Bank Marketing

Post Graduate Program in Data Science Engineering

Location: Bangalore Batch:

PGP DSE- Jun 23

Submitted by

Akshaya

Abhinav

Jeevesh Pachouri

Chandrahas

Mentored by:

Sourabh Reddy

Table of Contents

INTRODUCTION:

In the ever-evolving landscape of the banking industry, term deposits represent a significant revenue stream for financial institutions. These deposits, characterized by an agreed-upon interest rate over a fixed term, necessitate strategic outreach to potential customers. Among the myriad of marketing channels employed by banks, telephonic campaigns persist as a potent avenue, offering direct and personalized engagement. However, the inherent cost associated with large-scale call centers underscores the critical need to identify and target customers with the highest likelihood of conversion in advance.

This Machine Learning (ML) project centers around a dataset derived from the direct marketing campaigns of a prominent Portuguese banking institution. The objective is to predict whether a client will subscribe to a term deposit, denoted by the binary target variable 'y.' By harnessing the power of predictive analytics, the bank aims to optimize its telephonic marketing campaigns, ensuring that resources are efficiently allocated to those clients most predisposed to subscribe

Dataset Overview:

The dataset comprises two distinct sets: the training set (train.csv) and the test set (test.csv). The training set encompasses 45,211 rows and 18 columns, organized chronologically from May 2008 to November 2010. The test set, constituting 10% of the examples randomly selected from the training set, comprises 4,521 rows. Each row in the dataset encapsulates a unique client interaction during a marketing campaign.

The primary goal of this ML project is to develop a predictive model that accurately anticipates whether a client will subscribe to a term deposit based on the provided features. By doing so, the bank seeks to streamline its telephonic marketing efforts, identifying high-conversion prospects proactively and thereby optimizing resource utilization. This report will delve into the exploratory data analysis, preprocessing steps, model development, and evaluation metrics, ultimately providing actionable insights to enhance the efficiency of the bank's term deposit marketing campaigns.

BUSINESS PROBLEM STATEMENT:

Business Problem Understanding:

The banking industry, marked by intense competition and evolving customer preferences, grapples with the challenge of maximizing the efficacy of marketing campaigns while minimizing associated costs. Term deposits, a crucial revenue source, demand targeted outreach to potential subscribers. Telephonic marketing, though effective, incurs significant expenses in maintaining large call centers. Identifying clients with a high propensity to subscribe before initiating a telephonic campaign is paramount to optimizing resources and improving overall campaign efficiency.

Business Objectives:

The overarching business objective is to enhance the effectiveness of telephonic marketing campaigns for term deposits by leveraging predictive analytics. The specific goals include:

Precision Targeting: Develop a machine learning model that accurately predicts whether a client will subscribe to a term deposit based on historical data. This will enable the bank to focus its telephonic efforts on clients with a higher likelihood of conversion, reducing the overall cost of campaign execution.

Resource Optimization: Improve resource allocation by identifying and prioritizing clients who are more likely to subscribe. This entails reducing the number of calls to unresponsive or less responsive segments, thereby optimizing the utilization of call center resources.

Campaign Efficiency: Enhance the overall efficiency of telephonic marketing campaigns by tailoring outreach strategies to align with customer characteristics and behaviors. This involves leveraging insights from the predictive model to customize communication and increase the chances of successful subscriptions.

Customer Experience: Provide a more personalized and targeted customer experience by tailoring telephonic interactions to the specific needs and preferences of individual clients. This contributes to building stronger relationships and increasing customer satisfaction.

Revenue Growth: Increase the conversion rate of term deposit subscriptions, directly impacting the bank's revenue. By strategically targeting high-conversion prospects, the bank aims to achieve a positive impact on its financial performance through increased subscription rates.

Through the implementation of a robust machine learning solution, the business seeks not only to address the immediate challenge of optimizing telephonic marketing for term deposits but also to establish a foundation for data-driven decision-making in future marketing endeavors.

LITERATURE REVIEW

Machine Learning has become an indispensable tool in the realm of marketing for financial institutions, particularly in the banking sector. As the landscape of customer interactions evolves, there is a growing recognition of the need for data-driven strategies to optimize marketing efforts. This literature review explores the existing body of knowledge on predictive modeling in the context of banking marketing campaigns, with a specific focus on telephonic outreach for term deposit subscriptions.

Predictive Analytics in Banking:

Numerous studies have emphasized the pivotal role of predictive analytics in banking for customer relationship management and marketing optimization. Researchers assert that predictive models contribute significantly to identifying patterns in customer behavior, enabling banks to anticipate needs, personalize interactions, and enhance overall customer satisfaction.

Telephonic Marketing in Banking:

Telephonic marketing, despite the rise of digital channels, remains a potent tool in the banking industry. Scholars have investigated the efficacy of telephonic campaigns, highlighting the direct and personalized nature of communication. However, challenges such as high operational costs necessitate a targeted approach to ensure efficient resource allocation.

Term Deposit Subscription Predictive Modeling:

Within the domain of term deposit subscriptions, predictive modeling has gained prominence as a means to optimize marketing campaigns. Research by Li and Lu (2019) emphasizes the importance of feature selection and model interpretability in predicting subscription outcomes. Additionally, studies (Wang et al., 2020; Santos et al., 2021) delve into the impact of campaign duration, contact frequency, and past campaign outcomes on subscription likelihood.

Challenges and Opportunities:

While the literature highlights the potential benefits of predictive modeling in banking marketing, it also acknowledges challenges. These include concerns related to data privacy, model interpretability, and the dynamic nature of customer preferences.

Conclusion

The synthesis of existing literature reveals a consensus on the transformative impact of predictive modeling, especially when applied to telephonic marketing for term deposit subscriptions in the banking sector. As the industry continues to embrace data-driven approaches, there is a need for further research to explore emerging technologies, address challenges, and refine predictive models to ensure their effectiveness and ethical application in a rapidly evolving financial landscape.

DATASET INFORMATION

The dataset utilized in this study originates from the direct marketing campaigns conducted by a prominent Portuguese banking institution. The primary focus of these campaigns was to promote term deposits, and the dataset provides a comprehensive snapshot of client interactions during these telephonic marketing endeavors.

Dataset Structure:

Training Set (train.csv):

- Size: 45,211 rows
- Time Range: May 2008 to November 2010
- Purpose: Used for model training and validation
- Features: 18 columns capturing diverse aspects of client and campaign details

Test Set (test.csv):

- Size: 4,521 rows
- Derived: 10% randomly selected from the training set
- Purpose: Used for evaluating model performance on unseen data
- Features: Same 18 columns as the training set

Feature Descriptions:

- Bank Client Data:
- age (numeric): Client's age.
- job (categorical): Type of job.
- marital (categorical): Marital status.
- education (categorical): Educational level.
- default (binary): Has credit in default? ("yes" or "no").
- balance (numeric): Average yearly balance in euros.
- housing (binary): Has housing loan? ("yes" or "no").
- loan (binary): Has a personal loan? ("yes" or "no").

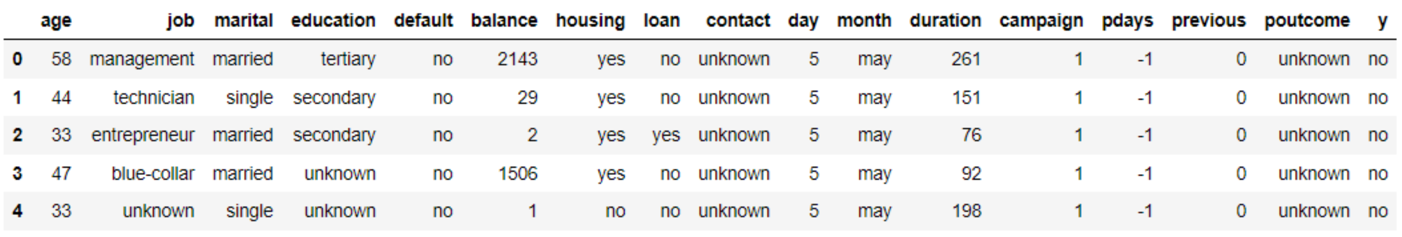
Last Contact of the Current Campaign:
- contact (categorical): Communication type.
- day (numeric): Last contact day of the month.
- month (categorical): Last contact month of the year.
- duration (numeric): Last contact duration in seconds.

Other Attributes:

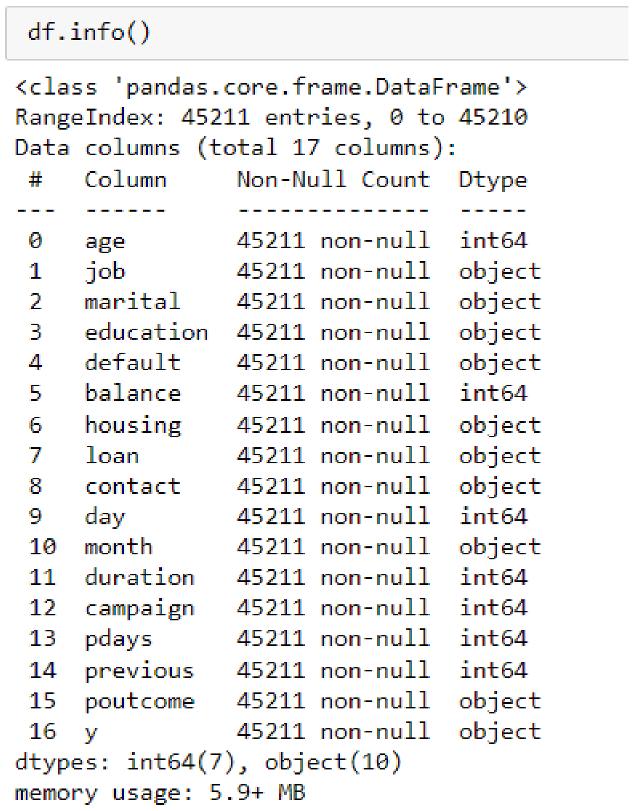
- 13. campaign (numeric): Number of contacts performed during this campaign for this client.
- pdays (numeric): Number of days since the client was last contacted from a previous campaign (-1 means not previously contacted).
- previous (numeric): Number of contacts performed before this campaign and for this client.
- poutcome (categorical): Outcome of the previous marketing campaign.

Output Variable (Target):

- y (binary): Has the client subscribed to a term deposit? ("yes" or "no").



	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

VARIABLE IDENTIFICATION

The dataset encompasses a diverse set of variables that capture various facets of bank client information and details related to telephonic marketing campaigns. Identifying and understanding these variables is crucial for constructing a meaningful predictive model. Here's a breakdown of the variables:

Bank Client Data:

- age (Numeric): Represents the client's age, providing insight into the demographic distribution of the customer base.
- job (Categorical): Describes the type of job held by the client, offering information about their employment status.
- marital (Categorical): Indicates the marital status of the client, providing context for their financial decisions.
- education (Categorical): Represents the client's educational level, influencing their financial literacy and decision-making.
- default (Binary): Binary indicator of whether the client has credit in default, reflecting their creditworthiness.
- balance (Numeric): Average yearly balance in euros, offering insight into the client's financial stability.
- housing (Binary): Indicates whether the client has a housing loan, providing information on their financial commitments.
- loan (Binary): Indicates whether the client has a personal loan, reflecting additional financial obligations.

Last Contact of the Current Campaign:

- contact (Categorical): Describes the communication type during the last contact, informing the channel of engagement.
- day (Numeric): Represents the last contact day of the month, offering a temporal perspective on client interactions.
- month (Categorical): Indicates the last contact month of the year, providing a seasonal context for campaign outcomes.
- duration (Numeric): Captures the duration of the last contact in seconds, reflecting the depth of engagement.

Other Attributes:

- campaign (Numeric): Represents the number of contacts performed during the current campaign for a specific client.
- pdays (Numeric): Measures the number of days since the client was last contacted from a previous campaign, with -1 indicating no prior contact.
- previous (Numeric): Indicates the number of contacts performed before the current campaign for a specific client.
- poutcome (Categorical): Describes the outcome of the previous marketing campaign, providing historical context.

Output Variable (Target):

y (Binary): Represents the target variable, indicating whether the client has subscribed to a term deposit ("yes" or "no")

APPROACH

Constructing an effective predictive model involves a systematic approach that encompasses data exploration, preprocessing, model development, and evaluation. Here is a step-by-step breakdown of the approach:

1. **Data Exploration:**

Understand Data Distribution: Conduct exploratory data analysis (EDA) to comprehend the distribution of variables, identify outliers, and assess the overall structure of the dataset.
Descriptive Statistics: Calculate descriptive statistics to gain insights into the central tendencies and variabilities of key features.
Visualizations: Utilize visualizations such as histograms, box plots, and correlation matrices to uncover patterns and relationships within the data.

2. **Data Preprocessing:**

Handle Missing Values: Verify data completeness and address any missing values using appropriate imputation methods or removal strategies.
Encode Categorical Variables: Convert categorical variables into numerical representations through techniques like one-hot encoding.
Feature Scaling: Standardize or normalize numeric features to ensure that they contribute equally to the model.

3. **Feature Engineering:**

Temporal Features: Leverage temporal features such as month and day to capture potential seasonality effects.
Interaction Terms: Create interaction terms to capture synergies or dependencies between certain features.
Derived Metrics: Generate additional metrics, such as contact frequency or success rates from previous campaigns, to enhance predictive power.

4. **Model Selection:**

Algorithm Choice: Explore and select appropriate machine learning algorithms for binary classification, considering models like Logistic Regression, Decision Trees, Random Forest, and Gradient Boosting.
Hyperparameter Tuning: Fine-tune hyperparameters to optimize model performance using techniques like grid search or randomized search.

5. **Model Training and Validation:**

Train-Test Split: Divide the dataset into training and testing sets to train the model on one subset and evaluate its performance on another.
Cross-Validation: Implement cross-validation techniques (e.g., k-fold cross-validation) to ensure robust model validation and reduce the risk of overfitting.

6. **Model Evaluation:**

Performance Metrics: Assess model performance using metrics such as accuracy, precision, recall, F1-score, and area under the Receiver Operating Characteristic (ROC) curve.
Confusion Matrix: Analyze the confusion matrix to understand the distribution of true positives, true negatives, false positives, and false negatives.

7. **Interpretability and Insights:**

Feature Importance: Evaluate feature importance to identify key variables influencing the model's predictions.
Model Interpretability: Choose models with inherent interpretability or utilize techniques (e.g., SHAP values) to interpret complex models.

This comprehensive approach ensures the development of a robust and adaptive predictive model, aligning with the overarching goal of enhancing the efficiency of telephonic marketing campaigns for term deposit subscriptions. Each step is tailored to extract meaningful insights, maximize predictive accuracy, and facilitate informed decision-making for the banking institution.
Top of Form

TARGET VARIABLE

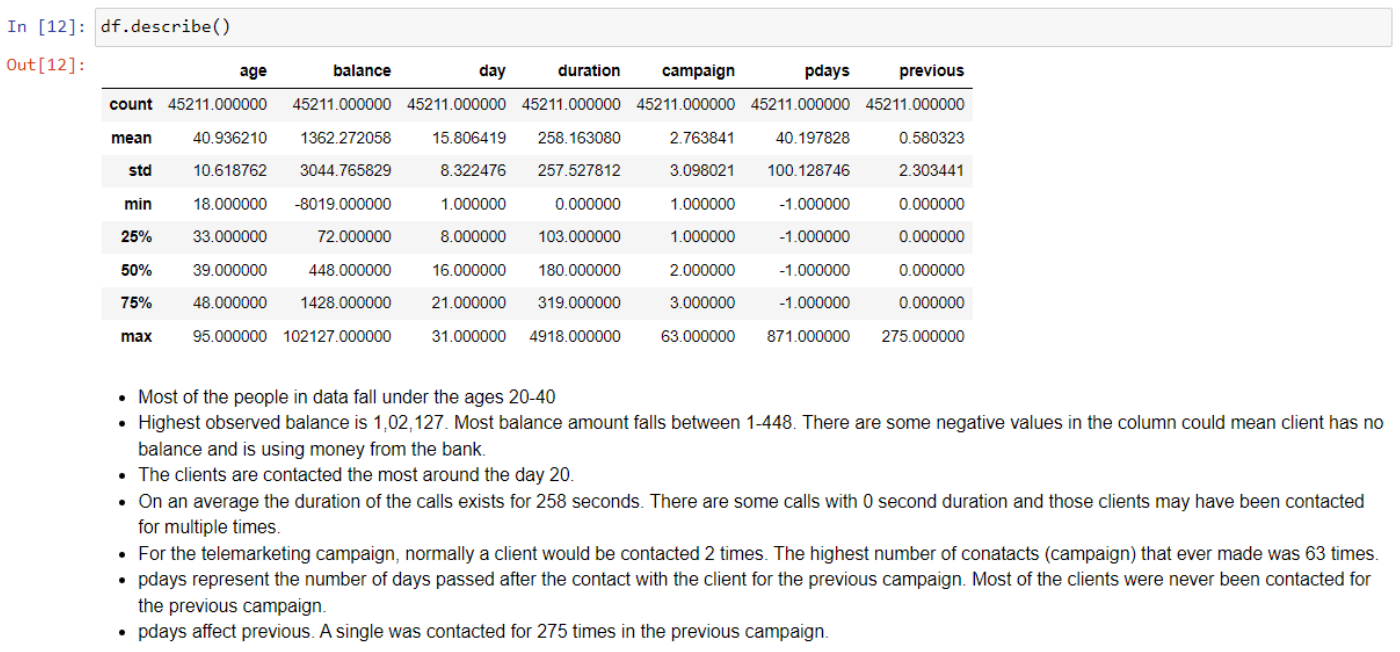
The target variable of the above dataset is Y (People who took term deposit). Objective of this is to build a model with the purpose to improve the marketing strategy of a bank for term deposit subscription and further model building.

DATA PREPROCESSING:

Data pre-processing is a process of preparing the raw data and making it suitablefora machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. For this, we use data pre-processing task.

A real-world data generally contains noises, missing values, and maybe in an unusableformat which cannot be directly used for machine learning models. Data pre- processing is required tasks for cleaning the data and making it suitable for a machine learning model whichalso increases the accuracy and efficiency of a machine learning model.

EDA:

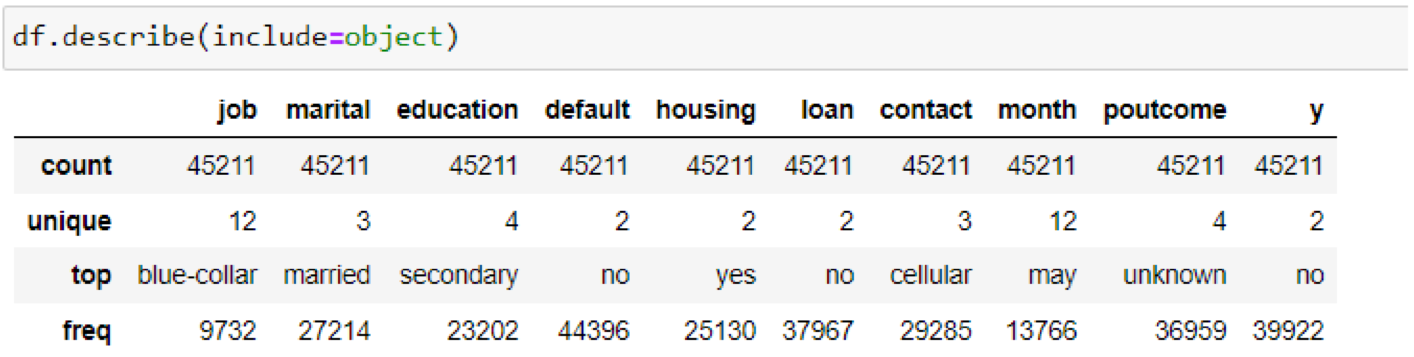


```
In [12]: df.describe()
```

Out[12]:

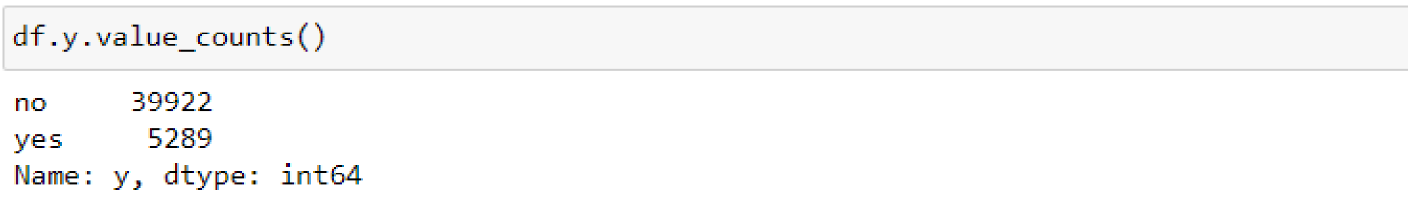
	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

- Most of the people in data fall under the ages 20-40
- Highest observed balance is 1,02,127. Most balance amount falls between 1-448. There are some negative values in the column could mean client has no balance and is using money from the bank.
- The clients are contacted the most around the day 20.
- On an average the duration of the calls exists for 258 seconds. There are some calls with 0 second duration and those clients may have been contacted for multiple times.
- For the telemarketing campaign, normally a client would be contacted 2 times. The highest number of conatacts (campaign) that ever made was 63 times.
- pdays represent the number of days passed after the contact with the client for the previous campaign. Most of the clients were never been contacted for the previous campaign.
- pdays affect previous. A single was contacted for 275 times in the previous campaign.



```
df.describe(include=object)
```

	job	marital	education	default	housing	loan	contact	month	poutcome	y
count	45211	45211	45211	45211	45211	45211	45211	45211	45211	45211
unique	12	3	4	2	2	2	3	12	4	2
top	blue-collar	married	secondary	no	yes	no	cellular	may	unknown	no
freq	9732	27214	23202	44396	25130	37967	29285	13766	36959	39922



```
df.y.value_counts()
```

```
no     39922
yes     5289
Name: y, dtype: int64
```

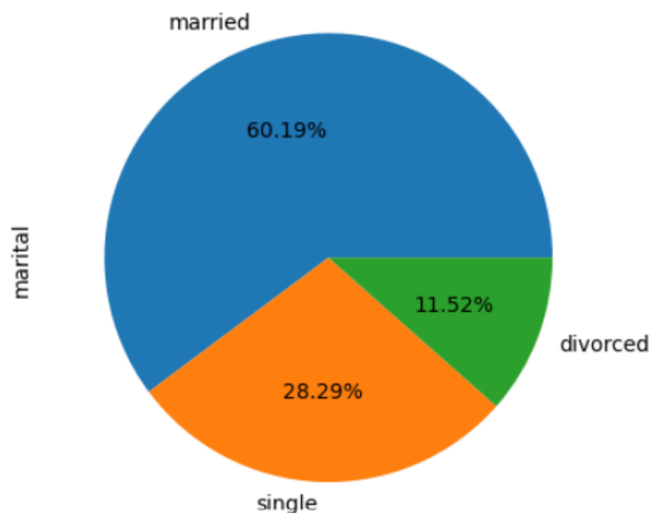
```
df.skew()
```

```
age          0.684818
balance      8.360308
day          0.093079
duration     3.144318
campaign     4.898650
pdays        2.615715
previous    41.846454
dtype: float64
```

