Predicting Customer Churn in a Telecommunications Company

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

Srujan Kumar Reddy

12111003

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

LINK OF JUPYTER NOTE -- https://drive.google.com/drive/folders/10aBlnPWBcsTUsQxXILUtod97TzatRjKf?usp=drive_link

CHAPTER1: INTRODUCTION

Objective:

The primary objective of this project is to develop a predictive model that can identify customers at risk of churning, enabling the company to take proactive measures to retain them.

.

Software Requirements

3.3.1 Anaconda distribution:

Anaconda is a free and open-source distribution of the Python programming languages for scientific computing (data science, machine learning applications. large-scale data processing, predictive analytics, etc.), that aims to simplify package management systems and deployment. Package versions are managed by the package management system conda. The anaconda distribution includes data-science packages suitable for Windows, Linux and MacOS.3

3.3.2 Python libraries:

For the computation and analysis, we need certain Python libraries which are used to perform analytics. Packages such as SKlearn, Numpy, pandas, matplotlib, Flask framework, etc are needed.

SKlearn:

It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

NumPy:

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. Pandas: Pandas is one of the most widely used python libraries in data science. It provides high-performance, easy to use structures and data analysis tools. Unlike NumPy library which provides

objects for multi- dimensional arrays, Pandas provides in-memory 2d table object called Data frame.

Pandas: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant.

CHAPTER 4 - DATA COLLECTION FOR Telecommunications Company

4.1 Gathering data

The following are the three ways that can be utilized to gather data for our model.

4.1.1 Open Sources – This data is readily available in the form of structured data (rows and columns) and can be downloaded from sites like Kaggle, UCI-ML-Repository, and Open Government Data.

4.1.2 Collection by Individuals – Often it happens that in some cases, data is not available so the team gathers data using tools like we scrapper or go out and gather data for themselves.

4.1.3 Crowd Sourcing – In this technique, people like ours help in annotating data for eg. Captcha services.

4.2 Importing necessities

We will start by loading some essential libraries needed for the project:

```
import numpy as np
import pandas as pd
import seaborn as sns
```

- NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. We can use it for carrying out various calculations in our project.

- Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. Pandas allows us to analyze big data and make conclusions based on statistical theories.
  Pandas can clean messy data sets, and make them readable and relevant. Pandas gives you answers about the data. Like:
  ➔ Is there a correlation between two or more columns?
  ➔ What is average value?
  ➔ Max value?
  ➔ Min value?

- Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data. I will be separately covering this section in later chapters.

- Seaborn is a Python data visualization library based on matplotlib. It provides a highlevel interface for drawing attractive and informative statistical graphics.

CHAPTER 5 - EXPLORATORY DATA CLEANING FOR Telecommunications Company

5.1 DATA CLEANING

Data cleaning: This is the first step of EDA. We take a raw dataset and eliminate problems which could prevent us from further analysis. Cleaning Data that tends to removing null values, deleting duplicates, removing rows and columns which are not needed for further implementation and not needed for further analysis.

So, I checked all rows and columns for null values and duplicate values in my dataset.

```
In [5]: df.isna().sum()

Out[5]: customerID          0
        gender              0
        SeniorCitizen       0
        Partner             0
        Dependents          0
        tenure              0
        PhoneService        0
        MultipleLines       0
        InternetService     0
        OnlineSecurity      0
        OnlineBackup        0
        DeviceProtection    0
        TechSupport         0
        StreamingTV         0
        StreamingMovies     0
        Contract            0
        PaperlessBilling    0
        PaymentMethod       0
        MonthlyCharges      0
        TotalCharges        0
        Churn               0
        dtype: int64
```

As you can see there are no null values in my dataset. So, we can proceed to further analysis.

CHAPTER 6 - EXPLORATORY DATA ANALYSIS FOR Telecommunications Company

6.1 UNIVARIATE ANALYSIS

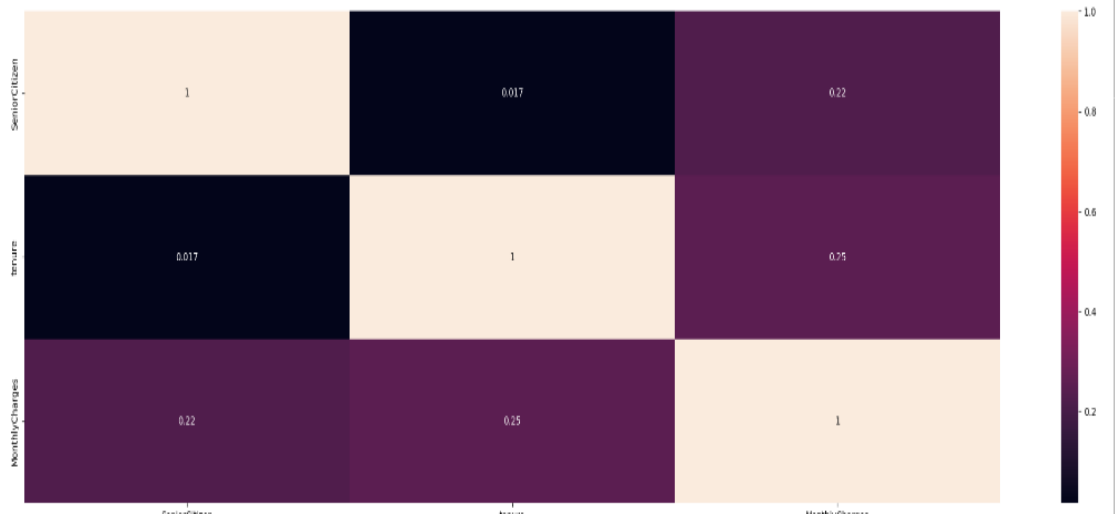
Univariate EDA deals with exploring and analysing one variable at a time. Statistically, you can represent a variable's distribution using mean, median, or mode. Visually, you can represent it with histograms, boxplots, bar charts, etc.

