PROJECT FILE

SevenMentor & Training Pvt Ltd.

(Shivaji Nagar Pune,411005)

SUBMITTED BY: -

Mr. ABASAHEB GOVIND SARGAR

UNDER SUPERVISION OF: -

Mr. NISHESH GOGIA

Acknowledgement

It is great pleasure for me to undertake this project. I feel highly doing the project entitled " CUSTOMER CHURN PREDICTION".

I am grateful to my project guide Mr. Nishesh Gogia Faculty Member of SEVEN MENTOR INSTITUTE and Miss. Arati, Administrative Officer of SEVEN MENTOR INSTITUTE.

This project would not have completed without their enormous help and worthy experience. Whenever I was in need, they were there behind me.

Although, this report has been prepared with utmost care and deep routed interest. Even then I accept respondent and imperfection.

Mr. Nishesh Gogia (Project Guide)

Mr. Abasaheb Govind Sargar (Data Science Batch I Seven Mentor Institute)

INDEX

INTRODUCTION

There is a Bank who wants to retain their existing customers who may churn, for that they have to know which existing customers has the highest probability of leaving the company and which customers have lowest probability of leaving the company. Churning of customers is a big problem for banks, they want to maximise the retention of the customers so that they can plan their future projects. Also, if a Bank wants to get a funding from big investors, Rate of churning plays an important role. So, Bank wants to build a Machine Learning model which can predict whether a particular customer will churn or not.

Now let's move toward the Machine learning project.Basically, we are going to predict here either person will stay or exit the bank using 13 different features that we have in our data. We are going toanalyse different machine learning models like SVM, KNN, LOGISTIC REGRESSION, RANDOM FOREST, and DECISION TREE because this is a binary classification problem. First, we will clean the data, and analyseit using different visualization techniques and then we will develop abovementioned ML models. We will compare these models on the basis of accuracy, precision, recall, F1 score, and confusion matrix and then choose one out of them with the highest accuracy and F1 score.

Let's go to our last project which is converting Long URL to Short URL. Longer URLs are truncated by search engines, in web browsers, and many other areas. Visitors like to see clean, human- readable URLs since it makes them easy to read, remember, and type. Basically, we are using dictionary here to store all data, we will use lowercase alphabets to create our shorts URLs and the length will be 6 to 10 characters. Along with this we are going to return Long URLs for short URLs. Here we have to understand that short URLs is a key for LongURLs and it will always unique.

Last project We are creating simple Interest calculator using Tkinter library and procedural way of programming.

LIBRARIES USED

NUMPY –NumPy is a Python library used for working with arrays.
It also has functions for working in the domain of linear algebra,Fourier transform, and matrices.

PANDAS – Pandas is a Python library used for working with data sets. It has functions for analysing, cleaning, exploring, and manipulating data.

Matplotlib. pyplot - Matplotlib is a low-level graph plotting library inpython that serves as a visualization utility.

SEABORN - Seaborn is a high-level graph plotting library in pythonthat serves as a visualization utility.

SK-LEARN - Scikit-learn (Sklearn) is **the most useful and robust library for machine learning in Python**. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression,clustering and dimensionality reduction via a consistence interface in Python.

PROJECT-I

CUSTOMER CHURN PREDICTION

BUSINESS PROBLEM: -

Banking is one of those traditional industries that has gone through a steady transformation over the decades. Yet, many banks today with a sizeable customer base hoping to gain a competitive edge have not tapped into the vast amounts of data they have, especially in solving one of the most acknowledged problems – customer churn.

While retaining existing customers and thereby increasing their lifetime value is something everyone acknowledges as being important, there is little the banks can do about customer churn when they don't see it coming in the first place. This is where predicting churn at the *right time* becomes important, especially when clear customer feedback is absent. Early and accurate churn prediction empowers CRM and customer experience teams to be creative and proactive in their engagement with the customer. In fact, by simply reaching out to the customer early enough, 11% of the churn can be avoided.
But how do you look for signs of churn? Collecting comprehensive feedback about the customer's experience can be a challenging task. For one, surveys are expensive and infrequent. Moreover, not all customers receive it or care to respond. So where else can you pick up signs of potential customer disengagement? The answer lies in extracting early warning signs from the already existing data.
There is a Bank who wants to retain their existing customers who may churn, for that they have to know which existing customers has the highest probability of leaving the company and which customers have lowest probability of leaving the company. Churning of customers is a big problem for banks, they want to maximise the retention of the customers so that they can plan their future projects. Also, if a Bank wants to get a funding from big investors, Rate of churning plays an important role. So, Bank wants to build a Machine Learning model which can predict whether a particular customer will churn or not.

SOLUTION: -

Advanced machine learning (ML) and data science (DS) techniques can learn from past customer behaviour and external triggers that led to churn and use this learning to predict the future occurrence of a churn-like event.

Data: -

We have 10000 rows with 13 features and 1 target variable, every row tells us different customer details and whether they have churned or not.

ML CONSTRAINTS: -

Time - In this project time is not a constraint for us because we are not making this project for users then training time and testing time could be long. Low latency is not the priority here.
Accuracy – Errors can be very costly because they could affect the financial condition of the bank so we want good Accuracy on the model. If we can get a probabilistic interpretation of the results then it will addvalue to our ML model.

ML FORMULATION: -

We know that this is a binary classification problem, we are finding here either people are staying or exiting the bank. In the output column, wehave '0' and '1' for either people staying or leaving the bank respectively.

We are going to use 13 different features which we have in the datato fit an ML model which will predict whether people are staying or exiting.

PERFORMANCE METRICS: -

This is a classification problem; we are predicting whether peopleare staying or exiting bank. We are going to use the following performance metrics.

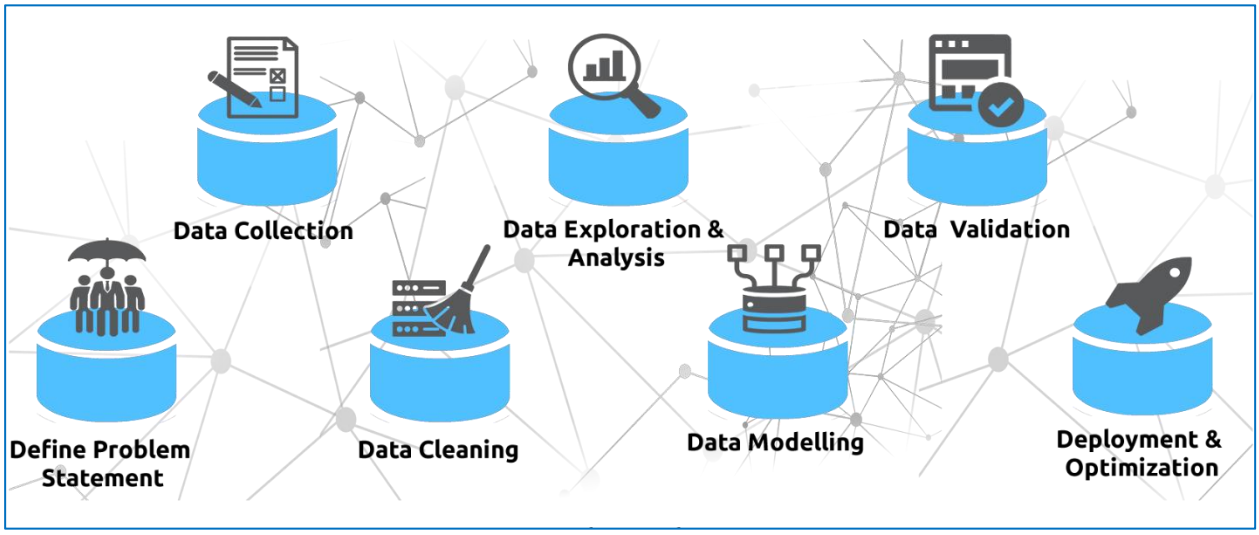
- Accuracy
- Precision
- Recall
- F1 Score
- Confusion Matrix

Most data science problems can be solved by following a standard process: -

Process Flow diagram with some details is given below. This is just one of the many ways to solve data science problems. But the general flow and concepts are similar. It does not cover deep learning and related algorithms but general idea is the same.

How to Solve Data Science Problems: -

A problem statement in data science can be solved by following the following steps:



Step 1: Define Problem Statement

Before you start a data science project, first you have to define your problem, which you will face in the project lifecycle. At this stage, you should be clear about all the objectives of your business.

Step 2: Data Collection

At this stage, you need to collect all the data needed to solve the problem. This process will be easier if you do prior research on what the data should be and where it should be collected. This process is not as easy as we think because the data does not exist in any database so that we can get it easily, we have to put a lot of effort into finding the data.

Step 3: Cleaning the Data

Data cleaning is the least preferred task for any data scientist. The task of cleaning the data includes the removal of redundant, missing words, duplication, and redundant data. According to the data scientist, it was observed that one of the time-consuming processes. But if we need explicit output then this step becomes necessary.

Step 4: Data Analysis and Exploration

Once the process of cleaning your data is complete then the next step is data analysis and data exploration. In this step, you should find out the methods and patterns used on the data. At the end of this step, you should start making hypotheses about your data and the problem you are dealing with.

Step 5: Data Modelling

Data Modelling is the process of creating a data model on the data that are stored on a database. Data modelling helps solve your problem to some extent. A model can be a Machine Learning Algorithm that is trained and tested using the data.

Step 6: Optimization and Deployment

This is the end of the process. Here you have to try to improve the efficiency of the created data model so that to make more accurate predictions. This stage is also used for checking the issues and error if caught then solve within this stage before deploying.

Coading :-

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.utils import class_weight
import warnings
warnings.filterwarnings("ignore")
```

```
df= pd.read_csv("Churn_Modelling.csv")
```

```
df.head(10)
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
5	6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	0	149756.71	1
6	7	15592531	Bartlett	822	France	Male	50	7	0.00	2	1	1	10062.80	0
7	8	15656148	Obinna	376	Germany	Female	29	4	115046.74	4	1	0	119346.88	1
8	9	15792365	He	501	France	Male	44	4	142051.07	2	0	1	74940.50	0
9	10	15592389	H?	684	France	Male	27	2	134603.88	1	1	1	71725.73	0

```
df.shape
```

```
(10000, 14)
```

```
# Checking whether there are null vaues.
df.isnull().sum()
```

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

Observation:

There are no null values in our data.

```
duplicate=df[df.duplicated()]
duplicate.head(10)
```

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited

Observation:

There are no duplicate values in our data.

```
df['Exited'].value_counts()
#1= will churn
#0= will not churn
```

```
0    7963
1    2037
Name: Exited, dtype: int64
```

Observation:

0 means those customers who have continued.

1 means those customers who have discontinued.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

```
print(df.columns)
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

Observation:

From Columns we can observe, there is 1 class label(y) i.e. "Exited" and 13 features(x).

'RowNumber', 'CustomerId', 'Surname' columns are useless in analysing this data.

So we will divide this data in Numerical and Categorical features. using this two features for next data analysis.