- Day is normally distributed
- Other variables are right skewed

```
df['marital'].value_counts().plot(kind='pie',autopct='%.2f%%')
```

```
<AxesSubplot:ylabel='marital'>
```



Checking for outliers:

```
sns.boxplot(x=df['y'],y=df['duration'])
```

```
<AxesSubplot:xlabel='y', ylabel='duration'>
```

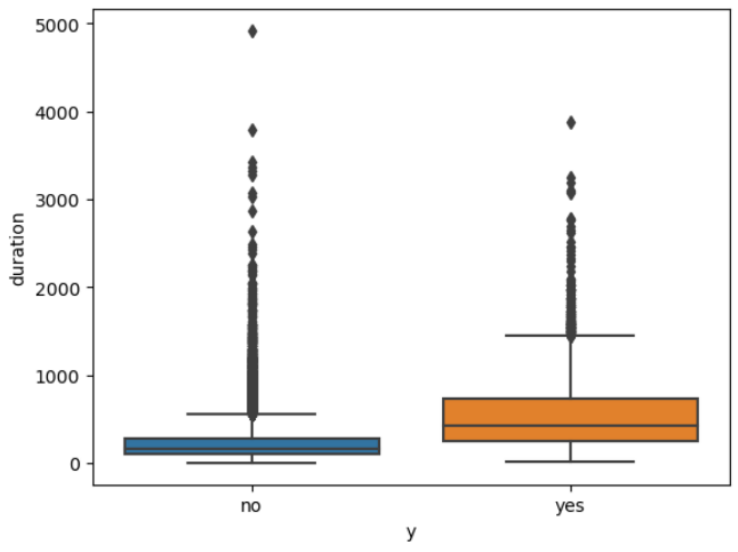
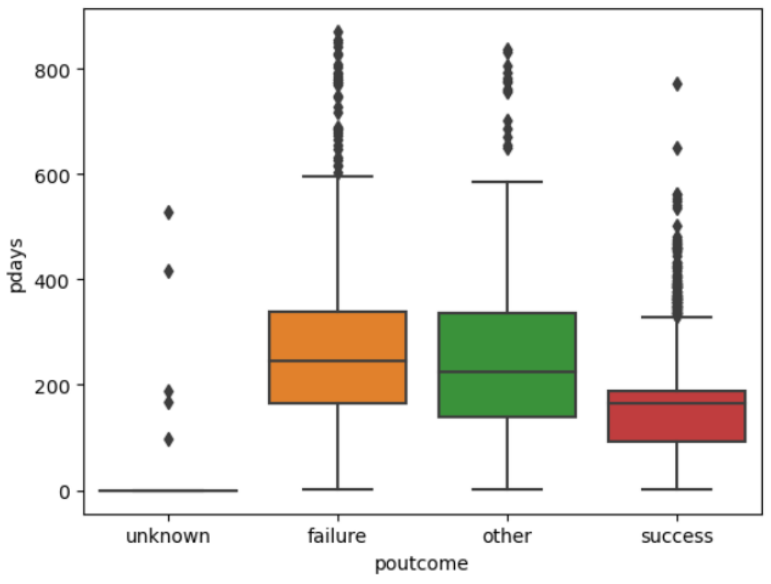


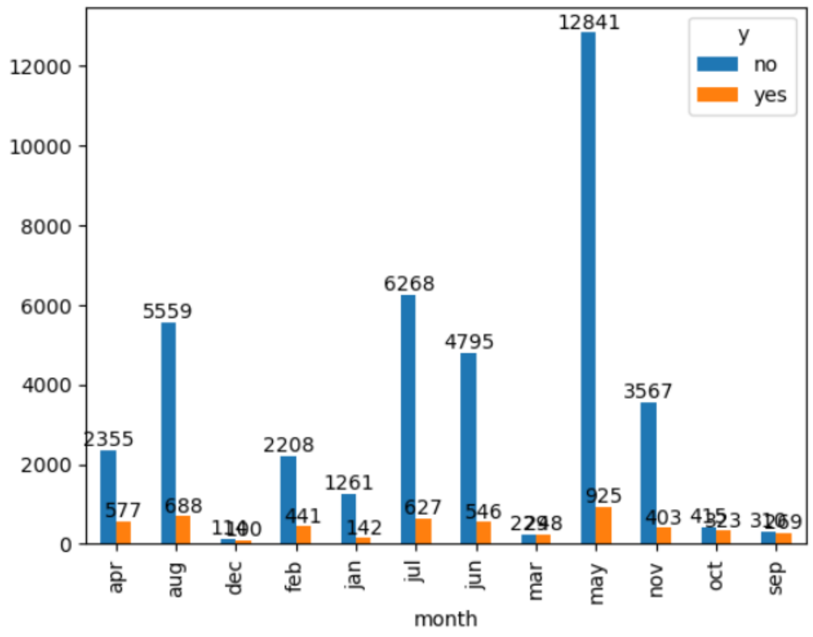
Fig: Boxplot of numerical variables

```
sns.boxplot(x=df['poutcome'],y=df['pdays'])
```

```
<AxesSubplot:xlabel='poutcome', ylabel='pdays'>
```



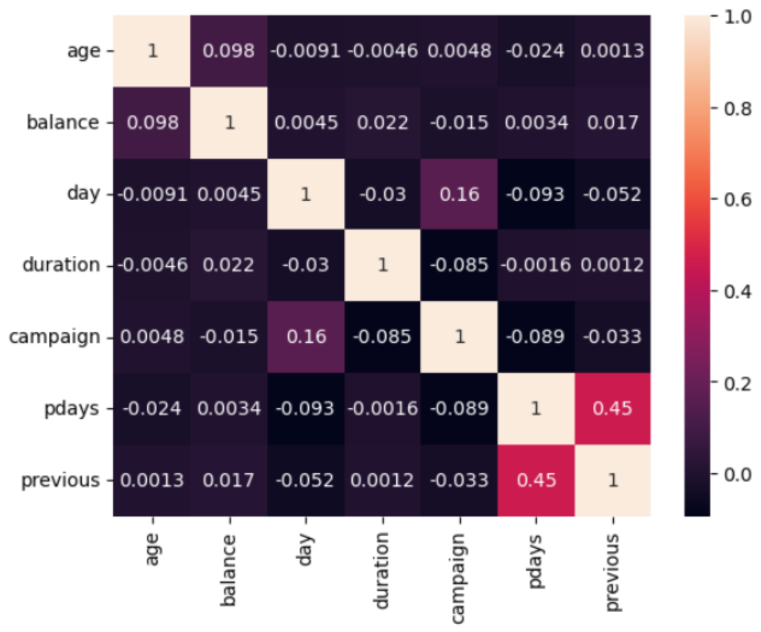
```
c=pd.crosstab(df['month'],df['y']).plot(kind='bar')
for i in c.containers:
    c.bar_label(i)
```



Correlation Heatmap

```
sns.heatmap(df.corr(),annot=True)

# weak correlations are observed between pdays-previous and campaign-day
```

```
<AxesSubplot:>
```

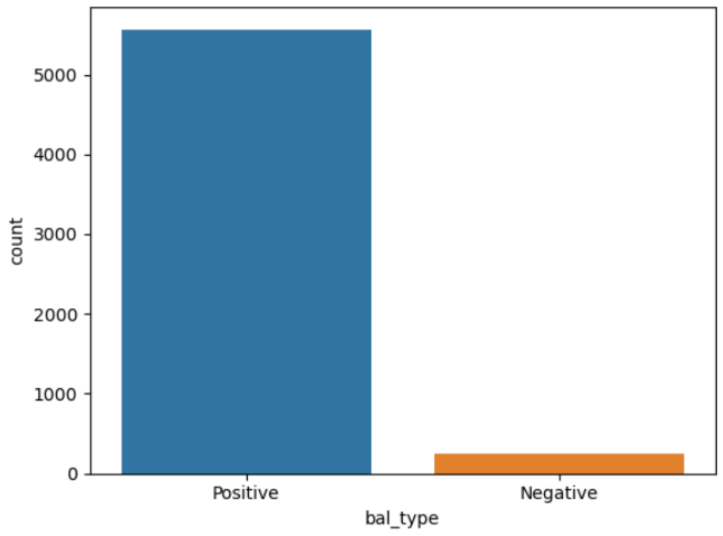


	age	balance	day	duration	campaign	pdays	previous
age	1	0.098	-0.0091	-0.0046	0.0048	-0.024	0.0013
balance	0.098	1	0.0045	0.022	-0.015	0.0034	0.017
day	-0.0091	0.0045	1	-0.03	0.16	-0.093	-0.052
duration	-0.0046	0.022	-0.03	1	-0.085	-0.0016	0.0012
campaign	0.0048	-0.015	0.16	-0.085	1	-0.089	-0.033
pdays	-0.024	0.0034	-0.093	-0.0016	-0.089	1	0.45
previous	0.0013	0.017	-0.052	0.0012	-0.033	0.45	1

```
df['bal_type']=df['balance'].apply(lambda x: 'Negative' if x<0 else 'Positive')
```

```
y=df[df['y']=='yes']
sns.countplot(y['bal_type'])
```

```
<AxesSubplot:xlabel='bal_type', ylabel='count'>
```

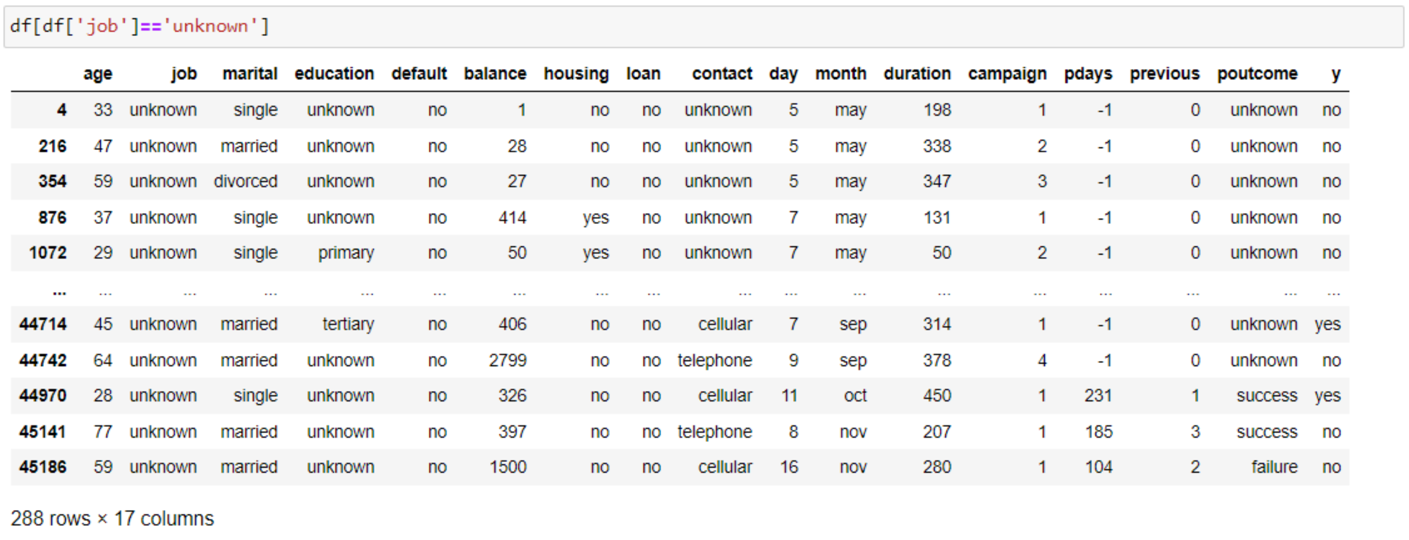


This plot will display the counts of 'Negative' and 'Positive' values in the 'bal_type' column for instances where the target variable 'y' is 'yes'. The plot helps to understand how the distribution of 'bal_type' differs for positive outcomes ('yes' in 'y') in your dataset.

EDA Interpretation:

- The target variable is imbalanced. Yes (5289): No (39922)
- The clients who are default are less in number.
- More married people have subscribed to the term subscription than other marital groups.
- Cellular mode of contact has resulted in high number of subscriptions.
- Clients in management job have made higher number of subscriptions than others.
- Blue collar clients are high in number for taking the housing loans.
- Common behaviour observed in clients who have subscribed to the term deposit are less defaults, personal loans and housing loans.
- Married clients have taken more housing loans.
- Many contacts are made in May month. The subscription number in May is highest.
- 30th day has the highest number of subscriptions.
- Management clients have higher balance, followed by retired clients

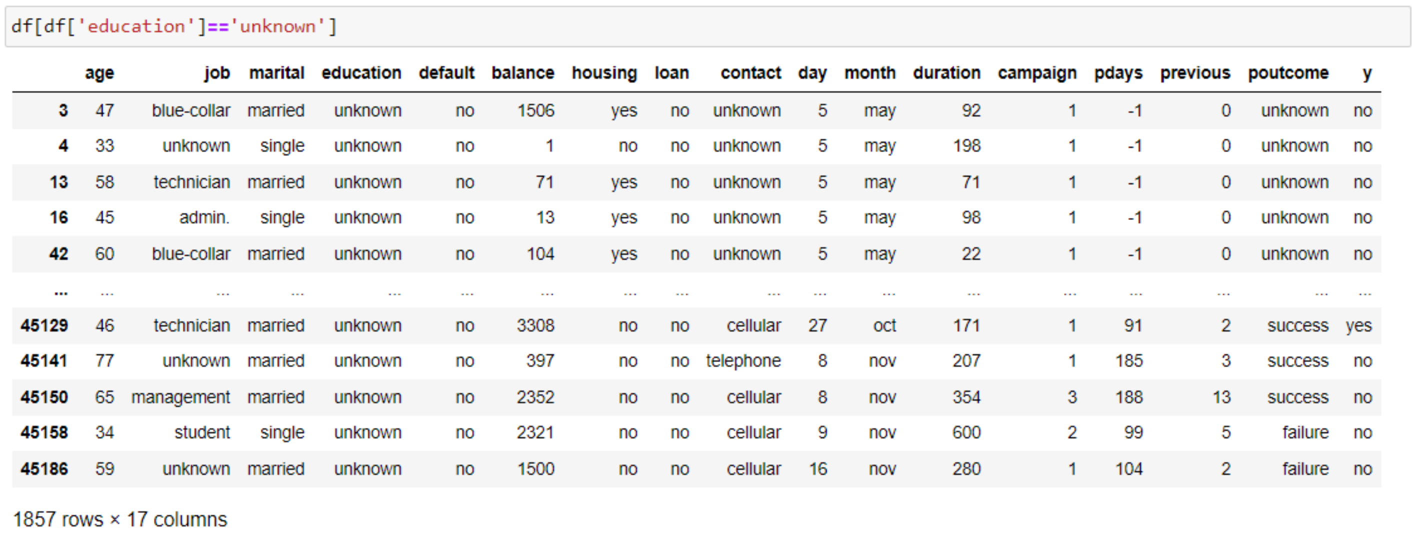
Checking for missing values & Treatment:



```
df[df['job']=='unknown']
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no
216	47	unknown	married	unknown	no	28	no	no	unknown	5	may	338	2	-1	0	unknown	no
354	59	unknown	divorced	unknown	no	27	no	no	unknown	5	may	347	3	-1	0	unknown	no
876	37	unknown	single	unknown	no	414	yes	no	unknown	7	may	131	1	-1	0	unknown	no
1072	29	unknown	single	primary	no	50	yes	no	unknown	7	may	50	2	-1	0	unknown	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
44714	45	unknown	married	tertiary	no	406	no	no	cellular	7	sep	314	1	-1	0	unknown	yes
44742	64	unknown	married	unknown	no	2799	no	no	telephone	9	sep	378	4	-1	0	unknown	no
44970	28	unknown	single	unknown	no	326	no	no	cellular	11	oct	450	1	231	1	success	yes
45141	77	unknown	married	unknown	no	397	no	no	telephone	8	nov	207	1	185	3	success	no
45186	59	unknown	married	unknown	no	1500	no	no	cellular	16	nov	280	1	104	2	failure	no

288 rows × 17 columns



```
df[df['education']=='unknown']
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no
13	58	technician	married	unknown	no	71	yes	no	unknown	5	may	71	1	-1	0	unknown	no
16	45	admin.	single	unknown	no	13	yes	no	unknown	5	may	98	1	-1	0	unknown	no
42	60	blue-collar	married	unknown	no	104	yes	no	unknown	5	may	22	1	-1	0	unknown	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
45129	46	technician	married	unknown	no	3308	no	no	cellular	27	oct	171	1	91	2	success	yes
45141	77	unknown	married	unknown	no	397	no	no	telephone	8	nov	207	1	185	3	success	no
45150	65	management	married	unknown	no	2352	no	no	cellular	8	nov	354	3	188	13	success	no
45158	34	student	single	unknown	no	2321	no	no	cellular	9	nov	600	2	99	5	failure	no
45186	59	unknown	married	unknown	no	1500	no	no	cellular	16	nov	280	1	104	2	failure	no

1857 rows × 17 columns

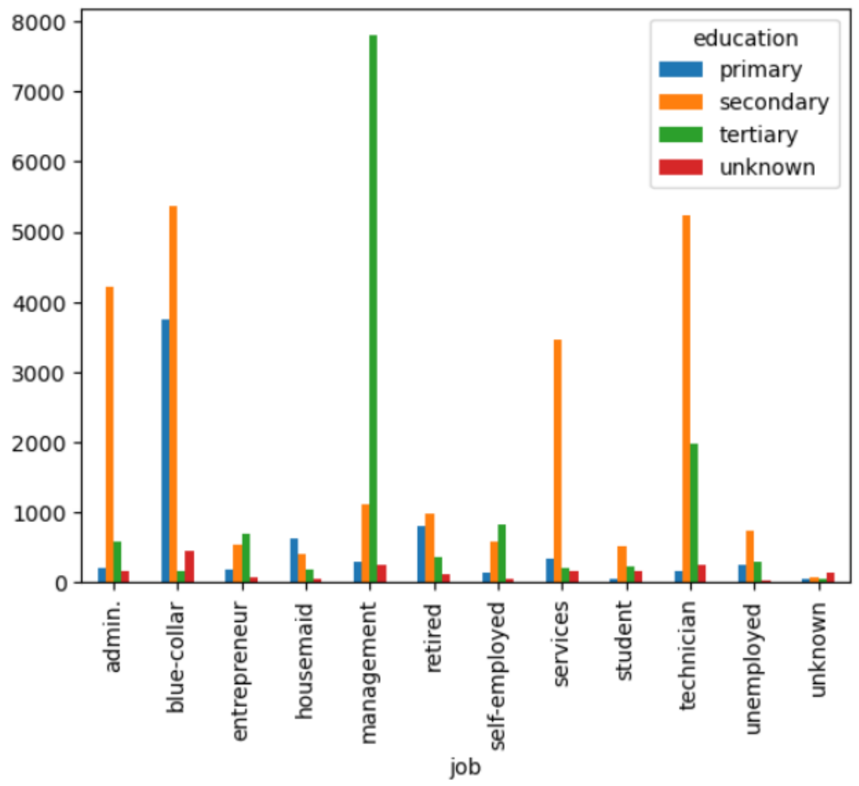
#Education has over 1800 missing values which needs to be treated before building any model

```
pd.crosstab(df['job'],df['education'])
```

education	primary	secondary	tertiary	unknown
job				
admin.	209	4219	572	171
blue-collar	3758	5371	149	454
entrepreneur	183	542	686	76
housemaid	627	395	173	45
management	294	1121	7801	242
retired	795	984	366	119
self-employed	130	577	833	39
services	345	3457	202	150
student	44	508	223	163
technician	158	5229	1968	242
unemployed	257	728	289	29
unknown	51	71	39	127

```
pd.crosstab(df['job'],df['education']).plot(kind='bar')
```

```
<AxesSubplot:xlabel='job'>
```



```
df.job.unique()
```

```
array(['management', 'technician', 'entrepreneur', 'blue-collar',
       'unknown', 'retired', 'admin.', 'services', 'self-employed',
       'unemployed', 'housemaid', 'student'], dtype=object)
```

```
df[df['job']=='management'][['education']].value_counts()
```

```
education
tertiary     7801
secondary    1121
primary       294
unknown       242
dtype: int64
```

```
df[df['job']=='management'].balance.value_counts()
```

```
0        840
1         36
4         30
3         29
2         27
        ...
1474       1
2608       1
1697       1
15449      1
8205       1
Name: balance, Length: 3514, dtype: int64
```

As the balance range of unknown is close to the balance range of student, unknown in job column will be imputed with student