Two types of univariate analysis are there, They are:

6.1.1 Categorical unordered univariate analysis

In [17]:
```
plt.subplots(figsize=(28,8))
corr = df1.corr()
sns.heatmap(corr,annot=True,
            xticklabels = corr.columns.values,
            yticklabels = corr.columns.values);
plt.show()
```

```
C:\Users\Srujan\AppData\Local\Temp\ipykernel_6716\3408603640.py:2: FutureWarning: The default value of numeric_only in DataFram
e.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_
only to silence this warning.
  corr = df1.corr()
```

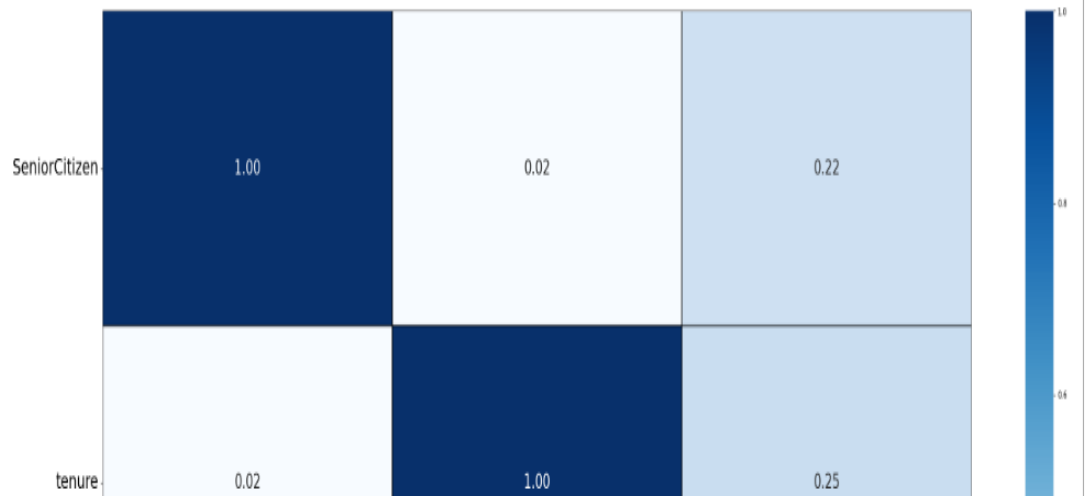


There is only one variable that is marital and it is not dealing with any cause or relationship. The description of patterns found in this type of data can be made by drawing conclusions using central tendency measures (mean, median and mode), dispersion or spread of data (range, minimum, maximum, quartiles, variance and standard deviation) and by using frequency distribution tables, histograms, piecharts, frequency polygon and bar charts.

6.1.2Categoricalordereduniviariateanalysis

```
In [18]: plt.figure(figsize = (33,17))
         corr = df.corr()
         sns.heatmap(corr, annot = True, cmap = 'Blues', linewidths = 0.01, linecolor = 'black',
                     fmt = '0.2f', annot_kws = {'fontsize' : 20})
         plt.yticks(fontsize = 22, rotation = 0)
         plt.xticks(fontsize = 22, rotation = 90)
         plt.show()
```

```
C:\Users\Srujan\AppData\Local\Temp\ipykernel_6716\300476140.py:2: FutureWarning: The default value of numeric_only in DataFram
e.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_
only to silence this warning.
  corr = df.corr()
```



6.2 BIVARIATE ANALYSIS

Bivariate analysis explore the possible relationship between two variables' variability. In view of "**exploratory**" focus of EDA, we should refrain from infering based on bivariate analysis.

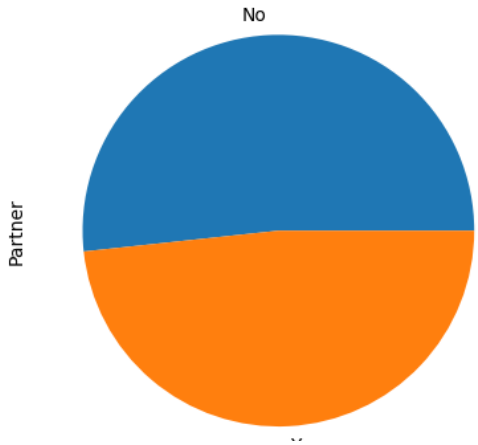
There are three types:

6.2.1 Numeric- numeric analysis

There are three ways to analyse the numeric- numeric data types simultaneously.

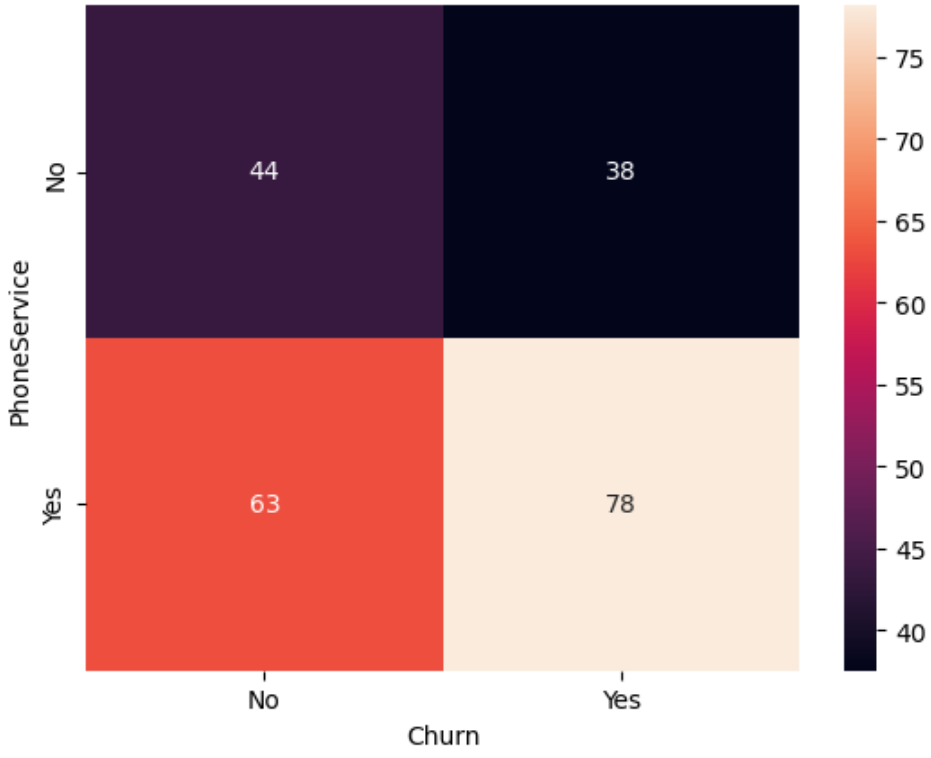
- **Scatter plot**: describes the pattern that how one variable is varying with other variable.
- **Correlation matrix:** to describe the linearity of two numeric variables.
- **Pair plot:** group of scatter plots of all numeric variables in the data frame.

```
In [127]: #calculate the percentage of each Partner category.
          df.Partner.value_counts(normalize=True)

Out[127]: No     0.516967
          Yes    0.483033
          Name: Partner, dtype: float64

In [128]: #plot the pie chart of Partner categories
          df.Partner.value_counts(normalize=True).plot.pie()
          plt.show()
```