```
# Dividing data into Numerical features and Categorical features.
df_numerical=df[['CreditScore','Age','Balance','EstimatedSalary','Exited']]
df_catagorical= df[['Geography','Gender','NumOfProducts', 'HasCrCard',
       'IsActiveMember','Tenure','Exited']]

df_numerical
```

	CreditScore	Age	Balance	EstimatedSalary	Exited
0	619	42	0.00	101348.88	1
1	608	41	83807.86	112542.58	0
2	502	42	159660.80	113931.57	1
3	699	39	0.00	93826.63	0
4	850	43	125510.82	79084.10	0
...	...	...	...	...	...
9995	771	39	0.00	96270.64	0
9996	516	35	57369.61	101699.77	0
9997	709	36	0.00	42085.58	1
9998	772	42	75075.31	92888.52	1
9999	792	28	130142.79	38190.78	0

10000 rows × 5 columns

```
df_catagorical
```

	Geography	Gender	NumOfProducts	HasCrCard	IsActiveMember	Tenure	Exited
0	France	Female	1	1	1	2	1
1	Spain	Female	1	0	1	1	0
2	France	Female	3	1	0	8	1
3	France	Female	2	0	0	1	0
4	Spain	Female	1	1	1	2	0
...	...	...	...	...	...	...	...
9995	France	Male	2	1	0	5	0
9996	France	Male	1	1	1	10	0
9997	France	Female	1	0	1	7	1
9998	Germany	Male	2	1	0	3	1
9999	France	Female	1	1	0	4	0

10000 rows × 7 columns

```
df.describe()
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

EDA (Exploratory Data Analysis)

Exploratory data analysis will provide us more insights of a data. Exploratory Data Analysis is an approach in analysing datasets to summarize their main characteristics, often using statistical graphics and other data visualization methods.

EDA assists Data science professionals in various ways: -

1. Getting a better understanding of data

2. Identifying various data patterns

3. Getting a better understanding of the problem statement in EDA. we will visualise every feature to understand its relation with class label i.e., "Exited". Here we will do EDA separately on numerical and categorical features.

We are going to do two types of analysis

1) Univariate analysis: - Uni means one and variate means variable, so in univariate analysis, there is only one dependable variable. The objective of univariate analysis is to derive the data, define and summarize it, and analyse the pattern present in it. In a dataset, it explores each variable separately. It is possible for two kinds of variables- Categorical and Numerical.

2) Bivariate analysis: - Bi means two and variate means variable, so here there are two variables. The analysis is related to cause and the relationship between the two variables. There are three types of bivariate analysis.

3) Multivariate analysis: - Multivariate analysis is required when more than two variables have to be analysed simultaneously. It is a tremendously hard task for the human brain to visualize a relationship among 4 variables in a graph and thus multivariate analysis is used to study more complex sets of data. Types of Multivariate Analysis include Cluster Analysis, Factor Analysis, Multiple Regression. Analysis, Principal Component Analysis, etc. More than 20 different ways to perform multivariate analysis exist and which one to choose depends upon the type of data and the end goal to achieve.

Exploratory data analysis tools: -

- Specific statistical functions and techniques you can perform with EDA tools include. Clustering and dimension reduction techniques, which help create graphical displays of high-dimensional data containing many variables.
- Univariate visualization of each field in the raw dataset, with summary statistics.
- Bivariate visualizations and summary statistics that allow you to assess the relationship between each variable in the dataset and the target variable you're looking at.
- Multivariate visualizations, for mapping and understanding interactions between different fields in the data.
- K-means Clustering is a clustering method in unsupervised learning where data points are assigned into K groups, i.e. the number of clusters, based on the distance from each group's centroid. The data points closest to a particular centroid will be clustered under the same category. K-means Clustering is commonly used in market segmentation, pattern recognition, and image compression.
- Predictive models, such as linear regression, use statistics and data to predict outcomes.

There are four primary types of EDA:

- **Univariate non-graphical.** This is simplest form of data analysis, where the data being analysed consists of just one variable. Since it's a single variable, it doesn't deal with causes or relationships. The main purpose of univariate analysis is to describe the data and find patterns that exist within it.
- **Univariate graphical.** Non-graphical methods don't provide a full picture of the data. Graphical methods are therefore required. Common types of univariate graphics include:
  - Stem-and-leaf plots, which show all data values and the shape of the distribution.
  - Histograms, a bar plot in which each bar represents the frequency (count) or proportion (count/total count) of cases for a range of values.
  - Box plots, which graphically depict the five-number summary of minimum, first quartile, median, third quartile, and maximum.

- **Multivariate nongraphical:** Multivariate data arises from more than one variable. Multivariate non-graphical EDA techniques generally show the relationship between two or more variables of the data through cross-tabulation or statistics.
- **Multivariate graphical:** Multivariate data uses graphics to display relationships between two or more sets of data. The most used graphic is a grouped bar plot or bar chart with each group representing one level of one of the variables and each bar within a group representing the levels of the other variable.

Other common types of multivariate graphics include:

- Scatter plot, which is used to plot data points on a horizontal and a vertical axis to show how much one variable is affected by another.

- Multivariate chart, which is a graphical representation of the relationships between factors and a response.
- Run chart, which is a line graph of data plotted over time.
- Bubble chart, which is a data visualization that displays multiple circles (bubbles) in a two-dimensional plot.
- Heat map, which is a graphical representation of data where values are depicted by colour.

Coding: -

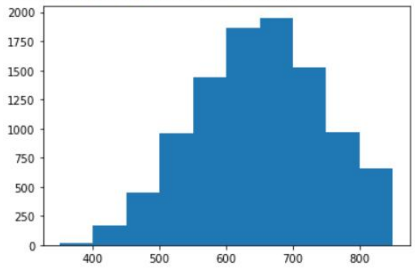
EDA on Numerical features:

UNIVARIATE EDA

1. CreditScore

```
plt.hist(df['CreditScore'])
```

```
(array([  19.,  166.,  447.,  958., 1444., 1866., 1952., 1525.,  968.,
         655.]),
 array([350., 400., 450., 500., 550., 600., 650., 700., 750., 800., 850.]),
 <BarContainer object of 10 artists>)
```

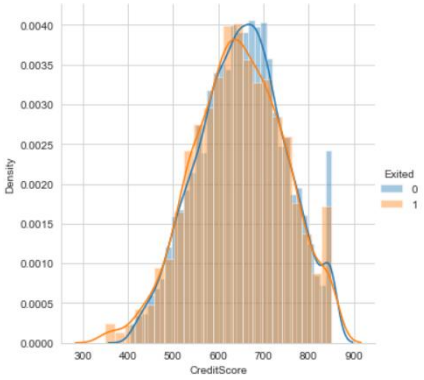


Observation:

Credit score histogram is giving us a hint to normal distributed curve, means most of the people have credit score betweeen 600

to 700 There are less people which have credit score more than 800 and which have credit score less than 500.

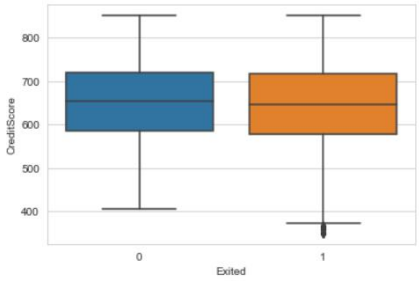
```
import warnings
warnings.filterwarnings("ignore")
sns.set_style("whitegrid");
sns.FacetGrid(df, hue="Exited", size=5) \
   .map(sns.distplot,"CreditScore")\
   .add_legend();
plt.show();
```



Observation:

PDF of CreditScore is not helping us to differentiate between customers who have stayed and left.

```
sns.boxplot(x='Exited',y='CreditScore', data=df)
plt.show()
```



Observation:

Observation: Box plot of CreditScore is not helping us to differentiate between customers who have stayed and left. But we can say that

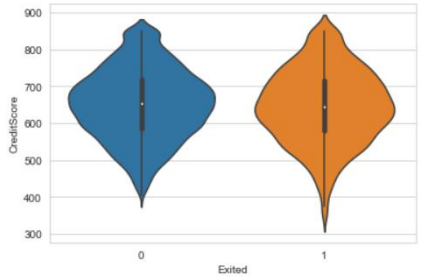
1) 75% Customers have Credit Score less than 710-720

2) 50% Customers have Credit Score less than 650-670

3) 25% customers have Credit Score less than 570-590

As per whiskers, customers who have left, have credit Score below 400. Most stable customer who have credit score 590-720.

```
sns.violinplot(x="Exited", y="CreditScore", data=df, size=8)
plt.show()
```



Observation:

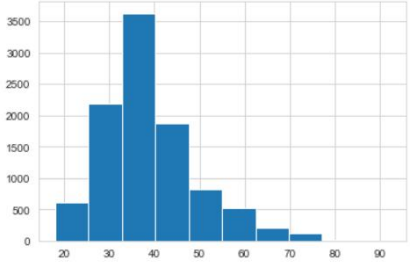
Violin Plot is combination of Box Plot and PDF.

Here violin plot is not helping in analysing data.

2. Age

```
plt.hist(df['Age'])
```

```
(array([ 611., 2179., 3629., 1871.,  828.,  523.,  208.,  127.,   20.,
          4.]),
 array([18. , 25.4, 32.8, 40.2, 47.6, 55. , 62.4, 69.8, 77.2, 84.6, 92. ]),
 <BarContainer object of 10 artists>)
```



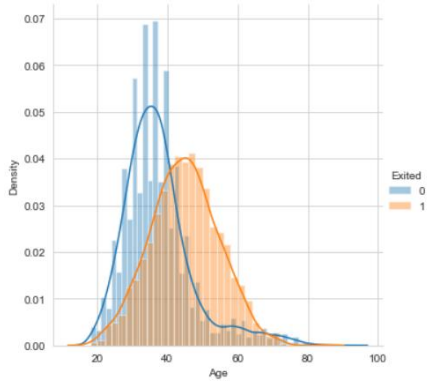
Observation:

Age histogram is giving us hint that distribution is Right skewed distrubation curve.

By observing this plot we can say that most of the customers have Age between 30-40

There are very less customers above age of 50.

```
import warnings
warnings.filterwarnings("ignore")
sns.set_style("whitegrid");
sns.FacetGrid(df, hue="Exited", size=5) \
   .map(sns.distplot,"Age")\
   .add_legend();
plt.show();
```

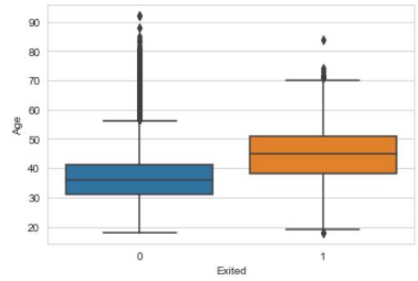


Observation:

PDF of Age is showing us that customers between age group of 30 to 40 are tend to stay.

Between age of 40 to 50 there are high chances of customers have left.

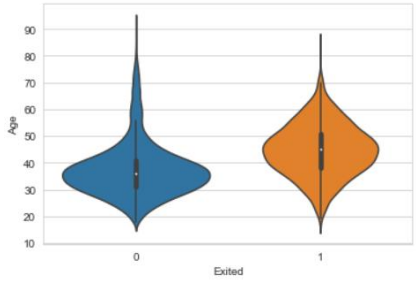
```
sns.boxplot(x='Exited',y='Age', data=df)
plt.show()
```



Observation:

As we can see IQR of Customers who stayed and who left are overplapping around age range between 39 to 41. it indication that the two groups have reasonably similar values ,the box plot not much usefull.

```
sns.violinplot(x="Exited", y="Age", data=df, size=8)
plt.show()
```



Observation:

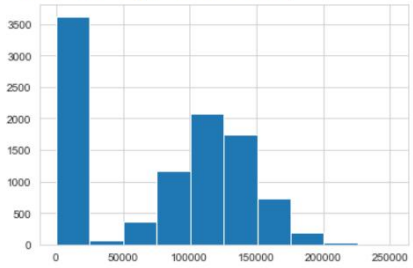
Violin Plot is combination of Box Plot and PDF.

Here violin plot is not helping in analysing data.