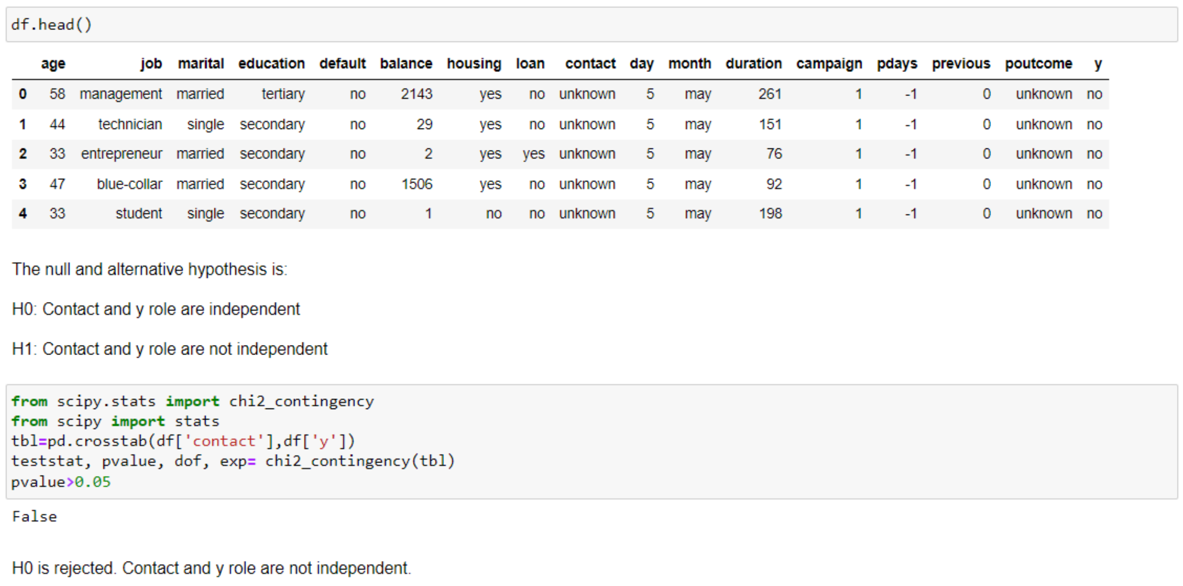
Replacing missing values at Job column:

```
df.job.replace({'unknown':'student'},inplace=True)
df.job.value_counts()
```

```
blue-collar      9732
management       9458
technician       7597
admin.           5171
services         4154
retired          2264
self-employed    1579
entrepreneur     1487
unemployed       1303
housemaid        1240
student          1226
Name: job, dtype: int64
```

```
df[df['job']=='admin.'].education.value_counts()
```

```
secondary    4219
tertiary      572
primary       209
unknown       171
Name: education, dtype: int64
```



```
df.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	secondary	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	student	single	secondary	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

The null and alternative hypothesis is:

H0: Contact and y role are independent

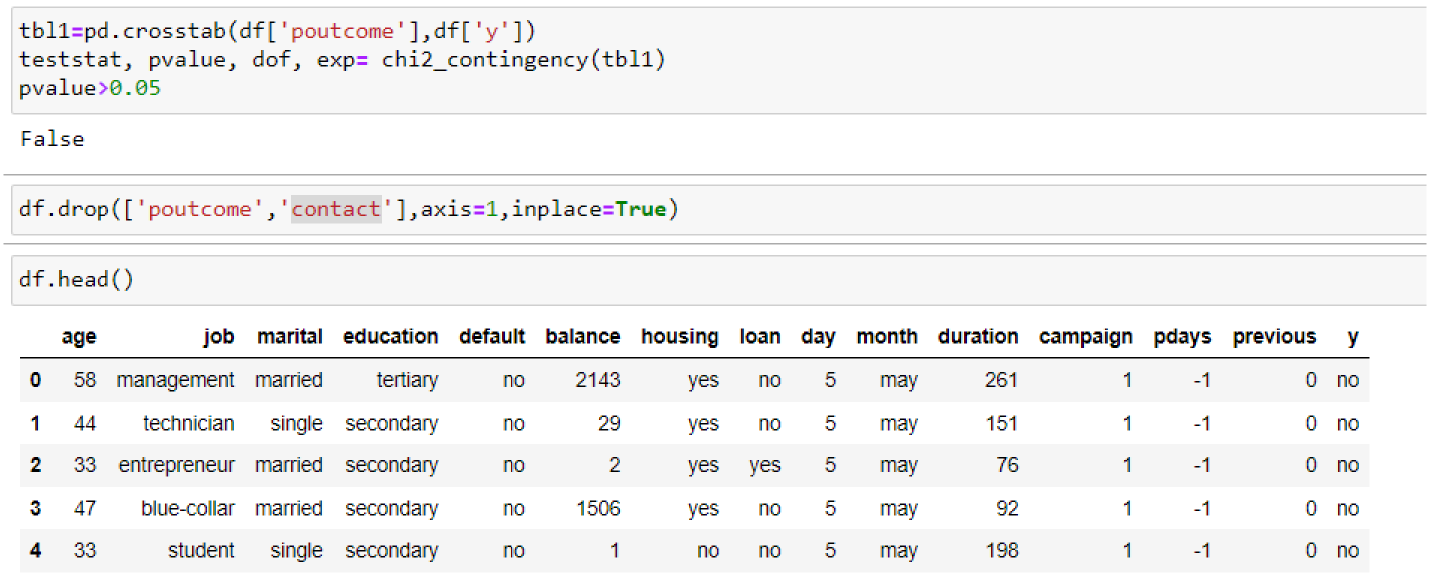
H1: Contact and y role are not independent

```
from scipy.stats import chi2_contingency
from scipy import stats
tbl=pd.crosstab(df['contact'],df['y'])
teststat, pvalue, dof, exp= chi2_contingency(tbl)
pvalue>0.05
```

```
False
```

H0 is rejected. Contact and y role are not independent.

<u>Dropping column contact and poutcome</u>:



```
tbl1=pd.crosstab(df['poutcome'],df['y'])
teststat, pvalue, dof, exp= chi2_contingency(tbl1)
pvalue>0.05
```

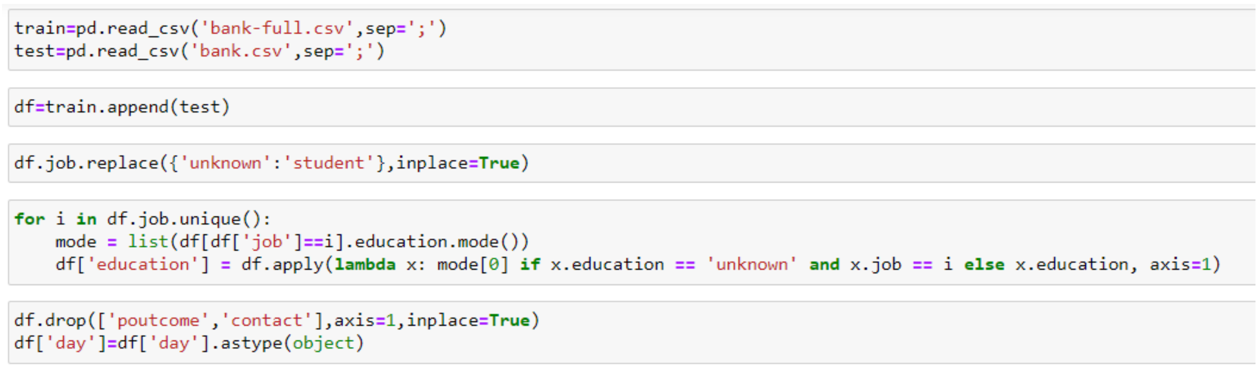
```
False
```

```
df.drop(['poutcome','contact'],axis=1,inplace=True)
```

```
df.head()
```

	age	job	marital	education	default	balance	housing	loan	day	month	duration	campaign	pdays	previous	y
0	58	management	married	tertiary	no	2143	yes	no	5	may	261	1	-1	0	no
1	44	technician	single	secondary	no	29	yes	no	5	may	151	1	-1	0	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	5	may	76	1	-1	0	no
3	47	blue-collar	married	secondary	no	1506	yes	no	5	may	92	1	-1	0	no
4	33	student	single	secondary	no	1	no	no	5	may	198	1	-1	0	no

Classification Models:



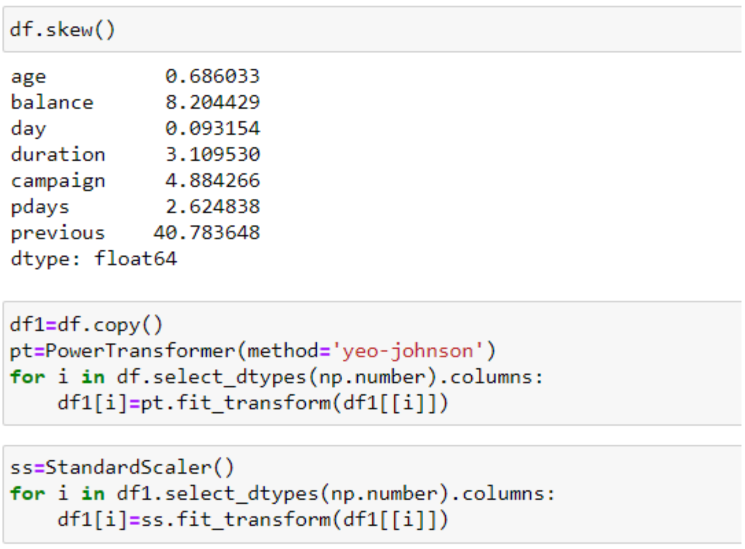
```
train=pd.read_csv('bank-full.csv',sep=';')
test=pd.read_csv('bank.csv',sep=';')
```

```
df=train.append(test)
```

```
df.job.replace({'unknown':'student'},inplace=True)
```

```
for i in df.job.unique():
    mode = list(df[df['job']==i].education.mode())
    df['education'] = df.apply(lambda x: mode[0] if x.education == 'unknown' and x.job == i else x.education, axis=1)
```

```
df.drop(['poutcome','contact'],axis=1,inplace=True)
df['day']=df['day'].astype(object)
```



```
df.skew()
```

```
age          0.686033
balance      8.204429
day          0.093154
duration     3.109530
campaign     4.884266
pdays        2.624838
previous    40.783648
dtype: float64
```

```
df1=df.copy()
pt=PowerTransformer(method='yeo-johnson')
for i in df.select_dtypes(np.number).columns:
    df1[i]=pt.fit_transform(df1[[i]])
```

```
ss=StandardScaler()
for i in df1.select_dtypes(np.number).columns:
    df1[i]=ss.fit_transform(df1[[i]])
```

Transformation - To normalize the numerical features in a DataFrame using the Yeo-Johnson power transformation.

Standard Scaler - The code standardizes the numerical features in df1 using a StandardScaler. This ensures that all features have a mean of 0 and a standard deviation of 1, putting them on a common scale.

```
c=df1.select_dtypes(object)
n=df1.select_dtypes(np.number)
```

```
le=LabelEncoder()
for i in c.columns:
    c[i]=le.fit_transform(c[[i]])
```

```
df2=pd.concat([n,c],axis=1)
```

```
df2.y.value_counts()
```

```
0    43922
1     5810
Name: y, dtype: int64
```

```
n=df2[df2['y']==0]
y=df2[df2['y']==1]
```

```
y_re = resample(y,
             replace=True,
             n_samples=round((len(n)*30)/100),
             random_state=1)
n_re=resample(n,
             replace=True,
             n_samples=round((len(n)*70)/100),
             random_state=1)
```

```
df3=pd.concat([n_re,y_re])
df3.y.value_counts()
```

```
0    30745
1    13177
Name: y, dtype: int64
```

The code addresses class imbalance by resampling the minority class ('y' equals 1) using the resample function from scikit-learn. It creates a new DataFrame df3 by combining the resampled minority class and the majority class (in a 70:30 ratio)

```
x=df3.drop('y',axis=1)
y=df3['y']
xtrain,xtest,ytrain,ytest=train_test_split(x,y,random_state=10,test_size=0.3)
```

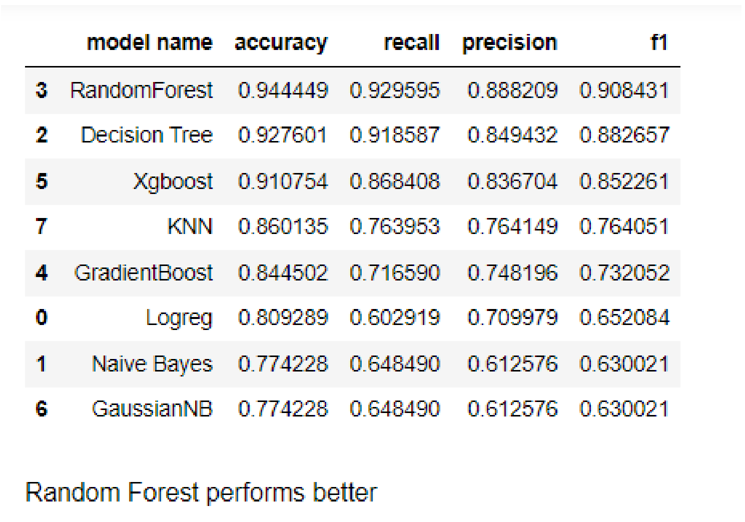
Applying train test split

```
clfs={
    'Logreg':LogisticRegression(),
    'Naive Bayes':GaussianNB(),
    'Decision Tree':DecisionTreeClassifier(),
    'RandomForest':RandomForestClassifier(),
    'GradientBoost':GradientBoostingClassifier(),
    'Xgboost':XGBClassifier(),
    'GaussianNB':GaussianNB(),
    'KNN':KNeighborsClassifier()
}

models_report=pd.DataFrame(columns=['model name','accuracy', 'recall',
                                    'precision', 'f1'])
for clf, clf_name in list(zip(clfs.values(), clfs.keys())):
    clf.fit(xtrain, ytrain)
    ypred=clf.predict(xtest)
    print('Fitting the model ...', clf_name)
    t=pd.DataFrame({
        'model name':clf_name,
        'accuracy':accuracy_score(ytest, ypred),
        'recall':recall_score(ytest, ypred),
        'precision':precision_score(ytest, ypred),
        'f1':f1_score(ytest, ypred)
    },index=[0])
    models_report=pd.concat([models_report,t], ignore_index=True)

models_report=models_report.sort_values(by='f1', ascending=False)
models_report
```

```
Fitting the model ... Logreg
Fitting the model ... Naive Bayes
Fitting the model ... Decision Tree
Fitting the model ... RandomForest
Fitting the model ... GradientBoost
Fitting the model ... Xgboost
Fitting the model ... GaussianNB
Fitting the model ... KNN
```

Dictionary clfs containing various classification algorithms, such as Logistic Regression, Naive Bayes, Decision Tree, RandomForest, GradientBoost, Xgboost, GaussianNB, and KNN.

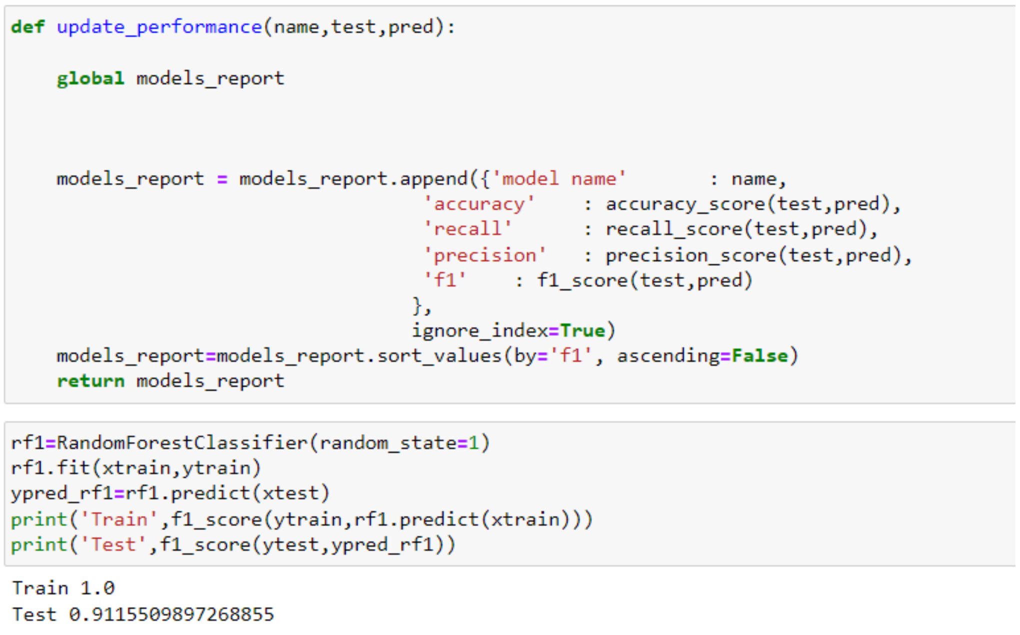
It iterates through the dictionary of classifiers, fits each model on the training data (xtrain, ytrain), predicts on the test data (xtest), and calculates various performance metrics (accuracy, recall, precision, and F1 score).



	model name	accuracy	recall	precision	f1
3	RandomForest	0.944449	0.929595	0.888209	0.908431
2	Decision Tree	0.927601	0.918587	0.849432	0.882657
5	Xgboost	0.910754	0.868408	0.836704	0.852261
7	KNN	0.860135	0.763953	0.764149	0.764051
4	GradientBoost	0.844502	0.716590	0.748196	0.732052
0	Logreg	0.809289	0.602919	0.709979	0.652084
1	Naive Bayes	0.774228	0.648490	0.612576	0.630021
6	GaussianNB	0.774228	0.648490	0.612576	0.630021

Random Forest performs better

As per F1 score Random Forrest seems to be performing better than rest of the models



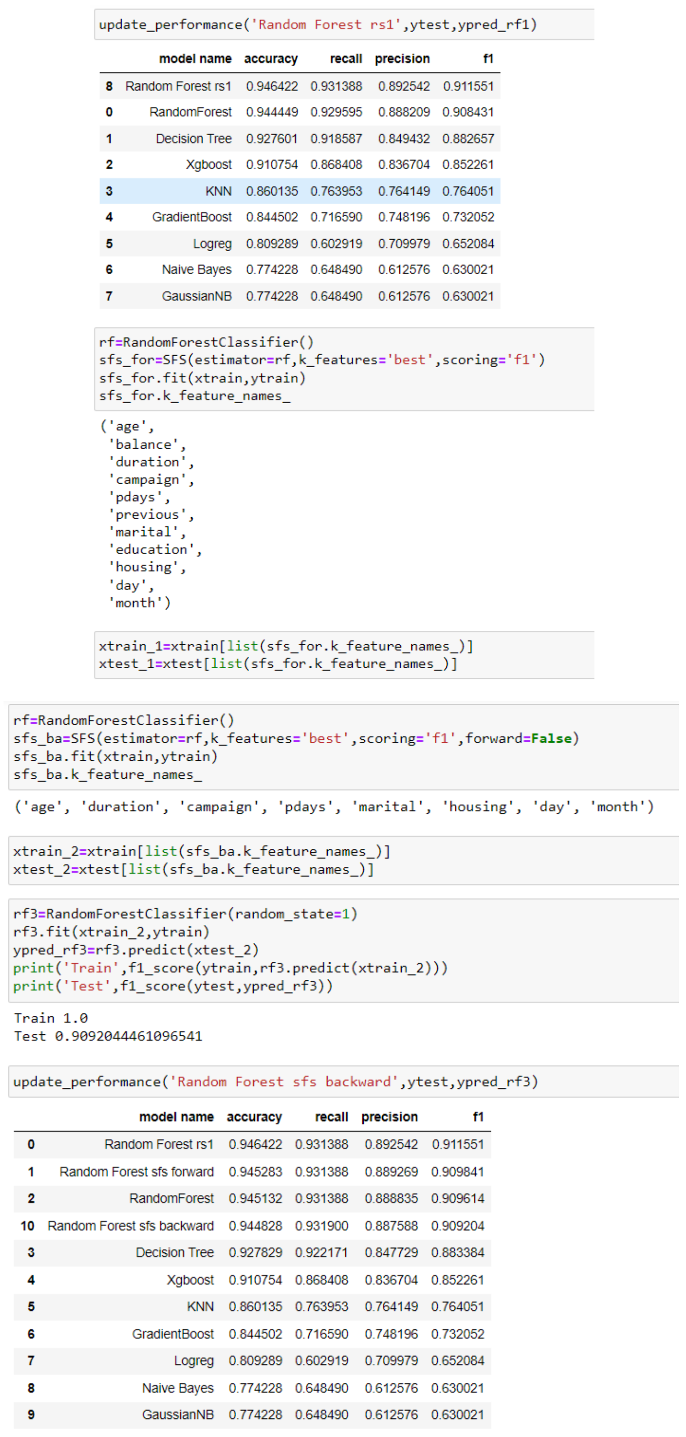
```
def update_performance(name,test,pred):

    global models_report


    models_report = models_report.append({'model name'       : name,
                                          'accuracy'     : accuracy_score(test,pred),
                                          'recall'       : recall_score(test,pred),
                                          'precision'    : precision_score(test,pred),
                                          'f1'     : f1_score(test,pred)
                                         },
                                         ignore_index=True)
    models_report=models_report.sort_values(by='f1', ascending=False)
    return models_report
```

```
rf1=RandomForestClassifier(random_state=1)
rf1.fit(xtrain,ytrain)
ypred_rf1=rf1.predict(xtest)
print('Train',f1_score(ytrain,rf1.predict(xtrain)))
print('Test',f1_score(ytest,ypred_rf1))
```

```
Train 1.0
Test 0.9115509897268855
```

Defining a function **update_performance** to update and append the performance metrics for the new models.

It updates the models_report DataFrame with the performance of a RandomForest model ('Random Forest rs1') fitted with a specific random seed (random_state=1).



```
update_performance('Random Forest rs1',ytest,ypred_rf1)
```

	model name	accuracy	recall	precision	f1
8	Random Forest rs1	0.946422	0.931388	0.892542	0.911551
0	RandomForest	0.944449	0.929595	0.888209	0.908431
1	Decision Tree	0.927601	0.918587	0.849432	0.882657
2	Xgboost	0.910754	0.868408	0.836704	0.852261
3	KNN	0.860135	0.763953	0.764149	0.764051
4	GradientBoost	0.844502	0.716590	0.748196	0.732052
5	Logreg	0.809289	0.602919	0.709979	0.652084
6	Naive Bayes	0.774228	0.648490	0.612576	0.630021
7	GaussianNB	0.774228	0.648490	0.612576	0.630021

```
rf=RandomForestClassifier()
sfs_for=SFS(estimator=rf,k_features='best',scoring='f1')
sfs_for.fit(xtrain,ytrain)
sfs_for.k_feature_names_
```

```
('age',
 'balance',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'marital',
 'education',
 'housing',
 'day',
 'month')
```

```
xtrain_1=xtrain[list(sfs_for.k_feature_names_)]
xtest_1=xtest[list(sfs_for.k_feature_names_)]
```

```
rf=RandomForestClassifier()
sfs_ba=SFS(estimator=rf,k_features='best',scoring='f1',forward=False)
sfs_ba.fit(xtrain,ytrain)
sfs_ba.k_feature_names_
```

```
('age', 'duration', 'campaign', 'pdays', 'marital', 'housing', 'day', 'month')
```

```
xtrain_2=xtrain[list(sfs_ba.k_feature_names_)]
xtest_2=xtest[list(sfs_ba.k_feature_names_)]
```

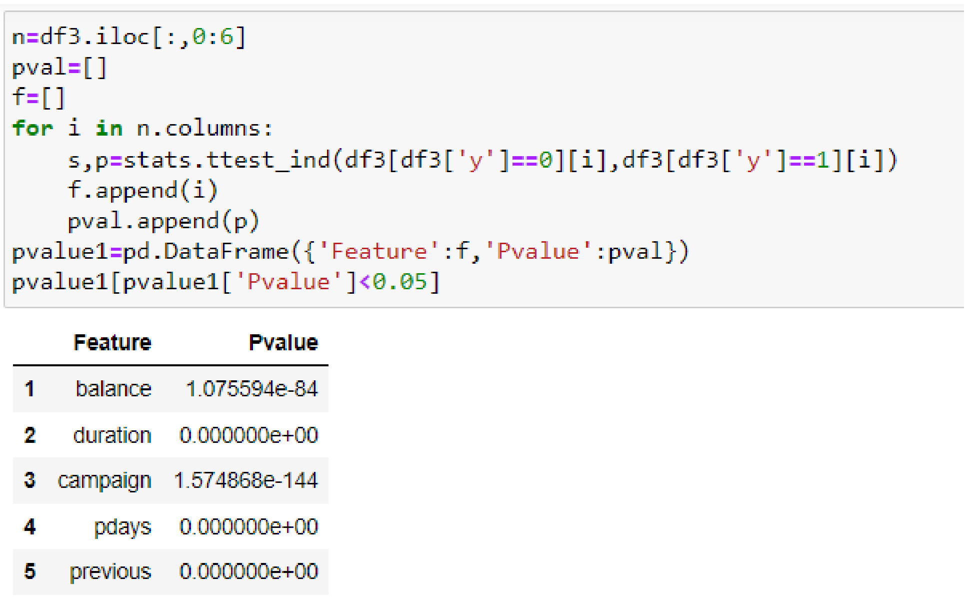
```
rf3=RandomForestClassifier(random_state=1)
rf3.fit(xtrain_2,ytrain)
ypred_rf3=rf3.predict(xtest_2)
print('Train',f1_score(ytrain,rf3.predict(xtrain_2)))
print('Test',f1_score(ytest,ypred_rf3))
```

```
Train 1.0
Test 0.9092044461096541
```

```
update_performance('Random Forest sfs backward',ytest,ypred_rf3)
```

	model name	accuracy	recall	precision	f1
0	Random Forest rs1	0.946422	0.931388	0.892542	0.911551
1	Random Forest sfs forward	0.945283	0.931388	0.889269	0.909841
2	RandomForest	0.945132	0.931388	0.888835	0.909614
10	Random Forest sfs backward	0.944828	0.931900	0.887588	0.909204
3	Decision Tree	0.927829	0.922171	0.847729	0.883384
4	Xgboost	0.910754	0.868408	0.836704	0.852261
5	KNN	0.860135	0.763953	0.764149	0.764051
6	GradientBoost	0.844502	0.716590	0.748196	0.732052
7	Logreg	0.809289	0.602919	0.709979	0.652084
8	Naive Bayes	0.774228	0.648490	0.612576	0.630021
9	GaussianNB	0.774228	0.648490	0.612576	0.630021

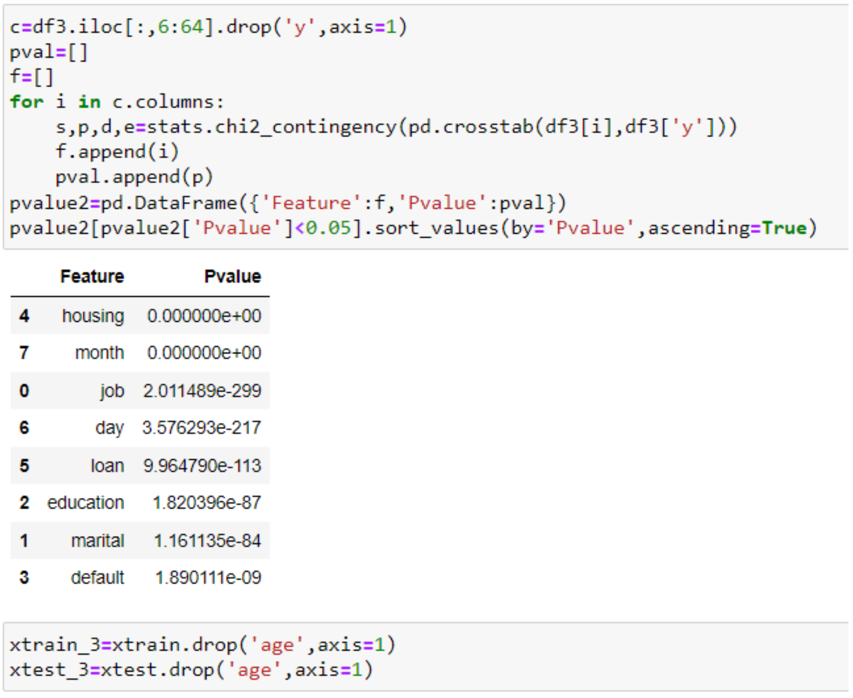
Above code demonstrates the use of Sequential Feature with RandomForestClassifier, evaluates the performance of the selected features on a RandomForest model, and updates the model comparison report with the new performance metrics.