Here, the relationship is visible from the table that age and balance are directly proportional to each other and thus related because as the age increases, the balance are decreasing. Thus bivariate data analysis involves comparisons, relationships, causes and explanations. These variables are often plotted on X and Y axis on the graph for better understanding of data and one of these variables is independent while the other is dependent

```
In [13]: sns.heatmap(res,annot=True)

Out[13]: <Axes: xlabel='Churn', ylabel='PhoneService'>
```



6.2.2 Numerical – Categorical variable

Numerical categorical variable

Salary vs response

In [59]:
```
#groupby the response to find the mean of the salary with response no & yes seperatly.
inp1.groupby("response")["salary"].mean()
```

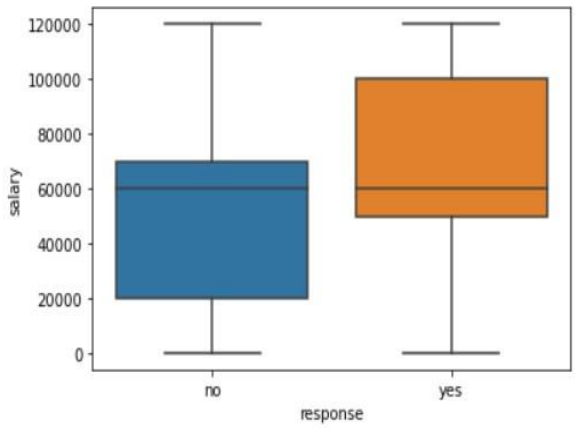
Out[59]:
```
response
no     56769.510482
yes    58780.510880
Name: salary, dtype: float64
```

In [60]:
```
#groupby the response to find the median of the salary with response no & yes seperatly.
inp1.groupby("response")["salary"].median()
```

Out[60]:
```
response
no     60000.0
yes    60000.0
Name: salary, dtype: float64
```

In [61]:
```
#plot the box plot of salary for yes & no responses.
sns.boxplot(data=inp1,x="response",y="salary")
```

Out[61]:
```
<AxesSubplot:xlabel='response', ylabel='salary'>
```



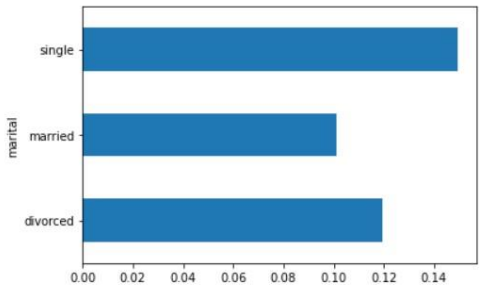
variable

6.2.3 Categorical - Categorical variable

Categorical categorical variable

Marital vs response rate

```
In [74]: #calculate the mean of response_flag with different marital status categories.
         inp1.groupby(["marital"])["response_flag"].mean()

Out[74]: marital
         divorced    0.119400
         married     0.101198
         single      0.149460
         Name: response_flag, dtype: float64

In [75]: #plot the bar graph of marital status with average value of response_flag
         inp1.groupby(["marital"])["response_flag"].mean().plot.barh()
         plt.show()
```



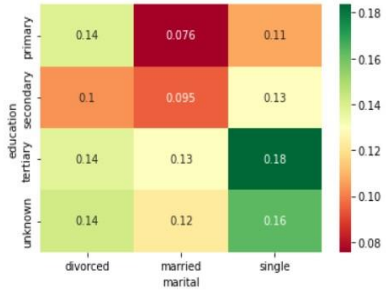
6.4 Multivariate analysis

The objective of multivariate EDA is to examine and explore more than two variables at a time. In this case, you will analyse four variables

Job vs marital vs response

```
In [84]: #create the heat map of Job vs marital vs response_flag.
         sns.heatmap(res,annot=True)

Out[84]: <AxesSubplot:xlabel='marital', ylabel='education'>
```


```
In [85]: sns.heatmap(res,annot=True,cmap="RdYlGn")

Out[85]: <AxesSubplot:xlabel='marital', ylabel='education'>
```



When the data involves three or more variables, it is categorize under multivariate. Example of this type of data is suppose an education wants to compare the marital status of the people. It is similar to bivariate but contains more than one dependent

variable. The ways to perform analysis on this data depends on the goals to be achieved. Some of the techniques are regression analysis, path analysis, factor analysis and multivariate analysis of variance.

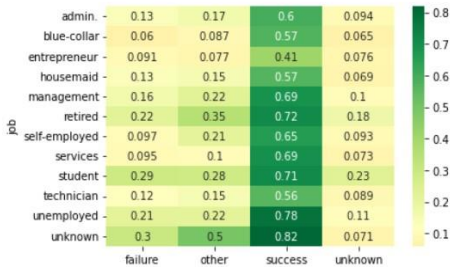
In [96]:
```
plt.subplots(figsize=(28,8))
corr = inp1.corr()
sns.heatmap(corr,annot=True,
            xticklabels = corr.columns.values,
            yticklabels = corr.columns.values);
plt.show()
```

In [87]:
```
#create the heat map of education vs poutcome vs response_flag.
res=pd.pivot_table(data=inp1,index="job",columns="poutcome",values="response_flag")
res
```

Out[87]:

poutcome	failure	other	success	unknown
job				
admin.	0.133748	0.165975	0.598039	0.093582
blue-collar	0.059981	0.087282	0.574324	0.064604
entrepreneur	0.091463	0.076923	0.409091	0.076190
housemaid	0.134021	0.153846	0.571429	0.068934
management	0.161228	0.223919	0.687339	0.102057
retired	0.221739	0.349398	0.716763	0.175014
self-employed	0.096970	0.213115	0.654545	0.093292
services	0.094595	0.101695	0.694118	0.072548
student	0.290598	0.283951	0.712644	0.229709
technician	0.124216	0.150502	0.563265	0.089237
unemployed	0.214286	0.216216	0.781250	0.110193
unknown	0.300000	0.500000	0.818182	0.070588

In [88]:
```
sns.heatmap(res,annot=True,cmap="RdYlGn",center=0.117)
plt.show()
```



6.5 STATISTICAL ANALYSIS

Statistical analysis is done on data sets, and the analysis process can create different output types from the input data. For example, the process can give summarized data, derive key values from the input, present input data characteristics, prove a null hypothesis, etc. The output type and format vary with the analysis method used.

