Prediction Model to Identify Result of Direct Marketing Campaigns Targeted to get Term Deposits

This project analyzes direct marketing campaigns conducted by a banking institution via phone calls. The primary objective is to predict whether a client will subscribe to a term deposit. This prediction is crucial for optimizing marketing strategies and improving campaign effectiveness.

Goal

The goal is to predict whether a client will subscribe ('yes') or not ('no') to a term deposit (y). This prediction leverages client data and campaign interaction details to build accurate predictive models.We shall use various classfiers to build models that provide satisfactory predictions

Dataset Characteristics

Multivariate: The dataset includes multiple features that can influence the outcome.
Subject Area: Data Analysis and Machine learning.
Associated Tasks: Predictive Modeling.
Feature Types: Categorical and Numerical.
Instances: 45,211 records.
Features: 16 variables, including client demographics, contact details, and campaign specifics.

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from imblearn.over_sampling import SMOTE
import math
from sklearn.feature_selection import RFECV
```

Libraries used:

Pandas:is a powerful data analysis and manipulation library for Python. It provides data structures (like DataFrame and Series) to work with structured data easily and efficiently.
NumPy:(Numerical Python) is a fundamental package for numerical computing in Python.
Matplotlib:is a 2D plotting library for Python that produces publication-quality figures in a variety of formats and interactive environments across platforms.
Seaborn:is built on top of Matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics.
Scikit-learn:is a machine learning library for Python that provides simple and efficient tools for data mining and data analysis tasks.
imbalanced-learn: Imbalanced-Learn (imblearn) is a Python library specifically designed to handle imbalanced datasets, where the number of instances in different classes varies significantly.

In [5]:
```
df_full=pd.read_csv("bank-additional-full.csv",sep=",")
```

In [6]:
```
pd.set_option('display.max_columns', None)
```

*Here we are using bank-additional-full.csv** with all examples (41188) and 20 inputs, ordered by date (from May 2008 to November 2010), very close to the data analyzed in [Moro et al., 2014][NOTE: This dataset was chosen because it has fewer unknown values

compared to the other datasets provide.]*

In [8]:
```
df_full
```

Out[8]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	149	1
2	37	services	married	high.school	no	yes	no	telephone	may	mon	226	1
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1
4	56	services	married	high.school	no	no	yes	telephone	may	mon	307	1
...	...	...	...	...	...	...	...	...	...	...	...	...
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	fri	334	1
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	383	1
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	189	2
41186	44	technician	married	professional.course	no	no	no	cellular	nov	fri	442	1
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	fri	239	3

41188 rows × 21 columns

DATASET INFORMATION:

Variable Name	Role	Type	Demographic	Description
age	Feature	Integer	Age	
job	Feature	Categorical	Occupation	type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','employed','services','student','technician','unemployed','unknown')
marital	Feature	Categorical	Marital Status	marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorce widowed)
education	Feature	Categorical	Education Level	(categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','un
default	Feature	Binary		has credit in default?
balance	Feature	Integer		average yearly balance
housing	Feature	Binary		has housing loan?
loan	Feature	Binary		has personal loan?
contact	Feature	Categorical		contact communication type (categorical: 'cellular','telephone')
day_of_week	Feature	Date		last contact day of the week
month	Feature	Date		last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
duration	Feature	Integer		last contact duration, in seconds (numeric). Important note: this attribute highly affects the outp target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. after the end of the call y is obviously known. Thus, this input should only be included for bench purposes and should be discarded if the intention is to have a realistic predictive model.
campaign	Feature	Integer		number of contacts performed during this campaign and for this client (numeric, includes last c
pdays	Feature	Integer		number of days that passed by after the client was last contacted from a previous campaign (n -1 means client was not previously contacted)
previous	Feature	Integer		number of contacts performed before this campaign and for this client
poutcome	Feature	Categorical		outcome of the previous marketing campaign (categorical: 'failure','nonexisten
y	Target	Binary		has the client subscribed

Renaming columns

- *Renaming housing as h_loan as it meaning housing loan*
- *Renaming loan as p_loan as it means personal loan*

In [13]:
```
df_full= df_full.rename(columns= {'housing':'h_loan','loan':"p_loan"})
```

In [14]:
```
len(df_full.columns)
```

Out[14]:
```
21
```

Dropping columns

By dropping unwanted columns, we simplify the data, speed up processing, and remove irrelevant information, thereby improving the accuracy of the model. Here, columns : 'duration', 'repay_loan', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed' acts as hinderance to our Pridiction goal, so it is preferable to remove them from the data..

In [17]:
```
df_full = df_full.drop(columns=['contact','emp.var.rate','cons.price.idx','cons.conf.idx','euribor3m','nr.emplo
```

In [18]:
```
len(df_full.columns)
```

Out[18]:
```
14
```

In [19]:
```
df_full['poutcome'].value_counts()
```

Out[19]:
```
poutcome
nonexistent    35563
failure         4252
success         1373
Name: count, dtype: int64
```

In [20]:
```
df_full['y'].value_counts()
```

Out[20]:
```
y
no     36548
yes     4640
Name: count, dtype: int64
```

In [21]:
```
df_full.to_csv("bankfulladd.csv")
```

In [22]:
```
df_full=pd.read_csv('bankfulladd.csv')
```

Replacing unknown value with MODE

*We use mode for categorical values** .Replacing unknown values with the mode is to ensure data completeness and reliability. The columns which had the unknow values were education, job, h_loan, p_loan .Also the dataset is smaller in size that is why we use mode in larger datasets we can just choose to remove the unknown observations.*

In [25]:
```
mode_edu = df_full['education'].mode()[0]
mode_job = df_full['job'].mode()[0]
mode_hl = df_full['h_loan'].mode()[0]
mode_pl = df_full['p_loan'].mode()[0]
```

In [26]:
```
print(mode_edu , mode_job, mode_hl, mode_pl)
```

```
university.degree admin. yes no
```

In [27]:
```
df_full.loc[df_full['education'] == 'unknown', 'education'] = mode_edu
df_full.loc[df_full['job'] == 'unknown', 'job'] = mode_job
df_full.loc[df_full['h_loan'] == 'unknown', 'h_loan'] = mode_hl
df_full.loc[df_full['p_loan'] == 'unknown', 'p_loan'] = mode_pl
```

In [28]:
```
df_full.isnull().sum()
```

Out[28]:
```
Unnamed: 0     0
age            0
job            0
marital        0
education      0
h_loan         0
p_loan         0
month          0
day_of_week    0
duration       0
campaign       0
pdays          0
previous       0
poutcome       0
y              0
dtype: int64
```

In [29]:
```
df_full['education'].value_counts()
```

Out[29]:
```
education
university.degree      13899
high.school             9515
basic.9y                6045
professional.course     5243
basic.4y                4176
basic.6y                2292
illiterate                18
Name: count, dtype: int64
```

In [30]:
```
df_full['job'].value_counts()
```

Out[30]:
```
job
admin.           10752
blue-collar       9254
technician        6743
services          3969
management        2924
retired           1720
entrepreneur      1456
self-employed     1421
housemaid         1060
unemployed        1014
student            875
Name: count, dtype: int64
```

Converting target variable(y) to binary variable

Converting the target variable y to a binary format helps classification algorithms work better. It simplifies the problem, makes model evaluation easier, and allows for predictions that are easier to understand.

In [33]:
```
df_full['y'] = df_full['y'].map({'yes': 1, 'no': 0})
```

In [34]:
```
df_full.head()
```

Out[34]:

	Unnamed: 0	age	job	marital	education	h_loan	p_loan	month	day_of_week	duration	campaign	pdays	previous	p
0	0	56	housemaid	married	basic.4y	no	no	may	mon	261	1	999	0	no
1	1	57	services	married	high.school	no	no	may	mon	149	1	999	0	no
2	2	37	services	married	high.school	yes	no	may	mon	226	1	999	0	no
3	3	40	admin.	married	basic.6y	no	no	may	mon	151	1	999	0	no
4	4	56	services	married	high.school	no	yes	may	mon	307	1	999	0	no

In [35]:
```
df_full['y'].value_counts()
```

Out[35]:
```
y
0    36548
1     4640
Name: count, dtype: int64
```

Encoding educational values to numeric value

We have determined that the 'education' column can be a critical factor influencing the target variable. Education alone may not determine someone's decision to take a term deposit, it can serve as a proxy for financial knowledge, stability, risk perception, and decision-making processes—all of which can influence financial behaviors, including the choice to invest in term deposits.

In [38]:
```
# Define the mapping dictionary
education_mapping = {
    'university.degree': 1,
    'high.school': 2,
    'basic.9y': 3,
    'professional.course': 4,
    'basic.4y': 5,
    'basic.6y': 6,
    'illiterate': 7
}