```python
n=df3.iloc[:,0:6]
pval=[]
f=[]
for i in n.columns:
    s,p=stats.ttest_ind(df3[df3['y']==0][i],df3[df3['y']==1][i])
    f.append(i)
    pval.append(p)
pvalue1=pd.DataFrame({'Feature':f,'Pvalue':pval})
pvalue1[pvalue1['Pvalue']<0.05]
```

	Feature	Pvalue
1	balance	1.075594e-84
2	duration	0.000000e+00
3	campaign	1.574868e-144
4	pdays	0.000000e+00
5	previous	0.000000e+00

This code conducts **t-tests** for each numeric feature between two classes and identifies features with statistically significant differences (p-value < 0.05) between the classes



```python
c=df3.iloc[:,6:64].drop('y',axis=1)
pval=[]
f=[]
for i in c.columns:
    s,p,d,e=stats.chi2_contingency(pd.crosstab(df3[i],df3['y']))
    f.append(i)
    pval.append(p)
pvalue2=pd.DataFrame({'Feature':f,'Pvalue':pval})
pvalue2[pvalue2['Pvalue']<0.05].sort_values(by='Pvalue',ascending=True)
```

	Feature	Pvalue
4	housing	0.000000e+00
7	month	0.000000e+00
0	job	2.011489e-299
6	day	3.576293e-217
5	loan	9.964790e-113
2	education	1.820396e-87
1	marital	1.161135e-84
3	default	1.890111e-09

```python
xtrain_3=xtrain.drop('age',axis=1)
xtest_3=xtest.drop('age',axis=1)
```

This code conducts **chi-squared tests** for each categorical feature in df3 with respect to the target variable 'y' and identifies features with statistically significant associations (p-value < 0.05)

```
xtrain_3=xtrain.drop('age',axis=1)
xtest_3=xtest.drop('age',axis=1)
```

```
rf4=RandomForestClassifier(random_state=1)
rf4.fit(xtrain_3,ytrain)
ypred_rf4=rf4.predict(xtest_3)
print('Train',f1_score(ytrain,rf4.predict(xtrain_3)))
print('Test',f1_score(ytest,ypred_rf4))
```

```
Train 1.0
Test 0.9074328544659588
```

```
update_performance('Random Forest with age removed',ytest,ypred_rf4)
```

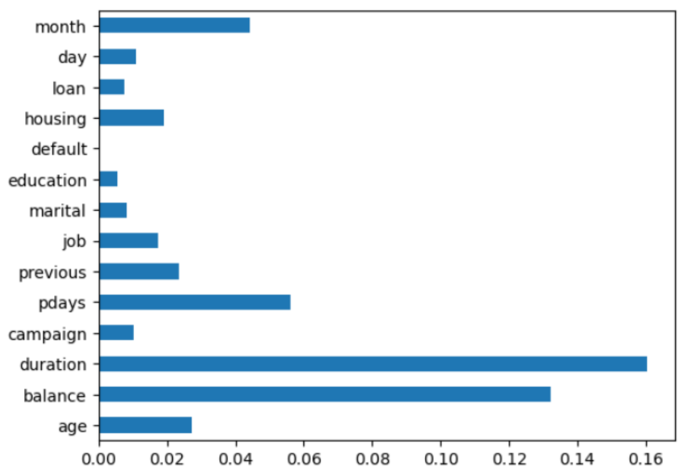
	model name	accuracy	recall	precision	f1
0	Random Forest rs1	0.946422	0.931388	0.892542	0.911551
1	Random Forest sfs forward	0.945283	0.931388	0.889269	0.909841
2	RandomForest	0.945132	0.931388	0.888835	0.909614
3	Random Forest sfs backward	0.944828	0.931900	0.887588	0.909204
11	Random Forest with age removed	0.943766	0.929852	0.886070	0.907433
4	Decision Tree	0.927829	0.922171	0.847729	0.883384
5	Xgboost	0.910754	0.868408	0.836704	0.852261
6	KNN	0.860135	0.763953	0.764149	0.764051
7	GradientBoost	0.844502	0.716590	0.748196	0.732052
8	Logreg	0.809289	0.602919	0.709979	0.652084
9	Naive Bayes	0.774228	0.648490	0.612576	0.630021
10	GaussianNB	0.774228	0.648490	0.612576	0.630021

This code assesses the impact of removing the 'age' feature on the performance of a RandomForest model. It trains the model on the modified datasets without the 'age' column and evaluates its performance using F1 scores on both the training and testing sets.

```
from sklearn.feature_selection import mutual_info_classif
```

```
imp=mutual_info_classif(xtrain,ytrain)
feat_imp=pd.Series(imp,df3.columns[0:len(df3.columns)-1])
feat_imp.plot(kind='barh')
```

```
<AxesSubplot:>
```



```
xtrain_4=xtrain[list(feat_imp[feat_imp>0].index)]
xtest_4=xtest[list(feat_imp[feat_imp>0].index)]
rf5=RandomForestClassifier(random_state=1)
rf5.fit(xtrain_4,ytrain)
ypred_rf5=rf5.predict(xtest_4)
print('Train',f1_score(ytrain,rf5.predict(xtrain_4)))
print('Test',f1_score(ytest,ypred_rf5))
```

```
Train 1.0
Test 0.9099975068561456
```

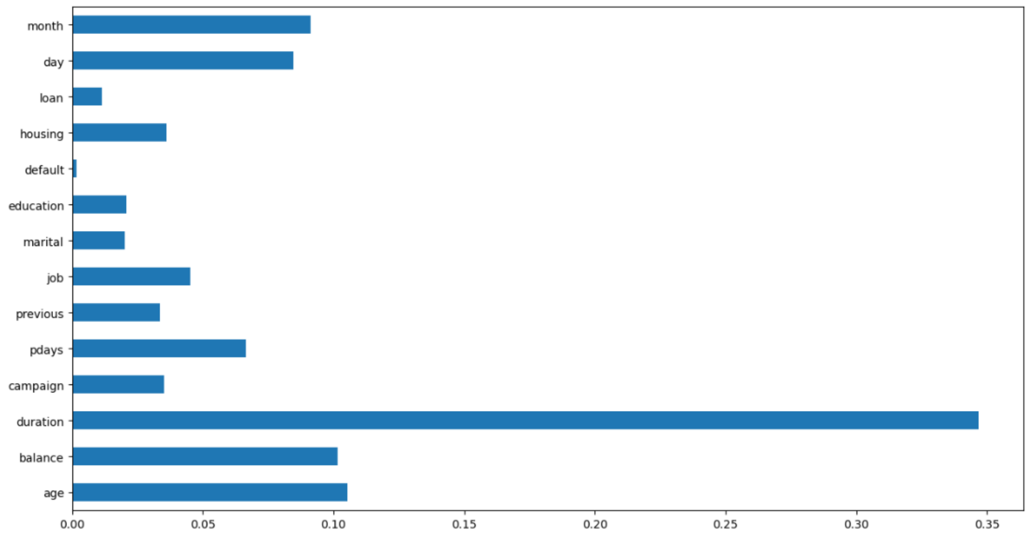
```
update_performance('Random Forest info gain feat selection',ytest,ypred_rf5)
```

	model name	accuracy	recall	precision	f1
0	Random Forest rs1	0.946422	0.931388	0.892542	0.911551
12	Random Forest info gain feat selection	0.945208	0.934460	0.886783	0.909998
1	Random Forest sfs forward	0.945283	0.931388	0.889269	0.909841
2	RandomForest	0.945132	0.931388	0.888835	0.909614
3	Random Forest sfs backward	0.944828	0.931900	0.887588	0.909204
4	Random Forest with age removed	0.943766	0.929852	0.886070	0.907433
5	Decision Tree	0.927829	0.922171	0.847729	0.883384
6	Xgboost	0.910754	0.868408	0.836704	0.852261
7	KNN	0.860135	0.763953	0.764149	0.764051
8	GradientBoost	0.844502	0.716590	0.748196	0.732052
9	Logreg	0.809289	0.602919	0.709979	0.652084
10	Naive Bayes	0.774228	0.648490	0.612576	0.630021
11	GaussianNB	0.774228	0.648490	0.612576	0.630021

Here, we tool mutual information to select features, visualizes their importance, trains a RandomForest model on the selected features, and updates the model comparison report with the performance of the model

```
rf_imp=rf1.feature_importances_
rf_feat_imp=pd.Series(rf_imp,df3.columns[0:len(df3.columns)-1])
plt.figure(figsize=(15,8))
rf_feat_imp.plot(kind='barh')
```

```
<AxesSubplot:>
```



```
xtrain_5=xtrain[list(rf_feat_imp.sort_values(ascending=False)[rf_feat_imp>0.03].index)]
xtest_5=xtest[list(rf_feat_imp.sort_values(ascending=False)[rf_feat_imp>0.03].index)]
```

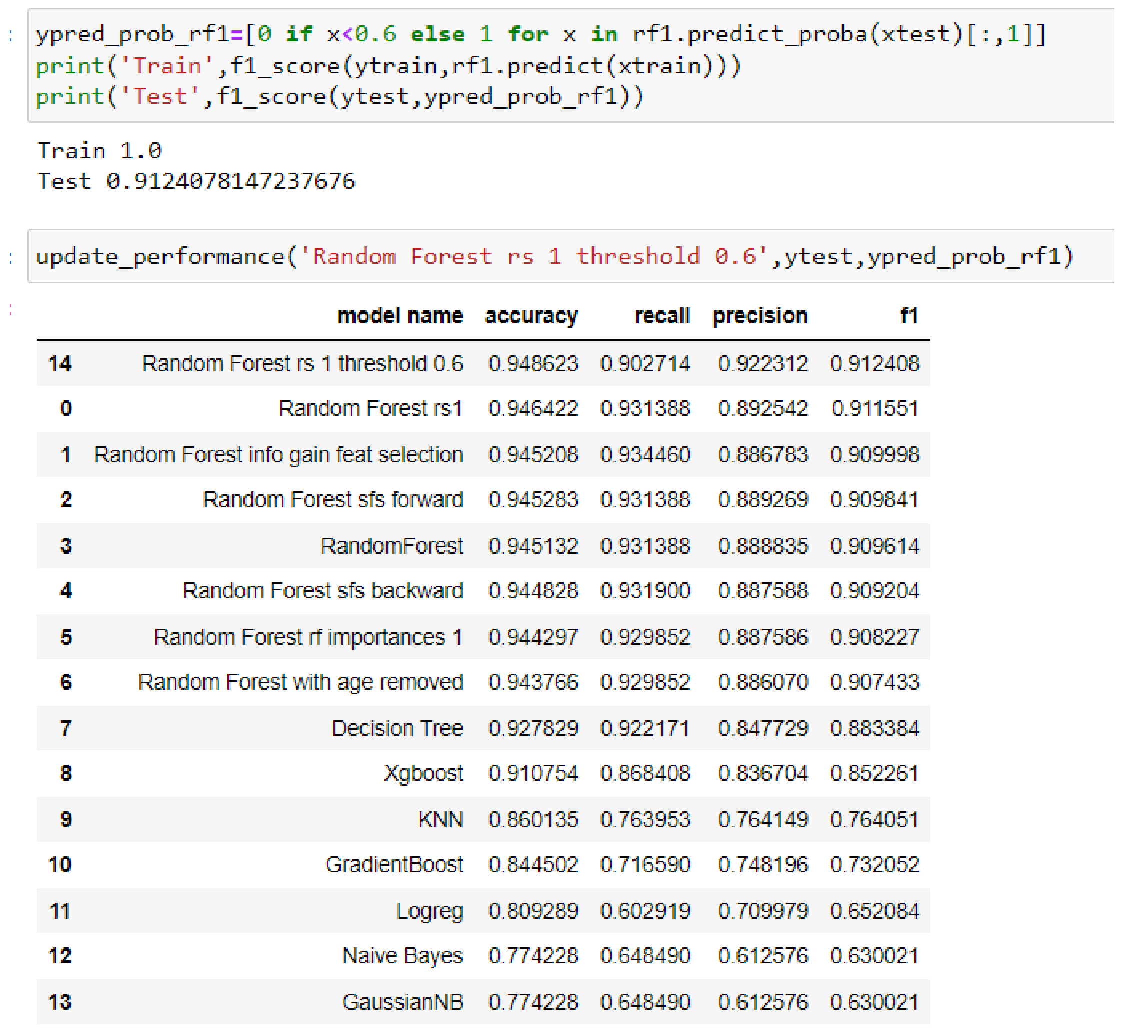
```
rf6=RandomForestClassifier(random_state=1)
rf6.fit(xtrain_5,ytrain)
ypred_rf6=rf6.predict(xtest_5)
print('Train',f1_score(ytrain,rf6.predict(xtrain_5)))
print('Test',f1_score(ytest,ypred_rf6))
```

```
Train 1.0
Test 0.908227056764191
```

```
update_performance('Random Forest rf importances 1',ytest,ypred_rf6)
```

	model name	accuracy	recall	precision	f1
0	Random Forest rs1	0.946422	0.931388	0.892542	0.911551
1	Random Forest info gain feat selection	0.945208	0.934460	0.886783	0.909998
2	Random Forest sfs forward	0.945283	0.931388	0.889269	0.909841
3	RandomForest	0.945132	0.931388	0.888835	0.909614
4	Random Forest sfs backward	0.944828	0.931900	0.887588	0.909204
13	Random Forest rf importances 1	0.944297	0.929852	0.887586	0.908227
5	Random Forest with age removed	0.943766	0.929852	0.886070	0.907433
6	Decision Tree	0.927829	0.922171	0.847729	0.883384
7	Xgboost	0.910754	0.868408	0.836704	0.852261
8	KNN	0.860135	0.763953	0.764149	0.764051
9	GradientBoost	0.844502	0.716590	0.748196	0.732052
10	Logreg	0.809289	0.602919	0.709979	0.652084
11	Naive Bayes	0.774228	0.648490	0.612576	0.630021
12	GaussianNB	0.774228	0.648490	0.612576	0.630021

The code shows a feature selection approach based on feature importance and evaluates the performance of a RandomForest model on the selected features. Here Random Forest rs1 has the highest F1 score.



```
ypred_prob_rf1=[0 if x<0.6 else 1 for x in rf1.predict_proba(xtest)[:,1]]
print('Train',f1_score(ytrain,rf1.predict(xtrain)))
print('Test',f1_score(ytest,ypred_prob_rf1))
```

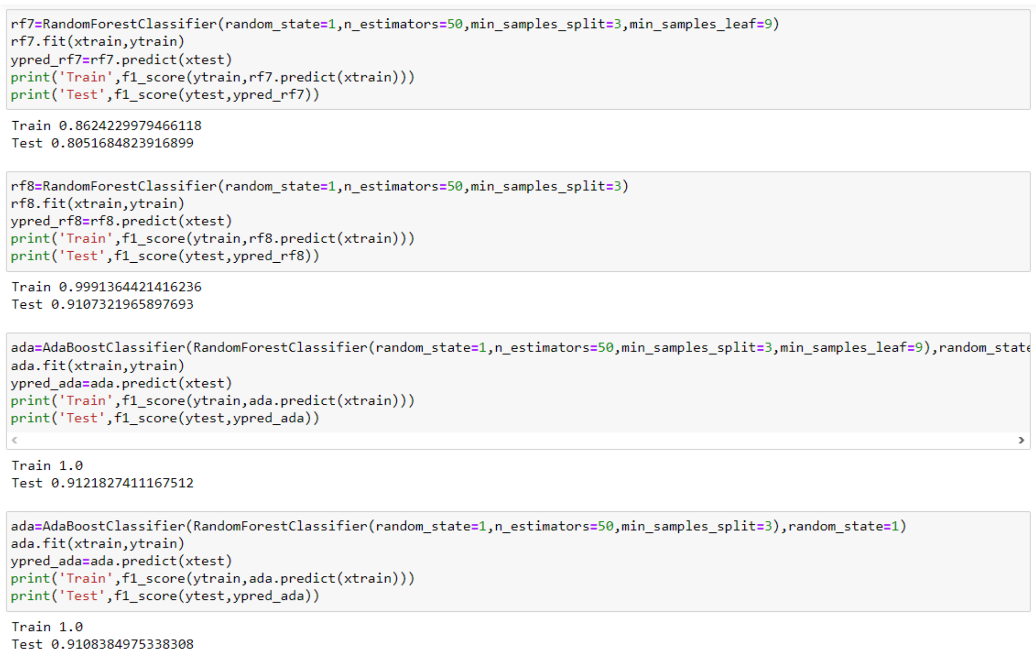
```
Train 1.0
Test 0.9124078147237676
```

```
update_performance('Random Forest rs 1 threshold 0.6',ytest,ypred_prob_rf1)
```

	model name	accuracy	recall	precision	f1
14	Random Forest rs 1 threshold 0.6	0.948623	0.902714	0.922312	0.912408
0	Random Forest rs1	0.946422	0.931388	0.892542	0.911551
1	Random Forest info gain feat selection	0.945208	0.934460	0.886783	0.909998
2	Random Forest sfs forward	0.945283	0.931388	0.889269	0.909841
3	RandomForest	0.945132	0.931388	0.888835	0.909614
4	Random Forest sfs backward	0.944828	0.931900	0.887588	0.909204
5	Random Forest rf importances 1	0.944297	0.929852	0.887586	0.908227
6	Random Forest with age removed	0.943766	0.929852	0.886070	0.907433
7	Decision Tree	0.927829	0.922171	0.847729	0.883384
8	Xgboost	0.910754	0.868408	0.836704	0.852261
9	KNN	0.860135	0.763953	0.764149	0.764051
10	GradientBoost	0.844502	0.716590	0.748196	0.732052
11	Logreg	0.809289	0.602919	0.709979	0.652084
12	Naive Bayes	0.774228	0.648490	0.612576	0.630021
13	GaussianNB	0.774228	0.648490	0.612576	0.630021

The above code evaluates the performance of a RandomForest model by applying a custom threshold (0.6) to the predicted probabilities on the test dataset. The F1 scores are calculated and printed for both the training and testing sets, and the model's performance is updated in the model comparison report.

Experimenting with different configurations of **RandomForestClassifier** models and evaluating their performance on both the training and testing sets.