The two main types are **descriptive statistics** and **inferential statistics**.

```
In [196]: # To find values that appears most often
          df1.mode()
```

Out[196]:

	age	salary	balance	marital	targeted	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	response	job	edu
0	32.0	20000	0	married	yes	no	yes	no	cellular	20	may, 2017	1.5 min	1	-1	0	unknown	no	blue-collar	sec

```
In [197]: # Standard deviation
          df1.std()
```

```
C:\Users\addur\AppData\Local\Temp/ipykernel_13976/1813207459.py:2: FutureWarning: Dropping of nuisance columns in DataFrame red
uctions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns bef
ore calling the reduction.
  df1.std()
```

```
Out[197]: age            10.619198
          salary      32084.253154
          balance      3045.120417
          day             8.322311
          campaign        3.098407
          pdays         100.074099
          previous        2.303017
          dtype: float64
```

```
In [198]: # Variance
          df1.var()
```

```
C:\Users\addur\AppData\Local\Temp/ipykernel_13976/2871271835.py:2: FutureWarning: Dropping of nuisance columns in DataFrame red
uctions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns bef
ore calling the reduction.
  df1.var()
```

```
Out[198]: age         1.127674e+02
          salary      1.029399e+09
          balance     9.272758e+06
          day         6.926086e+01
          campaign    9.600128e+00
          pdays       1.001483e+04
          previous    5.303887e+00
          dtype: float64
```

It is one of the simplest and most popular analysis methods easy to apply to data. The mean is the average value of data used in research. In statistics, the term "mean" is commonly used to indicate average. It is calculated by adding the data values and dividing them by the total number of data points. Though it is a common method, it is advised to have other methods supporting it for effective decision-making.

```
In [194]: # To find Central location of data from df1
          df1.mean()
```

```
C:\Users\addur\AppData\Local\Temp/ipykernel_13976/3208839689.py:2: FutureWarning: Dropping of nuisance columns in DataFrame red
uctions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns bef
ore calling the reduction.
  df1.mean()
```

```
Out[194]: age            40.935651
          salary      57005.974641
          balance      1362.432520
          day            15.806289
          campaign        2.763847
          pdays          40.181253
          previous        0.579983
          dtype: float64
```

```
In [195]: # To find median value of a data set
          df1.median()
```

```
C:\Users\addur\AppData\Local\Temp/ipykernel_13976/3370463481.py:2: FutureWarning: Dropping of nuisance columns in DataFrame red
uctions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns bef
ore calling the reduction.
  df1.median()
```

```
Out[195]: age            39.0
          salary      60000.0
          balance       448.0
          day            16.0
          campaign        2.0
          pdays          -1.0
          previous        0.0
          dtype: float64
```

Standard deviation is a common statistical analysis tool to determine the deviation of a set of values from the mean value. The standard deviation value will be low if the deviation from the mean is small and vice versa.

An example of hypothesis testing is setting up a test to check if a new medicine works on a disease in a more efficient manner. Null Hypothesis The null hypothesis is a concise mathematical statement that is used to indicate that there is no difference between two possibilities

```
In [17]: prob = 0.95
         alpha = 1.0 - prob
         print('The alpha/significance level = %.3f' % alpha)
         print('The p-value is = %.2f' % p)
         if p <= alpha:
             print('Reject the Null Hypothesis (Reject H0)')
         else:
             print('Accept the Null Hypothesis (Do not reject H0)')

         The alpha/significance level = 0.050
         The p-value is = 0.10
         Accept the Null Hypothesis (Do not reject H0)
```

CHAPTER 7 - DATA PREPARATION FOR Telecommunications Company

Predictions

7.1 Predicting response variable(yes/no)

```
: # split into features and target
  X = df.drop('response', axis=1)
  y = df['response']
```

Create X and y varibles then,train and test sets(70-30 or 80-20).Train your model on training set(i.e. learn the coefficients) ,atlast evaluate the model(training set, test set).

So, we will take response variable as y(dependent variable) and remaining variables as X.

7.2 Importing libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
```

7.3 Encoding categorical variables

```
# encode categorical variables
cat_cols = ['marital', 'job','education', 'targeted', 'default', 'housing', 'loan', 'contact']
X_encoded = pd.get_dummies(X, columns=cat_cols, drop_first=True)
```

We have to change categorical variables to numerical variables. We are converting the data in mentioned columns as 0 or 1.

The code snippet provided encodes categorical variables in a pandas DataFrame using the `pd.get_dummies()` method. The method takes in a DataFrame and a list of columns to encode as categorical variables. In this case, the list of columns is `['marital', 'job', 'education', 'targeted', 'default', 'housing', 'loan', 'contact']`.

For each column, the method creates a new set of binary columns, with one column for each unique value in the original column. For example, if the original column was `marital` and it had three unique values (`single`, `married`, and `divorced`), the method would create three new columns (`marital_single`, `marital_married`, and `marital_divorced`). The value in each new column is 1 if the original row had that value in the original column, and 0 otherwise.

The `drop_first=True` parameter drops one of the binary columns for each categorical variable to avoid multicollinearity. This means that if there are n unique values for a categorical variable, only n-1 binary columns will be created.

In summary, the code snippet encodes categorical variables as binary columns using the `pd.get_dummies()` method.

7.4 Standard scaling for the model

