

▼ Machine Learning

Logistic Regression

By \Rightarrow PRINCE 

▼ Formula For Accuracy in Classification

- **Accuracy** = $(TP + TN) / (TP + TN + FP + FN)$
- **Precision** = $TP / (TP + FP)$
- **Recall** = $TP / (TP + FN)$

where,

TP = True Positive

TN = True Negative

FP = False Positive

FN = False Negative

\rightarrow F1 Score is the **Harmonic Mean** of **Precision** and **Recall**.

- **F1 Score** = $(2 * Precision * Recall) / (Precision + Recall)$

▼ Mounting Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

Normalized Data \rightarrow Mean and Median are equivalent or almost equal.

▼ Python Implementation For Logistic Regression \Rightarrow

Importing Some Important Libraries

```
# Importing required Libraries
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

```
# Upload and Read the Data
df = pd.read_csv('/content/drive/MyDrive/Dataset/Bank.csv')
```

```
# First Five Rows
df.head()
```

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account
0	1	25	1	49	91107	4	1.6	1	0	0	1	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0

```
# Last Five Rows
df.tail()
```

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	Ac
4995	4996	29		3	40	92697	1	1.9	3	0	0	0
4996	4997	30		4	15	92037	4	0.4	1	85	0	0
4997	4998	63		39	24	93023	2	0.3	3	0	0	0

```
# Info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
#   Column              Non-Null Count  Dtype
---  -
0   ID                  5000 non-null   int64
1   Age                 5000 non-null   int64
2   Experience           5000 non-null   int64
3   Income              5000 non-null   int64
4   ZIP Code            5000 non-null   int64
5   Family              5000 non-null   int64
6   CCAvg               5000 non-null   float64
7   Education           5000 non-null   int64
8   Mortgage            5000 non-null   int64
9   Personal Loan       5000 non-null   int64
10  Securities Account  5000 non-null   int64
11  CD Account          5000 non-null   int64
12  Online              5000 non-null   int64
13  CreditCard          5000 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 547.0 KB
```

```
# Shape => (Rows, Columns)
df.shape
```

(5000, 14)

```
# Checking For Null Values
df.isna().apply(pd.value_counts).T
```

	False	
ID	5000	
Age	5000	
Experience	5000	
Income	5000	
ZIP Code	5000	
Family	5000	
CCAvg	5000	
Education	5000	
Mortgage	5000	
Personal Loan	5000	
Securities Account	5000	
CD Account	5000	
Online	5000	
CreditCard	5000	

```
# Describe
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
ID	5000.0	2500.500000	1443.520003	1.0	1250.75	2500.5	3750.25	5000.0
Age	5000.0	45.338400	11.463166	23.0	35.00	45.0	55.00	67.0
Experience	5000.0	20.104600	11.467954	-3.0	10.00	20.0	30.00	43.0
Income	5000.0	73.774200	46.033729	8.0	39.00	64.0	98.00	224.0
ZIP Code	5000.0	93152.503000	2121.852197	9307.0	91911.00	93437.0	94608.00	96651.0
Family	5000.0	2.396400	1.147663	1.0	1.00	2.0	3.00	4.0
CCAvg	5000.0	1.937938	1.747659	0.0	0.70	1.5	2.50	10.0

Visualization

```

import matplotlib.pyplot as plt
import seaborn as sns

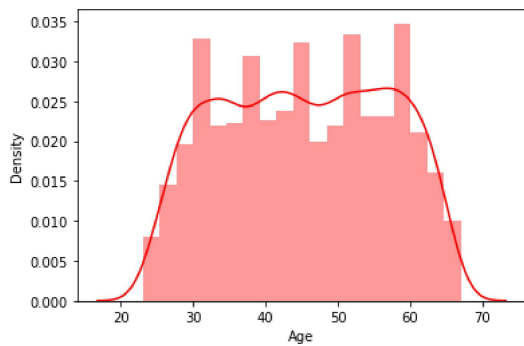
```

Distplot

```

sns.distplot(df['Age'], color = 'red')
plt.show()

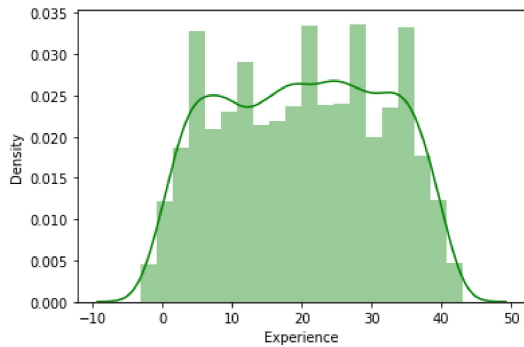
```



```

sns.distplot(df['Experience'], color = 'g')
plt.show()

```



'Age' and 'Experience' have equally distributed data (Normalized Graph).