# Encode the 'education' column in place using .loc
df_full.loc[:, 'education'] = df_full['education'].map(education_mapping)
df_full
```

Out[38]:

	Unnamed: 0	age	job	marital	education	h_loan	p_loan	month	day_of_week	duration	campaign	pdays	previous
0	0	56	housemaid	married	5	no	no	may	mon	261	1	999	0
1	1	57	services	married	2	no	no	may	mon	149	1	999	0
2	2	37	services	married	2	yes	no	may	mon	226	1	999	0
3	3	40	admin.	married	6	no	no	may	mon	151	1	999	0
4	4	56	services	married	2	no	yes	may	mon	307	1	999	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
41183	41183	73	retired	married	4	yes	no	nov	fri	334	1	999	0
41184	41184	46	blue-collar	married	4	no	no	nov	fri	383	1	999	0
41185	41185	56	retired	married	1	yes	no	nov	fri	189	2	999	0
41186	41186	44	technician	married	4	no	no	nov	fri	442	1	999	0
41187	41187	74	retired	married	4	yes	no	nov	fri	239	3	999	1

41188 rows × 15 columns

Dropping dupilcates

[Duplicate rows are removed] Dropping duplicates ensures that your dataset is clean, accurate, and ready for analysis by removing repeated records that can bias results, leading to more reliable and valid insights. This process eliminates unnecessary data inputs, reduces data size, reduces the weightage of data and improves the efficiency and accuracy of models.

In [41]:
```
df_full.drop_duplicates(inplace=True)
```

In [42]:
```
df_full
```

Out[42]:

	Unnamed: 0	age	job	marital	education	h_loan	p_loan	month	day_of_week	duration	campaign	pdays	previous
0	0	56	housemaid	married	5	no	no	may	mon	261	1	999	0
1	1	57	services	married	2	no	no	may	mon	149	1	999	0
2	2	37	services	married	2	yes	no	may	mon	226	1	999	0
3	3	40	admin.	married	6	no	no	may	mon	151	1	999	0
4	4	56	services	married	2	no	yes	may	mon	307	1	999	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
41183	41183	73	retired	married	4	yes	no	nov	fri	334	1	999	0
41184	41184	46	blue-collar	married	4	no	no	nov	fri	383	1	999	0
41185	41185	56	retired	married	1	yes	no	nov	fri	189	2	999	0
41186	41186	44	technician	married	4	no	no	nov	fri	442	1	999	0
41187	41187	74	retired	married	4	yes	no	nov	fri	239	3	999	1

41188 rows × 15 columns

In [43]:
```
df_full.to_csv('finaldupesremoved.csv', index=False)
```

Verifying the data type of education column

We are converting the Education column to int64 which was previously of object datatype

In [46]:
```
education_dtype = df_full.dtypes['education']
print(f"The data type of the 'education' column is: {education_dtype}")
df_full['education'] = df_full['education'].astype('int64')
df_full['education'].dtypes
```