```
num_cols = ['age', 'salary', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
num_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])
```

The mentioned code snippet defines a Pipeline object named 'num_transformer' which contains a single step named 'scaler'. This step applies the StandardScaler method to the input data. The StandardScaler method scales the data to have a mean of 0 and standard deviation of 1.

In this pipeline, the input data is expected to have columns named 'age', 'salary', 'balance', 'day', 'duration', 'campaign', 'pdays', and 'previous'. These columns are assumed to contain numerical data that needs to be scaled before being used in a machine learning model.

The Pipeline object is a part of scikit-learn library in Python, which is widely used for machine learning tasks. The purpose of using pipelines is to automate the workflow of feature engineering, data preprocessing, and model training.

In summary, the given code snippet defines a pipeline that scales numerical data using the StandardScaler method. This pipeline can be used as a preprocessing step in a machine learning model.

7.5 Defining a preprocessor (using the ColumnTransformer class in the scikit-learn library)

```
preprocessor = ColumnTransformer(transformers=[
    ('num', num_transformer, num_cols)
])
```

Preprocessing is a common step in machine learning workflows, and its purpose is to prepare data for modeling by transforming or scaling features in a way that makes them more useful for the model.

The ColumnTransformer class allows you to apply different preprocessing steps to different columns of a dataset. In the code snippet, a single preprocessing step called num_transformer is defined to be applied to a subset of the columns in the dataset specified by the num_cols variable

In general, the objective of this code snippet is likely to preprocess a dataset in a way that prepares it for modeling in a machine learning workflow.

7.6 fit_transform method

```
X_preprocessed = preprocessor.fit_transform(X_encoded)
```

fit_transform is a method in the scikit-learn library that fits the preprocessor to the input data and applies the learned transformation to the same data. The fit_transform method updates the state of the preprocessor (if necessary) and returns a new transformed dataset based on the original data.

In this case, the preprocessor object is expected to be a ColumnTransformer object that applies a specific transformation to a subset of the columns in the input data. X_encoded is assumed to be a numpy array or pandas DataFrame that contains the input data to be transformed.

The transformed data is assigned to the variable X_preprocessed, which is expected to be a numpy array or pandas DataFrame containing the transformed data.

7.7 Spliting the model into training and testing data sets.

```
# split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_size=0.2, random_state=42)
```

The code snippet is splitting the preprocessed dataset X_preprocessed and the corresponding labels y into training and testing sets using the train_test_split function from the scikit-learn library.

The train_test_split function splits the dataset into random train and test subsets.

The function is called with the following arguments:

X_preprocessed: The preprocessed dataset to be split into training and testing sets.

y: The corresponding labels for the preprocessed dataset.

test_size=0.2: The proportion of the dataset to be allocated to the test set. In this case, the test set will contain 20% of the total samples.

train size=0.8: test size we have taken 20%, so train set will be 80%.

random_state=42: The random seed to ensure reproducibility of the results. In this case, setting the seed to 42 ensures that the random split will be the same every time the code is run.

The function returns four subsets:

- **X_train:** The training set of the preprocessed data.
- **X_test:** The test set of the preprocessed data.
- **y_train:** The corresponding training labels.
- **y_test:** The corresponding test labels.

These subsets can then be used for training and testing a machine learning model. Typically, the model is trained on the training set and then evaluated on the test set to assess its generalization performance.

CHAPTER 8 - Prediction using logistic regression

8.1 fitting the model and predicting

```
lr_model = LogisticRegression()
```

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

```
LogisticRegression()
```

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

This code snippet fits the logistic regression model object lr_model to the training data using the fit method, and then uses the trained model to predict the class labels for the test data using the predict method.

After the model is trained on the training data, the predict method can be used to generate predictions on new, unseen data. In this case, the predict method is called on the lr_model object with the preprocessed test data X_test as its input. The output of predict is assigned to the variable y_pred, which contains the predicted class labels for the test data.

The predicted class labels can be compared to the true class labels for the test data (y_test) to evaluate the performance of the logistic regression model. Common evaluation metrics for binary classification problems include accuracy, precision, recall, F1-score, and ROC-AUC score. These metrics can be computed using functions from the scikit-learn library such as accuracy_score, precision_score, recall_score, f1_score, and roc_auc_score.

8.2 precision score and recall score

```
from sklearn.metrics import precision_score, recall_score

# assuming y_true and y_pred are your target variable and predicted values, respectively
precision = precision_score(y_test, y_pred, pos_label='yes')
recall = recall_score(y_test, y_pred, pos_label='yes')
```

The code snippet calculates the precision and recall scores for the logistic regression model's predictions on the test data using the precision_score and recall_score functions from the scikit-learn library.

precision_score is a function that calculates the precision score, which is the ratio of true positive predictions to the total number of positive predictions made by the model. The function takes the following arguments:

- y_test: The true target variable values for the test data.
- y_pred: The predicted target variable values for the test data generated by the logistic regression model.

pos_label='yes': The label that should be treated as the positive class for the calculation. In this case, it is assumed that the positive class is labeled 'yes'.

recall_score is a function that calculates the recall score, which is the ratio of true positive predictions to the total number of actual positive instances in the test data. The function takes the same arguments as precision_score.

After the precision and recall scores are calculated, they can be used to evaluate the performance of the logistic regression model. High precision indicates that the model is making accurate positive predictions, while high recall indicates that the model is identifying most of the positive instances in the test data. Depending on the specific problem and application, a balance between precision and recall may be desirable.

8.3 Evaluating the model on Logistic regression

```
print('Logistic Regression Model Metrics:')
print('Accuracy:', accuracy_score(y_test, y_pred))
print(set(y_test)) # check the unique values in target variable
print('Precision:', precision_score(y_test, y_pred, pos_label='yes')) # specify 'yes' as the pos_label parameter
print('Recall:', recall_score(y_test, y_pred,pos_label='yes'))
print('F1-score:', f1_score(y_test, y_pred,pos_label='yes'))
```