'ID' and "ZIP Code" are not needed because they have not training feature.

```

# 'ID' and "ZIP Code" are not needed because they have not training feature.
df.drop(['ID', 'ZIP Code'], axis = 1, inplace = True)
df.head()

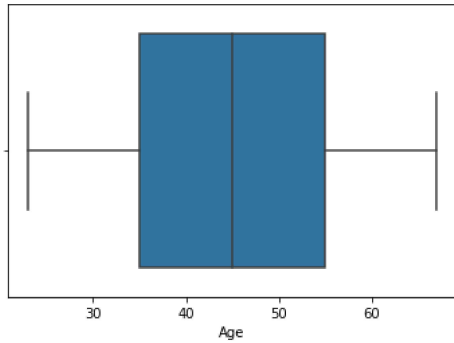
```

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online C
0	25	1	49	4	1.6	1	0	0	1	0	0
1	45	19	34	3	1.5	1	0	0	1	0	0
2	39	15	11	1	1.0	1	0	0	0	0	0
3	35	9	100	1	2.7	2	0	0	0	0	0
4	35	8	45	4	1.0	2	0	0	0	0	0

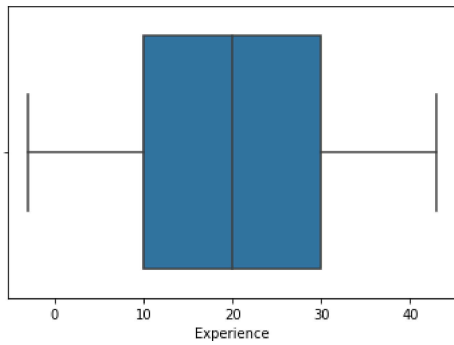
```
df.shape  
  
(5000, 12)
```

▼ Boxplot

```
sns.boxplot(df['Age'])  
plt.show()
```

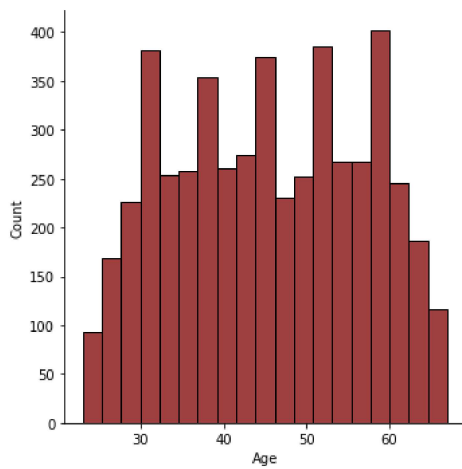


```
sns.boxplot(df['Experience'])  
plt.show()
```

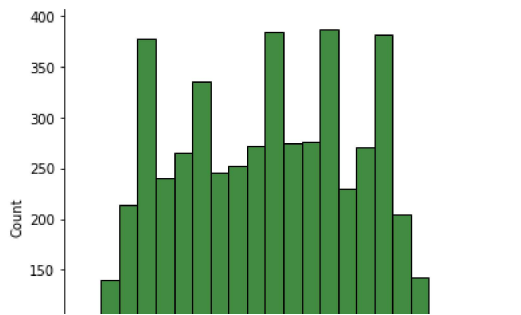


▼ Displot

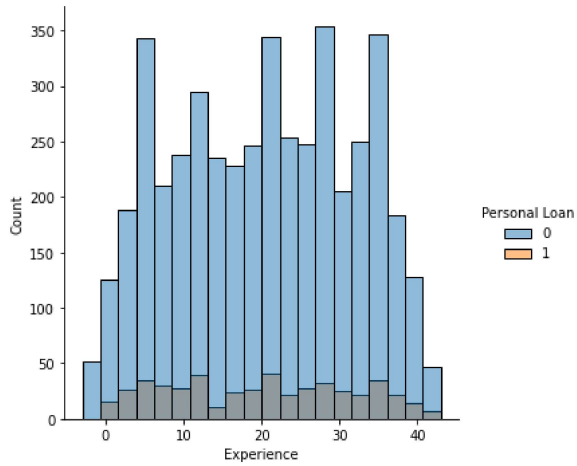
```
sns.displot(x = 'Age', data = df, color = 'maroon')  
plt.show()
```