```
The data type of the 'education' column is: object
```

Out[46]:
```
dtype('int64')
```

[*The datatype of the Education column was manually changed here to 'int64'*]

Creating seperate dataframe into categorical and numeric

- *We are creating seperate data frames for further encoding(one hot encoding). One hot encoding converts categorical** values numeric values. by creating a new features from the values of a column, means each element in a feature now gets to be a new feature in itself.*
- *It allows machine learning algorithms to process categorical data more effectively by treating each category as a separate binary feature, thus enabling the model to learn patterns more accurately without making implicit ordinal assumptions(Implicit ordinal assumptions refer to the incorrect assumption that the categories of a categorical variable have a meaningful order or ranking when they do not) about the categories.*

In [50]:
```
cat_columns = df_full.select_dtypes(include=['object']).columns
num_columns = df_full.select_dtypes(include=['int64','float64','bool']).columns
```

In [51]:
```
df_fullcat=df_full[cat_columns]
df_fullcat
```

Out[51]:

	job	marital	h_loan	p_loan	month	day_of_week	poutcome
0	housemaid	married	no	no	may	mon	nonexistent
1	services	married	no	no	may	mon	nonexistent
2	services	married	yes	no	may	mon	nonexistent
3	admin.	married	no	no	may	mon	nonexistent
4	services	married	no	yes	may	mon	nonexistent
...	...	...	...	...	...	...	...
41183	retired	married	yes	no	nov	fri	nonexistent
41184	blue-collar	married	no	no	nov	fri	nonexistent
41185	retired	married	yes	no	nov	fri	nonexistent
41186	technician	married	no	no	nov	fri	nonexistent
41187	retired	married	yes	no	nov	fri	failure

41188 rows × 7 columns

In [52]:
```
df_fullnum=df_full[num_columns]
df_fullnum
```

Out[52]:

	Unnamed: 0	age	education	duration	campaign	pdays	previous	y
0	0	56	5	261	1	999	0	0
1	1	57	2	149	1	999	0	0
2	2	37	2	226	1	999	0	0
3	3	40	6	151	1	999	0	0
4	4	56	2	307	1	999	0	0
...	...	...	...	...	...	...	...	...
41183	41183	73	4	334	1	999	0	1
41184	41184	46	4	383	1	999	0	0
41185	41185	56	1	189	2	999	0	0
41186	41186	44	4	442	1	999	0	1
41187	41187	74	4	239	3	999	1	0

41188 rows × 8 columns

In [53]:
```
df_full[num_columns].to_csv('finalnum_data.csv', index=False)
```

In [54]:
```
df_full[cat_columns].to_csv('finalcat_data.csv', index=False)
```

ONE HOT ENCODING

One hot encoding is a technique used in data preprocessing to convert categorical variables into a format that can be more easily used for machine learning algorithms.

id	color
1	red
2	blue
3	green
4	blue

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

- *One-hot encoding is typically preferred when dealing with categorical variables that lack intrinsic order or hierarchy, ensuring that models interpret the data correctly without introducing unintended biases or assumptions.*
- **All the Values of columns of the catergorical dataset now indivially is a feature which has values true or false accordingly.**

In [59]:
```
df_fullcat = pd.get_dummies(df_fullcat,drop_first=True)
```

In the Pandas library, get_dummies() is used to convert categorical variables into dummy/indicator variables.

In [61]:
```
df_fullcat
```

Out[61]:

	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	job_student	job_technic
0	False	False	True	False	False	False	False	False	Fa
1	False	False	False	False	False	False	True	False	Fa
2	False	False	False	False	False	False	True	False	Fa
3	False	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	True	False	Fa
...	...	...	...	...	...	...	...	...	
41183	False	False	False	False	True	False	False	False	Fa
41184	True	False	False	False	False	False	False	False	Fa
41185	False	False	False	False	True	False	False	False	Fa
41186	False	False	False	False	False	False	False	False	T
41187	False	False	False	False	True	False	False	False	Fa

41188 rows × 30 columns

Concatenating numeric and encoded dataframe

So far we created different dataframes that seperates categorical and numeric data now after the categorical data is dealt with ,we will concatenate the dataframes for further analysis.

In [64]:
```
df_combined = pd.concat([df_fullnum, df_fullcat], axis=1)
```

df_fullcat + df_fullnum = df_combined

In [66]:
```
df_combined
```

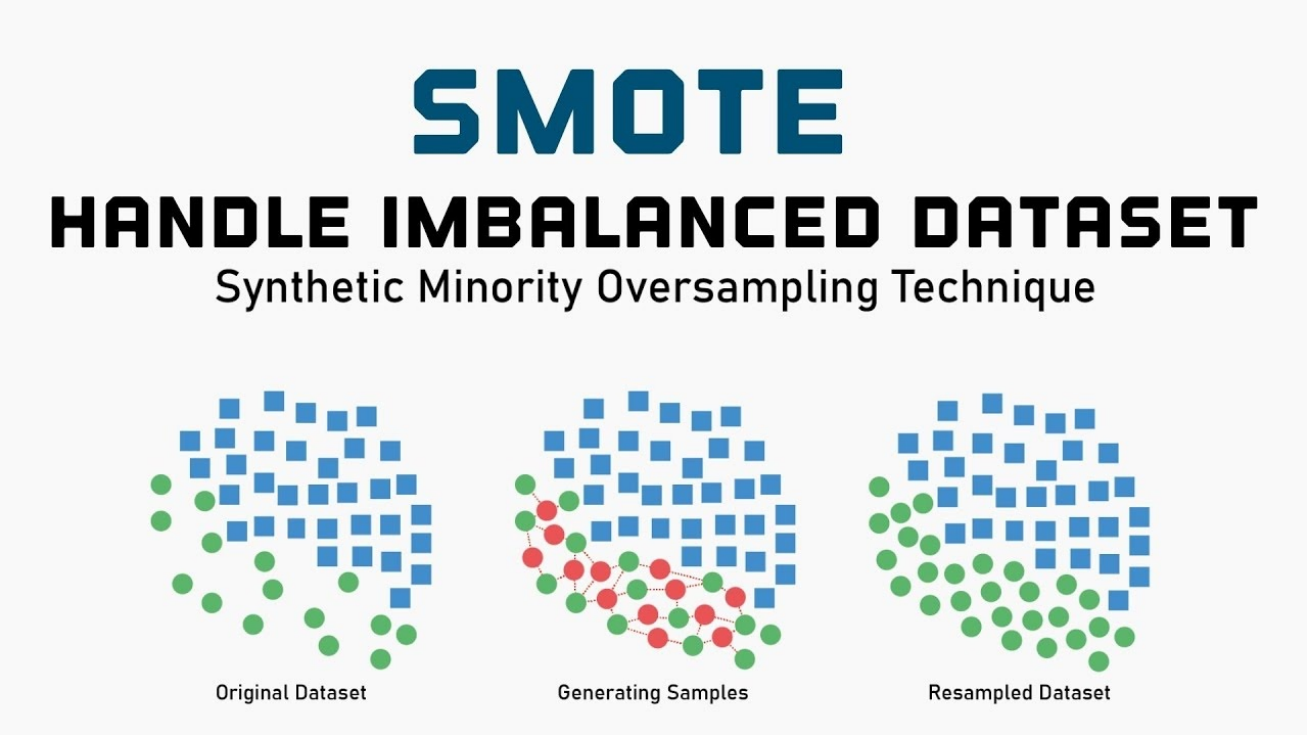
Out[66]:

	Unnamed: 0	age	education	duration	campaign	pdays	previous	y	job_blue-collar	job_entrepreneur	job_housemaid	job_mana
0	0	56	5	261	1	999	0	0	False	False	True	
1	1	57	2	149	1	999	0	0	False	False	False	
2	2	37	2	226	1	999	0	0	False	False	False	
3	3	40	6	151	1	999	0	0	False	False	False	
4	4	56	2	307	1	999	0	0	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	
41183	41183	73	4	334	1	999	0	1	False	False	False	
41184	41184	46	4	383	1	999	0	0	True	False	False	
41185	41185	56	1	189	2	999	0	0	False	False	False	
41186	41186	44	4	442	1	999	0	1	False	False	False	
41187	41187	74	4	239	3	999	1	0	False	False	False	

41188 rows × 38 columns

HANDLING IMBALANCED DATA USING SMOTE

SMOTE *(Synthetic Minority Over-sampling Technique) is a technique used to address the class imbalance in datasets where one class (usually the minority class) is significantly underrepresented compared to the other class(es).*



*In the figure** above we can visually understand the process of balancing the dataset for better prediction. The class with lesser number of datapoints or instances (in our case the target variable y has way less 1s than 0s). We create new instances referencing the orginal ones to balance the minority and majority class. **HERE we are creating new instances of 1s so that it can be equal to number 0s.***

In [71]:
```
df_combined = df_combined.drop('Unnamed: 0', axis=1)
```

In [72]:
```
df_combined
```

Out[72]:

	age	education	duration	campaign	pdays	previous	y	job_blue-collar	job_entrepreneur	job_housemaid	job_management	jo
0	56	5	261	1	999	0	0	False	False	True	False	
1	57	2	149	1	999	0	0	False	False	False	False	
2	37	2	226	1	999	0	0	False	False	False	False	
3	40	6	151	1	999	0	0	False	False	False	False	
4	56	2	307	1	999	0	0	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	
41183	73	4	334	1	999	0	1	False	False	False	False	
41184	46	4	383	1	999	0	0	True	False	False	False	
41185	56	1	189	2	999	0	0	False	False	False	False	
41186	44	4	442	1	999	0	1	False	False	False	False	
41187	74	4	239	3	999	1	0	False	False	False	False	

41188 rows × 37 columns

In [73]:
```
X = df_combined.drop('y', axis=1)
y = df_combined['y']
```

In [74]:
```
smt=SMOTE()
```

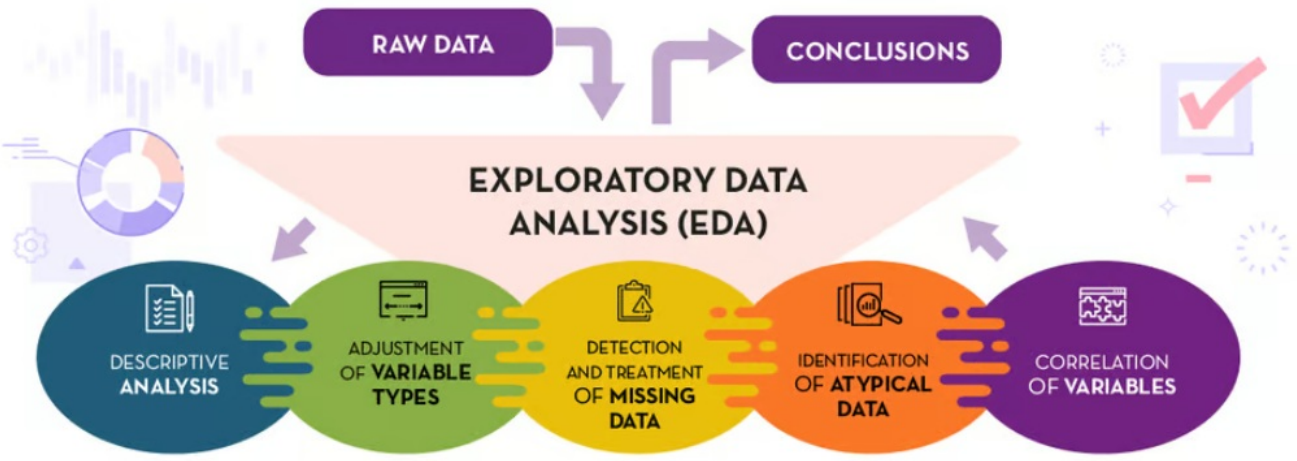
In [75]:
```
X ,y=smt.fit_resample(X,y)
```

In [76]:
```
y.value_counts()
```

Out[76]:
```
y
0    36548
1    36548
Name: count, dtype: int64
```

Exploratory Data Analysis (EDA)

__Exploratory Data Analysis (EDA)__ is a crucial step in the data analysis process. It involves summarizing the main characteristics of a dataset, often using visual methods, before making any assumptions or conclusions. The goal of EDA is to understand the data, detect patterns, anomalies, test hypotheses, and check assumptions with the help of summary statistics and graphical representations.Proper EDA can significantly enhance the quality of insights and improve the outcomes of data analysis and machine learning projects.



We will be understanding the features and its effects on each other by visualizing the data. EDA helps us find the hidden behaviour of the data which makes our model and its predictions better. To do that we will be using different Visualization techniques to understand the patterns of the data. After that we can decide which classifier to use to train a model. We will be using different classifiers such has Logistic regression and decision tree and see which one perform the best.

In [81]:
```
avg_calls = df_combined.groupby('age')['campaign'].mean().reset_index()
```

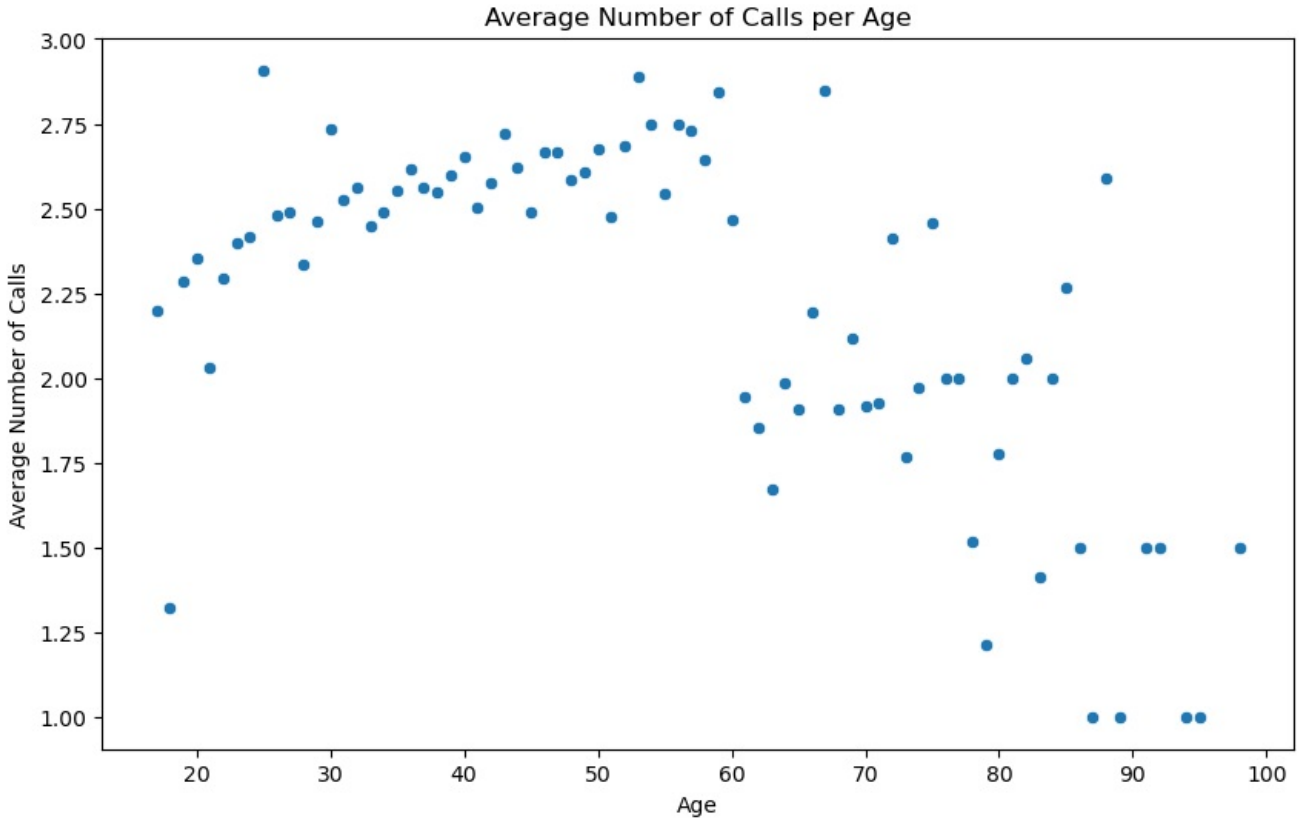
In [82]:
```
avg_calls.columns=['age','avg_campaign']
```

Scatter Plot

Scatter plots are a powerful tool for visualizing the relationship between two numerical variables, identifying patterns, and detecting outliers. They can be customized with additional features like color and size to enhance their interpretability and provide deeper insights.

In [85]:
```
plt.figure(figsize=(10,6))
sns.scatterplot(x='age',y='avg_campaign',data=avg_calls)
plt.title('Average Number of Calls per Age')
plt.xlabel('Age')
plt.ylabel('Average Number of Calls')
```

Out[85]:
```
Text(0, 0.5, 'Average Number of Calls')
```



Conclusion:

- **The Scatter Plot shows that from the age of around 20 to the age of 60 the average number of calls made *increase*. From the age of 60 to 100 the class *decrease*.**
- **We can observe 3 layers in number of calls. from younger ages(20) to late middle ages(60) we see the maximum number of calls, ages 60 to 80 see lesser calls and old ages(80 to 100) see the least number of calls made to them.**
- **This can also indicate that people in their old age might not be willing to say yes for them selves hence the number of calls made to them are also way less.**

In [87]:
```
grouped_data = df_combined[['age','y']]
grouped_data
```

Out[87]:

	age	y
0	56	0
1	57	0
2	37	0
3	40	0
4	56	0
...	...	...
41183	73	1
41184	46	0
41185	56	0
41186	44	1
41187	74	0

41188 rows × 2 columns

In [88]:
```
bins = range(0, 101, 10)
labels = [f'{i}-{i+10}' for i in bins[:-1]]