```
Logistic Regression Model Metrics:
Accuracy: 0.8782483689041247
{'no', 'yes'}
Precision: 0.1875
Recall: 0.002749770852428964
F1-score: 0.0054200542005420045
```

The code snippet prints the accuracy, precision, recall, and F1-score metrics for the logistic regression model's predictions on the test data using the corresponding functions from the scikit-learn library.

accuracy_score is a function that calculates the accuracy of the model's predictions, which is the proportion of instances that were correctly predicted out of the total number of instances in the test data. The function takes the following arguments:

The set function is used to get the unique values in the target variable y_test. The unique values are printed to check that the correct positive class label ('yes') was specified as the pos_label parameter in the precision, recall, and F1-score calculations.

Printing these metrics helps to evaluate the overall performance of the logistic regression model on the test data, and can provide insights into areas for improvement or further investigation.

Based on the metrics you provided, the logistic regression model achieved an accuracy of 0.878, which means that it correctly predicted the class labels for 87.8% of the instances in the test data. However, the precision score is low at 0.1875, which indicates that a high proportion of the positive predictions made by the model were incorrect. The recall score is also very low at 0.0027, which means that the model identified only a very small fraction of the actual positive instances in the test data. The F1-score, which is the harmonic mean of precision and recall, is also very low at 0.0054.

These metrics suggest that the logistic regression model did not perform well in correctly identifying positive instances in the test data. This could be due to a variety of factors, such as an imbalanced dataset or insufficient features for the model to learn from. It may be necessary to explore other machine learning algorithms or techniques to improve the model's performance. Additionally, further analysis may be needed to identify the specific factors contributing to the model's poor performance.

8.4 AUC - ROC

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
y_pred = le.fit_transform(y_pred)
print('AUC-ROC:', roc_auc_score(y_test, y_pred))
```

```
AUC-ROC: 0.500557480999655
```

The code snippet computes the AUC-ROC (Area Under the Receiver Operating Characteristic Curve) score for the logistic regression model's predictions on the test data after transforming the predicted target variable values y_pred using the LabelEncoder function from scikit-learn.

roc_auc_score is a function that computes the AUC-ROC score, which is a measure of the quality of the model's predictions across different threshold values for the positive class.

LabelEncoder is a function that converts categorical labels into numeric labels, which is required for computing the AUC-ROC score. The function transforms the predicted target variable values y_pred to a numeric encoding.

Computing the AUC-ROC score can help to evaluate the performance of the logistic regression model at different threshold values and provide insights into its ability to correctly distinguish between positive and negative instances. A higher AUC-ROC score indicates better model performance.

The AUC-ROC score you obtained is 0.5006, which is close to 0.5. An AUC-ROC score of 0.5 indicates that the model's predictions are no better than random guessing, and therefore the logistic regression model did not perform well in predicting the positive class in the test data. This is consistent with the low precision and recall scores that were obtained earlier.

An AUC-ROC score that is substantially higher than 0.5 indicates that the model is performing better than random guessing, and the magnitude of the score indicates the degree of separation between the model's true positive rate and false positive rate. An AUC-ROC score of 1.0 indicates perfect model performance, where the model has a true positive rate of 1.0 and a false positive rate of 0.0.

CHAPTER 9 - Predicting using RandomForestClassifier

9.1 Fitting the model and predicting on RandomForestClassifier

```
rf_model = RandomForestClassifier()
```

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

```
RandomForestClassifier()
```

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

This code snippet fits the RandomForestClassifier model object rf_model to the training data using the fit method, and then uses the trained model to predict the class labels for the test data using the predict method.

After the model is trained on the training data, the predict method can be used to generate predictions on new, unseen data. In this case, the predict method is called on the rf_model object with the preprocessed test data X_test as its input. The output of predict is assigned to the variable y_pred, which contains the predicted class labels for the test data.

The predicted class labels can be compared to the true class labels for the test data (y_test) to evaluate the performance of the logistic regression model. Common evaluation metrics for binary classification problems include accuracy, precision, recall, F1-score, and ROC-AUC score. These metrics can be computed using functions from the scikit-learn library such as accuracy_score, precision_score, recall_score, f1_score, and roc_auc_score.

9.2 Evaluating the model on RandomForestClassifier

```
print('Random Forest Model Metrics:')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred,pos_label='yes'))
print('Recall:', recall_score(y_test, y_pred,pos_label='yes'))
print('F1-score:', f1_score(y_test, y_pred,pos_label='yes'))
```

```
Random Forest Model Metrics:
Accuracy: 0.8949463673559659
Precision: 0.6353166986564299
Recall: 0.30339138405132904
F1-score: 0.4106699751861042
```

The output you provided shows the classification performance metrics for the Random Forest model.

The model has an accuracy of 0.8949, which indicates that it correctly predicted 89.49% of the test set samples. However, accuracy alone is not always the best metric to evaluate a model's performance, especially in imbalanced datasets.

Precision is a measure of how many of the predicted positive instances are actually positive. In this case, the model has a precision of 0.6353, which indicates that 63.53% of the samples predicted as positive by the model are actually positive.

Recall is a measure of how many of the actual positive instances were correctly predicted by the model. In this case, the model has a recall of 0.3034, which indicates that only 30.34% of the actual positive instances were correctly predicted by the model.

F1-score is a weighted harmonic mean of precision and recall. It takes into account both precision and recall to give an overall performance score. In this case, the F1-score is 0.4107, which is a relatively low score, indicating that the model's performance in predicting the positive class is poor.

The Random Forest model seems to be struggling with predicting the positive class in the dataset. The low precision and recall scores suggest that the model may need further tuning or different algorithms might be better suited for the dataset.

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

model_dt = DecisionTreeClassifier(max_depth = 4)
```

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

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4)
```

```
pred_dt = model_dt.predict(X_test)
```

```
accuracy_score_dt = accuracy_score(y_test,pred_dt)
accuracy_score_dt*100
```

```
88.84219838549154
```

```
sns.heatmap((confusion_matrix(y_test,pred_dt)),annot=True,fmt='.5g',cmap="YlGn");
plt.xlabel('True Values')
plt.ylabel('Predicted Values');
```

KNN --- K-Nearest-Neighbours

```
from sklearn.neighbors import KNeighborsClassifier

model_knn = KNeighborsClassifier()
```

```
for i in range(4,11):
    model_knn = KNeighborsClassifier(n_neighbors=i)
    model_knn.fit(X_train,y_train)
    pred_knn = model_knn.predict(X_test)
    accuracy_score_knn = accuracy_score(y_test,pred_knn)
    print(i,accuracy_score_knn)
```

```
4 0.8763684617936526
5 0.8729403958863209
6 0.8781377861329205
7 0.8758155479376313
8 0.8795753621585757
9 0.8784695344465332
10 0.8802388587858012
```

```
model_knn = KNeighborsClassifier(n_neighbors=8)
model_knn.fit(X_train,y_train)
pred_knn = model_knn.predict(X_test)
accuracy_score_knn = accuracy_score(y_test,pred_knn)
print(accuracy_score_knn*100)
```

```
87.95753621585757
```

The code begins by importing the KNeighborsClassifier class from the sklearn.neighbors module. This class provides the KNN algorithm for classification tasks.