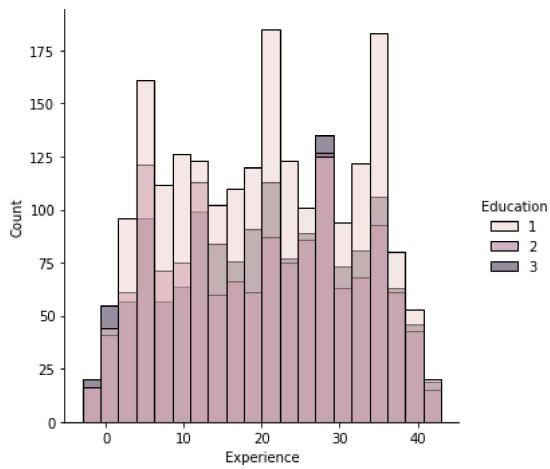
```
sns.displot(x = 'Experience', data = df, color = 'darkgreen')  
plt.show()
```



```
sns.displot(x = 'Experience', data = df, hue = 'Personal Loan', color = 'red')
plt.show()
```

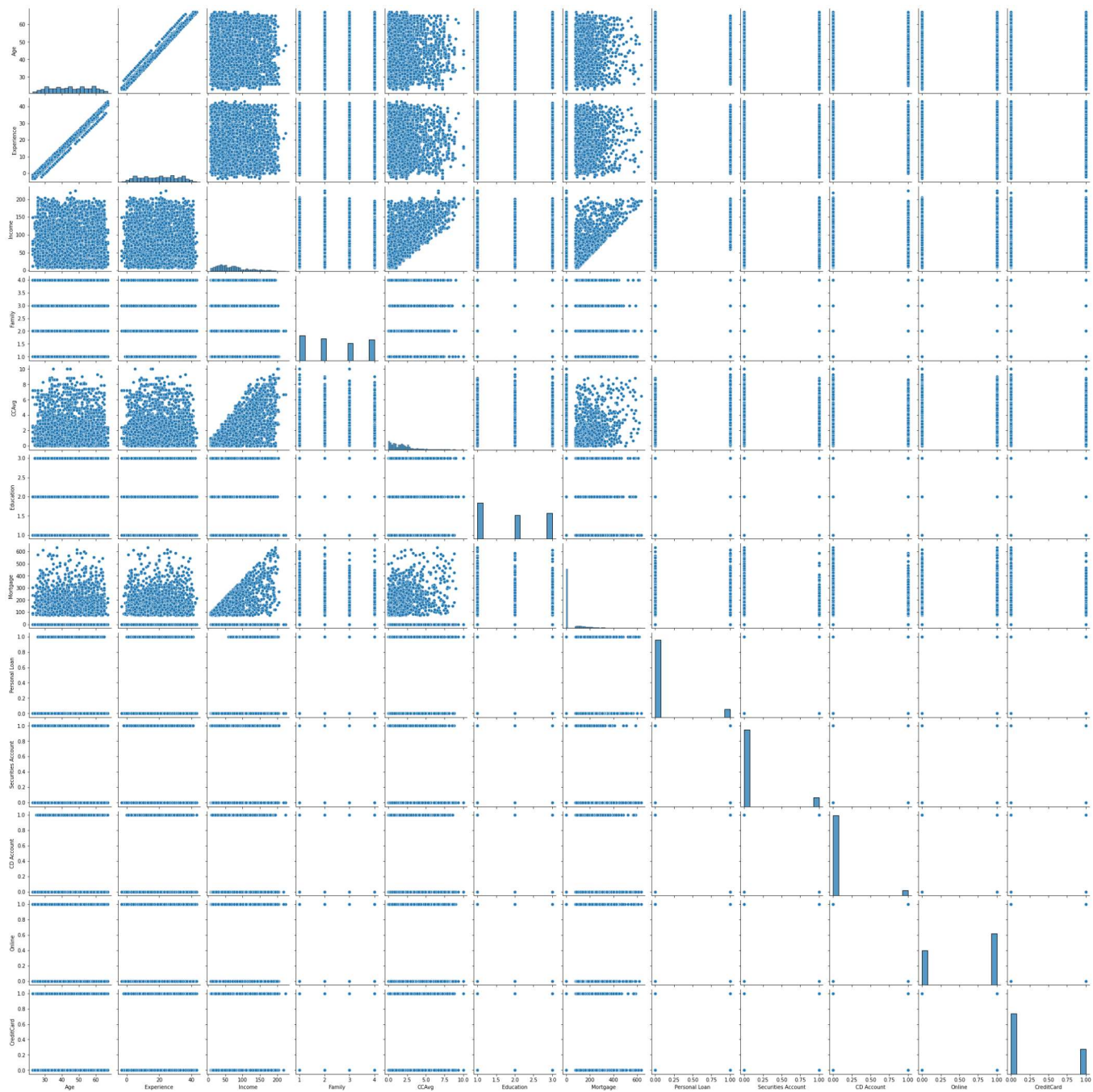


```
sns.displot(x = 'Experience', data = df, hue = 'Education')
plt.show()
```



