1. **Read the column description and ensure you understand each attribute well**

```
import numpy as np
import pandas as pd
import matplotlib as plt
import seaborn as sns
%matplotlib inline

data = pd.read_csv('Bank_Personal_Loan_Modelling.csv')
data.head()
```

```
In [12]: import numpy as np
         import pandas as pd
         import matplotlib as plt
         import seaborn as sns
         %matplotlib inline

In [14]: data = pd.read_csv('Bank_Personal_Loan_Modelling.csv')
         data.head()
```

Out[14]:

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

data.dtypes

```
In [16]: data.dtypes

Out[16]: ID                    int64
         Age                   int64
         Experience            int64
         Income                int64
         ZIP Code              int64
         Family                int64
         CCAvg               float64
         Education             int64
         Mortgage              int64
         Personal Loan         int64
         Securities Account    int64
         CD Account            int64
         Online                int64
         CreditCard            int64
         dtype: object
```

data.shape

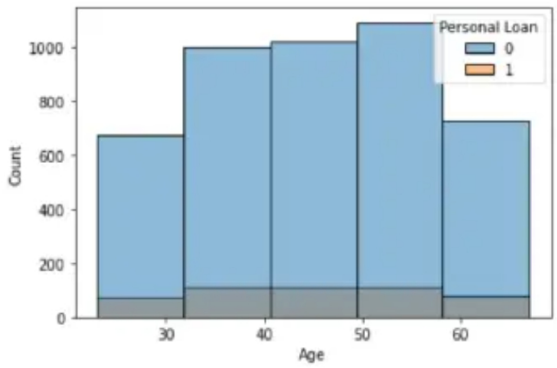
```
In [18]: data.shape
```

```
Out[18]: (5000, 14)
```


2. **Perform univariate analysis of each and every attribute - use an appropriate plot for a given attribute and mention your insights**

sns.histplot(x=data['Age'], hue =data['Personal Loan'], bins = 5)

```
In [19]: sns.histplot(x=data['Age'], hue = data['Personal Loan'],bins = 5 )
Out[19]: <AxesSubplot:xlabel='Age', ylabel='Count'>
```

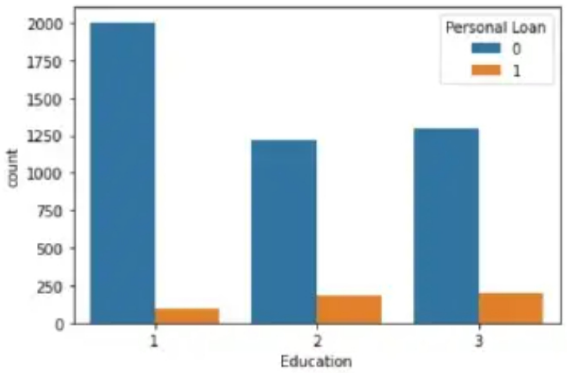


sns.countplot(x=data['Family'], hue=data['Personal Loan'])

```
In [36]: sns.countplot(x=data['Family'], hue=data['Personal Loan'])
Out[36]: <AxesSubplot:xlabel='Family', ylabel='count'>
```

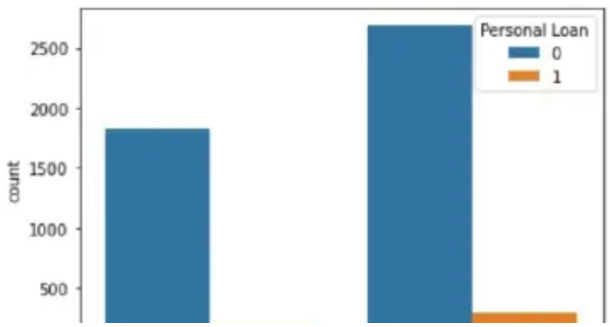
sns.countplot(x=data['Education'], hue=data['Personal Loan'])

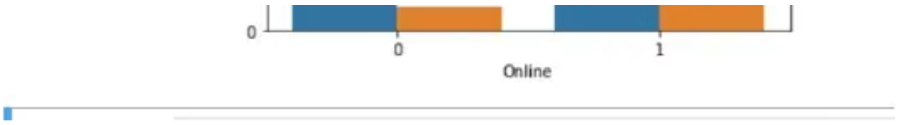
```
In [9]: sns.countplot(x=data['Education'], hue=data['Personal Loan'])
Out[9]: <AxesSubplot:xlabel='Education', ylabel='count'>
```



sns.countplot(x=data['Online'], hue=data['Personal Loan'])