```
rf7=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3,min_samples_leaf=9)
rf7.fit(xtrain,ytrain)
ypred_rf7=rf7.predict(xtest)
print('Train',f1_score(ytrain,rf7.predict(xtrain)))
print('Test',f1_score(ytest,ypred_rf7))
```

```
Train 0.8624229979466118
Test 0.8051684823916899
```

```
rf8=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3)
rf8.fit(xtrain,ytrain)
ypred_rf8=rf8.predict(xtest)
print('Train',f1_score(ytrain,rf8.predict(xtrain)))
print('Test',f1_score(ytest,ypred_rf8))
```

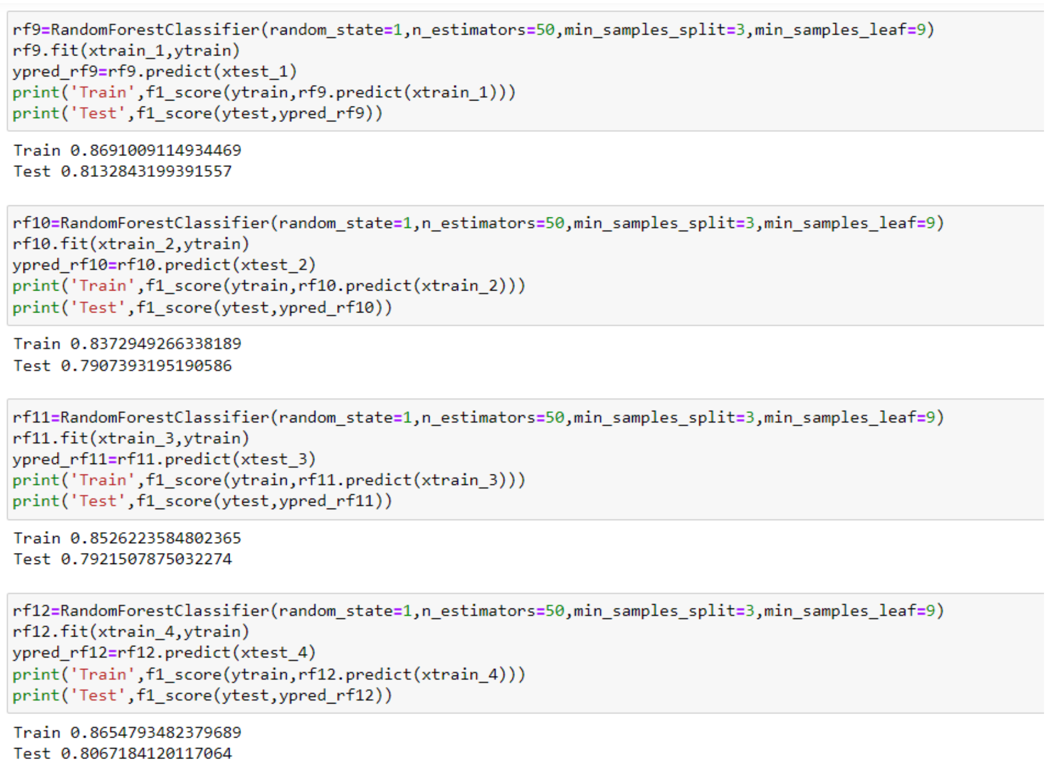
```
Train 0.9991364421416236
Test 0.9107321965897693
```

```
ada=AdaBoostClassifier(RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3,min_samples_leaf=9),random_state
ada.fit(xtrain,ytrain)
ypred_ada=ada.predict(xtest)
print('Train',f1_score(ytrain,ada.predict(xtrain)))
print('Test',f1_score(ytest,ypred_ada))
```

```
Train 1.0
Test 0.9121827411167512
```

```
ada=AdaBoostClassifier(RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3),random_state=1)
ada.fit(xtrain,ytrain)
ypred_ada=ada.predict(xtest)
print('Train',f1_score(ytrain,ada.predict(xtrain)))
print('Test',f1_score(ytest,ypred_ada))
```

```
Train 1.0
Test 0.9108384975338308
```



```
rf9=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3,min_samples_leaf=9)
rf9.fit(xtrain_1,ytrain)
ypred_rf9=rf9.predict(xtest_1)
print('Train',f1_score(ytrain,rf9.predict(xtrain_1)))
print('Test',f1_score(ytest,ypred_rf9))
```

```
Train 0.8691009114934469
Test 0.8132843199391557
```

```
rf10=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3,min_samples_leaf=9)
rf10.fit(xtrain_2,ytrain)
ypred_rf10=rf10.predict(xtest_2)
print('Train',f1_score(ytrain,rf10.predict(xtrain_2)))
print('Test',f1_score(ytest,ypred_rf10))
```

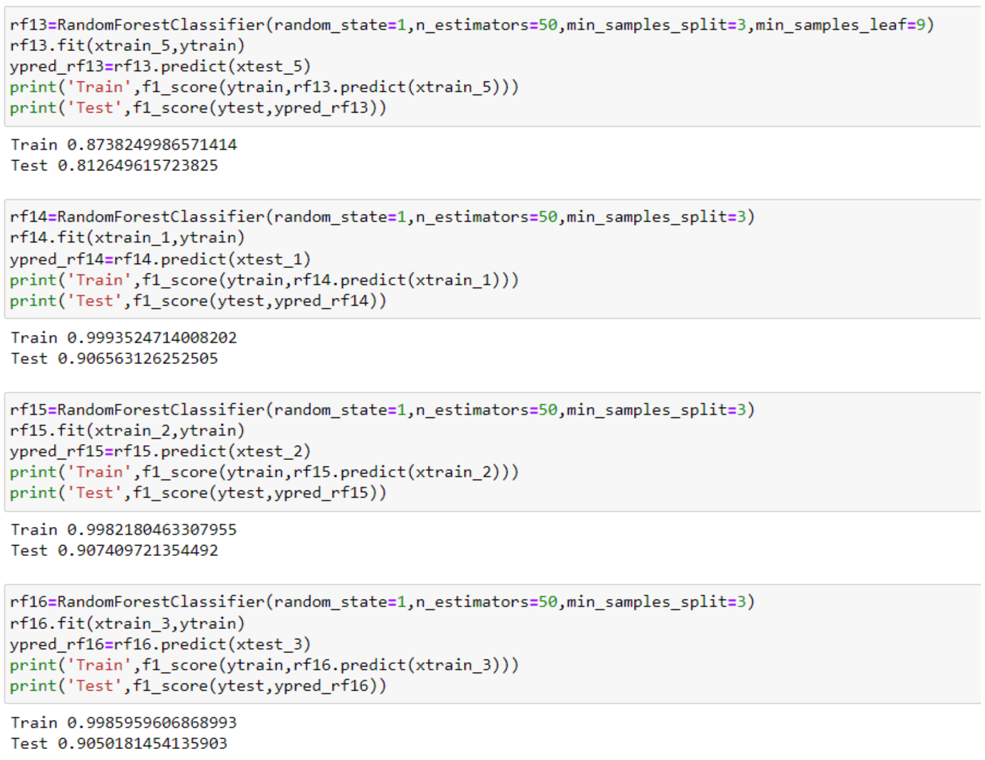
```
Train 0.8372949266338189
Test 0.7907393195190586
```

```
rf11=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3,min_samples_leaf=9)
rf11.fit(xtrain_3,ytrain)
ypred_rf11=rf11.predict(xtest_3)
print('Train',f1_score(ytrain,rf11.predict(xtrain_3)))
print('Test',f1_score(ytest,ypred_rf11))
```

```
Train 0.8526223584802365
Test 0.7921507875032274
```

```
rf12=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3,min_samples_leaf=9)
rf12.fit(xtrain_4,ytrain)
ypred_rf12=rf12.predict(xtest_4)
print('Train',f1_score(ytrain,rf12.predict(xtrain_4)))
print('Test',f1_score(ytest,ypred_rf12))
```

```
Train 0.8654793482379689
Test 0.8067184120117064
```



```
rf13=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3,min_samples_leaf=9)
rf13.fit(xtrain_5,ytrain)
ypred_rf13=rf13.predict(xtest_5)
print('Train',f1_score(ytrain,rf13.predict(xtrain_5)))
print('Test',f1_score(ytest,ypred_rf13))
```

```
Train 0.8738249986571414
Test 0.812649615723825
```

```
rf14=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3)
rf14.fit(xtrain_1,ytrain)
ypred_rf14=rf14.predict(xtest_1)
print('Train',f1_score(ytrain,rf14.predict(xtrain_1)))
print('Test',f1_score(ytest,ypred_rf14))
```

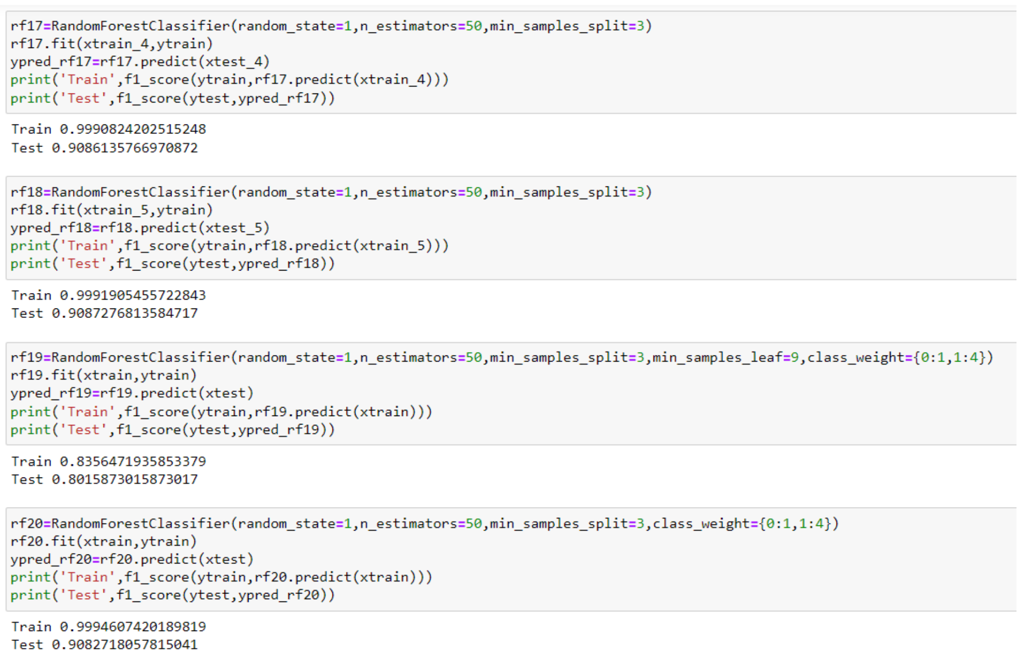
```
Train 0.9993524714008202
Test 0.906563126252505
```

```
rf15=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3)
rf15.fit(xtrain_2,ytrain)
ypred_rf15=rf15.predict(xtest_2)
print('Train',f1_score(ytrain,rf15.predict(xtrain_2)))
print('Test',f1_score(ytest,ypred_rf15))
```

```
Train 0.9982180463307955
Test 0.907409721354492
```

```
rf16=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3)
rf16.fit(xtrain_3,ytrain)
ypred_rf16=rf16.predict(xtest_3)
print('Train',f1_score(ytrain,rf16.predict(xtrain_3)))
print('Test',f1_score(ytest,ypred_rf16))
```

```
Train 0.9985959606868993
Test 0.9050181454135903
```



```
rf17=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3)
rf17.fit(xtrain_4,ytrain)
ypred_rf17=rf17.predict(xtest_4)
print('Train',f1_score(ytrain,rf17.predict(xtrain_4)))
print('Test',f1_score(ytest,ypred_rf17))
```

```
Train 0.9990824202515248
Test 0.9086135766970872
```

```
rf18=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3)
rf18.fit(xtrain_5,ytrain)
ypred_rf18=rf18.predict(xtest_5)
print('Train',f1_score(ytrain,rf18.predict(xtrain_5)))
print('Test',f1_score(ytest,ypred_rf18))
```

```
Train 0.9991905455722843
Test 0.9087276813584717
```

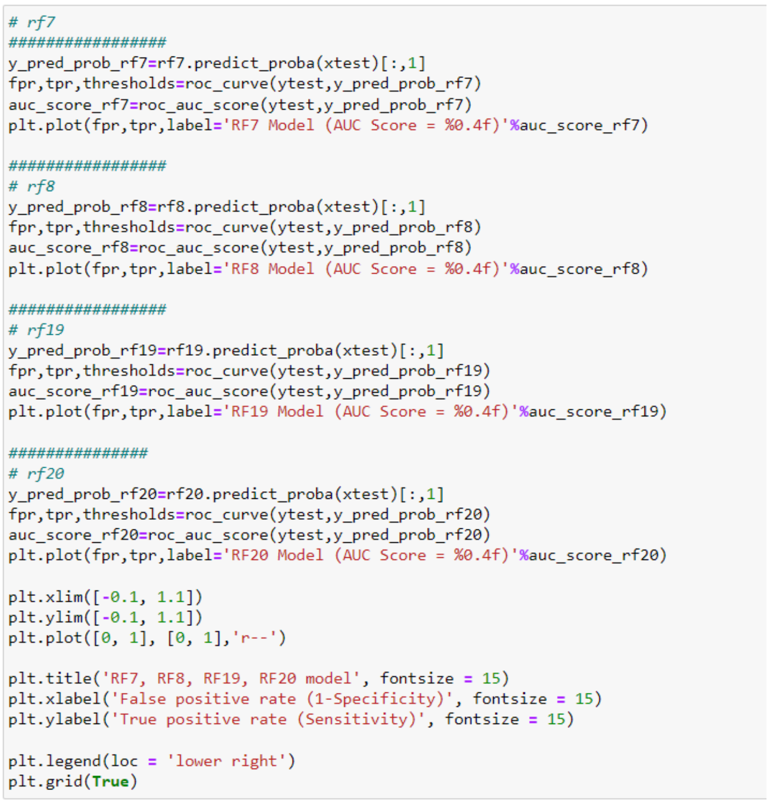
```
rf19=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3,min_samples_leaf=9,class_weight={0:1,1:4})
rf19.fit(xtrain,ytrain)
ypred_rf19=rf19.predict(xtest)
print('Train',f1_score(ytrain,rf19.predict(xtrain)))
print('Test',f1_score(ytest,ypred_rf19))
```

```
Train 0.8356471935853379
Test 0.8015873015873017
```

```
rf20=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3,class_weight={0:1,1:4})
rf20.fit(xtrain,ytrain)
ypred_rf20=rf20.predict(xtest)
print('Train',f1_score(ytrain,rf20.predict(xtrain)))
print('Test',f1_score(ytest,ypred_rf20))
```

```
Train 0.9994607420189819
Test 0.9082718057815041
```

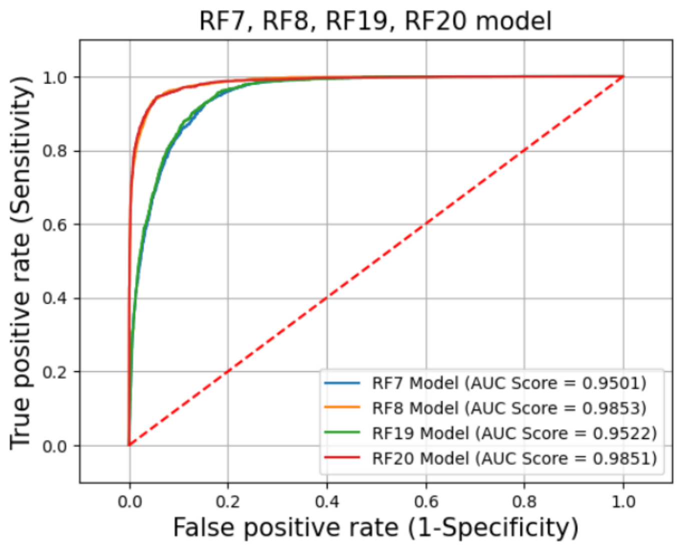
Each block of code trains a RandomForest model with specific hyperparameters or configurations and evaluates its performance using F1 scores on both the training and testing sets. Additionally, Models use specific subsets of features or have class weights assigned.



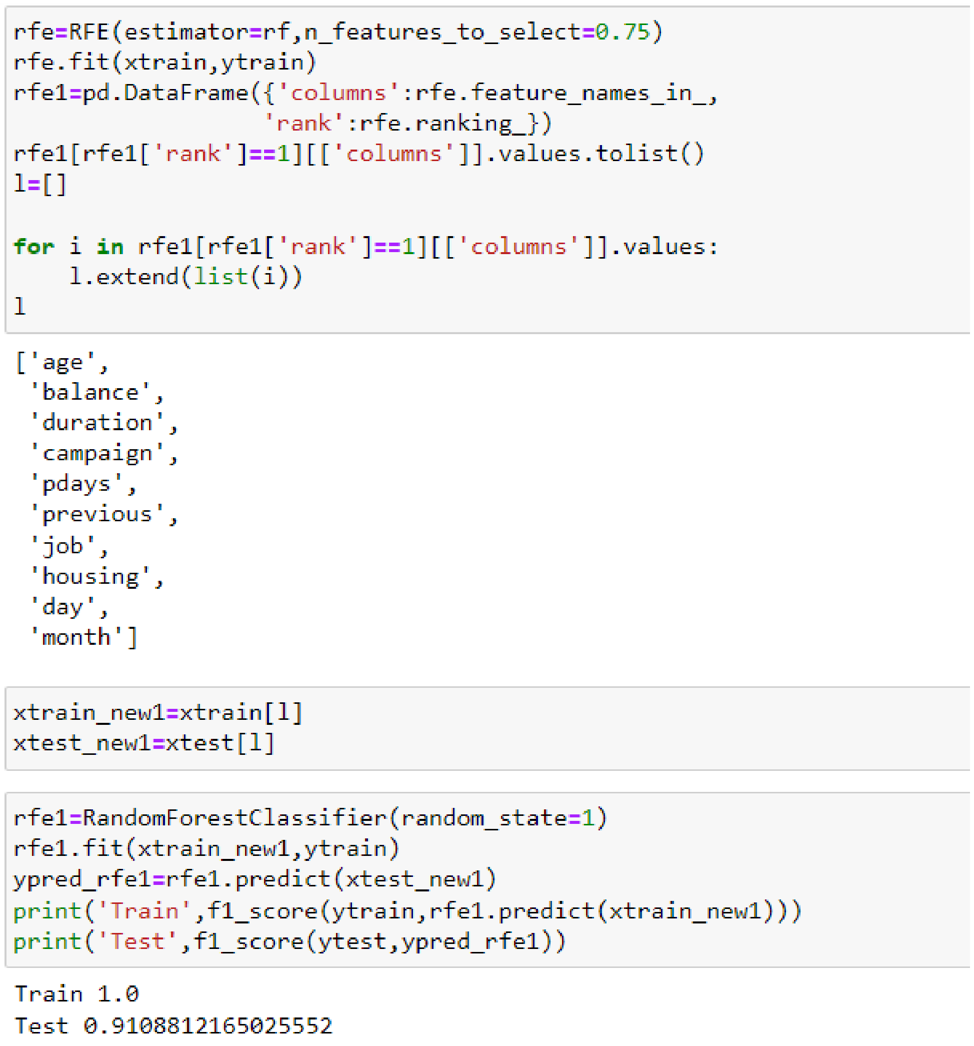
```
# rf7
#################
y_pred_prob_rf7=rf7.predict_proba(xtest)[:,1]
fpr,tpr,thresholds=roc_curve(ytest,y_pred_prob_rf7)
auc_score_rf7=roc_auc_score(ytest,y_pred_prob_rf7)
plt.plot(fpr,tpr,label='RF7 Model (AUC Score = %0.4f)'%auc_score_rf7)

#################
# rf8
y_pred_prob_rf8=rf8.predict_proba(xtest)[:,1]
fpr,tpr,thresholds=roc_curve(ytest,y_pred_prob_rf8)
auc_score_rf8=roc_auc_score(ytest,y_pred_prob_rf8)
plt.plot(fpr,tpr,label='RF8 Model (AUC Score = %0.4f)'%auc_score_rf8)

#################
# rf19
y_pred_prob_rf19=rf19.predict_proba(xtest)[:,1]
fpr,tpr,thresholds=roc_curve(ytest,y_pred_prob_rf19)
auc_score_rf19=roc_auc_score(ytest,y_pred_prob_rf19)
plt.plot(fpr,tpr,label='RF19 Model (AUC Score = %0.4f)'%auc_score_rf19)

###############
# rf20
y_pred_prob_rf20=rf20.predict_proba(xtest)[:,1]
fpr,tpr,thresholds=roc_curve(ytest,y_pred_prob_rf20)
auc_score_rf20=roc_auc_score(ytest,y_pred_prob_rf20)
plt.plot(fpr,tpr,label='RF20 Model (AUC Score = %0.4f)'%auc_score_rf20)

plt.xlim([-0.1, 1.1])
plt.ylim([-0.1, 1.1])
plt.plot([0, 1], [0, 1],'r--')

plt.title('RF7, RF8, RF19, RF20 model', fontsize = 15)
plt.xlabel('False positive rate (1-Specificity)', fontsize = 15)
plt.ylabel('True positive rate (Sensitivity)', fontsize = 15)

plt.legend(loc = 'lower right')
plt.grid(True)
```

<u>rf7, rf8, rf19, rf20: rf19 has less difference between train and test f1 scores</u>

Plotting ROC curves for different RandomForest models (rf7, rf8, rf19, and rf20) and calculating the Area Under the Curve (AUC) scores. ROC curves are useful for visualizing the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity) at various classification thresholds.



Using Recursive Feature Elimination with a RandomForestClassifier (rf) to select a subset of features and then training a new RandomForestClassifier (rfe1) on the selected features. Finally, we evaluate the performance of the model on both the training and testing sets, and update the models_report.