3. Balance

```
plt.hist(df['Balance'])
```

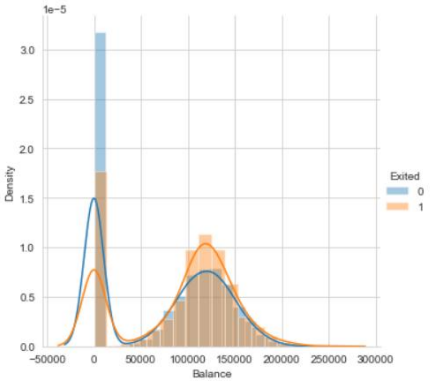
```
(array([3.623e+03, 6.900e+01, 3.600e+02, 1.173e+03, 2.081e+03, 1.747e+03,
        7.290e+02, 1.860e+02, 3.000e+01, 2.000e+00]),
 array([     0.   ,  25089.809,  50179.618,  75269.427, 100359.236,
        125449.045, 150538.854, 175628.663, 200718.472, 225808.281,
        250898.09 ]),
 <BarContainer object of 10 artists>)
```



Observation:

From Histogram we can observe that around 3600 customers have 0 balance,then 2000 customer who have balance 125000.

```
import warnings
warnings.filterwarnings("ignore")
sns.set_style("whitegrid");
sns.FacetGrid(df, hue="Exited", size=5) \
   .map(sns.distplot,"Balance")\
   .add_legend();
plt.show();
```

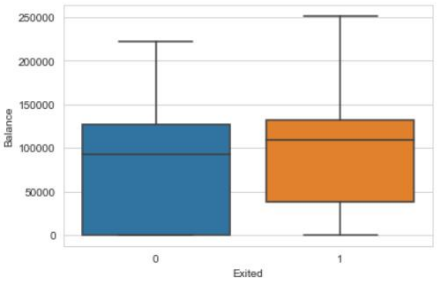


Observation:

A distplot plots a univariate distribution of observations ,Majority of Customers have 0 balance.

Though they have 0 balance, majority of them have stayed with the bank.

```
sns.boxplot(x='Exited',y='Balance', data=df)
plt.show()
```

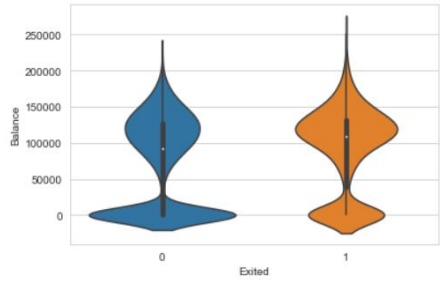


Observation:

Here we can observe that customer having balance less than 50000 are likely to stay with the bank.

Here the Median line of the plot B lies between Q2-Q3 the box of Plot A.

```
sns.violinplot(x="Exited", y="Balance", data=df, size=8)
plt.show()
```



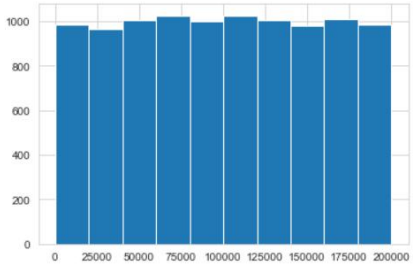
Observation:

A large number of customers who have 0 balance are likely to stay with bank.

4. EstimatedSalary

```
plt.hist(df['EstimatedSalary'])
```

```
(array([ 987.,  968., 1006., 1027., 1002., 1027., 1007.,  982., 1009.,
         985.]),
 array([1.1580000e+01, 2.0009670e+04, 4.0007760e+04, 6.0005850e+04,
        8.0003940e+04, 1.0000203e+05, 1.2000012e+05, 1.3999821e+05,
        1.5999630e+05, 1.7999439e+05, 1.9999248e+05]),
 <BarContainer object of 10 artists>)
```

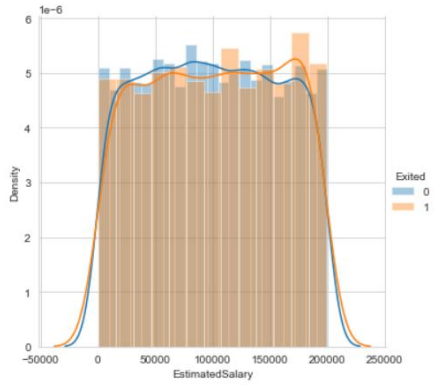


Observation:

We can see from histogram, there is not much difference in customers from different salary range.

Hence Histogram of EstimatedSalary is not useful for analysing data.both group values are likely same to other.

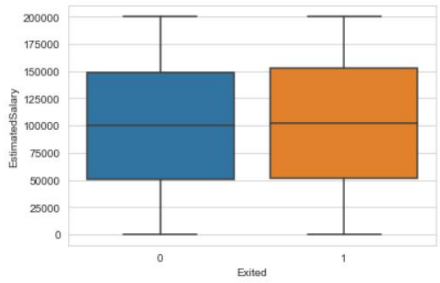
```
import warnings
warnings.filterwarnings("ignore")
sns.set_style("whitegrid");
sns.FacetGrid(df, hue="Exited", size=5) \
   .map(sns.distplot,'EstimatedSalary')\
   .add_legend();
plt.show();
```



Observation:

PDF of EstimatedSalary is also not much useful for this data analysis .

```
sns.boxplot(x='Exited',y='EstimatedSalary',data=df)
plt.show()
```

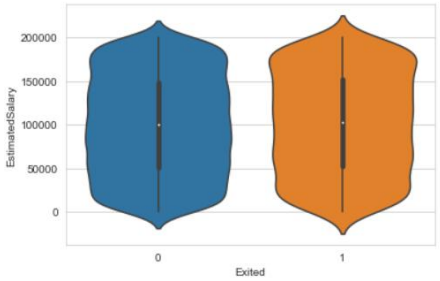


Observation:

There is no difference in box plot of stayed and exited Both belong to same salary range.

Hence Box plot is not hepful in data this analysis.

```
sns.violinplot(x="Exited", y="EstimatedSalary", data=df, size=8)
plt.show()
```



Observation:

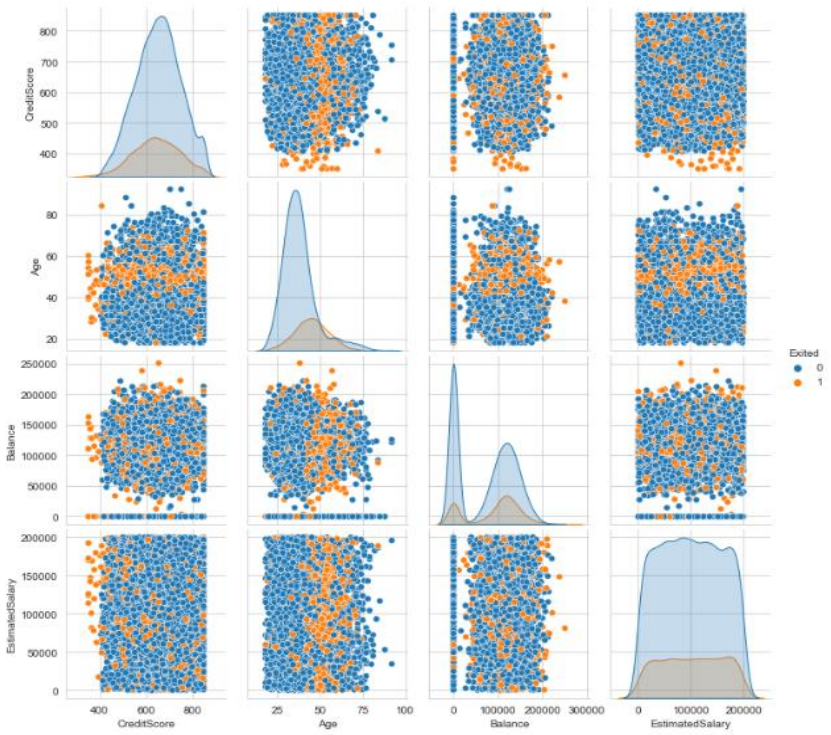
Even the Violin Plot of EstimatedSalary is not helpful in this data analysis.

BIVARIATE EDA

- **Pair-Plot: -**We will begin with Pair Plot to observe what is happening inside our numerical features.

```
sns.pairplot(df_numerical,hue='Exited')
```

```
<seaborn.axisgrid.PairGrid at 0x138d2223be0>
```



Observation:

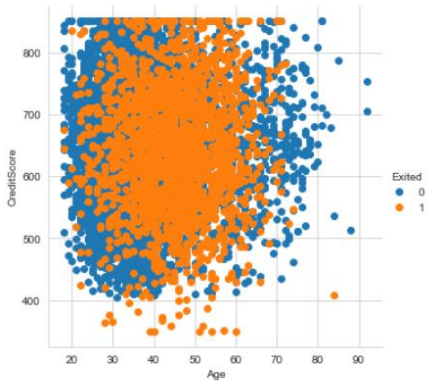
We can Observe some pattern formation between Age and EstimatedSalary.

We can also oserve pattern formation between Age and CreditScore.

2. Scatter plot using Age and CreditScore

A scatter plot represents individual pieces of data using dots

```
import warnings
warnings.filterwarnings("ignore")
sns.set_style("whitegrid");
sns.FacetGrid(df_numerical, hue="Exited", size=5) \
   .map(plt.scatter, "Age", "CreditScore")\
   .add_legend();
plt.show();
```



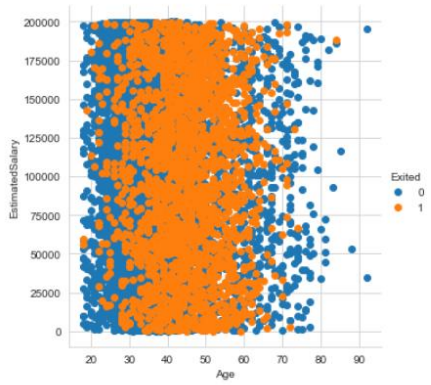
Observation:

We can see there is circular area in approximately middle of scatter plot which indicate customers who have left.

3. Scatter plot using Age and EstimatedSalary

A scatter plot represents individual pieces of data using dots.

```
import warnings
warnings.filterwarnings("ignore")
sns.set_style("whitegrid");
sns.FacetGrid(df_numerical, hue="Exited", size=5) \
   .map(plt.scatter, "Age", "EstimatedSalary" )\
   .add_legend();
plt.show();
```



Observation:

We can oserve from scatter plot of Age and EstimatedSalary that customers from age group below 30 and above 60 are likely to stay with the bank.

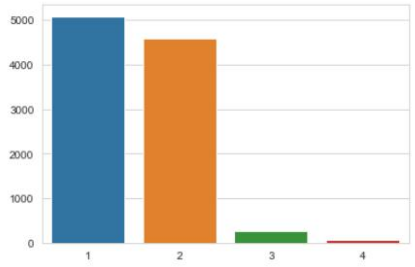
1) For NumOfProducts

```
import warnings
warnings.filterwarnings("ignore")
a=df_catagorical['NumOfProducts'].value_counts()
a
```

```
1    5084
2    4590
3     266
4      60
Name: NumOfProducts, dtype: int64
```

```
sns.barplot(x=a.index,y=a.values)
```

```
<AxesSubplot:>
```



Observation:

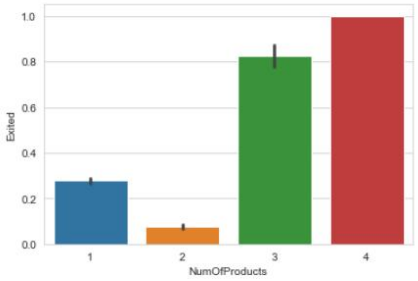
5084 customers have bought only one product from bank.

4590 customers have bought 2 products from bank.

266 customers have bought 3 products from bank.