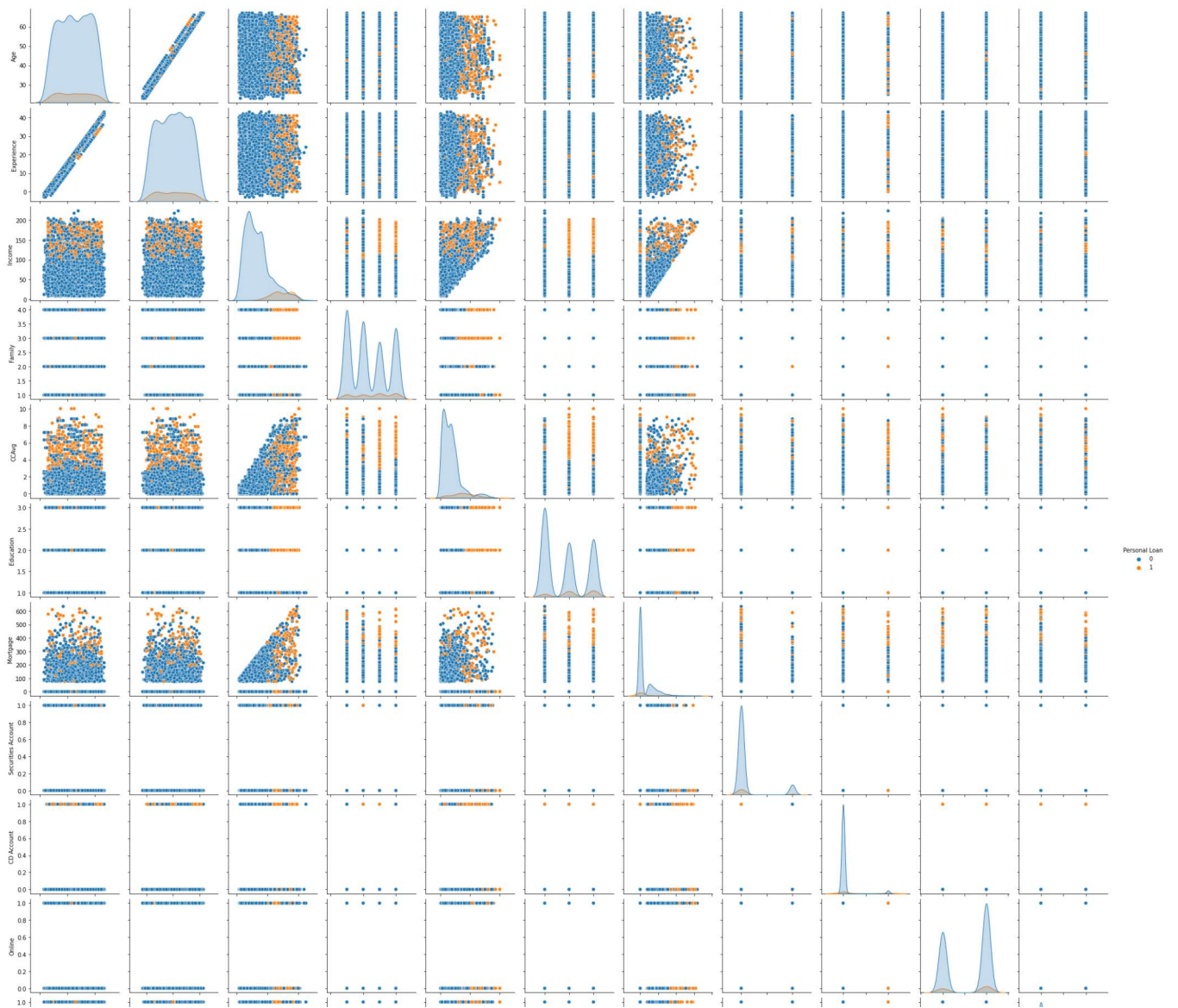
▼ Pairplot

```
sns.pairplot(df)
plt.show()
```



```
sns.pairplot(df, hue = 'Personal Loan')
plt.show()
```