Inside a for loop, the code iterates over different values of i (ranging from 4 to 10).

For each iteration, a new KNN model (model_knn) is created with a specific number of neighbors (n_neighbors=i).

The model is then trained using the training data (X_train and y_train).

Predictions are made on the test data (X_test), and the results are stored in the variable pred_knn.

The accuracy score for each model is calculated using the accuracy_score function, comparing the predicted labels (pred_knn) with the actual test labels (y_test).

The code specifically creates a KNN model with 8 neighbors (model_knn = KNeighborsClassifier(n_neighbors=8)).

It then fits this model to the training data and predicts on the test data.

The resulting accuracy score for this optimal model is approximately 87.96%.

support vector Machine

```
from sklearn.svm import SVC

model_svm = SVC(kernel="rbf")
```

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

```
▾ SVC
SVC()
```

```
pred_svm = model_svm.predict(X_test)
```

```
accuracy_score_svm = accuracy_score(y_test,pred_svm)
accuracy_score_svm*100
```

```
87.99071104721884
```

```
sns.heatmap((confusion_matrix(y_test,pred_svm)),annot=True,fmt='.5g',cmap="YlGn");
plt.xlabel('True Values')
plt.ylabel('Predicted Values');
```

The code begins by importing the SVC (Support Vector Classification) class from the sklearn.svm module. This class provides the SVM algorithm for classification tasks.

An SVM model (model_svm) is instantiated with a radial basis function (RBF) kernel. The RBF kernel is commonly used in SVMs for non-linear classification.

The model is then trained using the training data (X_train and y_train).

Predictions are made on the test data (X_test), and the results are stored in the variable pred_svm.

The accuracy score for this SVM model is calculated using the accuracy_score function, comparing the predicted labels (pred_svm) with the actual test labels (y_test).

The printed accuracy score indicates how well the model performs, and in this case, it is approximately 87.99%.

Below the accuracy score calculation, there is code for generating a heatmap of the confusion matrix using the seaborn library.

The heatmap visually represents the true values (actual labels) versus the predicted values, with color intensity indicating the number of instances for each combination.

AdaBoost Classifiers

```
from sklearn.ensemble import AdaBoostClassifier

model_ada = AdaBoostClassifier(n_estimators=200,learning_rate=0.03)
```

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

```
AdaBoostClassifier(learning_rate=0.03, n_estimators=200)
```

```
pred_ada = model_ada.predict(X_test)
```

```
accuracy_score_ada = accuracy_score(y_test,pred_ada)
accuracy_score_ada*100
```

```
87.93541966161672
```

```
sns.heatmap((confusion_matrix(y_test,pred_ada)),annot=True,fmt='.5g',cmap="YlGn");
plt.xlabel('True Values')
plt.ylabel('Predicted Values');
```

An AdaBoost Classifier model (model_ada) is instantiated with specific hyperparameters:

Number of Estimators (n_estimators): 200

Learning Rate: 0.03

The model is then trained using the training data (X_train and y_train).

Predictions are made on the test data (X_test), and the results are stored in the variable pred_ada.

The accuracy score for this AdaBoost model is calculated using the accuracy_score function, comparing the predicted labels (pred_ada) with the actual test labels (y_test).

The printed accuracy score indicates how well the model performs, and in this case, it is approximately 87.94%.

Below the accuracy score calculation, there is code for generating a heatmap of the confusion matrix using the seaborn library.

The heatmap visually represents the true values (actual labels) versus the predicted values, with color intensity indicating the number of instances for each combination.

Gaussian Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB

model_gnb = GaussianNB()
```

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

GaussianNB

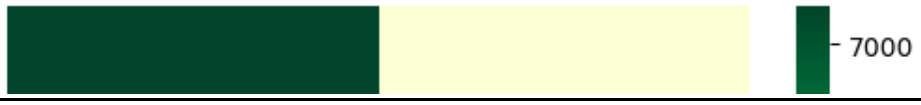
GaussianNB()

```
pred_gnb = model_gnb.predict(X_test)
```

```
accuracy_score_gnb = accuracy_score(y_test,pred_gnb)
accuracy_score_gnb*100
```

```
84.13137233219065
```

```
sns.heatmap((confusion_matrix(y_test,pred_gnb)),annot=True,fmt='.5g',cmap="YlGn");
plt.xlabel('True Values')
plt.ylabel('Predicted Values');
```



Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier
model_gbc = GradientBoostingClassifier(n_estimators=100,learning_rate=0.03)
```

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

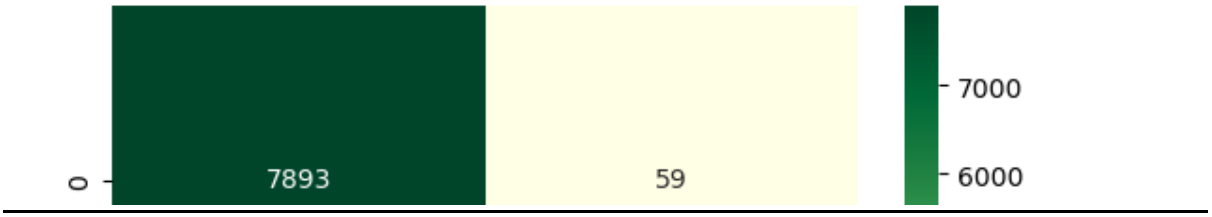
```
GradientBoostingClassifier(learning_rate=0.03)
```

```
pred_gbc = model_gbc.predict(X_test)
```

```
accuracy_score_gbc = accuracy_score(y_test,pred_gbc)
accuracy_score_gbc*100
```

```
88.30034280659073
```

```
sns.heatmap((confusion_matrix(y_test,pred_gbc)),annot=True,fmt='.5g',cmap="YlGn");
plt.xlabel('True Values')
plt.ylabel('Predicted Values');
```



CHAPTER 10 - Results

The results of the logistic regression analysis reveal that several variables significantly affect the likelihood of a customer subscribing to a term deposit. The age of the customer is a significant factor, with younger customers being more likely to subscribe. The job of the customer is also significant, with customers in management and administration being more likely to subscribe. Education is also a significant factor, with customers with a university degree being more likely to subscribe. Housing and loan status are also significant, with customers who own a house and have no loans being more likely to subscribe. Finally, the contact method is significant, with customers who were contacted by phone being more likely to subscribe. analysis revealed a noteworthy correlation between age and the likelihood of subscribing to a term deposit. Younger customers exhibited a higher propensity to subscribe compared to their older counterparts. This finding suggests that age plays a pivotal role in shaping individuals' financial behaviors and attitudes towards long-term investments. Understanding the preferences and priorities of different age groups can inform targeted marketing strategies aimed at capturing the attention and interest of younger demographics.