Only 60 Customers have bought 4 products from bank.

```
sns.barplot(x='NumOfProducts', y='Exited', data=df_catagorical)
plt.show()
```



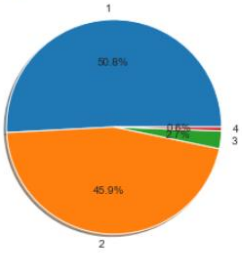
Observation:

Customers who have bought 2 products have shown least Exits.

Customers who have bought 3 or 4 products have shown highest Exits..

```
labels=a.index
sizes=a.values

plt.pie(sizes,labels=labels,autopct='%1.1f%%',
        shadow=True)
plt.axis('equal')
```

```
(-1.1059349205019084,
 1.1002826158397652,
 -1.116497450906123,
 1.1041122438861788)
```



Observation:

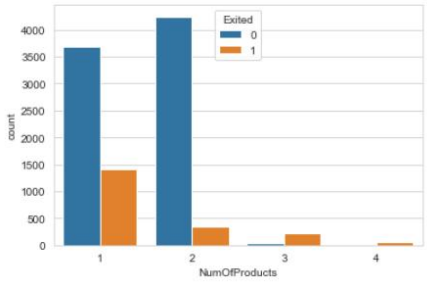
There are 50.8% customers who have bought only one product.

There are 45.9% customers who have bought 2 products.

The 2.7% customers have bought 3 products and only 0.6% customers have bought 4 products.

```
sns.countplot(data=df_catagorical, x='NumOfProducts', hue='Exited')
```

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



Observation:

Customers who have bought 2 products from bank are more likely to continue with same bank.

customers who have bought 3 or 4 products have Exited.

2) For Geography

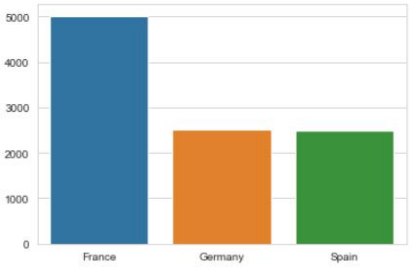
Here we are studying customer distribution across 3 countries

```
import warnings
warnings.filterwarnings("ignore")
b=df_catagorical['Geography'].value_counts()
b
```

```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

```
sns.barplot(x=b.index,y=b.values)
```

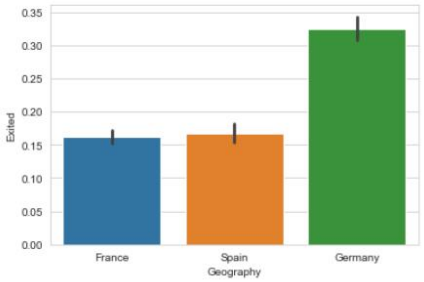
```
<AxesSubplot:>
```



Observation:

There are more number of customers in France(5014) as compared to Germany(2509) and Spain(2477).

```
sns.barplot(x='Geography', y='Exited', data=df_catagorical)
plt.show()
```

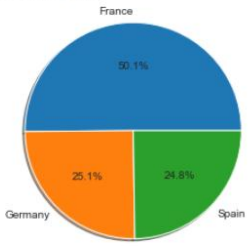


Observation:

More number of customers from Germany(33%) have Exited as compared to France(16.5%) and Spain(16%).

```
labels=b.index
sizes=b.values

plt.pie(sizes,labels=labels,autopct='%1.1f%%',
        shadow=True)
plt.axis('equal')
```

```
(-1.1011756716174903,
 1.1000559843627375,
 -1.101918599067293,
 1.1006880031680237)
```

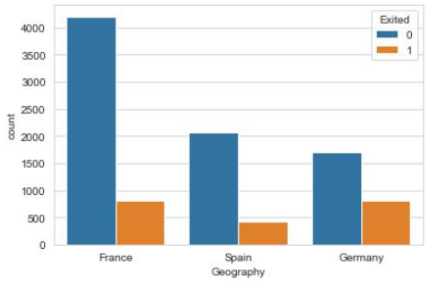


Observation:

50.1% customers belong to France, 25.1% customers belong to Germany and 24.8% customers are from Spain.

```
sns.countplot(data=df_catagorical, x='Geography', hue='Exited')
```

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



Observation:

More Churning is seen in Customers from Germany than Spain and France

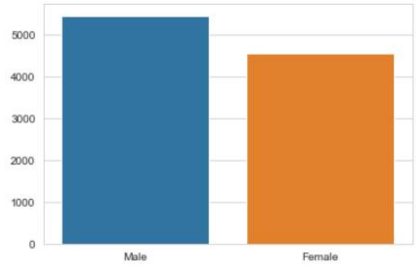
3) For Gender

```
import warnings
warnings.filterwarnings("ignore")
c=df_catagorical['Gender'].value_counts()
c
```

```
Male      5457
Female    4543
Name: Gender, dtype: int64
```

```
sns.barplot(x=c.index,y=c.values)
```

```
<AxesSubplot:>
```

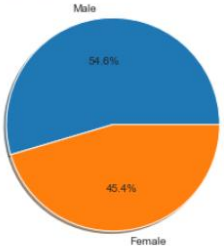


Observation:

More number of male customers(5457) than female(4543) customers.

```
labels=c.index
sizes=c.values

plt.pie(sizes,labels=labels,autopct='%1.1f%%',
        shadow=True)
plt.axis('equal')
```

```
(-1.1107416681807245,
 1.1005115080086059,
 -1.1258189261495668,
 1.1119466189728702)
```

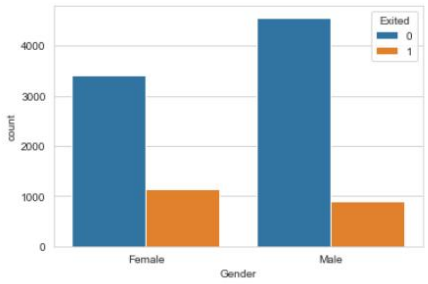


Observation:

Out of 100 % ,54.6% customers are males and 45.4% customers are females.

```
sns.countplot(data=df_catagorical, x='Gender', hue='Exited')
```

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



Observation:

Male customers are more loyal than female customers.

4) For Tenure

```
import warnings
warnings.filterwarnings("ignore")
d=df_catagorical['Tenure'].value_counts()
d
```

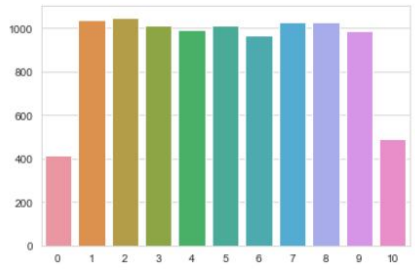
```
2     1048
1     1035
7     1028
8     1025
5     1012
3     1009
4      989
9      984
6      967
10     490
0      413
Name: Tenure, dtype: int64
```

Observation:

Here we can see that there are 11 different tenures from 0 to 10 years.

```
sns.barplot(x=d.index,y=d.values)
```

```
<AxesSubplot:>
```



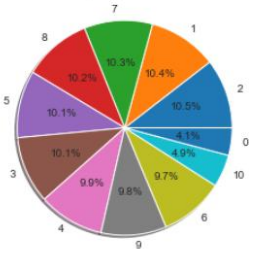
Observation:

From the graph we can see very few customers have tenure 0 years and 10 years.

Almost equal number of customers have tenure ranging from 1 to 9 years and highest tenure is 2 year.

```
labels=d.index
sizes=d.values

plt.pie(sizes,labels=labels,autopct='%1.1f%%',
        shadow=True)
plt.axis('equal')
```

```
(-1.1058178224184918,
 1.1002770391627854,
 -1.1056794625804278,
 1.1052094675315791)
```



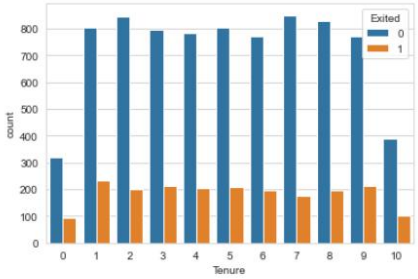
Observation:

Only 4.9% customers have tenure 10 years and only 4.1% customers have tenure 0 years.

Two year tenure have highest 10.5% customers.

```
sns.countplot(data=df_catagorical, x='Tenure', hue='Exited')
```

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



Observation:

Tenure feature is not helpful in understanding data.

Mean, Median, Mode, Percentile: -

Mean: -

The mean (or average) of observations is the sum of the values of all the observations divided by the total number of observations.

The mean of the data is given by $\mathbf{x = f_1x_1 + f_2x_2 + \ldots\ldots + f_nx_n/f1 + f2+\ldots\ldots\ldots + fn}$

Mean Formula is given by

$$\mathbf{Mean= \sum(f_i.x_i)/\sum f_i}$$

Median: -

We first arrange the given data values of the observations in ascending order. Then, if n is odd, the median is the (n+1/2). And if n is even, then the median will be the average of the n/2th and the (n/2 +1)th observation.

Formula for Calculating Median:

$$\mathbf{Median,\ M_e = l + \{h\ x\ (N/2 – cf\ )/f\}}$$

were,
l = lower limit of median class.

Mode: -

It is that value of a variate that occurs most often. More precisely, the mode is that value of the variable at which the concentration of the data is maximum.

Modal Class**:** In a frequency distribution, the class having maximum frequency is called the modal class.

Formula for Calculating Mode:

$$\mathbf{M_o = x_k + h\{(f_k – f_{k-1})/(2f_k - f_{k-1}-f_{k+1})\}}$$

were,

x_k = lower limit of the modal class interval.
f_k = frequency of the modal class.
f_{k-1}= frequency of the class preceding the modal class.
f_{k+1} = frequency of the class succeeding the modal class.
h=width of the class interval.

Percentile: -

Percentile formula is used in determining the performance of a person in comparison with others. It is used mostly in schools during results of tests to check where a person stands out from others. The percentile formula for a score 'x' can be defined as number of scores that fall under 'x' divided by total number of values in the given population.

Formula for Calculating Percentile:

Percentile(x) = (Number of values fall under 'x'/total number of values) × 100

$$P = (n/N) \times 100$$

Where,

P is percentile

n – Number of values below 'x'

N – Total count of population

Standard Deviation: -

The Standard Deviation is a measure of how spread-out numbers are.
Its symbol is **σ** (the Greek letter sigma)
The formula is easy: it is the **square root** of the **Variance.** So now you ask, "What is the Variance?"

Variance: -

The Variance is defined as, The average of the **squared** differences from the Mean.

To calculate the variance, follow these steps:

- Work out the Mean (the simple average of the numbers)
- Then for each number: subtract the Mean and square the result (the squared difference).
- Then work out the average of those squared differences.

The Standard Deviation is a measure of **how spread-out numbers are**.

This is the formula for Standard Deviation:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

In the formula above **μ** (the Greek letter "mu") is the mean of all our values, then for each number (Xi) & subtract the Mean(**μ**) and square the result. To work out the mean, **add up all the values** then **divide by how many.**

Coading:-

Mean, Median, Percentile

```
stayed=df[df['Exited']==0]
#pick those rows which have exited column=0
left=df[df['Exited']==1]
#these are people who have churned

stayed
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
6	7	15592531	Bartlett	822	France	Male	50	7	0.00	2	1	1	10062.80	0
8	9	15792365	He	501	France	Male	44	4	142051.07	2	0	1	74940.50	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9993	9994	15569266	Rahman	644	France	Male	28	7	155060.41	1	1	0	29179.52	0
9994	9995	15719294	Wood	800	France	Female	29	2	0.00	2	0	0	167773.55	0
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