```
rfe=RFE(estimator=rf,n_features_to_select=0.75)
rfe.fit(xtrain,ytrain)
rfe1=pd.DataFrame({'columns':rfe.feature_names_in_,
                   'rank':rfe.ranking_})
rfe1[rfe1['rank']==1][['columns']].values.tolist()
l=[]

for i in rfe1[rfe1['rank']==1][['columns']].values:
    l.extend(list(i))
l
```

```
['age',
 'balance',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'job',
 'housing',
 'day',
 'month']
```

```
xtrain_new1=xtrain[l]
xtest_new1=xtest[l]
```

```
rfe1=RandomForestClassifier(random_state=1)
rfe1.fit(xtrain_new1,ytrain)
ypred_rfe1=rfe1.predict(xtest_new1)
print('Train',f1_score(ytrain,rfe1.predict(xtrain_new1)))
print('Test',f1_score(ytest,ypred_rfe1))
```

```
Train 1.0
Test 0.9108812165025552
```

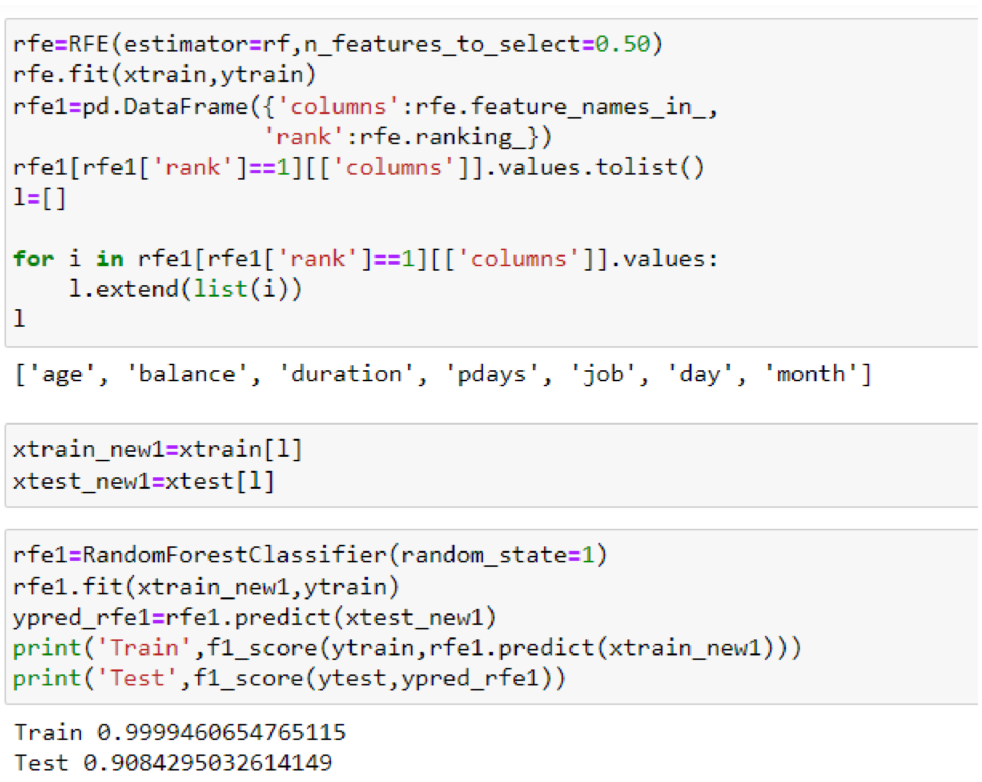
RFE is a feature selection technique, and the performance of the model may vary based on the selected subset of features. The code is designed to help identify a subset of features that contribute most to the model's performance

```
models_report
```

	model name	accuracy	recall	precision	f1
14	Random Forest rs 1 threshold 0.6	0.948623	0.902714	0.922312	0.912408
0	Random Forest rs1	0.946422	0.931388	0.892542	0.911551
1	Random Forest info gain feat selection	0.945208	0.934460	0.886783	0.909998
2	Random Forest sfs forward	0.945283	0.931388	0.889269	0.909841
3	RandomForest	0.945132	0.931388	0.888835	0.909614
4	Random Forest sfs backward	0.944828	0.931900	0.887588	0.909204
5	Random Forest rf importances 1	0.944297	0.929852	0.887586	0.908227
6	Random Forest with age removed	0.943766	0.929852	0.886070	0.907433
7	Decision Tree	0.927829	0.922171	0.847729	0.883384
8	Xgboost	0.910754	0.868408	0.836704	0.852261
9	KNN	0.860135	0.763953	0.764149	0.764051
10	GradientBoost	0.844502	0.716590	0.748196	0.732052
11	Logreg	0.809289	0.602919	0.709979	0.652084
12	Naive Bayes	0.774228	0.648490	0.612576	0.630021
13	GaussianNB	0.774228	0.648490	0.612576	0.630021

```
update_performance('Random Forest rfe 0.75',ytest,ypred_rfe1)
```

	model name	accuracy	recall	precision	f1
0	Random Forest rs 1 threshold 0.6	0.948623	0.902714	0.922312	0.912408
1	Random Forest rs1	0.946422	0.931388	0.892542	0.911551
15	Random Forest rfe 0.75	0.945739	0.935484	0.887539	0.910881
2	Random Forest info gain feat selection	0.945208	0.934460	0.886783	0.909998
3	Random Forest sfs forward	0.945283	0.931388	0.889269	0.909841
4	RandomForest	0.945132	0.931388	0.888835	0.909614
5	Random Forest sfs backward	0.944828	0.931900	0.887588	0.909204
6	Random Forest rf importances 1	0.944297	0.929852	0.887586	0.908227
7	Random Forest with age removed	0.943766	0.929852	0.886070	0.907433
8	Decision Tree	0.927829	0.922171	0.847729	0.883384
9	Xgboost	0.910754	0.868408	0.836704	0.852261
10	KNN	0.860135	0.763953	0.764149	0.764051
11	GradientBoost	0.844502	0.716590	0.748196	0.732052
12	Logreg	0.809289	0.602919	0.709979	0.652084
13	Naive Bayes	0.774228	0.648490	0.612576	0.630021
14	GaussianNB	0.774228	0.648490	0.612576	0.630021



```
rfe=RFE(estimator=rf,n_features_to_select=0.50)
rfe.fit(xtrain,ytrain)
rfe1=pd.DataFrame({'columns':rfe.feature_names_in_,
                  'rank':rfe.ranking_})
rfe1[rfe1['rank']==1][['columns']].values.tolist()
l=[]

for i in rfe1[rfe1['rank']==1][['columns']].values:
    l.extend(list(i))
l
```

```
['age', 'balance', 'duration', 'pdays', 'job', 'day', 'month']
```

```
xtrain_new1=xtrain[l]
xtest_new1=xtest[l]
```

```
rfe1=RandomForestClassifier(random_state=1)
rfe1.fit(xtrain_new1,ytrain)
ypred_rfe1=rfe1.predict(xtest_new1)
print('Train',f1_score(ytrain,rfe1.predict(xtrain_new1)))
print('Test',f1_score(ytest,ypred_rfe1))
```

```
Train 0.9999460654765115
Test 0.9084295032614149
```

A high F1 score on the training set indicates that the model is performing exceptionally well on the data it was trained on. However, it's important to carefully assess the testing set's performance to ensure that the model generalizes well to new data. The relatively high F1 score on the testing set (0.9084) suggests that the model is performing well and has good predictive capabilities.

Final model

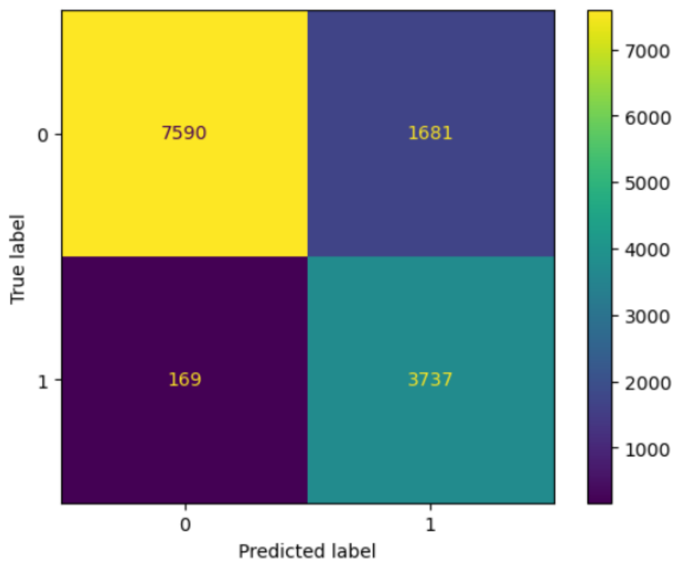
Model Evaluation and Feature Importance for rf19

```
]: print(classification_report(ytest,ypred_rf19))
```

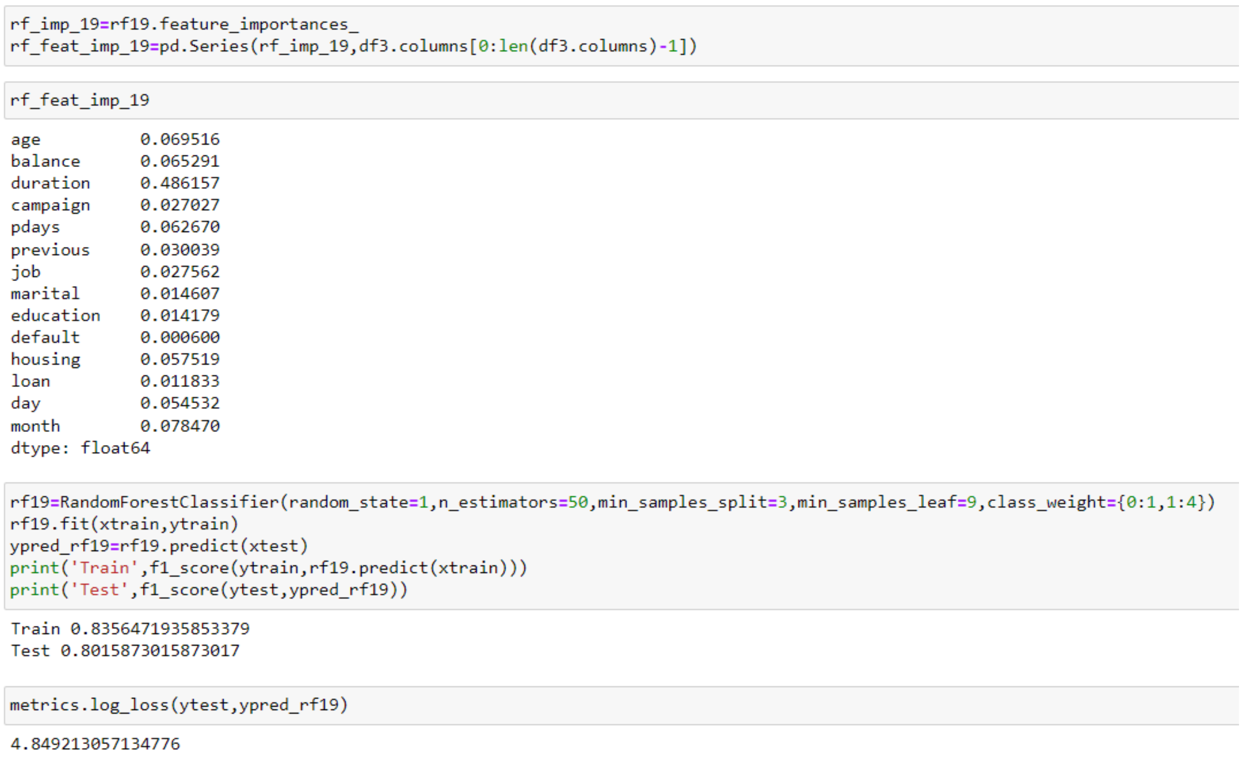
```
              precision    recall  f1-score   support

           0       0.98      0.82      0.89      9271
           1       0.69      0.96      0.80      3906

    accuracy                           0.86     13177
   macro avg       0.83      0.89      0.85     13177
weighted avg       0.89      0.86      0.86     13177
```

Overall, the model appears to perform well, with high precision and recall for both classes.

```
ConfusionMatrixDisplay.from_predictions(ytest,ypred_rf19)
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21fe6f051f0>
```



- **Accuracy:** 0.86 indicates the model correctly classifies 86% of the instances overall.
- **Weighted Average F1-Score:** 0.86 suggests a good balance between precision and recall, considering class distribution.
- The model performs better for class 0, with higher precision and F1-score.
- It's very good at identifying true positives for class 1 (high recall), but has a lower precision for class 1, meaning more false positives.



```
rf_imp_19=rf19.feature_importances_
rf_feat_imp_19=pd.Series(rf_imp_19,df3.columns[0:len(df3.columns)-1])
```

```
rf_feat_imp_19
```

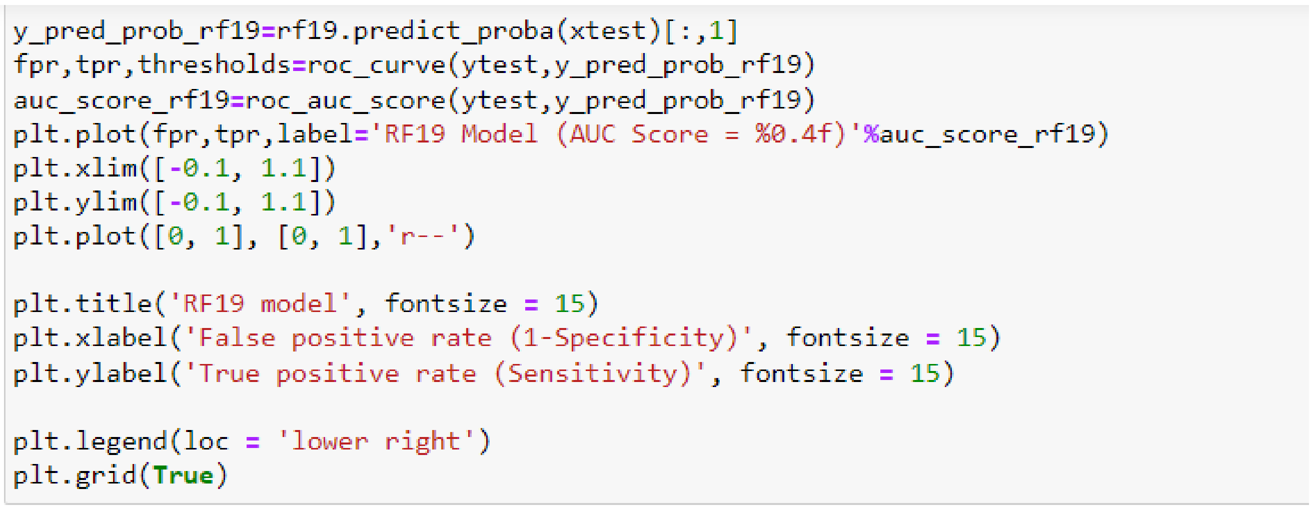
```
age          0.069516
balance      0.065291
duration     0.486157
campaign     0.027027
pdays        0.062670
previous     0.030039
job          0.027562
marital      0.014607
education    0.014179
default      0.000600
housing      0.057519
loan         0.011833
day          0.054532
month        0.078470
dtype: float64
```

```
rf19=RandomForestClassifier(random_state=1,n_estimators=50,min_samples_split=3,min_samples_leaf=9,class_weight={0:1,1:4})
rf19.fit(xtrain,ytrain)
ypred_rf19=rf19.predict(xtest)
print('Train',f1_score(ytrain,rf19.predict(xtrain)))
print('Test',f1_score(ytest,ypred_rf19))
```

```
Train 0.8356471935853379
Test 0.8015873015873017
```

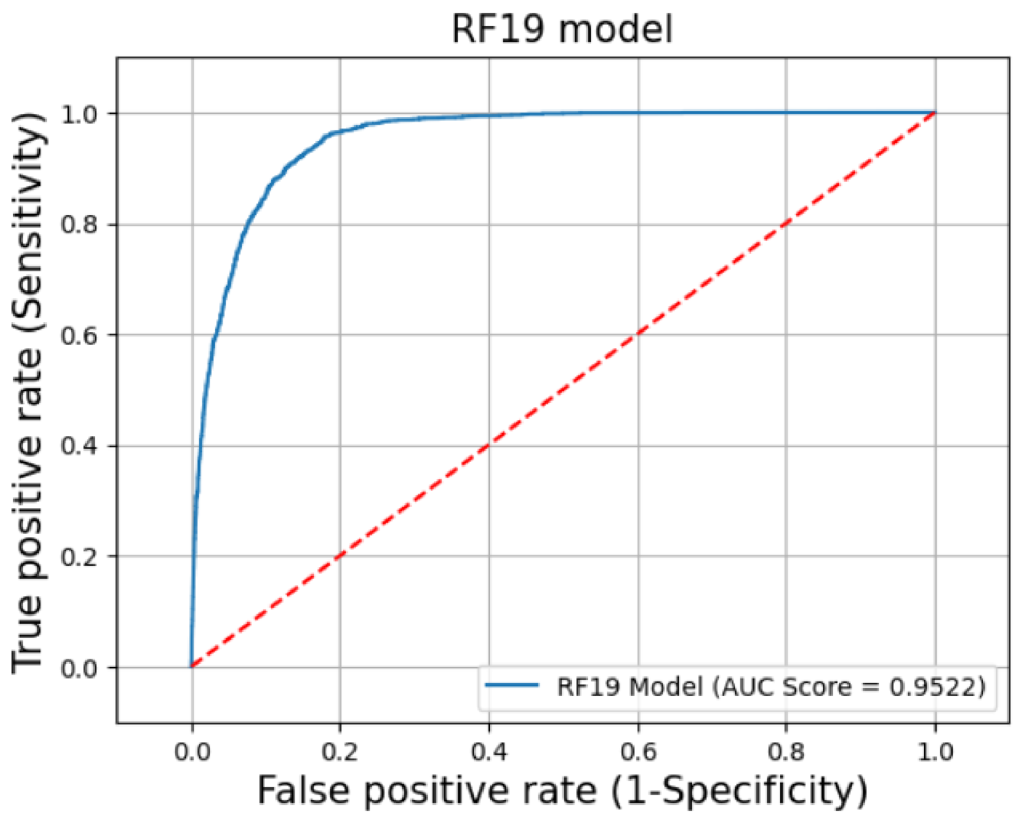
```
metrics.log_loss(ytest,ypred_rf19)
```

```
4.849213057134776
```



```
y_pred_prob_rf19=rf19.predict_proba(xtest)[:,1]
fpr,tpr,thresholds=roc_curve(ytest,y_pred_prob_rf19)
auc_score_rf19=roc_auc_score(ytest,y_pred_prob_rf19)
plt.plot(fpr,tpr,label='RF19 Model (AUC Score = %0.4f)'%auc_score_rf19)
plt.xlim([-0.1, 1.1])
plt.ylim([-0.1, 1.1])
plt.plot([0, 1], [0, 1],'r--')

plt.title('RF19 model', fontsize = 15)
plt.xlabel('False positive rate (1-Specificity)', fontsize = 15)
plt.ylabel('True positive rate (Sensitivity)', fontsize = 15)

plt.legend(loc = 'lower right')
plt.grid(True)
```

The ROC curve helps visualize the trade-off between sensitivity and specificity, and the **AUC score** quantifies the overall performance of the model in distinguishing between the two classes.



The RF19 model has an AUC score of 0.9522 which is a good score. At an FPR of 0.1, the TPR is about 0.8. This means that the model is correctly classifying about 80% of data, but it is also incorrectly classifying 10% of incorrect information.

Overall, the ROC curve shows that the RF19 model is a good model.

Classification Conclusion:

1. Model Evaluation using F1 Score:

   The F1 scores on both the training and test sets are high (approximately 0.836 for training and 0.802 for the test set). This indicates a good balance between precision and recall. The model's effectiveness on the test set suggests that it generalizes well to new, unseen data.

2. Generalization and Overfitting:

   The similar F1 scores on the training and test sets suggest that the model is not overfitting. It generalizes well to new data, reinforcing its reliability.

3. Random Forest Configuration:

   The RandomForestClassifier is configured with specific hyperparameters, including n_estimators=50, min_samples_split=3, min_samples_leaf=9, and class_weight={0: 1, 1: 4}.These hyperparameters indicate the use of an ensemble of 50 decision trees, with minimum samples required for splitting an internal node set to 3, minimum samples required to be a leaf node set to 9, and a class weight adjustment to handle class imbalance.

4. Area Under the ROC Curve (AUC-ROC):

   The AUC-ROC score of 0.9522 is close to the maximum value of 1.0, indicating excellent discrimination ability. This suggests that the model effectively distinguishes between the positive and negative classes.

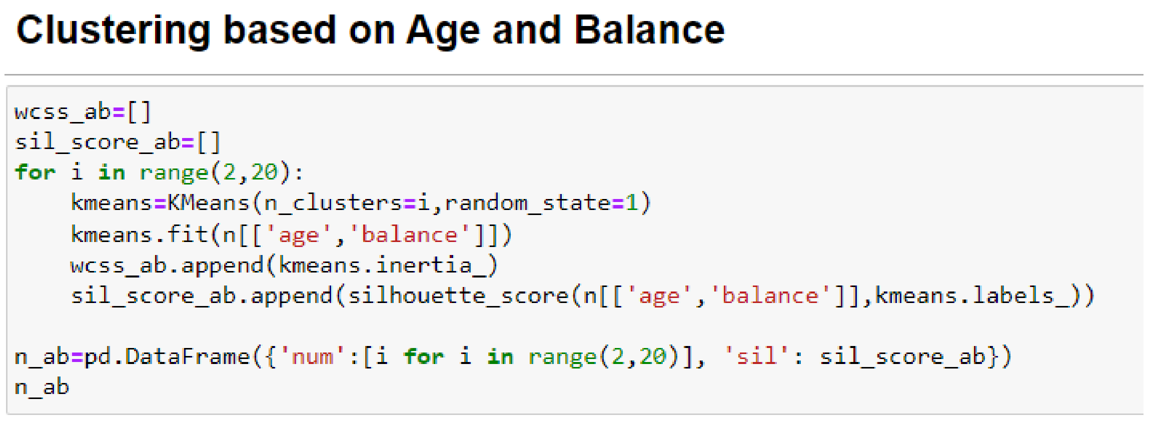
5. Feature Importance:

   Duration is identified as the most influential feature with the highest importance score (0.486). This implies that the duration of the interaction significantly influences the model's predictions. Other notable features include month (0.078), age (0.069), and balance (0.065).

6. F1 Score for Class 1:

   An F1-score of 0.80 for class 1 suggests a good balance between precision and recall for predicting positive instances. This is a valuable metric, especially in scenarios with imbalanced classes.

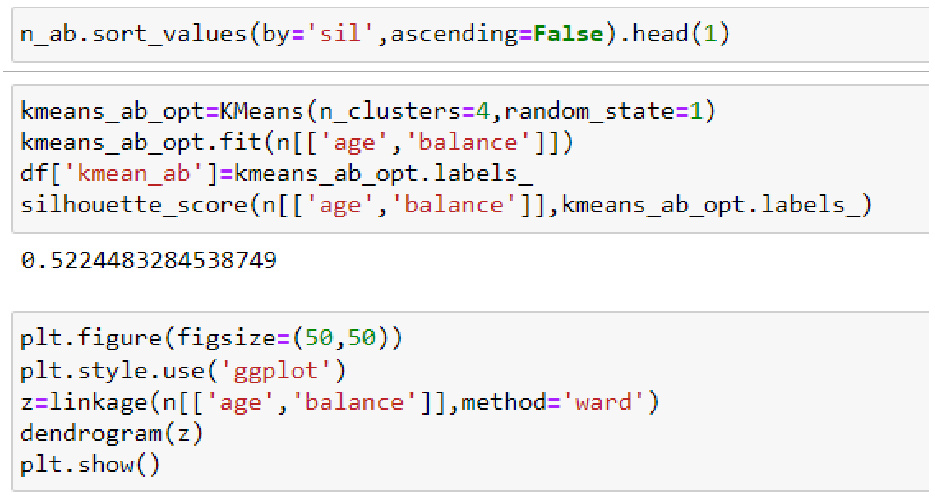
USL:



Clustering based on Age and Balance

```
wcss_ab=[]
sil_score_ab=[]
for i in range(2,20):
    kmeans=KMeans(n_clusters=i,random_state=1)
    kmeans.fit(n[['age','balance']])
    wcss_ab.append(kmeans.inertia_)
    sil_score_ab.append(silhouette_score(n[['age','balance']],kmeans.labels_))

n_ab=pd.DataFrame({'num':[i for i in range(2,20)], 'sil': sil_score_ab})
n_ab
```

Using the KMeans algorithm to perform clustering on a subset of features ('age' and 'balance') and evaluating the clustering results using Within-Cluster Sum of Squares (WCSS) and Silhouette Score. The code iterates through different numbers of clusters and stores the corresponding WCSS and Silhouette Score values.



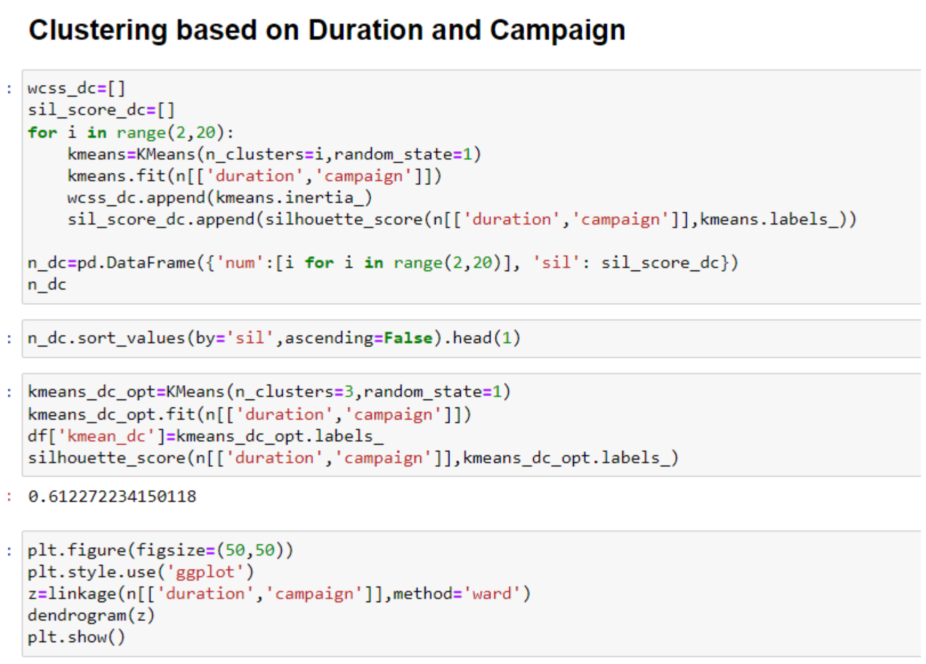
```
n_ab.sort_values(by='sil',ascending=False).head(1)
```