Correlation

```
corr = df.corr()
corr
```

	Age	Experience	Income	Family	CCAvg	Education	Mortg
Age	1.000000	0.994215	-0.055269	-0.046418	-0.052012	0.041334	-0.012
Experience	0.994215	1.000000	-0.046574	-0.052563	-0.050077	0.013152	-0.010
Income	-0.055269	-0.046574	1.000000	-0.157501	0.645984	-0.187524	0.206
Family	-0.046418	-0.052563	-0.157501	1.000000	-0.109275	0.064929	-0.020
CCAvg	-0.052012	-0.050077	0.645984	-0.109275	1.000000	-0.136124	0.109
Education	0.041334	0.013152	-0.187524	0.064929	-0.136124	1.000000	-0.033
Mortgage	-0.012539	-0.010582	0.206806	-0.020445	0.109905	-0.033327	1.000
Personal Loan	-0.007726	-0.007413	0.502462	0.061367	0.366889	0.136722	0.142
Securities Account	-0.000436	-0.001232	-0.002616	0.019994	0.015086	-0.010812	-0.005
CD Account	0.008043	0.010353	0.169738	0.014110	0.136534	0.013934	0.089

```
corr[['Personal Loan']]
```

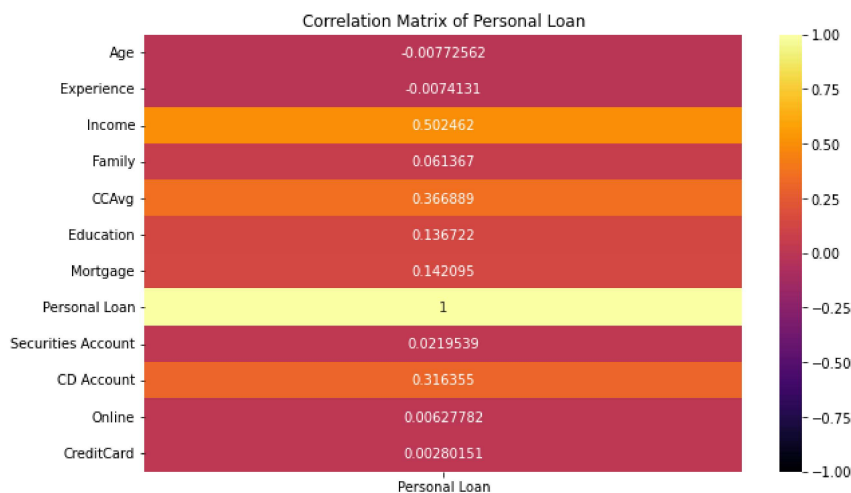
	Personal Loan	
Age	-0.007726	
Experience	-0.007413	
Income	0.502462	
Family	0.061367	
CCAvg	0.366889	
Education	0.136722	
Mortgage	0.142095	
Personal Loan	1.000000	
Securities Account	0.021954	
CD Account	0.316355	

Plot the Correlation Matrix

```
plt.figure(figsize = (10,6))
```

```
plt.title('Correlation Matrix of Personal Loan')
```

```
sns.heatmap(corr[['Personal Loan']],vmax = 1.0, vmin = -1.0, annot = True, cmap = 'inferno', fmt = 'g')
plt.show()
```



Define Independent and Dependent Variable

To define independent variable, drop dependent variable.

```
# Define Independent Variable
# To define independent variable, drop dependent variable.
```

```
x = df.drop('Personal Loan', axis = 1)
```

```
# Define Dependent Variable
y = df[['Personal Loan']]
```

Train Test Split

```
# Train Test Split at ration 70:30
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 1 )
```

Third Machine Learning Model

```
from sklearn.linear_model import LogisticRegression
# Our Third ML Model
model = LogisticRegression()
```