7963 rows × 14 columns

```
left
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
5	6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	0	149756.71	1
7	8	15656148	Obinna	376	Germany	Female	29	4	115046.74	4	1	0	119346.88	1
16	17	15737452	Romeo	653	Germany	Male	58	1	132602.88	1	1	0	5097.67	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9981	9982	15672754	Burbidge	498	Germany	Male	42	3	152039.70	1	1	1	53445.17	1
9982	9983	15768163	Griffin	655	Germany	Female	46	7	137145.12	1	1	0	115146.40	1
9991	9992	15769959	Ajuluchukwu	597	France	Female	53	4	88381.21	1	1	0	69384.71	1
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1

2037 rows × 14 columns

Here we have separated data of customers who have left from data of customers who have stayed. Now we will compare mean, median, standard deviation of stayed with that of left using different features.

1. Credit Score

```
# Mean, Mdian, Std-deviation
print("Mean for CreditScore:")
print(np.mean(stayed['CreditScore']), "for stayed")
print(np.mean(left['CreditScore']), "for left" )

print("\nMedian for CreditScore:")
print(np.median(stayed['CreditScore']), "for stayed")
print(np.median(left['CreditScore']), "for left")

print("\nStd-dev for CreditScore:")
print(np.std(stayed['CreditScore']), "for stayed")
print(np.std(left['CreditScore']), "for left")
```

```
Mean for CreditScore:
651.8531960316463 for stayed
645.3514972999509 for left

Median for CreditScore:
653.0 for stayed
646.0 for left

Std-dev for CreditScore:
95.64783071535247 for stayed
100.29687481012597 for left
```

2. Age

```
# Mean, Mdian, Std-deviation
print("Mean for Age:")
print(np.mean(stayed['Age']), "for stayed")
print(np.mean(left['Age']), "for left" )

print("\nMedian for Age:")
print(np.median(stayed['Age']), "for stayed")
print(np.median(left['Age']), "for left")

print("\nStd-dev for Age:")
print(np.std(stayed['Age']), "for stayed")
print(np.std(left['Age']), "for left")
```

```
Mean for Age:
37.40838879819164 for stayed
44.8379970544919 for left

Median for Age:
36.0 for stayed
45.0 for left

Std-dev for Age:
10.124727115441777 for stayed
9.759165198147958 for left
```

3. Balance

```
# Mean, Mdian, Std-deviation
print("Mean for Balance:")
print(np.mean(stayed['Balance']), "for stayed")
print(np.mean(left['Balance']), "for left" )

print("\nMedian for Balance:")
print(np.median(stayed['Balance']), "for stayed")
print(np.median(left['Balance']), "for left")

print("\nStd-dev for Balance:")
print(np.std(stayed['Balance']), "for stayed")
print(np.std(left['Balance']), "for left")
```

```
Mean for Balance:
72745.29677885193 for stayed
91108.53933726063 for left

Median for Balance:
92072.68 for stayed
109349.29 for left

Std-dev for Balance:
62844.094322747915 for stayed
58346.467874478956 for left
```

Observation:

Here we can see that there is noticeable difference between mean and median of Balance. It shows that there are some outliers.

4. Estimated Salary

```
# Mean, Mdian, Std-deviation
print("Mean for EstimatedSalary:")
print(np.mean(stayed['EstimatedSalary']), "for stayed")
print(np.mean(left['EstimatedSalary']), "for left" )

print("\nMedian for EstimatedSalary:")
print(np.median(stayed['EstimatedSalary']), "for stayed")
print(np.median(left['EstimatedSalary']), "for left")

print("\nStd-dev for EstimatedSalary:")
print(np.std(stayed['EstimatedSalary']), "for stayed")
print(np.std(left['EstimatedSalary']), "for left")
```

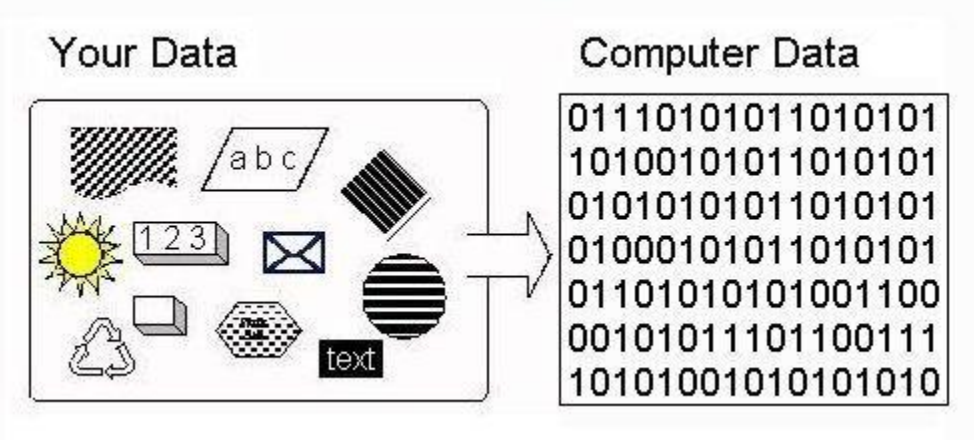
```
Mean for EstimatedSalary:
99738.39177194514 for stayed
101465.6775306824 for left

Median for EstimatedSalary:
99645.04 for stayed
102460.84 for left

Std-dev for EstimatedSalary:
57401.982332801934 for stayed
57898.201201044954 for left
```

Data Pre-processing:
One Hot Encoding for Categorical features

In one hot encoding we are converting the categorical data into numeric data to help understand the data properly to the machine. As a machine can only understand numbers and cannot understand the text in the first place, this essentially becomes the case with Deep Learning & Machine Learning algorithms.

What is Categorical Encoding?

Typically, any structured dataset includes multiple columns – a combination of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text. That's essentially the case with Machine Learning algorithms too.



That's primarily the reason we need to convert categorical columns to numerical columns so that a machine learning algorithm understands it. This process is called **categorical encoding**.

Categorical encoding is a process of converting categories to numbers.

In the next section, I will touch upon different ways of handling categorical variables.

Different Approaches to Categorical Encoding

So, how should we handle categorical variables? As it turns out, there are multiple ways of handling Categorical variables. In this article, I will discuss the two most widely used techniques:

- Label Encoding
- One-Hot Encoding

Now, let us see them in detail.

Label Encoding

Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

One-Hot Encoding

One-Hot Encoding is another popular technique for treating categorical variables. It simply creates additional features based on the number of unique values in the categorical feature. Every unique value in the category will be added as a feature. One-Hot Encoding is the process of creating dummy variables.

We apply One-Hot Encoding when:

- The categorical feature is not ordinal (like the countries above)
- The number of categorical features is less so one-hot encoding can be effectively applied
- We apply Label Encoding when: The categorical feature is ordinal (like Jr. kg, Sr. kg, Primary school, high school)
- The number of categories is quite large as one-hot encoding can lead to high memory consumption.

Coding: -

```
catagorical= ['Geography','Gender']
df_catagorical= pd.get_dummies(df_catagorical, columns=catagorical, drop_first=False)
df_catagorical
```

	NumOfProducts	HasCrCard	IsActiveMember	Tenure	Exited	Geography_France	Geography_Germany	Geography_Spain	Gender_Female	Gender_Male
0	1	1	1	2	1	1	0	0	1	0
1	1	0	1	1	0	0	0	1	1	0
2	3	1	0	8	1	1	0	0	1	0
3	2	0	0	1	0	1	0	0	1	0
4	1	1	1	2	0	0	0	1	1	0
...	...	...	...	...	...	...	...	...	...	...
9995	2	1	0	5	0	1	0	0	0	1
9996	1	1	1	10	0	1	0	0	0	1
9997	1	0	1	7	1	1	0	0	1	0
9998	2	1	0	3	1	0	1	0	0	1
9999	1	1	0	4	0	1	0	0	1	0

10000 rows × 10 columns

Observation:

By one hot encoding our number of features have increased and that will make our data of too many dimensions.

And to minimise the number of columns(features) we Adjusted the data in a way that , where Male and female were 2 columns we diducted into 1 (Gender_male) and 0 (Gender_Female). which tales us the customer is male or not that will ultimately tell you that if customer is not male then she's female.

Scaling for Numerical Features:

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range.

Here we have used MinMaxScaler(Data Normalisation) for preprocessing of numerical features to bring all featues in comparable

range.

```
from sklearn.preprocessing import MinMaxScaler
scaling=MinMaxScaler()
cs=scaling.fit_transform(df_numerical[['CreditScore']])
Age=scaling.fit_transform(df_numerical[['Age']])
Balance=scaling.fit_transform(df_numerical[['Balance']])
EstimatedSalary=scaling.fit_transform(df_numerical[['EstimatedSalary']])

df_numerical['CreditScore']= cs

df_numerical['Age']= Age

df_numerical['Balance']= Balance

df_numerical['EstimatedSalary']= EstimatedSalary
```

```
df_numerical
```

	CreditScore	Age	Balance	EstimatedSalary	Exited
0	0.538	0.324324	0.000000	0.506735	1
1	0.516	0.310811	0.334031	0.562709	0
2	0.304	0.324324	0.636357	0.569654	1
3	0.698	0.283784	0.000000	0.469120	0
4	1.000	0.337838	0.500246	0.395400	0
...	...	...	...	...	...
9995	0.842	0.283784	0.000000	0.481341	0
9996	0.332	0.229730	0.228657	0.508490	0
9997	0.718	0.243243	0.000000	0.210390	1
9998	0.844	0.324324	0.299226	0.464429	1
9999	0.884	0.135135	0.518708	0.190914	0

10000 rows × 5 columns

As we have done preprocessing of both Numerical as well as Categorical features, we will now joined back all the features and class label for training of a model.

```
# Here we are dropping the Exited column, because Numerical data also contains Exited column.
df_catagorical
df_catagorical2= df_catagorical.drop("Exited",axis=1)
df_catagorical2
```

	NumOfProducts	HasCrCard	IsActiveMember	Tenure	Geography_France	Geography_Germany	Geography_Spain	Gender_Female	Gender_Male
0	1	1	1	2	1	0	0	1	0
1	1	0	1	1	0	0	1	1	0
2	3	1	0	8	1	0	0	1	0
3	2	0	0	1	1	0	0	1	0
4	1	1	1	2	0	0	1	1	0
...	...	...	...	...	...	...	...	...	...
9995	2	1	0	5	1	0	0	0	1
9996	1	1	1	10	1	0	0	0	1
9997	1	0	1	7	1	0	0	1	0
9998	2	1	0	3	0	1	0	0	1
9999	1	1	0	4	1	0	0	1	0

10000 rows × 9 columns

```
# Concatination of Numerical and categorical dataframes.
df_concat2 = pd.concat([df_catagorical2,df_numerical], axis=1)

df_concat2
```

	NumOfProducts	HasCrCard	IsActiveMember	Tenure	Geography_France	Geography_Germany	Geography_Spain	Gender_Female	Gender_Male	CreditScore	Age	Balar
0	1	1	1	2	1	0	0	1	0	0.538	0.324324	0.0000
1	1	0	1	1	0	0	1	1	0	0.516	0.310811	0.3340
2	3	1	0	8	1	0	0	1	0	0.304	0.324324	0.6363
3	2	0	0	1	1	0	0	1	0	0.698	0.283784	0.0000
4	1	1	1	2	0	0	1	1	0	1.000	0.337838	0.5002
...	...	...	...	...	...	...	...	...	...	...	...	
9995	2	1	0	5	1	0	0	0	1	0.842	0.283784	0.0000
9996	1	1	1	10	1	0	0	0	1	0.332	0.229730	0.2286
9997	1	0	1	7	1	0	0	1	0	0.718	0.243243	0.0000
9998	2	1	0	3	0	1	0	0	1	0.844	0.324324	0.2992
9999	1	1	0	4	1	0	0	1	0	0.884	0.135135	0.5187

10000 rows × 14 columns

Separation of features from Class label:

Here we are separating first 13 columns i.e. our features from 14th column i.e. our class label.