# Create a new column 'age_range' with the defined bins
grouped_data.loc[:, 'age_range'] = pd.cut(grouped_data['age'], bins=bins, labels=labels, right=False)


grouped_data
```

```
C:\Users\priya\AppData\Local\Temp\ipykernel_4804\1037491717.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  grouped_data.loc[:, 'age_range'] = pd.cut(grouped_data['age'], bins=bins, labels=labels, right=False)
```

Out[88]:

	age	y	age_range
0	56	0	50-60
1	57	0	50-60
2	37	0	30-40
3	40	0	40-50
4	56	0	50-60
...	...	...	...
41183	73	1	70-80
41184	46	0	40-50
41185	56	0	50-60
41186	44	1	40-50
41187	74	0	70-80

41188 rows × 3 columns

*Bins are used to convert age into age range. Bins** are predefined intervals that divide the range of data values into segments. Each bin covers a specific range of data points, and the count of data points falling within each bin is recorded.*

Now we are creating three dataframes using age ranges and y to visually understand their relationship as ratio of people who accepted the offer v/s age range

In [91]:
```
grouped_data = grouped_data.groupby('age_range', observed=True)['y'].count().reset_index()
```

In [92]:
```
grouped_data
```

Out[92]:

	age_range	y
0	10-20	75
1	20-30	5594
2	30-40	16938
3	40-50	10526
4	50-60	6862
5	60-70	724
6	70-80	319
7	80-90	140
8	90-100	10

In [93]:
```
grouped_data= grouped_data.rename(columns= {'y':'y_all'})
```

Contains age ranges and sum of all the responses of y

In [95]:
```
grouped_data
```

Out[95]:

	age_range	y_all
0	10-20	75
1	20-30	5594
2	30-40	16938
3	40-50	10526
4	50-60	6862
5	60-70	724
6	70-80	319
7	80-90	140
8	90-100	10

In [96]:
```
positive_df = df_combined[['age','y']]
```

In [97]:
```
bins = range(0, 101, 10)
labels = [f'{i}-{i+10}' for i in bins[:-1]]

# Create a new column 'age_range' with the defined bins
positive_df.loc[:, 'age_range'] = pd.cut(positive_df['age'], bins=bins, labels=labels, right=False)
```

```
C:\Users\priya\AppData\Local\Temp\ipykernel_4804\1258644111.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  positive_df.loc[:, 'age_range'] = pd.cut(positive_df['age'], bins=bins, labels=labels, right=False)
```

In [98]:
```
positive_df = df_combined[['age', 'y']]
# Filter out only rows where 'y' is 1
positive_df = positive_df[positive_df['y'] == 1]

bins = range(0, 101, 10)
labels = [f'{i}-{i+10}' for i in bins[:-1]]

# Create a new column 'age_range' with the defined bins
positive_df['age_range'] = pd.cut(positive_df['age'], bins=bins, labels=labels, right=False)
```

In [99]:
```
positive_df = positive_df[positive_df['y'] == 1].groupby('age_range', observed=False).size().reset_index(name='
```

Contains sum of all the positive responses of y and age ranges

In [101…
```
positive_df
```

Out[101…

	age_range	y_positive
0	0-10	0
1	10-20	34
2	20-30	888
3	30-40	1715
4	40-50	834
5	50-60	697
6	60-70	251
7	70-80	144
8	80-90	72
9	90-100	5

In [102…
```
result =positive_df.join(grouped_data.set_index('age_range'), on='age_range', lsuffix='_ositive_df', rsuffix='_
```

Contains both positive values of y , sum of all values of y and age range

In [104…
```
result
```

Out[104…

	age_range	y_positive	y_all
0	0-10	0	NaN
1	10-20	34	75.0
2	20-30	888	5594.0
3	30-40	1715	16938.0
4	40-50	834	10526.0
5	50-60	697	6862.0
6	60-70	251	724.0
7	70-80	144	319.0
8	80-90	72	140.0
9	90-100	5	10.0

In [105…
```
result['ratio'] = result['y_positive'] / result['y_all']
```

In [106…
```
result
```

Out[106…

	age_range	y_positive	y_all	ratio
0	0-10	0	NaN	NaN
1	10-20	34	75.0	0.453333
2	20-30	888	5594.0	0.158742
3	30-40	1715	16938.0	0.101252
4	40-50	834	10526.0	0.079232
5	50-60	697	6862.0	0.101574
6	60-70	251	724.0	0.346685
7	70-80	144	319.0	0.451411
8	80-90	72	140.0	0.514286
9	90-100	5	10.0	0.500000

Bar Plot

A bar plot (or bar chart) is a type of visualization that represents categorical data with rectangular bars. The length or height of each bar is proportional to the value it represents. Bar plots are useful for comparing the quantities of different categories or groups.

In [109…
```
result
```

Out[109…

	age_range	y_positive	y_all	ratio
0	0-10	0	NaN	NaN
1	10-20	34	75.0	0.453333
2	20-30	888	5594.0	0.158742
3	30-40	1715	16938.0	0.101252
4	40-50	834	10526.0	0.079232
5	50-60	697	6862.0	0.101574
6	60-70	251	724.0	0.346685
7	70-80	144	319.0	0.451411
8	80-90	72	140.0	0.514286
9	90-100	5	10.0	0.500000

In [110…

```
mean_value = result['ratio'].mean()
result['deviation']=result['ratio']-mean_value
result['se'] = result['deviation'] / math.sqrt(10)
```

In [111…

```
result
```

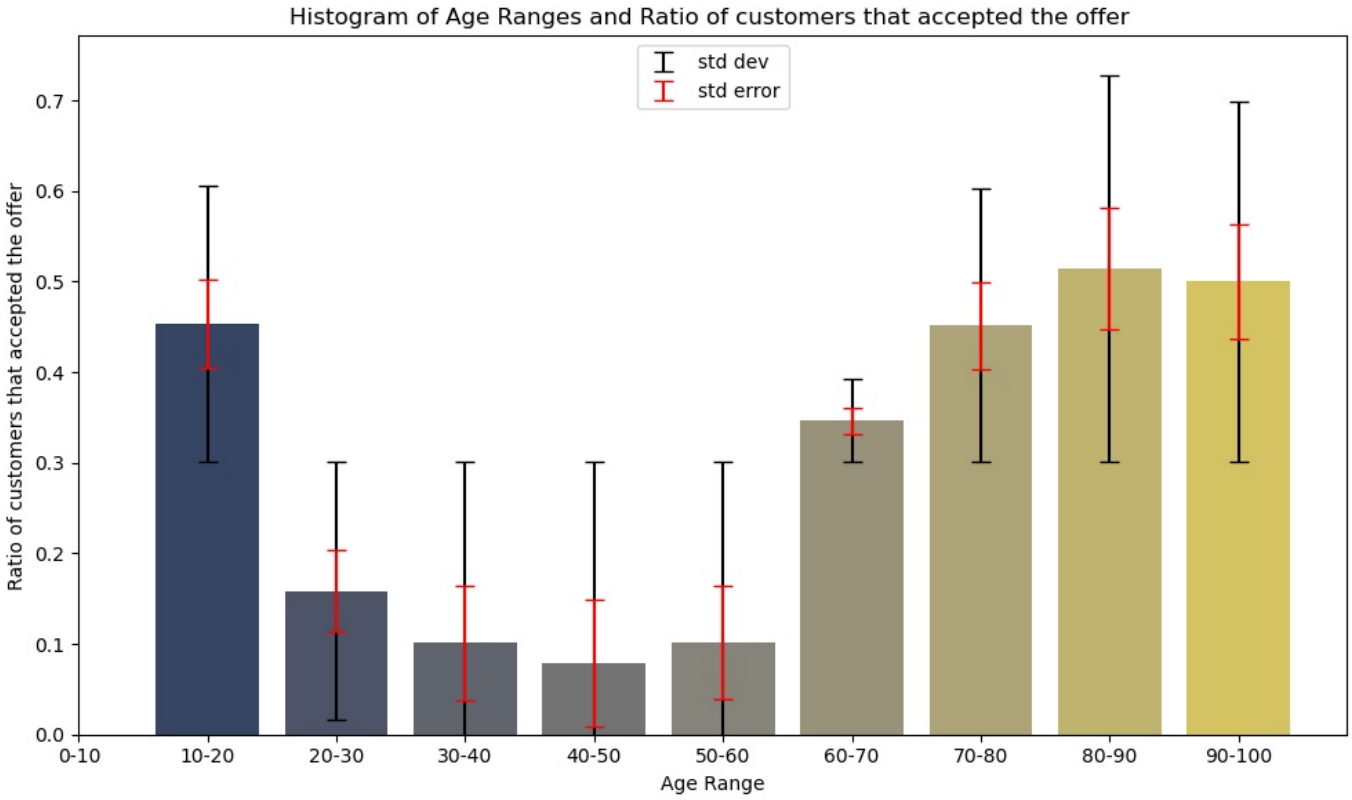
Out[111…

	age_range	y_positive	y_all	ratio	deviation	se
0	0-10	0	NaN	NaN	NaN	NaN
1	10-20	34	75.0	0.453333	0.152610	0.048259
2	20-30	888	5594.0	0.158742	-0.141982	-0.044899
3	30-40	1715	16938.0	0.101252	-0.199472	-0.063079
4	40-50	834	10526.0	0.079232	-0.221491	-0.070042
5	50-60	697	6862.0	0.101574	-0.199150	-0.062977
6	60-70	251	724.0	0.346685	0.045961	0.014534
7	70-80	144	319.0	0.451411	0.150687	0.047651
8	80-90	72	140.0	0.514286	0.213562	0.067534
9	90-100	5	10.0	0.500000	0.199276	0.063017

In [112…

