |      0.93       |        0.48        |      0.47       |
|    Random Forest   | 0.903   |        0.92        |      0.98       |        0.7         |      0.37       |
|    XGB Classifier  | 0.904   |        0.93        |      0.96       |        0.62        |      0.49       |
+--------------------+----------+--------------------+-----------------+--------------------+-----------------+
```

Fig 11: Table of Comparison

- **Accuracy:**
  It is a metric used to evaluate the performance of a classification model. It represents the ratio of correctly predicted instances to the total instances in a dataset.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total number of Predictions}}$$

- **Precision:**
  It is a metric used to assess the accuracy of positive predictions made by a classification model.

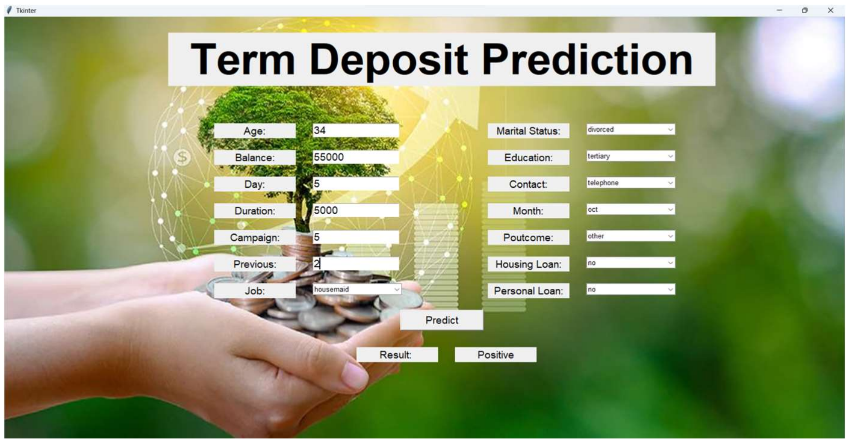
$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$

- **Recall:**
  It is a metric used to evaluate the ability of a classification model to capture all the relevant instances of a particular class.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

5. RESULTS

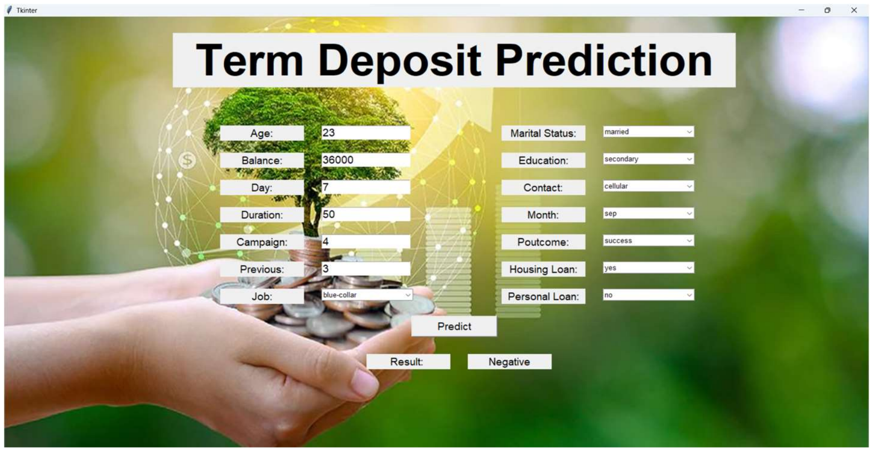
The project's output is a predictive model identifying customers likely to subscribe to term deposits. Analyzing client data and call details enables targeted telephonic marketing, optimizing outreach efforts and minimizing costs. This approach aims to boost term deposit subscriptions and enhance the effectiveness of the bank's marketing campaigns, ultimately contributing to increased revenue.



Age:	34	Marital Status:	divorced
Balance:	55000	Education:	tertiary
Day:	5	Contact:	telephone
Duration:	5000	Month:	oct
Campaign:	5	Poutcome:	other
Previous:	2	Housing Loan:	no
Job:	housemaid	Personal Loan:	no

Result: Positive

Fig 12: Positive Prediction for Term Deposit by a Client



Age:	23	Marital Status:	married
Balance:	36000	Education:	secondary
Day:	7	Contact:	cellular
Duration:	50	Month:	sep
Campaign:	4	Poutcome:	success
Previous:	3	Housing Loan:	yes
Job:	blue-collar	Personal Loan:	no

Result: Negative

Fig 13: Negative Prediction for Term Deposit by a Client

6. APPENDIX

6.1 Stacking Classifier:

```
import NumPy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

import seaborn as sns


train_bank_data = pd.read_csv('/content/train.csv')


train_bank_data.head()


train_bank_data.info()


# Describe numerical columns

train_bank_data.describe()


# Find missing values

features_na = [features for features in train_bank_data.columns if train_bank_data[features].isnull().sum() > 0]

for feature in features_na:

    print(feature, np.round(train_bank_data[feature].isnull().mean(), 4), ' % missing values')

else:
    print("No missing value found")


categorical_features=[feature for feature in train_bank_data.columns if ((train_bank_data[feature].dtypes=='object') & (feature not in ['subscribed']))]

categorical_features


#check count based on categorical features

plt.figure(figsize=(15,80), facecolor='white')

plotnumber =1

for categorical_feature in categorical_features:

    ax = plt.subplot(12,3,plotnumber)

    sns.countplot(y=categorical_feature,data=train_bank_data)

    plt.xlabel(categorical_feature)

    plt.title(categorical_feature)

    plotnumber+=1

plt.show()


#check target label split over categorical features

#Find out the relationship between categorical variable and dependent variable

for categorical_feature in categorical_features:

    sns.catplot(x='subscribed', col=categorical_feature, kind='count', data= train_bank_data)

plt.show()



#Check target label split over categorical features and find the count

for categorical_feature in categorical_features:

    print(train_bank_data.groupby(['subscribed',categorical_feature]).size())
```

```
train_bank_data = train_bank_data.drop('ID', axis=1)


# list of numerical variables

numerical_features = [feature for feature in train_bank_data.columns if ((train_bank_data[feature].dtypes != 'object') & (feature not in ['subscribed']))]

print('Number of numerical variables: ', len(numerical_features))

# visualise the numerical variables

train_bank_data[numerical_features].head()


discrete_feature=[feature for feature in numerical_features if len(train_bank_data[feature].unique())<25]

print("Discrete Variables Count: {}".format(len(discrete_feature)))


continuous_features=[feature for feature in numerical_features if feature not in discrete_feature+['subscribed']]

print("Continuous feature Count {}".format(len(continuous_features)))


#plot a univariate distribution of continues observations

plt.figure(figsize=(20,60), facecolor='white')

plotnumber =1

for continuous_feature in continuous_features:

    ax = plt.subplot(12,3,plotnumber)

    sns.distplot(train_bank_data[continuous_feature])

    plt.xlabel(continuous_feature)

    plotnumber+=1

plt.show()
```

#boxplot to show target distribution with respect numerical features

```
plt.figure(figsize=(20,60), facecolor='white')
plotnumber =1
for feature in continuous_features:
    ax = plt.subplot(12,3,plotnumber)
    sns.boxplot(x="subscribed", y= train_bank_data[feature], data=train_bank_data)
    plt.xlabel('subscribed')
    plotnumber+=1
plt.show()
```

#boxplot on numerical features to find outliers

```
plt.figure(figsize=(20,60), facecolor='white')
plotnumber =1
for numerical_feature in numerical_features:
    ax = plt.subplot(12,3,plotnumber)
    sns.boxplot(train_bank_data[numerical_feature])
    plt.xlabel(numerical_feature)
    plotnumber+=1
plt.show()
```

# Checking for correlation

```
cor_mat=train_bank_data.corr()
fig = plt.figure(figsize=(15,7))
sns.heatmap(cor_mat,annot=True)
```

# dependent variable distribution

```
sns.countplot(x='subscribed',data=train_bank_data)
plt.show()
```

```
train_bank_data['subscribed'].groupby(train_bank_data['subscribed']).count()
```

```
df2=train_bank_data.copy()
```

#defaut features does not play imp role

```
df2.groupby(['subscribed','default']).size()
```

```
df2.drop(['default'],axis=1, inplace=True)
df2.groupby(['subscribed','pdays']).size()
```

# drop pdays as it has -1 value for around 75%

```
df2.drop(['pdays'],axis=1, inplace=True)
```

# remove outliers in feature age

```
df2.groupby('age',sort=True)['age'].count()
# these can be ignored and values lies in between 18 to 95
```

# remove outliers in feature balance

```
df2.groupby(['subscribed','balance'],sort=True)['balance'].count()
```

# these outlier should not be remove as balance goes high, client show interest on term deposit

# remove outliers in feature duration