```
X=df_concat2.iloc[:,0:13]
X
```

	NumOfProducts	HasCrCard	IsActiveMember	Tenure	Geography_France	Geography_Germany	Geography_Spain	Gender_Female	Gender_Male	CreditScore	Age	Balar
0	1	1	1	2	1	0	0	1	0	0.538	0.324324	0.0000
1	1	0	1	1	0	0	1	1	0	0.516	0.310811	0.3340
2	3	1	0	8	1	0	0	1	0	0.304	0.324324	0.6363
3	2	0	0	1	1	0	0	1	0	0.698	0.283784	0.0000
4	1	1	1	2	0	0	1	1	0	1.000	0.337838	0.5002
...	...	...	...	...	...	...	...	...	...	...	...	
9995	2	1	0	5	1	0	0	0	1	0.842	0.283784	0.0000
9996	1	1	1	10	1	0	0	0	1	0.332	0.229730	0.2286
9997	1	0	1	7	1	0	0	1	0	0.718	0.243243	0.0000
9998	2	1	0	3	0	1	0	0	1	0.844	0.324324	0.2992
9999	1	1	0	4	1	0	0	1	0	0.884	0.135135	0.5187

10000 rows × 13 columns

```
Y=df_concat2.iloc[:,13:]
Y
```

	Exited
0	1
1	0
2	1
3	0
4	0
...	...
9995	0
9996	0
9997	1
9998	1
9999	0

10000 rows × 1 columns

```
# Train-Test Split
# Spliting the data into train and test
import sklearn
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test= train_test_split(X, Y, test_size=0.2, random_state=42)
```

```
# Train-Test Split
# Spliting the data into train and test
import sklearn
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test= train_test_split(X, Y, test_size=0.2, random_state=42)
```

```
X_train.shape
```

```
(8000, 13)
```

```
X_test.shape
```

```
(2000, 13)
```

```
Y_train.shape
```

```
(8000, 1)
```

```
Y_test.shape
```

```
(2000, 1)
```

Modelling:

A machine learning model is a file that has been trained to recognize certain types of patterns. You train a model over a set of data, providing it an algorithm that it can use to reason over and learn from those data. We are using machine learning to solve the customer churn problem. There are several ways to formulate the task.

1. KNN (K-NEAREST NEIGHBOUR)

KNN is nearest neighbour belongs to same class. We will apply KNN to our data and then we will check its performance.

When do we use KNN algorithm?

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique, we generally look at 3 important aspects:

1. Ease to interpret output
2. Calculation time
3. Predictive Power

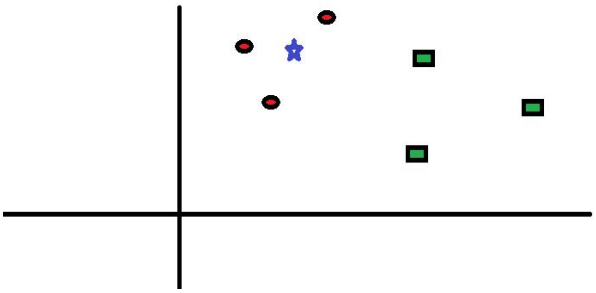
Let us take a few examples to place KNN in the scale:

	Logistic Regression	CART	Random Forest	KNN
1. Ease to interpret output	2	3	1	3
2. Calculation time	3	2	1	3
3. Predictive Power	2	2	3	2

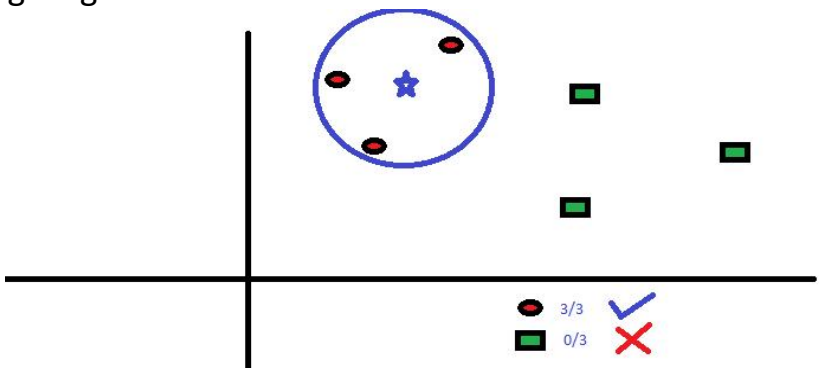
KNN algorithm fairs across all parameters of considerations. It is commonly used for its easy of interpretation and low calculation time.

How does the KNN algorithm work?

Let's take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS) :



You intend to find out the class of the blue star (BS). BS can either be RC or GS and nothing else. The “K” is KNN algorithm is the nearest neighbour we wish to take the vote from. Let’s say K = 3. Hence, we will now make a circle with BS as the centre just as big as to enclose only three datapoints on the plane. Refer to the following diagram for more details:



The three closest points to BS are all RC. Hence, with a good confidence level, we can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbour went to RC. The choice of the parameter K is very crucial in this algorithm. Next, we will understand what are the factors to be considered to conclude the best.

```
# Applying KNN

# KNN(k nearest neighbour)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# Create KNN classifier

hyper_parameter_k=[{'n_neighbors':[3,5,7,9]}]
clf= KNeighborsClassifier()
modelknn=GridSearchCV( clf,hyper_parameter_k,scoring='accuracy')

modelknn.fit(X_train,Y_train)

print(modelknn.best_estimator_)
# Fit the classifier to the data
```

```
KNeighborsClassifier(n_neighbors=7)
```

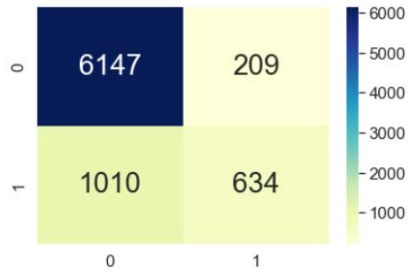
```
#Predictions
knn_train_predictions=modelknn.predict(X_train) #Y_pred_train
knn_test_prediction=modelknn.predict(X_test)    #Y_pred_test
```

```
# Evaluation:


# Confusion matrix:

#Training confusion matrix for KNN

from sklearn.metrics import confusion_matrix
x_knn_cf_train=confusion_matrix(Y_train,knn_train_predictions)
print(x_knn_cf_train)
```

```
[[6147  209]
 [1010  634]]
```

```
print("Train confusion matrix")
sns.set(font_scale=1.4)#for label size
sns.heatmap(x_knn_cf_train, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

```
Train confusion matrix
<AxesSubplot:>
```

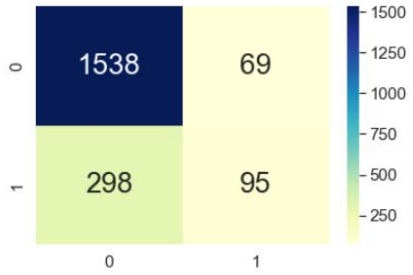



```
# Testing confusion matrix for KNN

from sklearn.metrics import confusion_matrix
x_knn_cf=confusion_matrix(Y_test,knn_test_prediction)
print(x_knn_cf)
```

```
[[1538   69]
 [ 298   95]]
```

```
print("Test confusion matrix")
sns.set(font_scale=1.4)#for label size
sns.heatmap(x_knn_cf, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

```
Test confusion matrix
<AxesSubplot:>
```




```
# Precison for KNN

# Training data

from sklearn.metrics import precision_score
x_knn_p=precision_score(Y_train,knn_train_predictions)
print("The precision of x_knn_p for training data",x_knn_p)

# Testing data

from sklearn.metrics import precision_score
x_knn_p_test=precision_score(Y_test,knn_test_prediction)
print("The precision of x_knn_p for testing data",x_knn_p_test)
```

```
The precision of x_knn_p for training data 0.7520759193357058
The precision of x_knn_p for testing data 0.5792682926829268
```

```
#RECALL for KNN

#Training data

from sklearn.metrics import recall_score
x_knn_r_train=recall_score(Y_train,knn_train_predictions)
print("The recall of training data for knn is",x_knn_r_train)

#Test data

from sklearn.metrics import recall_score
x_knn_r_test=recall_score(Y_test,knn_test_prediction)
print("The recall of test data for knn is",x_knn_r_test)
```

```
The recall of training data for knn is 0.38564476885644766
The recall of test data for knn is 0.24173027989821882
```

```
# F1 Score for KNN

from sklearn.metrics import f1_score

X_knntrain_F1score=f1_score(Y_train,knn_train_predictions)
print("Training f1 score is",X_knntrain_F1score) #f1 score on train data

X_knntest_F1score=f1_score(Y_test,knn_test_prediction)
print("Testing f1 score is",X_knntest_F1score) #f1 score on test data
```

```
Training f1 score is 0.5098512263771612
Testing f1 score is 0.34111310592459604
```

```
# Accuracy for KNN

from sklearn.metrics import accuracy_score
a=accuracy_score(Y_train,knn_train_predictions)

print("training accuracy is",a)

from sklearn.metrics import accuracy_score
b=accuracy_score(Y_test,knn_test_prediction)

print("testing accuracy is",b)
```

```
training accuracy is 0.847625
testing accuracy is 0.8165
```

Logistic Regression is used to solve classification problems. **Models are trained on historical labelled datasets** and aim to predict which category new observations will belong to.

2. Logistic Regression

Logistic Regression is a Classification model. It helps to make predictions where the output variable is categorical.

```
# We are using simple GridSearchCV to tune the hyperparameter C.
# Here we are not using smote samples.

tuned_parameters=[{'C':[10**-4,10**-2,10**0,10**2,10**4]}]

LRmodel=GridSearchCV(LogisticRegression(max_iter=400,class_weight='balanced'),tuned_parameters)

LRmodel.fit(X_train,Y_train)

print(LRmodel.best_estimator_)
print("Training Accuracy:",LRmodel.score(X_train,Y_train))
print("Testing Accuracy:",LRmodel.score(X_test,Y_test))

#this is accuracy
```

```
LogisticRegression(C=100, class_weight='balanced', max_iter=400)
Training Accuracy: 0.70775
Testing Accuracy: 0.7195
```

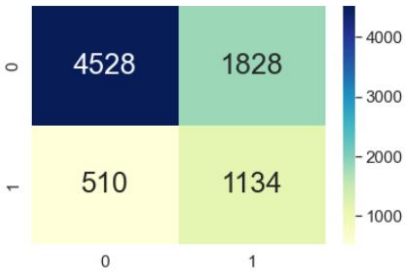
```
# Prediction
LR_predictions_train=LRmodel.predict(X_train)
LR_predictions_test=LRmodel.predict(X_test)
```

```
# Evaluation

# Confusion Matrix

# Training Confusion Matrix for Logistic Regression

x_LRmodel_train=confusion_matrix(Y_train,LR_predictions_train)
print(x_LRmodel_train)
```

```
[[4528 1828]
 [ 510 1134]]
```

```
print("Train confusion matrix for Logistic Regression")
sns.set(font_scale=1.4)#for label size
sns.heatmap(x_LRmodel_train, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

```
Train confusion matrix for Logistic Regression
<AxesSubplot:>
```