```
# Make sure error values are positive
result['se'] = result['se'].abs()
result['deviation'] = result['deviation'].abs()

# Plotting with seaborn and matplotlib
plt.figure(figsize=(10, 6))

# Bar plot for ratios with seaborn
sns.barplot(x='age_range', y='ratio', data=result, palette='cividis')

# Add error bars for standard deviation
plt.errorbar(result['age_range'], result['ratio'], yerr=result['deviation'], fmt='none', c='black', capsize=5, 

# Add error bars for standard error
plt.errorbar(result['age_range'], result['ratio'], yerr=result['se'], fmt='none', c='red', capsize=5, label='st

# Labeling the plot
plt.xlabel('Age Range')
plt.ylabel('Ratio')
plt.title('Ratio vs Age Range with Std Deviation and Std Error')
plt.legend()

plt.xlabel('Age Range')
plt.ylabel('Ratio of customers that accepted the offer')
plt.title('Histogram of Age Ranges and Ratio of customers that accepted the offer ')
plt.ylim(0, None)  # Set the y-axis limit to start at zero
plt.xticks(np.arange(len(result)), result['age_range'])
# Rotate x-axis labels for better readability
plt.xticks(rotation=0)

# Display the plot
plt.tight_layout()
plt.show()
```

```
C:\Users\priya\anaconda3\Lib\site-packages\seaborn\categorical.py:641: FutureWarning: The default of observed=Fa
lse is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain curre
nt behavior or observed=True to adopt the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
```



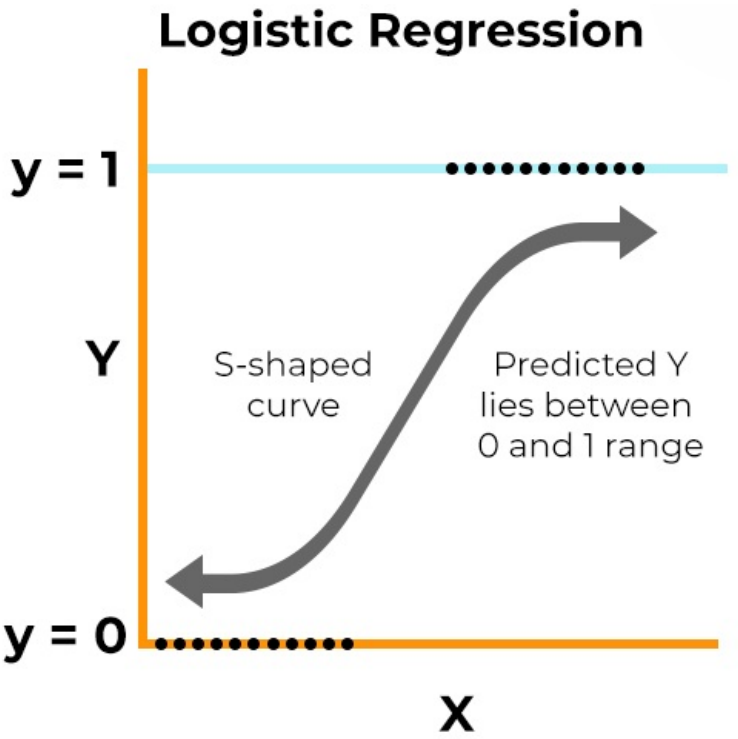
*__Standard deviation__: The black circles in the plot represent standard deviation error bars. These show how much the ratios (the percentage of customers who accepted the offer) vary within each age range. By looking at these error bars, you can see if some age ranges have more or less variation in their acceptance ratios.
__Standard error__: SE indicates the precision of the sample mean as an estimate of the population mean. The red caps in the plot represent standard error bars. These show how precise our estimate of the population ratio is based on our sample data. Smaller red caps mean we are more confident in our estimate (it's more precise), while larger red caps mean we are less confident (it's less precise).*

Conclusion:

- **The histogram above shows that people in their middle ages around (30 to 60) have a lower possibility of saying yes to the term deposit.The younger(around 20) and older people (80 to 100) are more likely to say yes. The reason maybe that the middle ages generally carry the responsibilty of their entire family which make them less likely to say yes as their funds might already be tight.**
- **Although the calls made to the middle ages are more than that of the two extremes.**

Logistic Regression

Logistic regression is a statistical model used for binary classification tasks, where the target variable y. y (also known as the dependent or response variable) is categorical with two possible outcomes, typically encoded as 0 and 1. The logistic regression model estimates the probability that a given observation belongs to a particular class (usually the positive class, labeled as 1).



- **Logistic regression requires fewer computational resources and can be trained quickly even on small datasets.**
- **Logistic regression can be sensitive to imbalanced datasets where one class is much more frequent than the other.So we balanced our data before hand using smote.**

In [119…
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

__StandardScaler__ makes all the features have the same average value (zero) and the same variation (one). It's like transforming all your data to a common unit of measurement, so when you use it in machine learning models, they can understand the data better and make more accurate predictions.

70% of the data will be the traing set and rest 30% will be the testing data

In [122…
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
```

L1 regularization is a form of embedded feature selection technique which selects the most relevat feature

- *Initializing a logistic regression model with L1 regularization** (penalty='l1') and uses the liblinear solver. L1 regularization helps in feature selection by shrinking the less important features' coefficients to zero.*
- ***random_state** is a crucial parameter in machine learning algorithms to control randomness and ensure consistent and reproducible results across different runs of the same algorithm on the same data.*
- *The Datasets (The balanced ones using smote) is split into training and testing data sets.*

In [125…
```
model = LogisticRegression(penalty='l1', solver='liblinear')
model.fit(X_train, y_train)
selected_features_l1 = np.abs(model.coef_)[0] > 0
X_l1_train = X_train[:, selected_features_l1]
X_l1_test = X_test[:, selected_features_l1]
print("L1 Regularization Selected Features:", X.columns[selected_features_l1])
```

```
L1 Regularization Selected Features: Index(['age', 'education', 'duration', 'campaign', 'pdays', 'previous',
       'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
       'job_management', 'job_retired', 'job_self-employed', 'job_services',
       'job_student', 'job_technician', 'job_unemployed', 'marital_married',
       'marital_single', 'marital_unknown', 'h_loan_yes', 'p_loan_yes',
       'month_aug', 'month_dec', 'month_jul', 'month_jun', 'month_mar',
       'month_may', 'month_nov', 'month_oct', 'month_sep', 'day_of_week_mon',
       'day_of_week_thu', 'day_of_week_tue', 'day_of_week_wed',
       'poutcome_nonexistent', 'poutcome_success'],
      dtype='object')
```

In [126…
```
model.fit(X_l1_train, y_train)
```

Out[126…
```
LogisticRegression(penalty='l1', solver='liblinear')
```

Accuracy	Predictions/ Classifications	$\frac{\text{Correct}}{\text{Correct + Incorrect}}$
Precision	Predictions/ Classifications	$\frac{\text{True Positive}}{\text{True Positive + False Positive}}$
Recall	Predictions/ Classifications	$\frac{\text{True Positive}}{\text{True Positive + False Negative}}$
F1	Predictions/ Classifications	$\frac{\text{2 * True Positive}}{\text{True Positive + 0.5 (False Positive + False Negative)}}$

Performance evaluation

In [129…
```
# Predict
y_pred_l1 = model.predict(X_l1_test)
y_pred = model.predict(X_l1_train)
# Evaluate
accuracy_test = accuracy_score(y_test, y_pred_l1)
accuracy_train= accuracy_score(y_train,y_pred)

train_preds = model.predict(X_train)
test_preds = model.predict(X_test)


# Create a DataFrame
accuracy_df = pd.DataFrame({
    'Dataset': ['Training', 'Test'],
    'Accuracy': [accuracy_test, accuracy_train]
})

# Display the table

accuracy_df
```

Out[129…

	Dataset	Accuracy
0	Training	0.923754
1	Test	0.925831

*__Precision__: The ratio of true positive predictions to the total predicted positives (i.e., how many of the predicted positives are actual positives).
Recall (Sensitivity): The ratio of true positive predictions to the total actual positives (i.e., how many of the actual positives are correctly identified).
The **F1 score** is the harmonic mean of precision and recall, ensuring that a model only scores high if both precision and recall are high.*

WHY DO WE USE F1 SCORE

- *In datasets with imbalanced classes, metrics like accuracy can be misleading. For instance, in a dataset with 95% negatives and 5% positives, a model predicting all negatives would have 95% accuracy but would fail to identify any positives.The F1 score focuses on the performance of the positive class, providing a more meaningful evaluation in such scenarios.*
- *While precision and recall individually highlight different aspects of performance, the F1 score encapsulates both in a single metric, making it easier to compare models.*
- *It is particularly useful when the cost of false positives and false negatives is high or varies.