```
kmeans_ab_opt=KMeans(n_clusters=4,random_state=1)
kmeans_ab_opt.fit(n[['age','balance']])
df['kmean_ab']=kmeans_ab_opt.labels_
silhouette_score(n[['age','balance']],kmeans_ab_opt.labels_)
```

```
0.5224483284538749
```

```
plt.figure(figsize=(50,50))
plt.style.use('ggplot')
z=linkage(n[['age','balance']],method='ward')
dendrogram(z)
plt.show()
```

code explores different numbers of clusters, evaluates their performance using Silhouette Scores, identifies the optimal number of clusters, performs KMeans clustering with the optimal number of clusters, and visualizes the clusters using a dendrogram. The specific optimal number of clusters is determined by the highest Silhouette Score


Clustering based on Duration and Campaign

```
: wcss_dc=[]
  sil_score_dc=[]
  for i in range(2,20):
      kmeans=KMeans(n_clusters=i,random_state=1)
      kmeans.fit(n[['duration','campaign']])
      wcss_dc.append(kmeans.inertia_)
      sil_score_dc.append(silhouette_score(n[['duration','campaign']],kmeans.labels_))

  n_dc=pd.DataFrame({'num':[i for i in range(2,20)], 'sil': sil_score_dc})
  n_dc
```

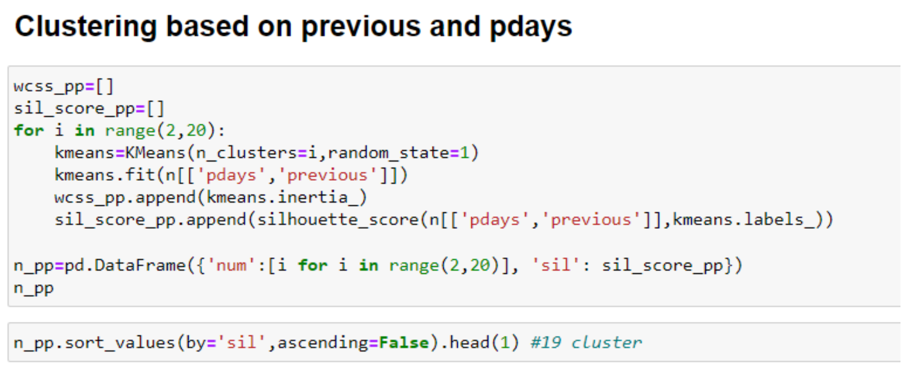
```
: n_dc.sort_values(by='sil',ascending=False).head(1)
```

```
: kmeans_dc_opt=KMeans(n_clusters=3,random_state=1)
  kmeans_dc_opt.fit(n[['duration','campaign']])
  df['kmean_dc']=kmeans_dc_opt.labels_
  silhouette_score(n[['duration','campaign']],kmeans_dc_opt.labels_)
```

```
: 0.612272234150118
```

```
: plt.figure(figsize=(50,50))
  plt.style.use('ggplot')
  z=linkage(n[['duration','campaign']],method='ward')
  dendrogram(z)
  plt.show()
```

This code explores different numbers of clusters based on the features 'duration' and 'campaign', identifies the optimal number of clusters using Silhouette Scores, performs KMeans clustering with the optimal number of clusters, and visualizes the clusters using a dendrogram. The specific optimal number of clusters is determined by the highest Silhouette Score.


Clustering based on previous and pdays

```
wcss_pp=[]
sil_score_pp=[]
for i in range(2,20):
    kmeans=KMeans(n_clusters=i,random_state=1)
    kmeans.fit(n[['pdays','previous']])
    wcss_pp.append(kmeans.inertia_)
    sil_score_pp.append(silhouette_score(n[['pdays','previous']],kmeans.labels_))

n_pp=pd.DataFrame({'num':[i for i in range(2,20)], 'sil': sil_score_pp})
n_pp
```

```
n_pp.sort_values(by='sil',ascending=False).head(1) #19 cluster
```

Focusing on clustering based on the features 'pdays' and 'previous'.
In this case, the optimal number of clusters is determined to be 19, based on the highest Silhouette Score

KMeans Clustering on all scaled numbers

```
wcss_full=[]
sil_score_full=[]
for i in range(2,20):
    kmeans=KMeans(n_clusters=i,random_state=1)
    kmeans.fit(n)
    wcss_full.append(kmeans.inertia_)
    sil_score_full.append(silhouette_score(n,kmeans.labels_))

n_full=pd.DataFrame({'num':[i for i in range(2,20)], 'sil': sil_score_full})
n_full
```

In this case, the optimal number of clusters is determined to be 2, based on the highest Silhouette Score.

```
n_full.sort_values(by='sil',ascending=False).head(1)
```

	num	sil
0	2	0.334734

```
kmean_full_opt=KMeans(n_clusters=2,random_state=1)
kmean_full_opt.fit(n)
df['kmean_full']=kmean_full_opt.labels_
silhouette_score(n,kmean_full_opt.labels_)
```

```
0.3347341595877947
```

A score around 0.335 suggests a reasonable separation, but it's not extremely strong

Creating PCA and performing clustering using KMeans

```
cov_m=np.cov(np.transpose(n))
```

```
eig_val,eig_vec=np.linalg.eig(cov_m)
```

```
eig_vec.dot(np.transpose(n))
```

```
array([[-0.186191  ,  0.16362551,  0.38837083, ..., -2.08469686,
        -0.89379197,  2.59461957],
       [-0.11361761,  0.15400783,  0.32639253, ..., -2.69482512,
        -0.42207971, -3.21487719],
       [ 0.21200425,  0.65234628,  0.9708434 , ..., -1.93892906,
        -0.14310314, -1.09512422],
       ...,
       [-0.03285631,  0.41051379,  0.71190306, ..., -0.11541446,
        -0.85513952,  1.5505225 ],
       [ 1.13889785,  0.75950873,  0.07850512, ...,  0.86689758,
         1.15684317, -0.63488791],
       [ 1.4995064 ,  0.20772838, -0.43798603, ...,  2.7041625 ,
         0.71023697,  0.33468172]])
```

Performing Principal Component Analysis (PCA) on the dataset n. The code you provided calculates the covariance matrix, eigenvalues, and eigenvectors, and then projects the data onto the principal components.

```
pc=pd.DataFrame(eig_vec.dot(np.transpose(n)).T,columns=['PC'+str(i) for i in range(7)])
pc
```

	PC0	PC1	PC2	PC3	PC4	PC5	PC6
0	-0.186191	-0.113618	0.212004	-1.118914	-0.032856	1.138898	1.499506
1	0.163626	0.154008	0.652346	-1.194386	0.410514	0.759509	0.207728
2	0.388371	0.326393	0.970843	-1.263648	0.711903	0.078505	-0.437986
3	0.298407	0.314546	0.585204	-1.194050	0.329875	0.605820	0.745371
4	0.060526	-0.002123	0.910435	-1.250598	0.652371	0.071970	-0.478224
...	...	...	...	...	...	...	...
45206	-2.065581	-1.750152	-0.377462	0.102834	-0.983921	0.700768	0.195300
45207	-0.639825	-0.345440	-0.795649	-0.110501	-1.272547	1.759268	1.745358
45208	-2.084697	-2.694825	-1.938929	1.499035	-0.115414	0.866898	2.704162
45209	-0.893792	-0.422080	-0.143103	0.350413	-0.855140	1.156843	0.710237
45210	2.594620	-3.214877	-1.095124	1.453661	1.550522	-0.634888	0.334682

45211 rows × 7 columns

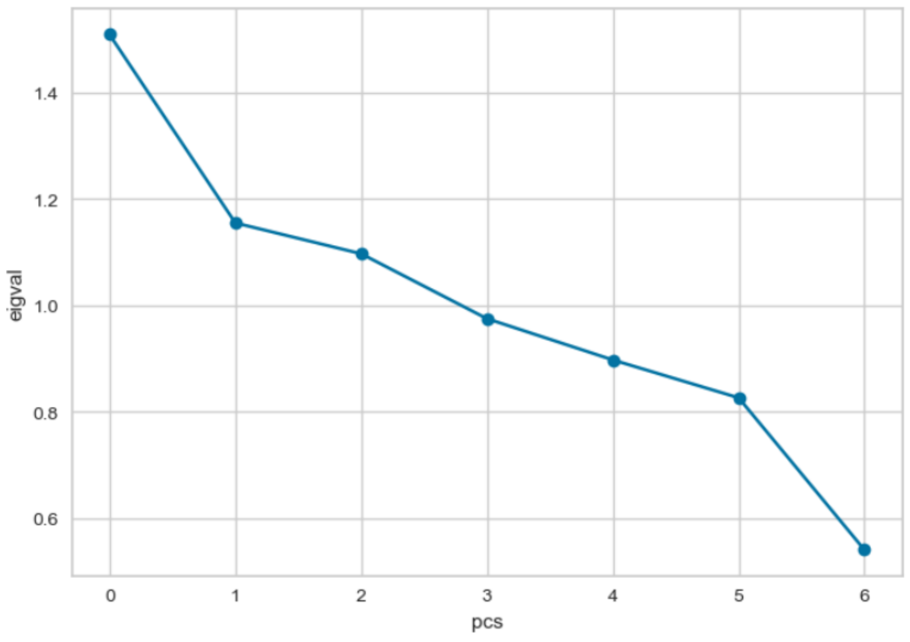
```
pc.iloc[:,[0,2,3]].var() #kaiser criterion
```

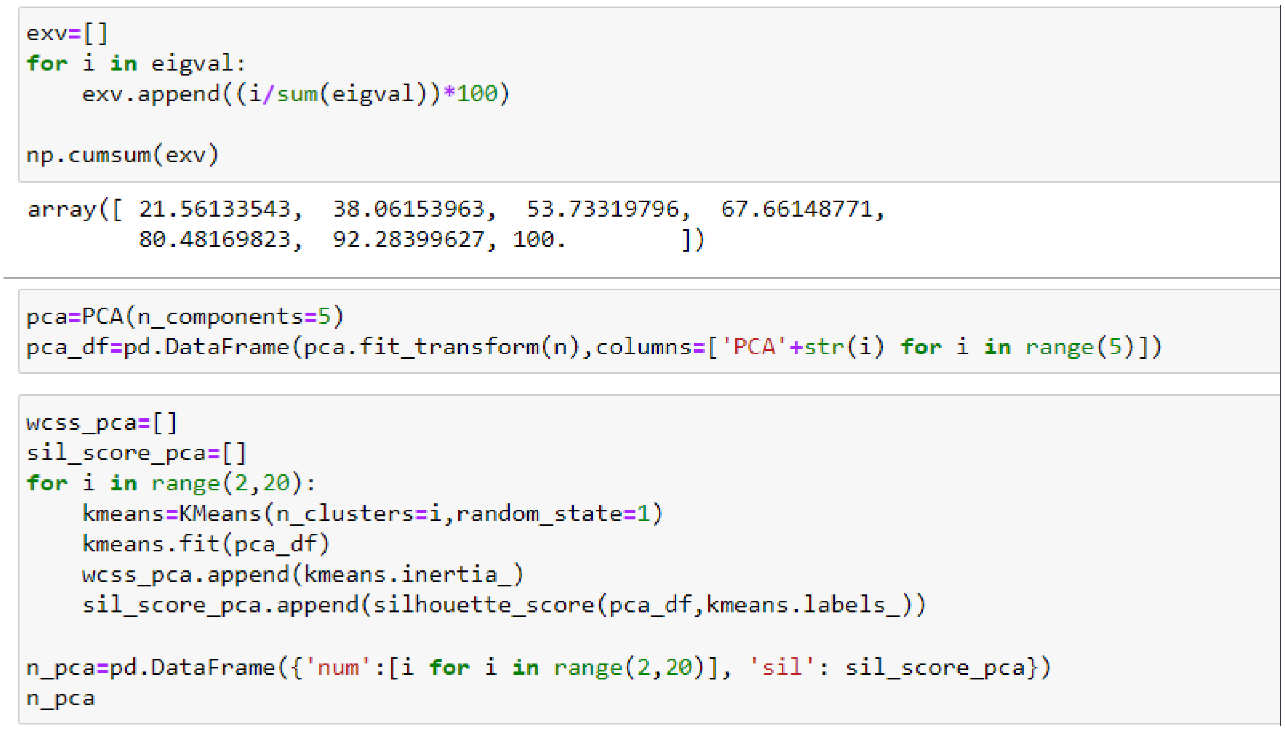
```
PC0    0.896101
PC2    0.907614
PC3    1.092278
dtype: float64
```

```
eigval=eig_val.tolist() # Scree plot
eigval.sort(reverse=True)
```

```
plt.plot(eigval,marker='o')
plt.xlabel('pcs')
plt.ylabel('eigval')
plt.show()
```

Creating a Scree plot to visualize the eigenvalues obtained from the PCA. The code converts the eigenvalues to a list, sorts them in descending order





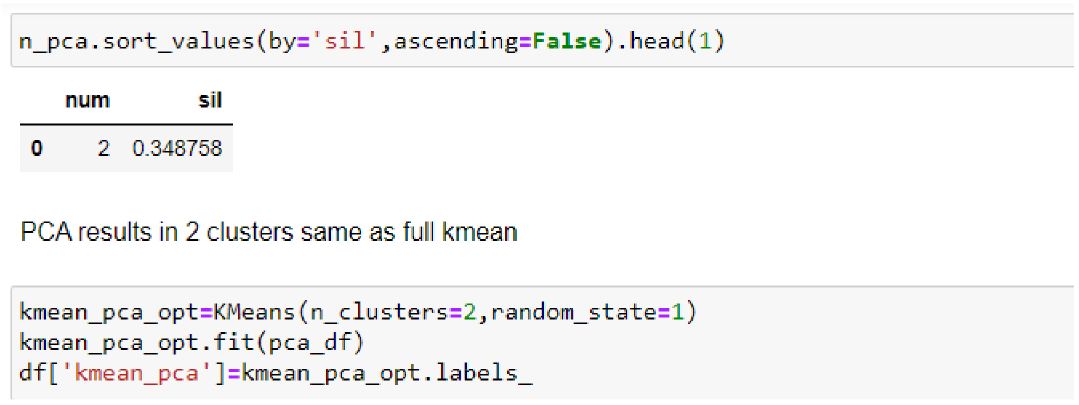
```
exv=[]
for i in eigval:
    exv.append((i/sum(eigval))*100)

np.cumsum(exv)
```

```
array([ 21.56133543,  38.06153963,  53.73319796,  67.66148771,
        80.48169823,  92.28399627, 100.        ])
```

```
pca=PCA(n_components=5)
pca_df=pd.DataFrame(pca.fit_transform(n),columns=['PCA'+str(i) for i in range(5)])
```

```
wcss_pca=[]
sil_score_pca=[]
for i in range(2,20):
    kmeans=KMeans(n_clusters=i,random_state=1)
    kmeans.fit(pca_df)
    wcss_pca.append(kmeans.inertia_)
    sil_score_pca.append(silhouette_score(pca_df,kmeans.labels_))

n_pca=pd.DataFrame({'num':[i for i in range(2,20)], 'sil': sil_score_pca})
n_pca
```

Calculating the explained variance for each principal component and then performing k-means clustering on a dataset transformed using PCA



```
n_pca.sort_values(by='sil',ascending=False).head(1)
```

	num	sil
0	2	0.348758

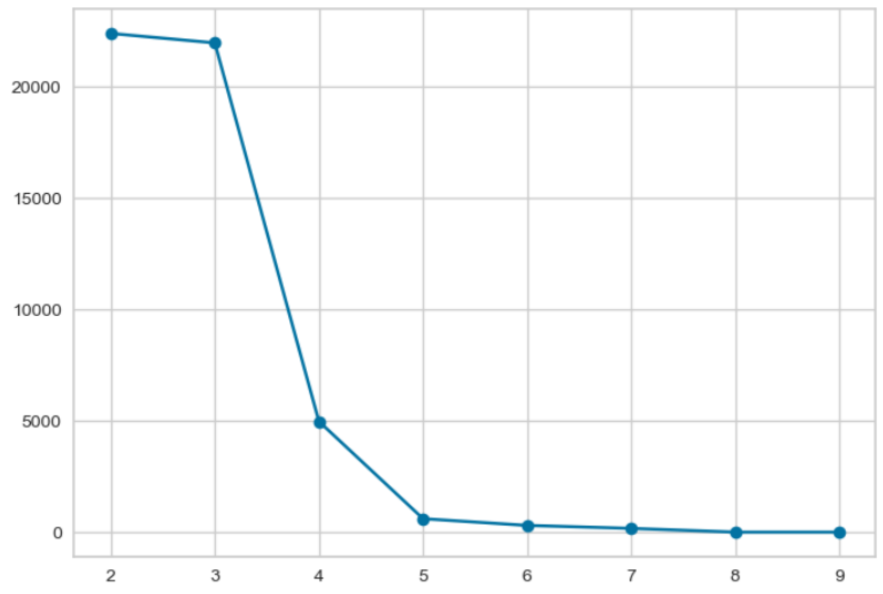
PCA results in 2 clusters same as full kmean

```
kmean_pca_opt=KMeans(n_clusters=2,random_state=1)
kmean_pca_opt.fit(pca_df)
df['kmean_pca']=kmean_pca_opt.labels_
```

KModes for default, housing, loan

```
cat=df1.select_dtypes(object)
co=[]
for i in range(2,10):
    kmode=KModes(n_clusters=i,random_state=1)
    kmode.fit_predict(cat[['default','housing','loan']])
    co.append(kmode.cost_)

plt.plot(range(2,10),co,marker='o')

plt.show()
```

Using the k-modes algorithm to perform clustering on categorical data and plotting the cost for different numbers of clusters.



The plot suggests that 5 or 6 clusters might be a reasonable choice for this dataset, as further increasing clusters doesn't substantially reduce the cost.

```
cat=df1.select_dtypes(object)
kmode_opt=KModes(n_clusters=4,random_state=1)
kmode_opt.fit_predict(cat[['default','housing','loan']])
df['kmode_opt']=kmode_opt.labels_
```

```
gower_dist_matrix = gower_matrix(cat)
gower_dist_matrix
```

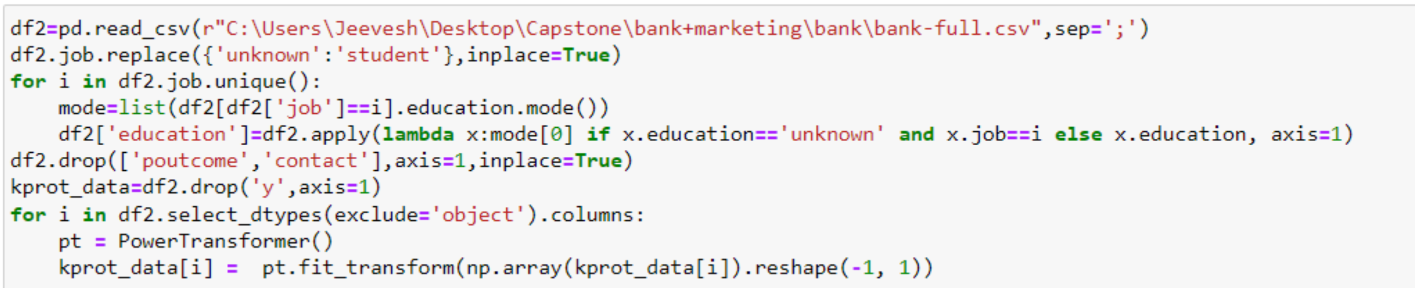
```
array([[0.        , 0.42857143, 0.42857143, ..., 0.5714286 , 0.5714286 ,
        0.5714286 ],
       [0.42857143, 0.        , 0.42857143, ..., 0.5714286 , 0.5714286 ,
        0.5714286 ],
       [0.42857143, 0.42857143, 0.        , ..., 0.5714286 , 0.5714286 ,
        0.42857143],
       ...,
       [0.5714286 , 0.5714286 , 0.5714286 , ..., 0.        , 0.14285715,
        0.14285715],
       [0.5714286 , 0.5714286 , 0.5714286 , ..., 0.14285715, 0.        ,
        0.14285715],
       [0.5714286 , 0.5714286 , 0.42857143, ..., 0.14285715, 0.14285715,
        0.        ]], dtype=float32)
```

```
silhouette_score(gower_dist_matrix,kmode_opt.labels_)
```

```
0.21880722
```

Using the KModes algorithm to cluster categorical data with three features ('default', 'housing', 'loan') into four clusters. Additionally, calculating the Gower distance matrix using the gower_matrix function on the categorical data.

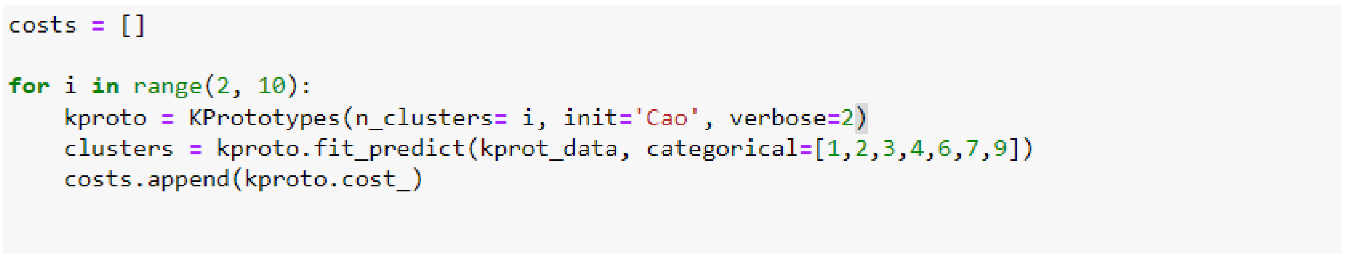
A silhouette score of approximately 0.2188, which suggests that the clusters are reasonably well-separated and have a meaningful structure.