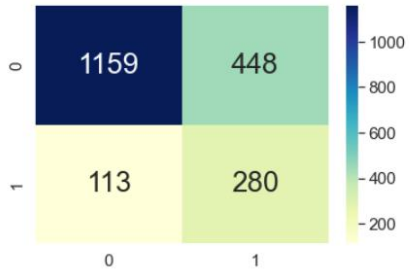



```
# Testing Confusion Matrix for Logistic Regression

x_LRmodel_test=confusion_matrix(Y_test,LR_predictions_test)
print(x_LRmodel_test)
```

```
[[1159  448]
 [ 113  280]]
```

```
print("Test confusion matrix for Logistic Regression")
sns.set(font_scale=1.4)#for label size
sns.heatmap(x_LRmodel_test, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

```
Test confusion matrix for Logistic Regression
<AxesSubplot:>
```




```
# Precison for Logistic Regression

# Training data

x_LRmodel_p_train=precision_score(Y_train,LR_predictions_train)
print("The precision of x_LRmodel_p_train for training data",x_LRmodel_p_train)

# Testing data

x_LRmodel_p_test=precision_score(Y_test,LR_predictions_test)
print("The precision of x_LRmodel_p_test for testing data",x_LRmodel_p_test)
```

```
The precision of x_LRmodel_p_train for training data 0.3828494260634706
The precision of x_LRmodel_p_test for testing data 0.38461538461538464
```

```
#RECALL for Logistic Regression

#Training data

x_LRmodel_r_train=recall_score(Y_train,LR_predictions_train)
print("The recall of training data for Logistic Regression is",x_LRmodel_r_train)

#Test data

x_LRmodel_r_test=recall_score(Y_test,LR_predictions_test)
print("The recall of test data for Logistic Regression is",x_LRmodel_r_test)
```

```
The recall of training data for Logistic Regression is 0.6897810218978102
The recall of test data for Logistic Regression is 0.712468193384224
```

```
# F1 Score for Logistic Regression

X_LRmodel_F1score_train=f1_score(Y_train,LR_predictions_train)
print("Training f1 score for Logistic Regression is",X_LRmodel_F1score_train) #f1 score on train data

X_LRmodel_F1score_test=f1_score(Y_test,LR_predictions_test)
print("Testing f1 score for Logistic Regression is",X_LRmodel_F1score_test) #f1 score on test data
```

```
Training f1 score for Logistic Regression is 0.49240121580547114
Testing f1 score for Logistic Regression is 0.49955396966993754
```

3. SVM(SUPPORT VECTOR MACHINE

Support Vector Machine" (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.To finding best fit line .

```
#svm(support vector machine)
from sklearn import svm
tuned_parameters=[{'C':[0.001, 0.01, 0.1, 1, 10]}]
clf= svm.SVC(kernel='rbf')
SVMmodel=GridSearchCV(clf,tuned_parameters,cv=3)

SVMmodel.fit(X_train,Y_train)

print(SVMmodel.best_estimator_)
print("Training Accuracy using SVM:",SVMmodel.score(X_train,Y_train))
print("Testing Accuracy using SVM:",SVMmodel.score(X_test,Y_test))
# Fit the classifier to the data
```

```
SVC(C=10)
Training Accuracy using SVM: 0.826125
Testing Accuracy using SVM: 0.83
```

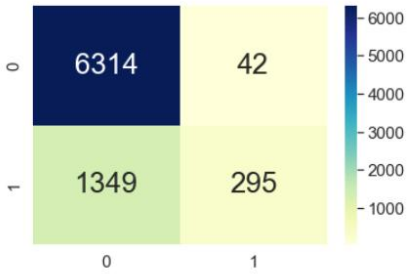
```
# Prediction
SVM_predictions_train=SVMmodel.predict(X_train)
SVM_predictions_test=SVMmodel.predict(X_test)
```

```
# Evaluation

# Confusion Matrix

# Training Confusion Matrix for SVM

x_SVM_train=confusion_matrix(Y_train,SVM_predictions_train)
print(x_SVM_train)
```

```
[[6314   42]
 [1349  295]]
```

```
print("Train confusion matrix for SVM")
sns.set(font_scale=1.4)#for label size
sns.heatmap(x_SVM_train, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

```
Train confusion matrix for SVM
<AxesSubplot:>
```

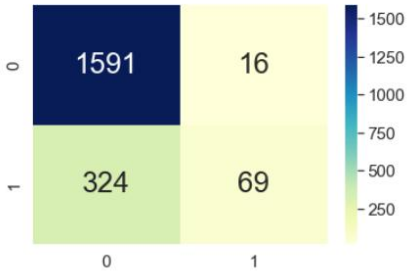



```
# Testing Confusion Matrix for Logistic Regression

x_SVM_test=confusion_matrix(Y_test,SVM_predictions_test)
print(x_SVM_test)
```

```
[[1591   16]
 [ 324   69]]
```

```
print("Test confusion matrix for SVM")
sns.set(font_scale=1.4)#for label size
sns.heatmap(x_SVM_test, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

```
Test confusion matrix for SVM
<AxesSubplot:>
```



```
# Precison for SVM

# Training data

x_SVM_p_train=precision_score(Y_train,SVM_predictions_train)
print("The precision of x_SVM_p_train for training data",x_SVM_p_train)

# Testing data

x_SVM_p_test=precision_score(Y_test,SVM_predictions_test)
print("The precision of x_SVM_p_test for testing data",x_SVM_p_test)
```

```
The precision of x_SVM_p_train for training data 0.8753709198813057
The precision of x_SVM_p_test for testing data 0.8117647058823529
```

```
#RECALL for SVM

#Training data

x_SVM_r_train=recall_score(Y_train,SVM_predictions_train)
print("The recall of training data for SVM is",x_SVM_r_train)

#Test data

x_SVM_r_test=recall_score(Y_test,SVM_predictions_test)
print("The recall of test data for SVM is",x_SVM_r_test)
```

```
The recall of training data for SVM is 0.1794403892944039
The recall of test data for SVM is 0.17557251908396945
```

```
# F1 Score for Logistic Regression

X_SVM_F1score_train=f1_score(Y_train,SVM_predictions_train)
print("Training f1 score for SVM is",X_SVM_F1score_train) #f1 score on train data

X_SVM_F1score_test=f1_score(Y_test,SVM_predictions_test)
print("Testing f1 score for SVM is",X_SVM_F1score_test) #f1 score on test data
```

```
Training f1 score for SVM is 0.2978293791014639
Testing f1 score for SVM is 0.28870292887029286
```

4. Decision Tree

A Decision Tree is a supervised learning algorithm. It is a graphical representation of all the possible solutions.

All the decisions were made based on some conditions.

```
from sklearn import tree
tuned_parameters=[{'max_depth':[4,5,6]}]

clf = tree.DecisionTreeClassifier(min_samples_split=5,random_state=42)


DTmodel=GridSearchCV(clf,tuned_parameters)

DTmodel.fit(X_train,Y_train)

print(DTmodel.best_estimator_)
print("Training Accuracy using Decision Tree:",DTmodel.score(X_train,Y_train))
print("Training Accuracy using Decision Tree:",DTmodel.score(X_test,Y_test))
```

```
DecisionTreeClassifier(max_depth=6, min_samples_split=5, random_state=42)
Training Accuracy using Decision Tree: 0.865875
Training Accuracy using Decision Tree: 0.8595
```

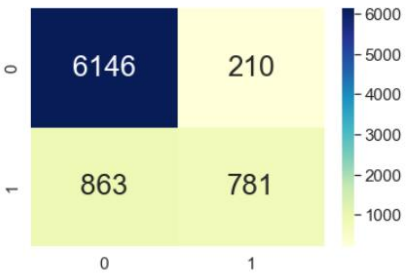
```
# Prediction

DT_predictions_train=DTmodel.predict(X_train)
DT_predictions_test=DTmodel.predict(X_test)
```

```
# Evaluation

# Confusion Matrix

# Training Confusion Matrix for Decision Tree

x_DT_train=confusion_matrix(Y_train,DT_predictions_train)
print(x_DT_train)
```

```
[[6146  210]
 [ 863  781]]
```

```
print("Train confusion matrix for Decision Tree")
sns.set(font_scale=1.4)#for label size
sns.heatmap(x_DT_train, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

```
Train confusion matrix for Decision Tree
<AxesSubplot:>
```

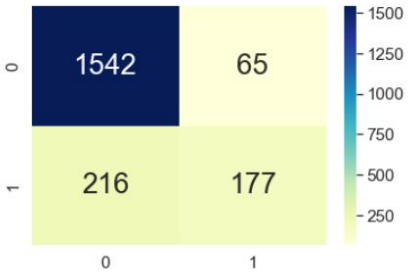



```
# Testing Confusion Matrix for Decision Tree

x_DT_test=confusion_matrix(Y_test,DT_predictions_test)
print(x_DT_test)
```

```
[[1542   65]
 [ 216  177]]
```

```
print("Test confusion matrix for Decision Tree")
sns.set(font_scale=1.4)#for label size
sns.heatmap(x_DT_test, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

```
Test confusion matrix for Decision Tree
<AxesSubplot:>
```




```
# Precison for Decision tree

# Training data

x_DT_p_train=precision_score(Y_train,DT_predictions_train)
print("The precision of x_DT_p_train for training data",x_DT_p_train)

# Testing data

x_DT_p_test=precision_score(Y_test,DT_predictions_test)
print("The precision of x_DT_p_test for testing data",x_DT_p_test)
```

```
The precision of x_DT_p_train for training data 0.7880928355196771
The precision of x_DT_p_test for testing data 0.731404958677686
```

```
#RECALL for Decision Tree

#Training data

x_DT_r_train=recall_score(Y_train,DT_predictions_train)
print("The recall of training data for Decision Tree is",x_DT_r_train)

#Test data

x_DT_r_test=recall_score(Y_test,DT_predictions_test)
print("The recall of test data for Decision Tree is",x_DT_r_test)
```

```
The recall of training data for Decision Tree is 0.4750608272506083
The recall of test data for Decision Tree is 0.45038167938931295
```

```
# F1 Score for Decision Tree

X_DT_F1score_train=f1_score(Y_train,DT_predictions_train)
print("Training f1 score for Decision Tree is",X_DT_F1score_train) #f1 score on train data

X_DT_F1score_test=f1_score(Y_test,DT_predictions_test)
print("Testing f1 score for Decision Tree is",X_DT_F1score_test) #f1 score on test data
```

```
Training f1 score for Decision Tree is 0.5927893738140418
Testing f1 score for Decision Tree is 0.5574803149606299
```

5. Random Forest

Random Forest is an ensemble algorithm that follows the bagging approach. The Decision Tree is the base estimator for a Random

Forest. As the name suggests, a forest is a group of many trees, and a random forest is a set of various Decision Trees.

Random Forest selects a feature set randomly to choose the best split at each decision tree node.