Train


```
# Train
model.fit(x_train, y_train)

LogisticRegression()
```

▼ Test or Prediction

```
# Test or predict
y_pred = model.predict(x_test)
y_pred

array([0, 0, 0, ..., 0, 0, 0])
```

▼ Accuracy

Test Accuracy and Training Accuracy should be closer to each other and test accuracy will be taken under consideration, not the training accuracy

```
# **Test Accuracy and Training Accuracy should be closer to each other and
#test accuracy will be taken under consideration, not the training accuracy.**
model.score(x_train, y_train)    # Training Accuracy

0.95

model.score(x_test, y_test)      #Accuracy => Test Accuracy

0.942

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)

0.942
```

▼ Confusion Matrix

```
from sklearn.metrics import confusion_matrix, classification_report
cf = confusion_matrix(y_test, y_pred)
cf

array([[1334,  17],
       [ 70,   79]])
```



TP = 1334 | FP = 17

FN = 70 | TN = 79


▼ Plotting the Confusion Matrix

```
# Plotting the confusion matrix
plt.figure(figsize = (10,5))
plt.title('Confusion Matrix of Logistic Regression', fontsize = 16)
sns.heatmap(cf, annot = True, cmap = 'spring', fmt = 'g')
plt.show()
```

Prove Accuracy by Confusion Matrix


(1334+79)/(1334+79+70+17) # Should be equal to 0.942
0.942


Classification Report


Classification Report : Use print keyword for proper (better) format.
print(classification_report(y_test, y_pred))

	precision	recall	f1-score	support
0	0.95	0.99	0.97	1351
1	0.82	0.53	0.64	149
accuracy			0.94	1500
macro avg	0.89	0.76	0.81	1500
weighted avg	0.94	0.94	0.94	1500

How KNN Fails ?

Drawbacks →

- KNN mistreats or misclassifies outliers.
- The misclassification will reduce our accuracy.

```
from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier(n_neighbors = 3)  
knn.fit(x_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
# Prediction  
knn_pred = knn.predict(x_test)  
knn_pred  
  
array([0, 0, 0, ..., 0, 0, 0])
```

```
# Accuracy  
accuracy_score(y_test, knn_pred)  
  
0.9033333333333333
```

```
knn.score(x_train, y_train) # Training Accuracy  
  
0.9551428571428572
```

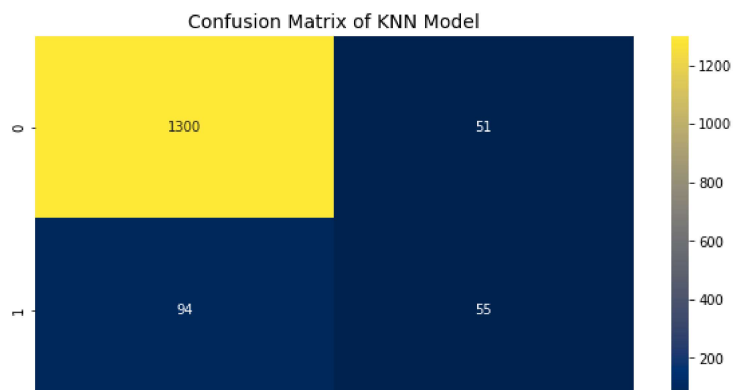
In KNN training accuracy = 0.9551 and test accuracy or accuracy = 0.9033

Difference between training accuracy and test accuracy should be minimal, hardly 0.8 but In KNN this difference is almost 5 .

So, K-Nearest Neighbors Model fails here.

Confusion Matrix of KNN Model

```
# Confusion Matrix of KNN  
cf_knn = confusion_matrix(y_test, knn_pred)  
cf_knn  
  
array([[1300, 51],  
       [ 94, 55]])  
  
plt.figure(figsize = (10,5))  
plt.title('Confusion Matrix of KNN Model', fontsize = 14)  
sns.heatmap(cf_knn, annot = True, cmap = 'cividis', fmt = 'g')  
plt.show()
```



Classification Report of KNN Model

```
# Classification Report of KNN Model  
print(classification_report(y_test, knn_pred))
```

	precision	recall	f1-score	support
0	0.93	0.96	0.95	1351
1	0.52	0.37	0.43	149
accuracy			0.90	1500
macro avg	0.73	0.67	0.69	1500
weighted avg	0.89	0.90	0.90	1500