The F1 score helps in understanding the trade-offs between precision and recall. Improving one often leads to a decrease in the other, and the F1 score ensures that the model performs well on both fronts.*

- **Our dataset is balanced as we used SMOTE before that is why accuracy and f1 score are both relaiable here.**

In [132…
```
# Evaluate F1 score
f1 = f1_score(y_test, y_pred_l1)

# Compute Precision and Recall
precision = precision_score(y_test, y_pred_l1)
recall = recall_score(y_test, y_pred_l1)
```

```
precision_train = precision_score(y_train, y_pred)
recall_train = recall_score(y_train, y_pred)
f1_train = f1_score(y_train, y_pred)
```

In [133…
```
metrics_df = pd.DataFrame({
    'Dataset': ['Training', 'Test'],
    'Precision': [precision_train, precision],
    'Recall': [recall_train, recall],
    'F1 Score': [f1_train, f1]
})

# Formatting the 'Score' columns to 2 decimal places
metrics_df[['Precision', 'Recall', 'F1 Score']] = metrics_df[['Precision', 'Recall', 'F1 Score']].applymap("{:.

# Displaying the DataFrame
metrics_df
```

```
C:\Users\priya\AppData\Local\Temp\ipykernel_4804\2008444669.py:9: FutureWarning: DataFrame.applymap has been dep
recated. Use DataFrame.map instead.
  metrics_df[['Precision', 'Recall', 'F1 Score']] = metrics_df[['Precision', 'Recall', 'F1 Score']].applymap("{:
.2f}".format)
```

Out[133…

	Dataset	Precision	Recall	F1 Score
0	Training	0.93	0.92	0.93
1	Test	0.93	0.91	0.92

Training and Testing accuracy, precision , recall and f1-score all are similar which can indicate that the model is not overfitted.

In [135…
```
# Compute cross-validation scores
cv_scores_lr = cross_val_score(model, X_l1_test, y_test, cv=5).mean()  # Using 5-fold cross-validation
print("Mean Cross-Validation Scores of logistic regression:", cv_scores_lr)
```

```
Mean Cross-Validation Scores of logistic regression: 0.9248939483512638
```

Cross-Validation Process:

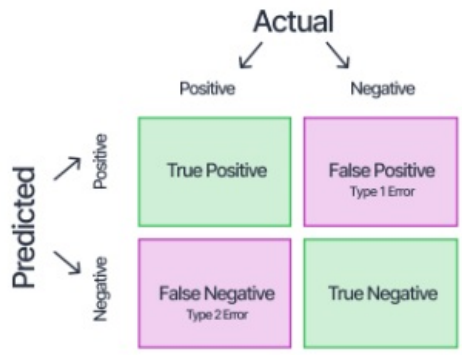
- *The dataset is divided into k subsets (folds).*
- *The model is trained on k-1 folds and validated on the remaining fold.*
- *This process is repeated k times, with each fold used exactly once as the validation data.*

Mean Cross-Validation Score:

- *After performing k-fold cross-validation, the performance metric (such as accuracy, precision, recall, or F1 score) is computed for each fold.*
- *The mean of these performance metrics across all folds gives the mean cross-validation score.*
- *It provides an aggregated measure of how well the model is expected to perform on unseen data based on the cross-validation process.*

The model will be 92% accurate when encountered with unseen data

Confusion matrix



*A confusion matrix** is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. It allows visualization of the performance of an algorithm by comparing predicted values with actual values.*

In [142…
```
cm = confusion_matrix(y_test, y_pred_l1)

TN = cm[0, 0]
```

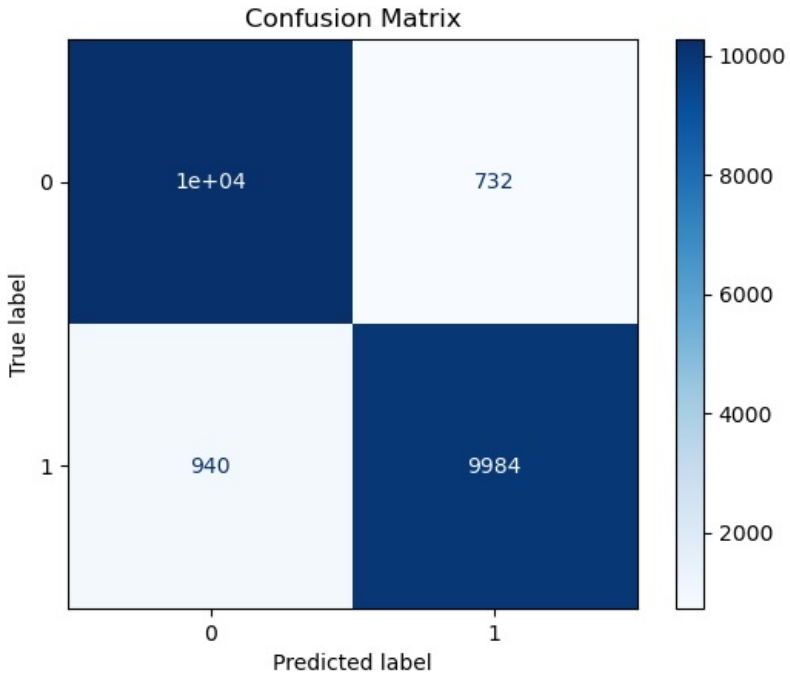
```
FP = cm[0, 1]
FN = cm[1, 0]
TP = cm[1, 1]
```

In [143…
```
print("True Negatives (TN):", TN)
print("False Positives (FP):", FP)
print("False Negatives (FN):", FN)
print("True Positives (TP):", TP)
```

```
True Negatives (TN): 10273
False Positives (FP): 732
False Negatives (FN): 940
True Positives (TP): 9984
```

In [144…
```
cm = confusion_matrix(y_test, y_pred_l1)


print("Confusion Matrix:\n", cm)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```

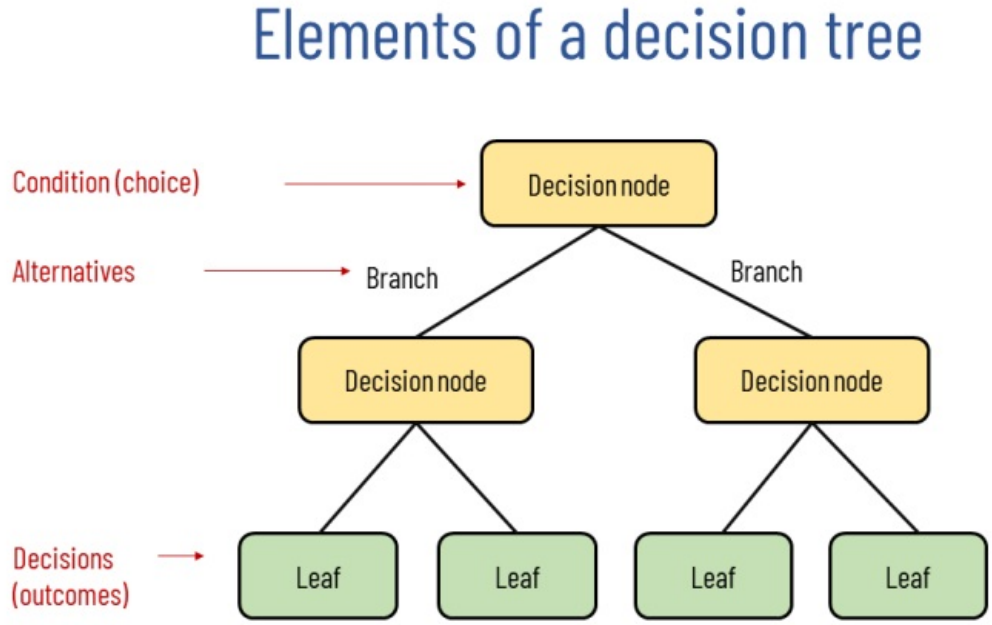
```
Confusion Matrix:
 [[10273   732]
 [  940  9984]]
```



DecisionTree

A decision tree is a supervised machine learning algorithm that models decisions based on a tree-like structure.

A decision tree is structured like a flowchart, where each internal node represents a "test" on an attribute (feature), each branch represents the outcome of the test, and each leaf node represents a class label or a numerical value (in case of regression).



- **DECISION TREE here is used to make a desicion on various features heirarchially placed indicating if the person will subscribe to the term deposit or not**
- **Decision trees do not require feature scaling because they are not affected by the magnitude of the features.**
- **Decision trees can handle irrelevant features reasonably well by ignoring them in the splits.**

*Decision trees can easily overfit** the training data, especially if they are deep. Pruning techniques, setting a maximum depth, or using ensemble methods like random forests or gradient boosting can mitigate this issue. Here we will set a maximun depth of the tree.*

In [151…
```
# Create and train the decision tree classifier with the best depth
clf = DecisionTreeClassifier(max_depth=25,random_state=42)
clf.fit(X_train, y_train)
```

Out[151…
```
DecisionTreeClassifier(max_depth=25, random_state=42)
```

In [152…
```
# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

train_preds = model.predict(X_train)
test_preds = model.predict(X_test)