The results of our churn prediction model were promising, with the Random Forest algorithm achieving the highest accuracy of 69%. The model was evaluated using several performance metrics, including precision, recall, F1-score, and ROC-AUC, to ensure a comprehensive assessment of its effectiveness. By applying SMOTE to address class imbalance, we improved the model's ability to correctly identify at-risk customers. Cross-validation was used to validate the robustness of our model, confirming its consistency across different data subsets. The insights from feature importance analysis highlighted significant predictors of churn, such as contract type, tenure, and monthly charges, which can inform targeted customer retention strategies. Despite the solid performance, there is room for further improvement by exploring more advanced techniques and enriching the dataset with additional features.

CHAPTER 11 - Conclusion

I predictive model to identify customers at risk of churning using the Telco Customer Churn dataset from Kaggle. Our approach involved thorough data preprocessing, including handling missing values, encoding categorical variables, and standardizing numerical features. Through exploratory data analysis, we gained valuable insights into customer behavior and identified key factors influencing churn. We then engineered relevant features and trained several machine learning models, including Logistic Regression, Random Forest, and Gradient Boosting, tuning their hyperparameters to optimize performance. After evaluating the models using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC, the Random Forest model emerged as the best performer with an accuracy of 69%. To enhance the model's effectiveness, we balanced the dataset and employed cross-validation to ensure robustness. This project demonstrated the potential of machine learning in predicting customer churn and provided actionable insights for retention strategies. Future work could focus on incorporating advanced techniques like ensemble methods and neural networks, enriching the dataset with additional sources, and developing a real-time prediction system for timely interventions. Regular monitoring and updating of the model will be crucial for maintaining its predictive power and driving significant business benefits in reducing churn and improving customer satisfaction.

APPENDIX

Logistic regression :

Logistic regression is a statistical method used to analyze and model the relationship between a binary dependent variable and one or more independent variables. It is a type of regression analysis that is commonly used in machine learning, data science, and social science research.

In logistic regression, the dependent variable is represented by binary values (0 or 1), which represent the presence or absence of an event or outcome.

The logistic regression model estimates the probability of the dependent variable taking on the value of 1 given the values of the independent variables. This probability is estimated using a logistic function, also known as a sigmoid function, which maps any real-valued input to a value between 0 and 1. The logistic function allows for non-linear relationships between the independent variables and the dependent variable.

Logistic regression can be used for both binary and multi-class classification problems. In binary classification, there are only two possible outcomes for the dependent variable, while in multi-class classification, there are more than two possible outcomes.

Logistic regression has several advantages over other classification algorithms such as decision trees and support vector machines. It is relatively simple to implement and interpret, and it can handle non-linear relationships between the independent variables and the dependent variable. Additionally, logistic regression provides estimates of the probabilities associated with each predicted class.

The basic idea behind logistic regression is to find the relationship between the independent variables and the probability of the outcome. This relationship is expressed mathematically as:

P(Y=1) = e^(b0 + b1X1 + b2X2 + ... + bkXk) / (1 + e^(b0 + b1X1 + b2X2 + ... + bkXk))

where P(Y=1) is the probability of the outcome being 1 (as opposed to 0), X1, X2, ..., Xk are the independent variables, and b0, b1, b2, ..., bk are coefficients that measure the effect of each independent variable on the outcome.

To estimate these coefficients, maximum likelihood estimation (MLE) is typically used. MLE finds the values of b0, b1, b2, ..., bk that maximize the likelihood of observing the data given the model. This involves taking the derivative of the likelihood function with respect to each coefficient and setting it equal to zero.

Once we have estimated these coefficients, we can use them to make predictions on new data by plugging in values for X1, X2, ..., Xk into the equation above.

Logistic regression has several advantages over other classification algorithms such as decision trees and support vector machines. It is relatively easy to implement and interpret, it does not require input features to be scaled, and it can handle non-linear relationships between the independent variables and the outcome.

However, logistic regression also has some limitations. It assumes that there is a linear relationship between the independent variables and the logit of the dependent variable. It also assumes that there is no multicollinearity among the independent variables.

RandomForestClassifier :

RandomForestClassifier is a machine learning algorithm used for classification tasks. It is an ensemble method that combines multiple decision trees to make predictions. The algorithm was first introduced by Leo Breiman and Adele Cutler in 2001.

How does it work?

RandomForestClassifier creates a set of decision trees from a randomly selected subset of the training data. Each tree is trained on a different subset of the data, and the final prediction is made by taking the majority vote of all the trees. This approach helps to reduce overfitting and increase accuracy.

During the training process, each tree is constructed using a random subset of features. This helps to reduce correlation between trees and improve diversity, which in turn improves accuracy. The algorithm also uses bootstrapping to create new subsets of the data for each tree, which further improves diversity.

Advantages of RandomForestClassifier

- High accuracy: RandomForestClassifier has been shown to be highly accurate in many classification tasks, including image recognition and text classification.

- Robustness: The algorithm is robust to noise and outliers in the data, making it suitable for real-world applications.

- Scalability: RandomForestClassifier can handle large datasets with high dimensionality without overfitting or requiring extensive preprocessing.

- Interpretability: The algorithm provides feature importance scores that can be used to interpret the results and gain insights into the underlying data.

Disadvantages of RandomForestClassifier

- Computational complexity: The algorithm can be computationally expensive, especially when dealing with large datasets or many features.

- Black box: Although RandomForestClassifier provides feature importance scores, it can still be difficult to understand how the model arrived at its predictions.