```
df2.groupby(['subscribed','duration'],sort=True)['duration'].count()
```

# these outlier should not be remove as duration goes high, client show interest on term deposit

# remove outliers in feature campaign

```
d=df2.groupby(['subscribed','campaign'],sort=True)['campaign'].count()

print(d[0:20])

print(d[20:60])


df3 = df2[df2['campaign'] < 41]


df3.groupby(['subscribed','campaign'],sort=True)['campaign'].count()
```

# remove outliers in feature previous

```
df3.groupby(['subscribed','previous'],sort=True)['previous'].count()


df4 = df3[df3['previous'] < 31]


cat_columns = ['job', 'marital', 'education', 'contact', 'month', 'poutcome']

for col in  cat_columns:

   df4 = pd.concat([df4.drop(col, axis=1),pd.get_dummies(df4[col], prefix=col,
prefix_sep='_',drop_first=False, dummy_na=False)], axis=1)

bool_columns = ['housing','loan', 'subscribed']

for col in  bool_columns:

   df4[col+'_new']=df4[col].apply(lambda x : 1 if x == 'yes' else 0)

bool_columns = ['housing','loan', 'subscribed']

for col in  bool_columns:

  df4.drop(col, axis=1, inplace=True)

df4.head()
```

#Spliting Dataset into Training set and Testing set

```
from sklearn.model_selection import train_test_split
```

```
X = df4.drop(['subscribed_new'],axis=1)

y = df4['subscribed_new']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,
random_state=123)
```

#Model Selection: Logistic Regression

```
from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import cross_val_score

from sklearn.metrics import roc_auc_score
,accuracy_score,classification_report,roc_curve,confusion_matrix


lmodel=LogisticRegression()

lmodel_score =cross_val_score(estimator=lmodel,X=X_train, y=y_train,
cv=5,scoring='accuracy')

print(lmodel_score)

print(lmodel_score.mean())
```

# Fit the model on the training set

```
lmodel.fit(X_train, y_train)
```

# Use the model to make predictions on X_test

```
y_pred = lmodel.predict(X_test)
```

# Calculate the testing accuracy

```
acc = accuracy_score(y_test, y_pred)

print(acc)
print('Classification Report:')

print(classification_report(y_test,y_pred))


print('Confusion matrix:')

print(confusion_matrix(y_test, y_pred))


#Model Selection: Decision Tree

from sklearn.tree import DecisionTreeClassifier

dmodel=DecisionTreeClassifier()

dmodel_score =cross_val_score(estimator=dmodel,X=X_train, y=y_train, cv=5)

print(dmodel_score)

print(dmodel_score.mean())


# Fit the model on the training set

dmodel.fit(X_train, y_train)


# Use the model to make predictions on X_test

y_pred = dmodel.predict(X_test)


# Calculate the testing accuracy

acc = accuracy_score(y_test, y_pred)

print(acc)


print('Classification Report:')

print(classification_report(y_test,y_pred))


print('Confusion matrix:')
print(confusion_matrix(y_test, y_pred))


#Model Selection: Random Forest
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score


rmodel=RandomForestClassifier()
rmodel_score =cross_val_score(estimator=rmodel,X=X_train, y=y_train, cv=5)
print(rmodel_score)
print(rmodel_score.mean())
# Fit the model on the training set
rmodel.fit(X_train, y_train)
# Use the model to make predictions on X_test
y_pred = rmodel.predict(X_test)


# Calculate the testing accuracy
acc = accuracy_score(y_test, y_pred)
print(acc)


print('Classification Report:')
print(classification_report(y_test,y_pred))
print('Confusion matrix:')
print(confusion_matrix(y_test, y_pred))
```

```
#Model Selection: XGBoost
from sklearn.model_selection import cross_val_score


xmodel = XGBClassifier()
xmodel_score =cross_val_score(estimator = xmodel, X = X_train, y = y_train, cv=5)
print(xmodel_score)
print(xmodel_score.mean())