# Calculate accuracy
train_accuracy = accuracy_score(y_train, train_preds)
test_accuracy = accuracy_score(y_test, test_preds)

# Create a DataFrame
accuracy_df = pd.DataFrame({
    'Dataset': ['Training', 'Test'],
    'Accuracy': [train_accuracy, test_accuracy]
})

# Display the table
accuracy_df
```

Out[152…

	Dataset	Accuracy
0	Training	0.925831
1	Test	0.923754

Accuracy

In [154…
```
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
```

```
f1 = f1_score(y_test, y_pred)
f1_df = pd.DataFrame({
    'data':["Train","test"],
    'Precision': [precision_score(y_train, train_preds), precision],
    'Recall': [recall_score(y_train, train_preds), recall],
    'F1 Score': [f1_score(y_train, train_preds), f1]
})
f1_df
```

Out[154…

	data	Precision	Recall	F1 Score
0	Train	0.933924	0.916758	0.925261
1	test	0.908178	0.927133	0.917558

Training and Testing accuracy, precision , recall and f1-score all are similar which can indicate that the model is not overfitted.

In [156…

```
cv_scores_dt = cross_val_score(clf, X_train, y_train, cv=5,scoring='accuracy').mean() # Using 5-fold cross-vali
print("Mean Cross-Validation Score Desicion tree:", cv_scores_dt)
```

```
Mean Cross-Validation Score Desicion tree: 0.9156292448868852
```

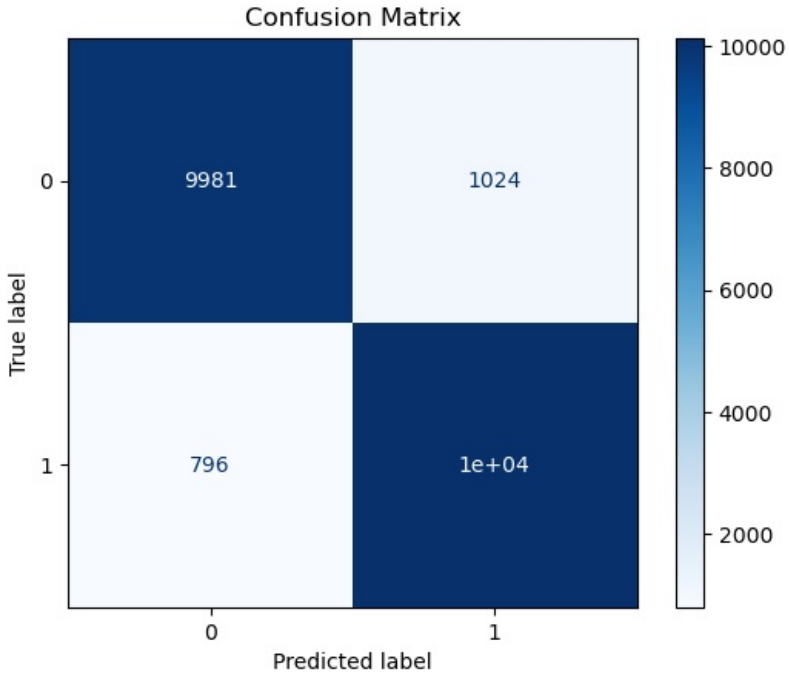
Mean Cross-Validation Score

The model will be 91% accurate when encountered with unseen data

In [159…

```
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
 [[ 9981  1024]
 [  796 10128]]
```

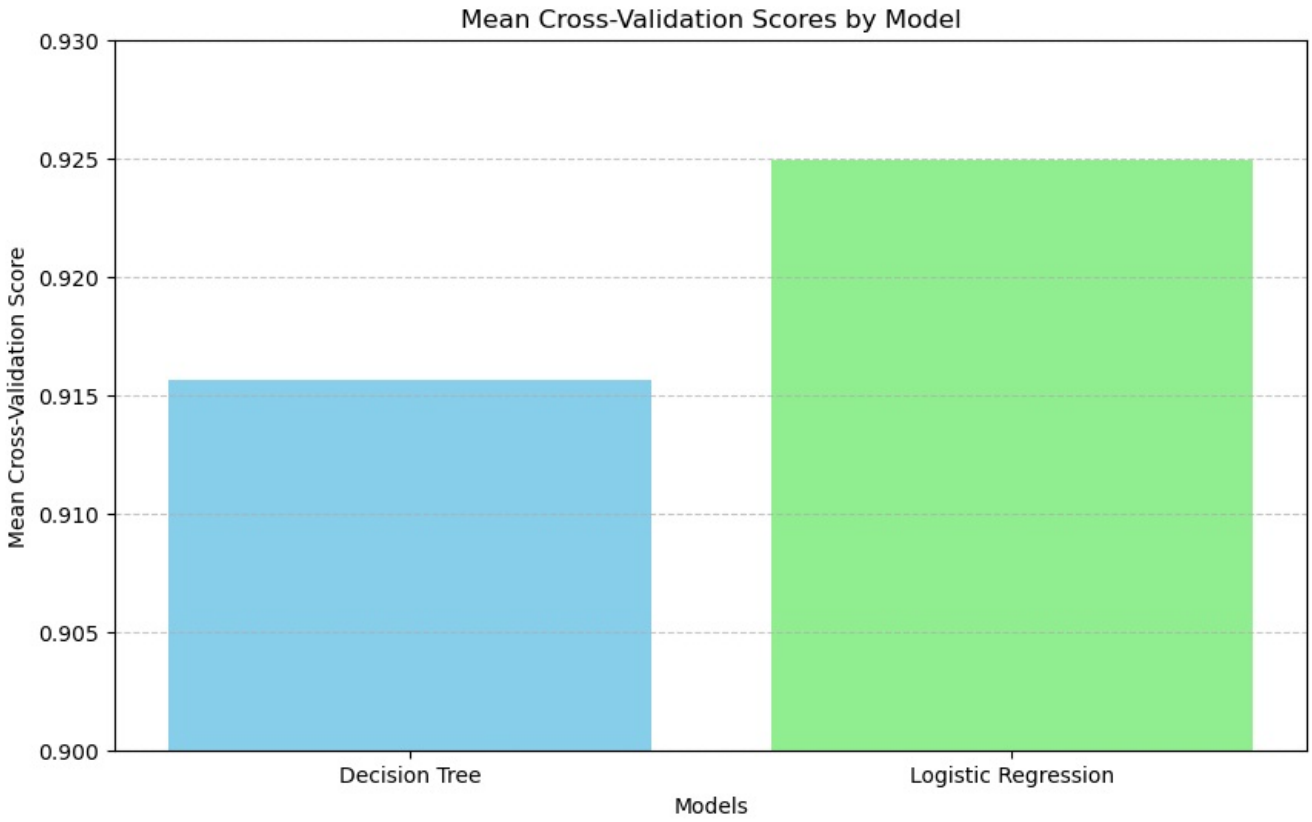


Bar plot for mean cross validation scores of Logistic Regression and Decision tree

In [161…

```
models = ['Decision Tree', 'Logistic Regression']
mean_scores = [cv_scores_dt, cv_scores_lr]