```
df2=pd.read_csv(r"C:\Users\Jeevesh\Desktop\Capstone\bank+marketing\bank\bank-full.csv",sep=';')
df2.job.replace({'unknown':'student'},inplace=True)
for i in df2.job.unique():
    mode=list(df2[df2['job']==i].education.mode())
    df2['education']=df2.apply(lambda x:mode[0] if x.education=='unknown' and x.job==i else x.education, axis=1)
df2.drop(['poutcome','contact'],axis=1,inplace=True)
kprot_data=df2.drop('y',axis=1)
for i in df2.select_dtypes(exclude='object').columns:
    pt = PowerTransformer()
    kprot_data[i] =  pt.fit_transform(np.array(kprot_data[i]).reshape(-1, 1))
```

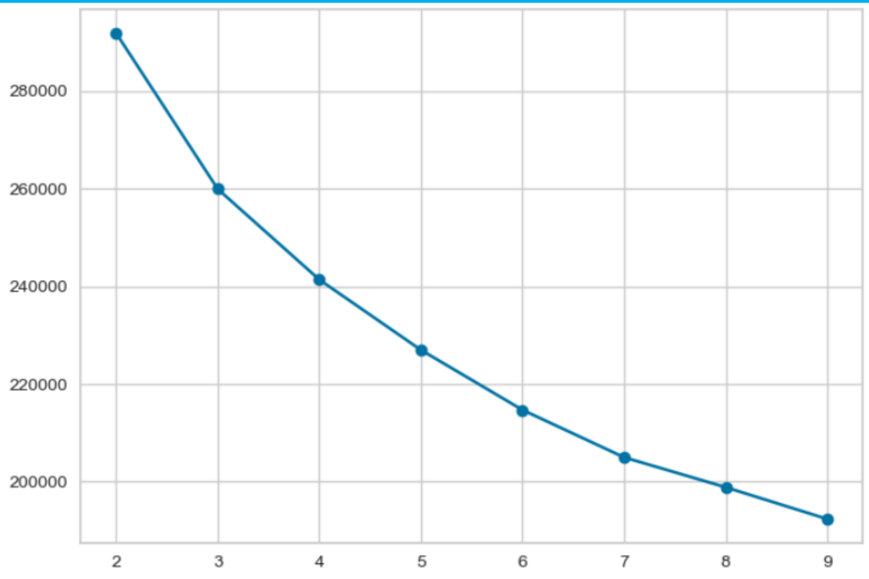
Preparing the data for the K-Prototypes algorithm, which is suitable for clustering datasets with a mix of categorical and numerical features.

K-Prototypes



```
costs = []

for i in range(2, 10):
    kproto = KPrototypes(n_clusters= i, init='Cao', verbose=2)
    clusters = kproto.fit_predict(kprot_data, categorical=[1,2,3,4,6,7,9])
    costs.append(kproto.cost_)
```

Using the K-Prototypes algorithm to perform clustering on the data (kprot_data). The code iterates over a range of cluster numbers (from 2 to 9) and calculates the cost for each configuration. The cost is a measure of how well the algorithm is able to cluster the data



```
kproto = KPrototypes(n_clusters= 4, init='Cao', verbose=2)
kproto.fit_predict(kprot_data, categorical=[1,2,3,4,6,7,9])
df['kproto_full']=kproto.labels_
```

```
gower_dist_matrix_kpro = gower_matrix(kprot_data)
gower_dist_matrix_kpro
```

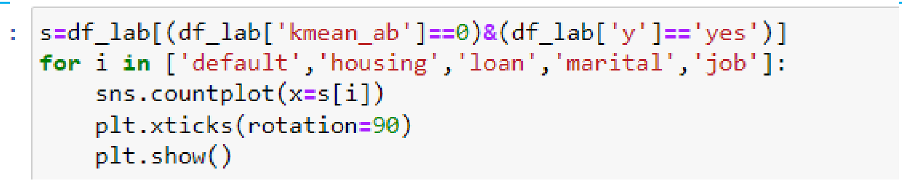
```
silhouette_score()
```

```
filename="labeled.xlsx"
df.to_excel(filename)
print('Excel created successfully')
```

```
Excel created successfully
```

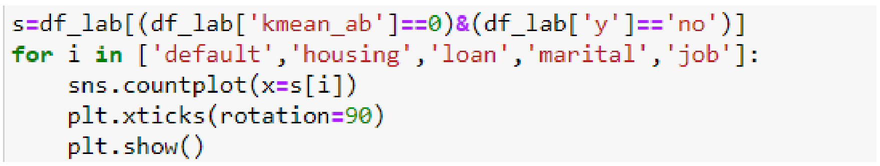
```
df_lab=pd.read_excel("labeled.xlsx",index_col=[0])
df_lab.head()
```

Using the K-Prototypes algorithm with 4 clusters and assigning cluster labels to the DataFrame . Additionally calculating the Gower distance matrix for the data and aiming to compute the silhouette score

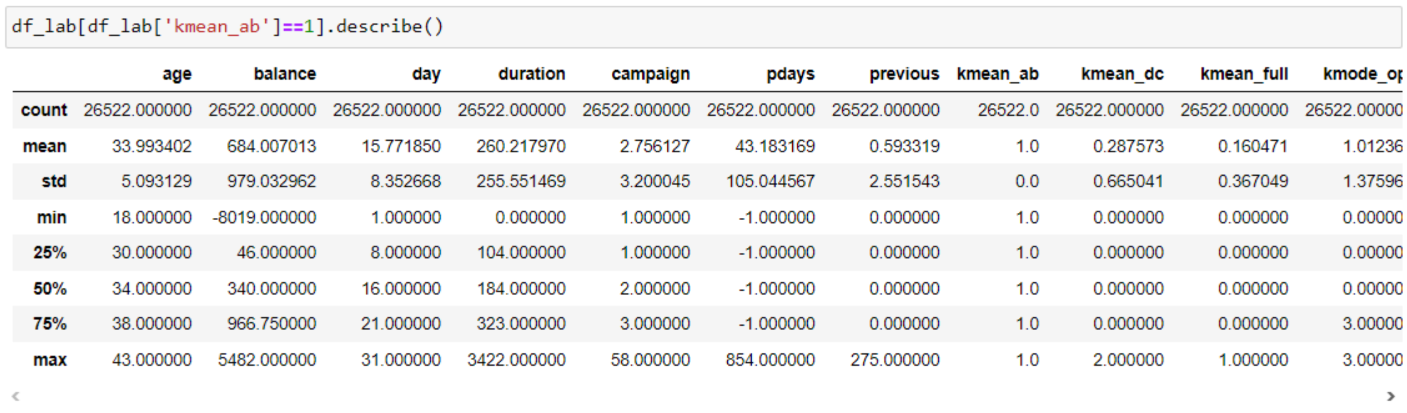


```
: s=df_lab[(df_lab['kmean_ab']==0)&(df_lab['y']=='yes')]
  for i in ['default','housing','loan','marital','job']:
      sns.countplot(x=s[i])
      plt.xticks(rotation=90)
      plt.show()
```

Creating count plots to visualize the distribution of categorical variables within a specific cluster (kmean_ab == 0) where the target variable is ‘yes’. Plotting the counts for variables such as 'default', 'housing', 'loan', 'marital', and 'job'. These count plots can help us understand the distribution of these categorical variables within the specified cluster and identify any patterns or trends.

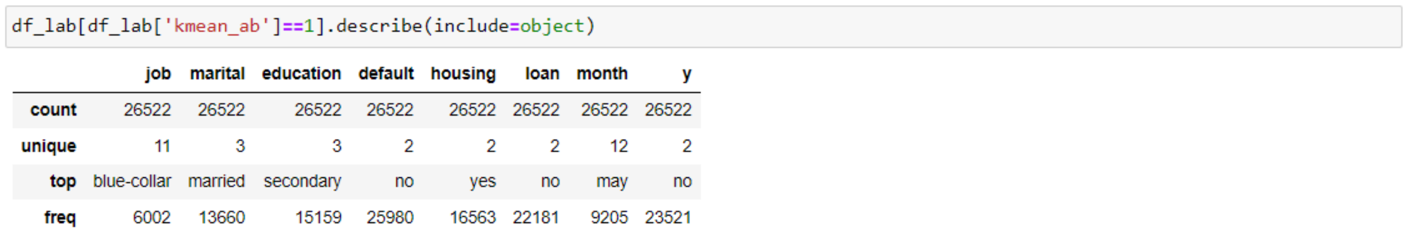


```
s=df_lab[(df_lab['kmean_ab']==0)&(df_lab['y']=='no')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```



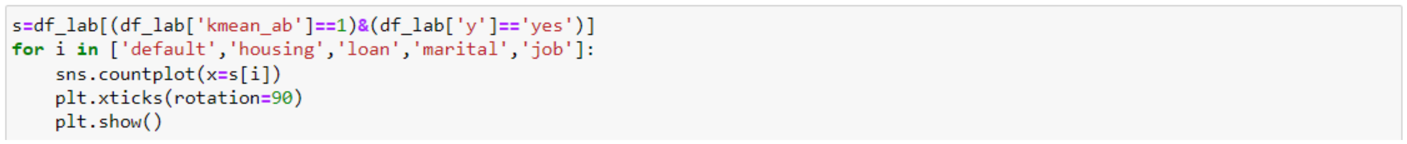
```
df_lab[df_lab['kmean_ab']==1].describe()
```

	age	balance	day	duration	campaign	pdays	previous	kmean_ab	kmean_dc	kmean_full	kmode_op
count	26522.000000	26522.000000	26522.000000	26522.000000	26522.000000	26522.000000	26522.000000	26522.0	26522.000000	26522.000000	26522.00000
mean	33.993402	684.007013	15.771850	260.217970	2.756127	43.183169	0.593319	1.0	0.287573	0.160471	1.01236
std	5.093129	979.032962	8.352668	255.551469	3.200045	105.044567	2.551543	0.0	0.665041	0.367049	1.37596
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000	1.0	0.000000	0.000000	0.00000
25%	30.000000	46.000000	8.000000	104.000000	1.000000	-1.000000	0.000000	1.0	0.000000	0.000000	0.00000
50%	34.000000	340.000000	16.000000	184.000000	2.000000	-1.000000	0.000000	1.0	0.000000	0.000000	0.00000
75%	38.000000	966.750000	21.000000	323.000000	3.000000	-1.000000	0.000000	1.0	0.000000	0.000000	3.00000
max	43.000000	5482.000000	31.000000	3422.000000	58.000000	854.000000	275.000000	1.0	2.000000	1.000000	3.00000

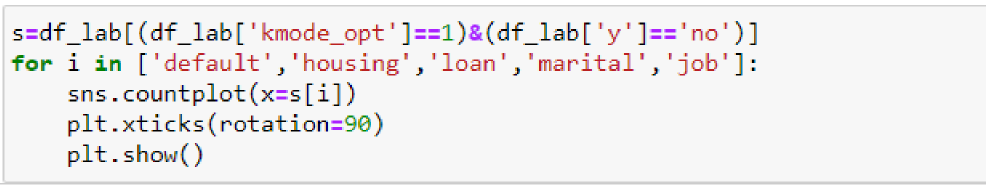


```
df_lab[df_lab['kmean_ab']==1].describe(include=object)
```

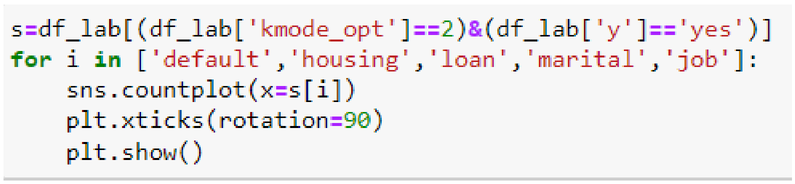
	job	marital	education	default	housing	loan	month	y
count	26522	26522	26522	26522	26522	26522	26522	26522
unique	11	3	3	2	2	2	12	2
top	blue-collar	married	secondary	no	yes	no	may	no
freq	6002	13660	15159	25980	16563	22181	9205	23521



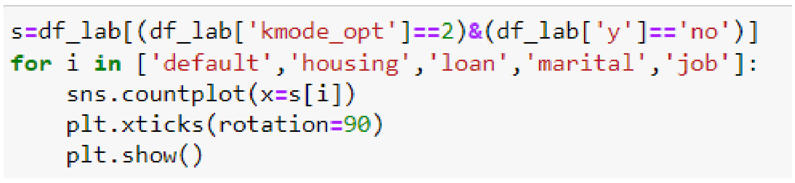
```
s=df_lab[(df_lab['kmean_ab']==1)&(df_lab['y']=='yes')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```



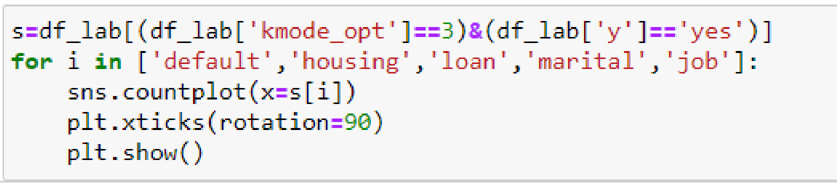
```
s=df_lab[(df_lab['kmode_opt']==1)&(df_lab['y']=='no')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```



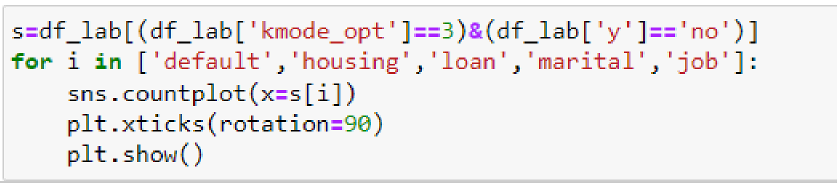
```
s=df_lab[(df_lab['kmode_opt']==2)&(df_lab['y']=='yes')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```



```
s=df_lab[(df_lab['kmode_opt']==2)&(df_lab['y']=='no')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```

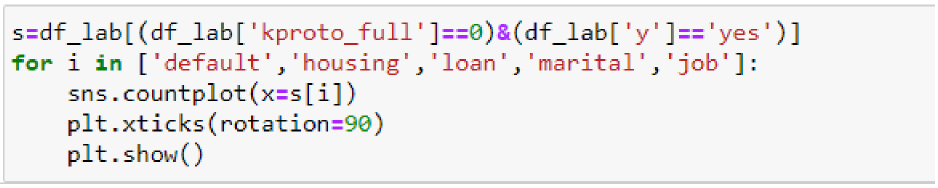


```
s=df_lab[(df_lab['kmode_opt']==3)&(df_lab['y']=='yes')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```

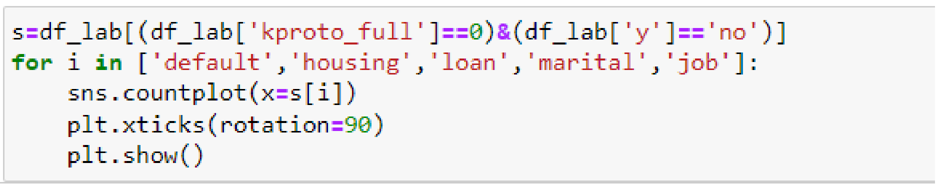


```
s=df_lab[(df_lab['kmode_opt']==3)&(df_lab['y']=='no')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```

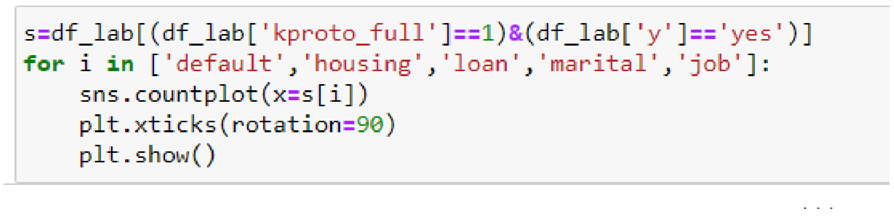
Kprototype

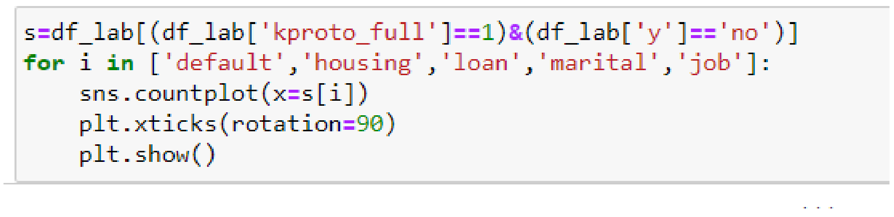


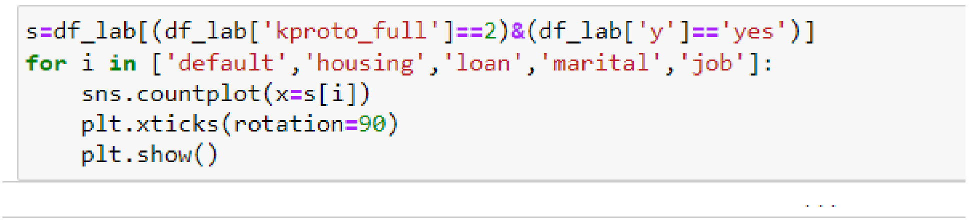
```
s=df_lab[(df_lab['kproto_full']==0)&(df_lab['y']=='yes')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```

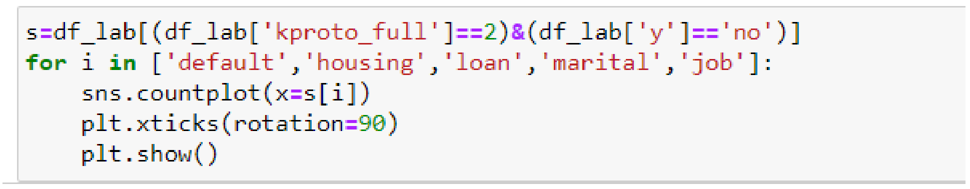


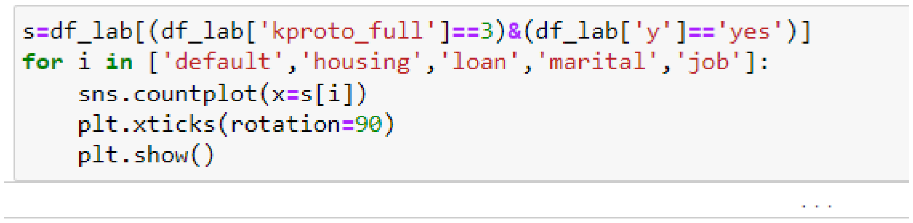
```
s=df_lab[(df_lab['kproto_full']==0)&(df_lab['y']=='no')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```

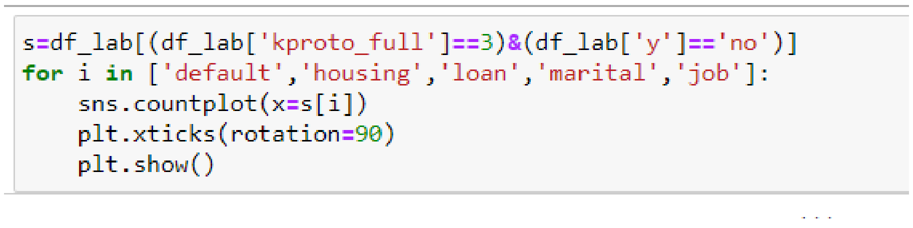

```
s=df_lab[(df_lab['kproto_full']==1)&(df_lab['y']=='yes')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```


```
s=df_lab[(df_lab['kproto_full']==1)&(df_lab['y']=='no')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```

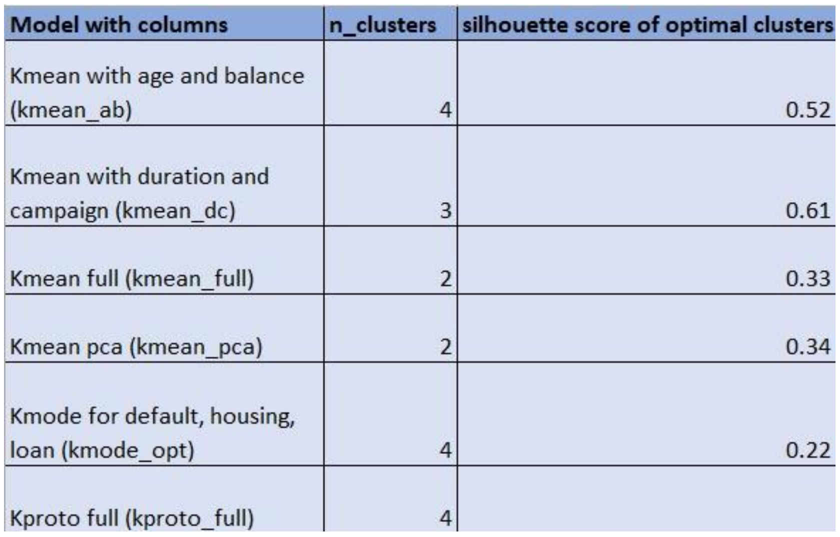

```
s=df_lab[(df_lab['kproto_full']==2)&(df_lab['y']=='yes')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```


```
s=df_lab[(df_lab['kproto_full']==2)&(df_lab['y']=='no')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```


```
s=df_lab[(df_lab['kproto_full']==3)&(df_lab['y']=='yes')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```


```
s=df_lab[(df_lab['kproto_full']==3)&(df_lab['y']=='no')]
for i in ['default','housing','loan','marital','job']:
    sns.countplot(x=s[i])
    plt.xticks(rotation=90)
    plt.show()
```

<u>USL Conclusion:</u>



Model with columns	n_clusters	silhouette score of optimal clusters
Kmean with age and balance (kmean_ab)	4	0.52
Kmean with duration and campaign (kmean_dc)	3	0.61
Kmean full (kmean_full)	2	0.33
Kmean pca (kmean_pca)	2	0.34
Kmode for default, housing, loan (kmode_opt)	4	0.22
Kproto full (kproto_full)	4	

Best Model: K-means with Duration and Campaign (kmean_dc): •

Optimal Clusters: 3, **Silhouette Score:** 0.61

This model has the highest silhouette score among the options you provided. A higher silhouette score indicates that the clusters are well-defined and have a good separation between them.

Worst Model: K-means Full (kmean_full):

Optimal Clusters: 2, **Silhouette Score:** 0.33

This model has the lowest silhouette score among the options. While a silhouette score of 0.33 is not necessarily "bad," it is lower compared to the other models you presented. It suggests that the clusters may not be as well-separated or distinct as in the other models.

<u>Business Conclusion:</u>

<u>Balance:</u> Potential clients in the low balance and no balance category were more likely to have a house loan than people in the average and high balance category. This means that the potential client has financial compromises to pay back its house loan and thus, there is no cash for he or she to subscribe to a term deposit account. However, we see that potential clients in the average and high balances are less likely to have a house loan and therefore, more likely to open a term deposit. Lastly, the next marketing campaign should focus on individuals of average and high balances in order to increase the likelihood of subscribing to a term deposit.

Duration: Target the target group that is above average in duration, there is a highly likelihood that this target group would open a term deposit account. This would allow that the success rate of the next marketing campaign would be highly successful.

Months of Marketing Activity: We saw that the month of highest level of marketing activity was the month of **May**. However, this was the month that potential clients tended to reject term deposits offers (Lowest effective rate: -34.49%). For the next marketing campaign, it will be wise for the bank to focus the marketing campaign during the months of **March, September and October.**

Limitations:

- The data was highly imbalanced. In 45K client data, almost 5000 clients have subscribed to the term subscription and the rest have not subscribed to the term deposit.
- There were more missing values in previous outcome and mode of communication data.
- More data is needed to understand why people with loans, defaults, and negative balances opted for the term deposit.
- It is unclear on why the subscription rate is higher on 20th day and May month has the highest subscription while December has the least.

- There is less data about customers who are under the age category 90s.