```
#Training random forest after applying SMOTE technique.
#here we are using GridSearchCV to tune the hyperparamters we have in randomforest to see which
# parameters works best.
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [200,300, 500],
    'max_depth' : [4,5,6]
}
rfc=RandomForestClassifier(random_state=42)

grid_search = GridSearchCV(estimator = rfc, param_grid = param_grid,
                           cv = 3, n_jobs = -1)
```

```
grid_search.fit(X_train, Y_train)

print(grid_search.best_params_)
```

```
{'max_depth': 6, 'n_estimators': 500}
```

```
#training the random forest model
rfc_new=RandomForestClassifier(criterion= 'entropy', max_depth= 6, max_features= 'auto', n_estimators=500,random_state=42)
random_forest_model=rfc_new.fit(X_train,Y_train)
```

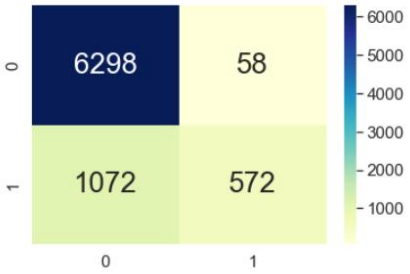
```
#predictions
RF_predictions_train=random_forest_model.predict(X_train)
RF_predictions_test=random_forest_model.predict(X_test)
```

```
# Evaluation

# Confusion Matrix

# Training Confusion Matrix for Random Forest

x_RF_train=confusion_matrix(Y_train,RF_predictions_train)
print(x_RF_train)
```

```
[[6298   58]
 [1072  572]]
```

```
print("Train confusion matrix for Random Forest")
sns.set(font_scale=1.4)#for label size
sns.heatmap(x_RF_train, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

```
Train confusion matrix for Random Forest
<AxesSubplot:>
```

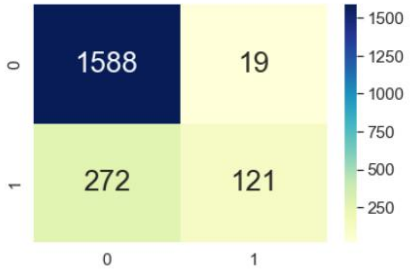


```
# Testing Confusion Matrix for Random Forest

x_RF_test=confusion_matrix(Y_test,RF_predictions_test)
print(x_RF_test)
```

```
[[1588   19]
 [ 272  121]]
```

```
print("Test confusion matrix for Random Forest")
sns.set(font_scale=1.4)#for label size
sns.heatmap(x_RF_test, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

```
Test confusion matrix for Random Forest
<AxesSubplot:>
```




```
# Precison for Random Forest

# Training data

x_RF_p_train=precision_score(Y_train,RF_predictions_train)
print("The precision of x_RF_p_train for training data",x_RF_p_train)

# Testing data

x_RF_p_test=precision_score(Y_test,RF_predictions_test)
print("The precision of x_RF_p_test for testing data",x_RF_p_test)
```

```
The precision of x_RF_p_train for training data 0.9079365079365079
The precision of x_RF_p_test for testing data 0.8642857142857143
```

```
#RECALL for Random Forest

#Training data

x_RF_r_train=recall_score(Y_train,RF_predictions_train)
print("The recall of training data for Random Forest is",x_RF_r_train)

#Test data

x_RF_r_test=recall_score(Y_test,RF_predictions_test)
print("The recall of test data for Random Forest is",x_RF_r_test)
```

```
The recall of training data for Random Forest is 0.34793187347931875
The recall of test data for Random Forest is 0.30788804071246817
```

```
# F1 Score for Random Forest

X_RF_F1score_train=f1_score(Y_train,RF_predictions_train)
print("Training f1 score for Random Forest is",X_RF_F1score_train) #f1 score on train data

X_RF_F1score_test=f1_score(Y_test,RF_predictions_test)
print("Testing f1 score for Random Forest is",X_RF_F1score_test) #f1 score on test data
```

```
Training f1 score for Random Forest is 0.5030782761653474
Testing f1 score for Random Forest is 0.45403377110694176
```

```
# Accuracy for Random Forest

a=accuracy_score(Y_train,RF_predictions_train)

print("training accuracy is",a)

b=accuracy_score(Y_test,RF_predictions_test)

print("testing accuracy is",b)
```

```
training accuracy is 0.85875
testing accuracy is 0.8545
```

```
pip install prettytable
```

```
Requirement already satisfied: prettytable in c:\users\abasaheb\anaconda3\lib\site-packages (3.4.1)
Requirement already satisfied: wcwidth in c:\users\abasaheb\anaconda3\lib\site-packages (from prettytable) (0.2.5)
Note: you may need to restart the kernel to use updated packages.
```

```
from prettytable import PrettyTable
columns=(["ALGORITHM", "ACCURACY","PRECISION","RECALL","F1"])
myTable = PrettyTable()
myTable.add_column(columns[0], ["KNN","Logistic Regression","SVM", "DECISION TREE" ,"Random Forest"])
myTable.add_column(columns[1], ["81.65%","72.00%","83.00%","86.00%","85.45%"])
myTable.add_column(columns[2], ["58.00%","38.46%","81.17%","73.14%","86.42%"])
myTable.add_column(columns[3], ["24.17%","71.20%","17.55%","45.03%","30.78%"])
myTable.add_column(columns[4], ["34.11%","50.00%","28.87%","55.75%","45.40%"])

print(myTable)
```

```
+---------------------+----------+-----------+--------+--------+
|      ALGORITHM      | ACCURACY | PRECISION | RECALL |   F1   |
+---------------------+----------+-----------+--------+--------+
|         KNN         |  81.65%  |   58.00%  | 24.17% | 34.11% |
| Logistic Regression |  72.00%  |   38.46%  | 71.20% | 50.00% |
|         SVM         |  83.00%  |   81.17%  | 17.55% | 28.87% |
|    DECISION TREE    |  86.00%  |   73.14%  | 45.03% | 55.75% |
|    Random Forest    |  85.45%  |   86.42%  | 30.78% | 45.40% |
+---------------------+----------+-----------+--------+--------+
```

Conclusion of the Project:

1. we have walked through a complete end-to-end machine learning project using the bank customer Churn dataset. We can see here this is a binary classification problem.

2. We started by cleaning the data and analysing it with visualization.

3. Then, to be able to build a machine learning model, we transformed the categorical data into numeric variables (feature engineering).

4. We have created here an ML model which predicts whether people are staying or exiting the bank.

5. We use 13 different features here.

6. After transforming the data, we tried five different machine learning algorithms (knn, logistic regression, svm, decision tree, and random forest) using default parameters.

7. The given problem is a classification problem so we have used here accuracy, precision, recall, f1 score and confusion matrix these performance metrics.

8. Finally, we tuned the hyperparameters of the **RANDOM FOREST** is giving the best **accuracy and F1 score.**

9. Also, **training accuracy and testing accuracy of RANDOM FOREST is close and high. So, we can use this algorithm for the production**.

PROJECT -II

LONG URL- SHORT URL

Business Problem:

In this Project, we are considering real world problem of URL. Generally, URLs are very long to read and write also. It takes lot of time to type a complete URL during heavy workload.

Solution:

So, I have come up with the solution by assigning Short URL to original Long URL. Here we will be using Procedural way of implementation of Short URL system.

Coading:-

```
#Class: group all variables/attributes and functions/methods into a single logical unit
import random
import string
from tkinter import *
import random
import string


root=tk.Tk()
root.geometry('500x500')
root.title('ShortURL.com')
root.configure(background='#02345c')
global selected
selected=False
v=tk.StringVar()
d=dict()

class ShortURL:

    def __init__(self): # constructor; not must, but, good to have;  initialize all attributes here
        self.d=dict()
        self.e=list()


    # given a long URL, get a short URL
    def getShortURL(self, longURL): # first argument to all methods is "self" => this object
        # length = random value in 6-10
        l = random.randint(6,10);


        # generate random characters into a string of length l
        chars = string.ascii_lowercase
        shortURL = ''.join(random.choice(chars) for i in range(l))


        # check if this string is already present in dict d
        if shortURL in self.d:
            return getShortURL(longURL);
        else:
            self.d[shortURL] = longURL;


        s = "https://www.shortURL.com/"+shortURL
        return s;
```

```
    def getLongURL(self, shortURL):

       # print(self.d); # print statemnt for debugging

        # extarct key from URL https://www.shortURL.com/mxzmuis ---> mxzmuis
        k = shortURL[25:];


        if k in self.d:
            return self.d[k];
        else:
            return None;
     #Class: datatype/DS & Object: variable of a class
s = ShortURL() # constructor being called; memory allocated.

print(s.getShortURL("https://github.com/Nishesh2115/Stock-Prediction-Project"))

my_label=tk.Label(root, text='Enter URL to Shorten', font=('arial',16))
my_label.configure(background='#03254c', foreground='white')
my_label.pack(pady=20)

my_entry=tk.Entry(root, font=('arial',16))
my_entry.configure(width=35)
my_entry.pack(pady=20)

my_button=tk.Button(root, text='Make ShortURL', command=lambda: getShortURL(my_entry), font=('arial',16))
my_button.configure(background='black', foreground='white')
my_button.pack(pady=20)

ShortURL_label=tk.Label(root, text='Here is your ShortURL', font=('arial',16))
ShortURL_label.configure(background='#03254c', foreground='white', width=35)
ShortURL_label.pack(pady=30)

ShortURL=tk.Label(root, width=35, height=1, font=('arial',16))
ShortURL.pack(pady=5)


ShortURL_label=tk.Label(root,text='Just copy and paste in Browser', font=('arial',12))
ShortURL_label.configure(background='#03254c', foreground='white', width=35)
ShortURL_label.pack(pady=5)


root.mainloop()
```

https://www.shortURL.com/zdoabtrfxx

PROJECT -III

SIMPLE INTEREST CALCULATOR

Business Problem

Requirement of simple to use and quick interest calculator.

Solution: - Calculator by using Tkinter

We are creating simple Interest calculator using Tkinter library and procedural way of programming.

Coding: -

```
from tkinter import *

root=Tk()
root.title('Simple interest Calculator')
root.geometry('570x600+100+200')
root.resizable(0,0)
root.configure(background='#17161b')


def calculate():
    prin=int(principalentry.get())
    rat=int(rateentry.get())
    tim=int(timeentry.get())
    amount=(prin*rat*tim)/100
    Label(text= f"{amount}", font="arial 30 bold").place(x=200,y=220)

principal=Label(root,text="Principal:",font="arial 15")
rate=Label(root,text="Rate of Interest:",font="arial 15")
time=Label(root,text="Time:",font="arial 15")


principal.place(x=50,y=20)
rate.place(x=50,y=90)
time.place(x=50,y=160)

interest=Label(root,text="Interest:",font="arial 15")
interest.place(x=50,y=230)

principalvalue= StringVar()
ratevalue= StringVar()
timevalue= StringVar()

principalentry=Entry(root,textvariable=principalvalue,font="arial 20",width=8)
rateentry=Entry(root,textvariable=ratevalue,font="arial 20",width=8)
timeentry=Entry(root,textvariable=timevalue,font="arial 20",width=8)

principalentry.place(x=200,y=20)
rateentry.place(x=200,y=90)
timeentry.place(x=200,y=160)

Button(text="Calculate",font="arial 15",command=calculate).place(x=350,y=20)
Button(root,text='Exit',command=lambda:exit(),font="arial 15",width=8).place(x=350,y=90)


root.mainloop()
```

LEARNINGS

- I learn to import different libraries and its uses.
- Learn to load data and perform different activities to cleandata.
- I got knowledge of Exploratory data analysis.
- I learn about ML modelling.
- Got knowledge of KNN, Logistic Regression, SVM, DecisionTree, Linear Regression and Random Forest these ML models.
- I learn to implement different performance metrics according to our problem statement. if it is a classification problem we will use Accuracy, Precision, Recall, F1 Score and Confusion Matrix These Performance Metrics. If our problem is a regression problem then we will use R sq., RMSE as performance metrics.
- Along with ML models and Performance metrics we learnabout Gradient Descent, One Hot Encoding, Hyperparameter tuning, and Cross-Validation.
- I learn to handle the different situations in ML like Imbalance data, Missing values, Scaling, Categorical features and Multiclass classification.
- I learn to make code from the scratch.
- I Used Python to build applications for the Web as well as desktop and Mobile platforms. Like a simple interest Calculator
- I used Python to create a variety of different programs.

REFERENCES

- www.kaggle.com
- www.w3school.com
- www.sklearn.com
- www.analyticsvidhya.com
- www.medium.com
- www.geeksforgeeks.org
- www.wikipedia.org
- www.simplilearn.com