# Create the bar plot
plt.figure(figsize=(10, 6))
plt.bar(models, mean_scores, color=['skyblue', 'lightgreen'])
plt.xlabel('Models')
plt.ylabel('Mean Cross-Validation Score')
plt.title('Mean Cross-Validation Scores by Model')
plt.ylim(0.9, 0.93)  # Set the y-axis limits for better visualization
plt.grid(axis='y', linestyle='--', alpha=0.7)
```

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



Logistic Regression has a slightly better mean cross validation score than decision tree classifier.

RFE (Recursive Feature Elimination).

It is a feature selection technique in machine learning that recursively removes features from a model based on their importance.

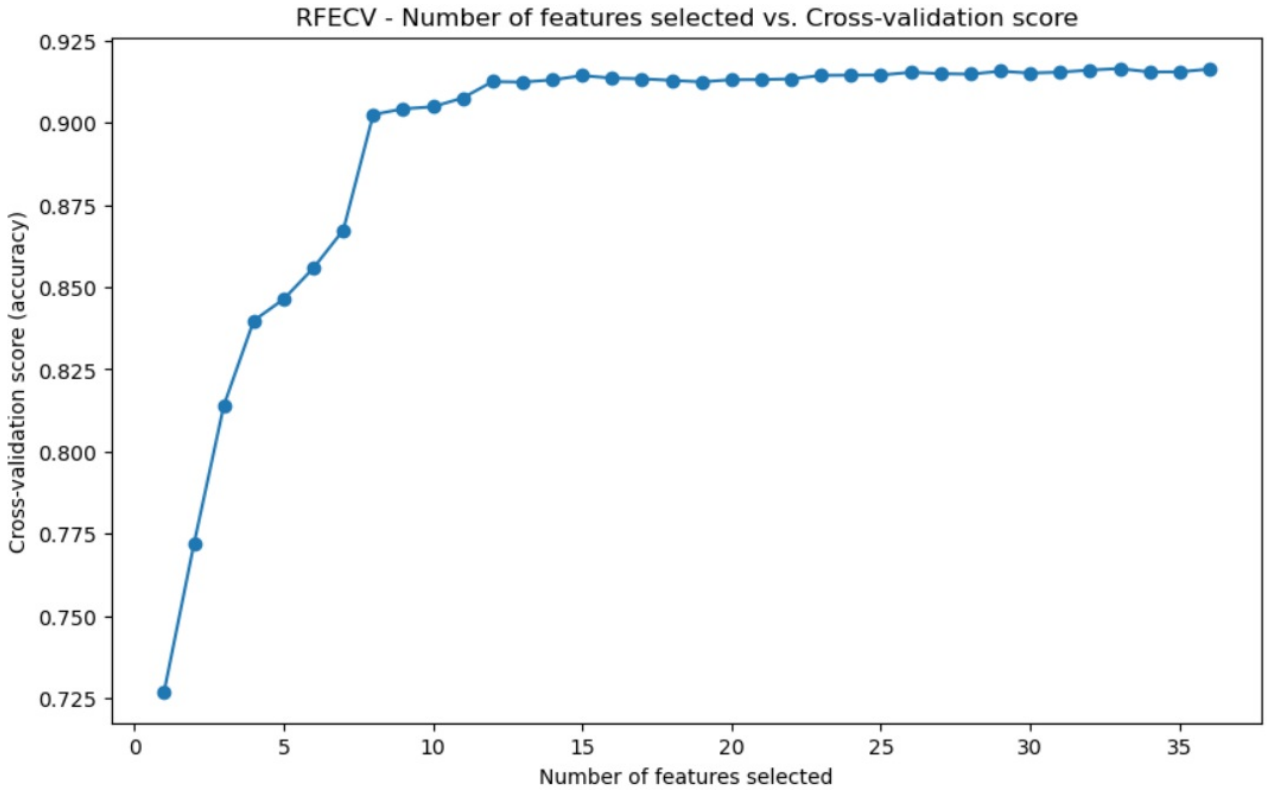
Large datasets often suffer from the "curse of dimensionality." With a vast number of features, training machine learning models becomes computationally expensive and prone to overfitting. This is where feature selection comes in.

Advantages of RFE

- **Reduces Overfitting**: *Eliminates irrelevant features to decrease model complexity.*
- **Enhances Generalization**: *Improves model performance on unseen data.*
- **Increases Interpretability**: *Results in simpler, more understandable models.*
- **Highlights Important Features**: *Focuses on the most impactful features for the model.*
- **Efficient Resource Use**: *Lowers computational cost, storage, and memory usage.*
- **Handles Multicollinearity**: *Identifies and removes redundant features.*

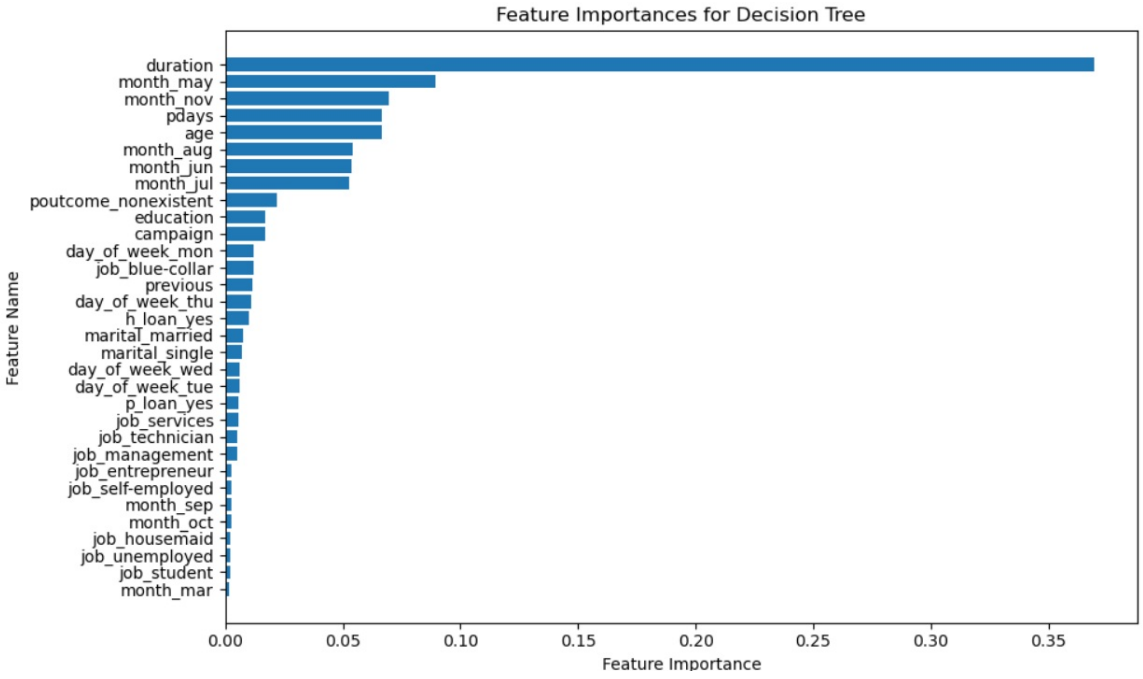
RFE: A Step-by-Step Selection Process:

- **Start with All Features**: *It begins by considering all features in your dataset.*
- **Train and Evaluate**: *It trains a machine learning model (e.g., Decision Tree classifier) using the entire feature set.*
- **Rank and Eliminate**: *It analyzes the model's performance and ranks features based on their importance. The least important features (those contributing minimally to the model's accuracy) are then eliminated.*
- **Repeat and Refine**: *This process is repeated iteratively. The model is retrained on the reduced feature set, and features are continuously removed until the desired number (e.g., top 5-6) or a pre-defined performance metric is achieved.*



- *The cross-validation scores suggest that using around 10 features provides a good balance between model complexity and performance.*
- *The decision tree with the selected features achieves a high accuracy of approximately 91.76%, and the cross-validation scores indicate stable and reliable performance.*
- *The plateauing of the cross-validation scores after around 10 features validates the effectiveness of the RFE process, confirming that further feature addition does not significantly enhance the model's predictive ability.*

```
Accuracy of the decision tree with selected features:  0.9176433033882074
Cross-validation scores:
[0.72695282 0.77196238 0.81404023 0.83981863 0.8463463  0.85582513
 0.86739502 0.90251537 0.90425481 0.90488013 0.90757719 0.91256089
 0.91240449 0.91304951 0.91437843 0.91359671 0.91340122 0.91291264
 0.91246317 0.91310809 0.91316669 0.91336217 0.91441754 0.9144957
 0.91453482 0.91537518 0.91496479 0.9148084  0.91570741 0.91514069
 0.91541434 0.91605921 0.91650878 0.91547298 0.91551203 0.9163524 ]
```



The above Bar plot shows that some features have a greater impact on the prediction than others.The first 10 to 12 features should make up around 90% of the whole data set. We could choose only those to train a model to get accurate predictions(In case of a large dataset).

Conclusion

Given the manageable size of our dataset, employing Recursive Feature Elimination (RFE) may not be necessary.

Conclusion

This project focused on analyzing direct marketing campaigns conducted by a banking institution via phone calls to predict whether clients would subscribe to a term deposit. This prediction aims to optimize marketing strategies and improve campaign effectiveness.

Data Preparation:

- *The dataset was thoroughly analyzed and cleaned to enhance its quality.*
- *Columns were either cleaned or dropped as necessary.*
- *Duplicate rows were removed.*

Feature Engineering:

- *All the features were encoded to numeric values for easy modeling and analysis.*
- *Implementing SMOTE to address class imbalance was essential to enhance the robustness of subsequent modeling efforts.*
- *we also use feature engineering in selection features for logistic regression(embedded method)*

Exploratory Data Analysis (EDA):

- *Visualizations provided profound insights into the dataset.Notably, the histogram revealed that individuals at younger and older ages demonstrated a higher likelihood of subscribing to a term deposit.*
- *Conversely, those in the middle age range (30-60 years) exhibited lower inclination, possibly due to financial commitments associated with familial responsibilities.*
- *The scatter plot informed us of average calls made to different age groups*

Modeling and Comparative Analysis:

- *Logistic Regression and Decision Tree classifiers were employed and their performances compared.*
- *Logistic Regression emerged slightly superior with a mean cross-validation score of 0.9253** compared to Decision Tree's **0.9161**.*
- *Random Forest is indeed a viable algorithm for model training. However, when dealing with a small and straightforward dataset, simpler, faster algorithms are often preferred. These algorithms, such as Logistic Regression or Decision Trees, offer advantages like quicker training times and easier interpretability without compromising predictive accuracy.*

Consideration of Feature Selection Techniques:

- *While Recursive Feature Elimination (RFE) was deemed unnecessary for this dataset, its potential utility, particularly with more complex datasets, was acknowledged.*

The findings emphasize data-driven decision-making's pivotal role in optimizing marketing strategies sand enabling organizations to achieve better outcomes in subscription initiatives.

***Educational Campaigns:** Launch educational campaigns aimed at younger and older demographics to explain the benefits and flexibility of term deposits. Emphasize features such as low-risk investment, guaranteed returns, and long-term financial planning benefits to alleviate concerns about financial constraints or time availability.*

***Product Innovation:** Introduce flexible term deposit options that cater to varying financial capabilities and time constraints. For example, offer shorter-term deposits or customizable options that align with different customer needs and preferences.*