# Fit the model on the training set
xmodel.fit(X_train, y_train)


# Use the model to make predictions on X_test
y_pred = xmodel.predict(X_test)


# Calculate the testing accuracy
acc = accuracy_score(y_test, y_pred)
print(acc)
print('Classification Report:')
print(classification_report(y_test,y_pred))


print('Confusion matrix:')
print(confusion_matrix(y_test, y_pred))


plt.bar(['Logistic Regression', 'Decision Tree', 'Random Forest', 'XGB Classifier'],
[lmodel_score.mean(),dmodel_score.mean(),rmodel_score.mean(),xmodel_score.mea
n()])
plt.xlabel('Classifier')
plt.ylabel('Accuracy')
plt.title('Classifier Accuracy Comparison')

plt.show()


print("Logistic Regression accuracy:", lmodel_score.mean())

print("Decision tree accuracy:", dmodel_score.mean())

print("Random Forest accuracy:", rmodel_score.mean())

print("XGB Classifier accuracy:", xmodel_score.mean())
```

#Model Building

```
from sklearn.ensemble import StackingClassifier

from sklearn.metrics import accuracy_score,classification_report

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

rf_model = RandomForestClassifier(n_estimators=100,
min_samples_split=10,min_samples_leaf=1,max_depth=30)

xgb_model1 =
XGBClassifier(subsample=0.9,colsample_bytree=0.9,learning_rate=0.1 ,
max_depth=5, n_estimators=200)

meta_classifier = LogisticRegression(solver= 'liblinear', penalty = 'l2', C= 10)
```

# Define the stacking model

```
stacking_model = StackingClassifier(

    estimators=[

        ('randomforest', rf_model),

        ('xgboost', xgb_model1),

    ],

    final_estimator=meta_classifier

)
```

```
# Train the stacking model
stacking_model.fit(X_train, y_train)


# Make predictions
y_pred = stacking_model.predict(X_test)


# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")


print(classification_report(y_test,y_pred))
```

6.2 Principal Component Analysis:

```
import pandas as pd

import NumPy as np

import seaborn as sns

import matplotlib.pyplot as plt

import statsmodels.api as sm

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model_selection import train_test_split, RandomizedSearchCV

from sklearn.decomposition import PCA

from imblearn.over_sampling import SMOTE

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from xgboost import XGBClassifier

from sklearn.ensemble import StackingClassifier

%matplotlib inline

pd.pandas.set_option('display.max_columns', None)


df_train = pd.read_csv('/content/train.csv')

df_test = pd.read_csv('/content/test.csv')
```

```
df_train.info()

df_train.head()


df_train_dummies = pd.get_dummies(df_train[['job', 'marital', 'default', 'housing',
'loan', 'contact', 'poutcome']], drop_first = True)

df_train_label = df_train[['education', 'month']].apply(LabelEncoder().fit_transform)

df_test_dummies = pd.get_dummies(df_test[['job', 'marital', 'default', 'housing', 'loan',
'contact', 'poutcome']], drop_first = True)

df_test_label = df_test[['education', 'month']].apply(LabelEncoder().fit_transform)

df_train = pd.concat([df_train.drop(['job', 'marital', 'default', 'housing', 'loan', 'contact',
'poutcome', 'education', 'month'], axis = 1), df_train_dummies, df_train_label], axis =
1)

df_test = pd.concat([df_test.drop(['job', 'marital', 'default', 'housing', 'loan', 'contact',
'poutcome', 'education', 'month'], axis = 1), df_test_dummies, df_test_label], axis = 1)


df_train['subscribed'] = df_train['subscribed'].map({'yes': 1, 'no': 0})


df_train.head()


df_train_scaled                                                                          =
pd.DataFrame(StandardScaler().fit_transform(df_train.drop('subscribed', axis = 1)),
columns = df_test.columns)

df_test_scaled = pd.DataFrame(StandardScaler().fit_transform(df_test), columns =
df_test.columns)


df_train_scaled.head()


pca_columns = []

for i in range(df_train_scaled.shape[1]):

    pca_columns.append('PC' + str(i+1))
```

```
pca_model = PCA()

pca_model.fit(df_train_scaled)

df_pca_train = pd.DataFrame(pca_model.transform(df_train_scaled), columns = pca_columns)

explained_info_train = pd.DataFrame(pca_model.explained_variance_ratio_, columns=['Explained Info']).sort_values(by = 'Explained Info', ascending = False)

imp = []

for i in range(explained_info_train.shape[0]):

    imp.append(explained_info_train.head(i).sum())

explained_info_train_sum = pd.DataFrame()

explained_info_train_sum['Variable'] = pca_columns

explained_info_train_sum['Importance'] = imp

explained_info_train_sum


pca_columns = []

for i in range(25):

    pca_columns.append('PC' + str(i+1))

pca_model = PCA(n_components = 25)

pca_model.fit(df_train_scaled)

df_pca_train = pd.DataFrame(pca_model.transform(df_train_scaled), columns = pca_columns)


df_pca_train.head()


df=df_train.drop(columns=['subscribed'],axis=1)

df


pca_model = PCA(n_components = 25)
```

```
pca_model.fit(df_test_scaled)

df_pca_test = pd.DataFrame(pca_model.transform(df_test_scaled), columns = pca_columns)

X = df_pca_train

y = df_train['subscribed']

X.head()


smote = SMOTE()

X_smote, y_smote = smote.fit_resample(X, y)

y_smote.value_counts()


X_train, X_val, y_train, y_val = train_test_split(X_smote, y_smote, test_size = 0.3, random_state = 17)

X_test = df_pca_test


models = [LogisticRegression(), DecisionTreeClassifier(), RandomForestClassifier(), XGBClassifier()]

model_names = ['LogisticRegression', 'DecisionTreeClassifier', 'RandomForestClassifier', 'XGBClassifier']

accuracy_train = []

accuracy_val = []

for model in models:

    mod = model

    mod.fit(X_train, y_train)

    y_pred_train = mod.predict(X_train)

    y_pred_val = mod.predict(X_val)

    accuracy_train.append(accuracy_score(y_train, y_pred_train))

    accuracy_val.append(accuracy_score(y_val, y_pred_val))
```

```
data = {'Modelling Algorithm' : model_names, 'Train Accuracy' : accuracy_train,
'Validation Accuracy' : accuracy_val}

data = pd.DataFrame(data)

data['Difference'] = ((np.abs(data['Train Accuracy'] - data['Validation Accuracy'])) *
100)/(data['Train Accuracy'])

data.sort_values(by = 'Validation Accuracy', ascending = False)
```

7. CONCLUSION & FUTURE SCOPE

In this analysis, we implemented preprocessing techniques like one-hot encoding, label encoding, and standard scaling for a binary classification task. PCA and SMOTE were used for dimensionality reduction and handling class imbalance, respectively. The Random Forest classifier achieved the highest accuracy of 93% after PCA.

The stacked ensemble with Logistic Regression as the meta-classifier achieved an impressive 92% accuracy on the test set, showcasing the efficacy of blending traditional and ensemble methods for robust predictive modeling.

The analysis explored a banking dataset, addressing missing values, exploring feature distributions, and studying the impact of categorical features. Preprocessing included one-hot encoding, label encoding, and outlier removal. Individual model performance and subsequent stacking showcased superior accuracy.

Looking forward, optimization of hyperparameters, exploring advanced ensemble techniques, and investigating additional features or external datasets could enhance predictive performance. Continuous monitoring and periodic retraining are recommended for adaptability to evolving data, deployment in real-world scenarios, and iterative refinement based on feedback.

8. REFERENCES

[1] Hayder, Israa M., Ghazwan Abdul Nabi Al Ali, and Hussain A. Younis. "Predicting reaction based on customer’s transaction using machine learning approaches. “International Journal of Electrical and Computer Engineering 13, no. 1 (2023): 1086.

[2] Hou, Sipu, Zongzhen Cai, Jiming Wu, Hongwei Du, and Peng Xie. "Applying Machine Learning to the Development of Prediction Models for Bank Deposit Subscription. “International Journal of Business Analytics (IJBAN) 9, no. 1 (2022): 1-14.

[3] Borugadda, Premkumar, Prabhakar Nandru, and Chendragiri Madhavaiah. "Predicting the success of bank telemarketing for selling long-term deposits: An application of machine learning algorithms." St. Theresa Journal of Humanities and Social Sciences 7, no. 1 (2021): 91-108.

[4] Singh, M., Dhanda, N., Farooqui, U.K., Gupta, K.K. and Verma, R., 2023, March. Prediction of Client Term Deposit Subscription Using Machine Learning. In *International Conference on Communication, Devices and Computing* (pp. 83-93). Singapore: Springer Nature Singapore.

[5] Muslim, M.A., Dasril, Y., Alamsyah, A. and Mustaqim, T., 2021, June. Bank predictions for prospective long-term deposit investors using machine learning LightGBM and SMOTE. In *Journal of Physics: Conference Series* (Vol. 1918, No. 4, p. 042143). IOP Publishing.