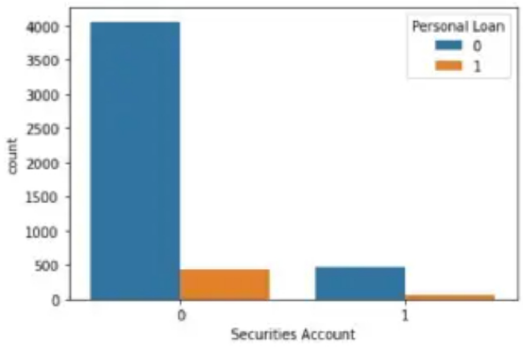
```
In [14]: sns.countplot(x=data['Online'], hue=data['Personal Loan'])
Out[14]: <AxesSubplot:xlabel='Online', ylabel='count'>
```





sns.countplot(x=data['Securities Account'], hue=data['Personal Loan'])

```
In [16]: sns.countplot(x=data['Securities Account'], hue=data['Personal Loan'])
Out[16]: <AxesSubplot:xlabel='Securities Account', ylabel='count'>
```

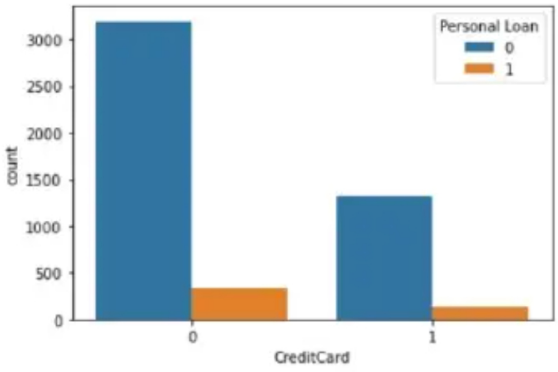


sns.countplot(x=data['CD Account'], hue=data['Personal Loan'])

```
In [17]: sns.countplot(x=data['CD Account'], hue=data['Personal Loan'])
Out[17]: <AxesSubplot:xlabel='CD Account', ylabel='count'>
```

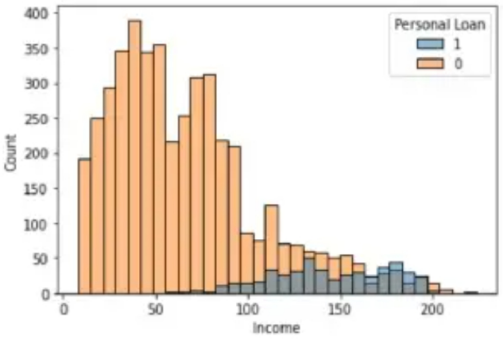
sns.countplot(x=data['CreditCard'], hue=data['Personal Loan'])

```
In [18]: sns.countplot(x=data['CreditCard'], hue=data['Personal Loan'])
Out[18]: <AxesSubplot:xlabel='CreditCard', ylabel='count'>
```



sns.histplot(x=data['Income'], hue=data['Personal Loan'], hue_order = [1,0])

```
In [28]: sns.histplot(x=data['Income'], hue=data['Personal Loan'], hue_order = [1,0])
Out[28]: <AxesSubplot:xlabel='Income', ylabel='Count'>
```



Inference:

i. The age groups of 35-45 take the maximum number of personal loans
ii. The bank has maximum customers with 1 family member, but this customer group has the lowest personal loan number whereas customers with family of 3 and 4 members have taken a greater number of personal loans
iii. More customers with advanced/professional education take personal loan as

compared to the customers with undergrad or graduate education

iv. A greater number of customers who avail internet banking facilities avail persona loans than those who do not avail internet banking but the percentage of both is approximately same.

v. Customers without securities account with the bank avail more personal loan

vi. More number of customers without CD account use personal loans although more percentage of customers with CD account use personal loan

vii. Higher number of customers without credit cards avail personal loan

viii. The count of people who have taken personal loan is the highest between the income range of $120,000 – $140,000 and then rises again in the $170,000 – $190,000. Lower income groups have not taken personal loans

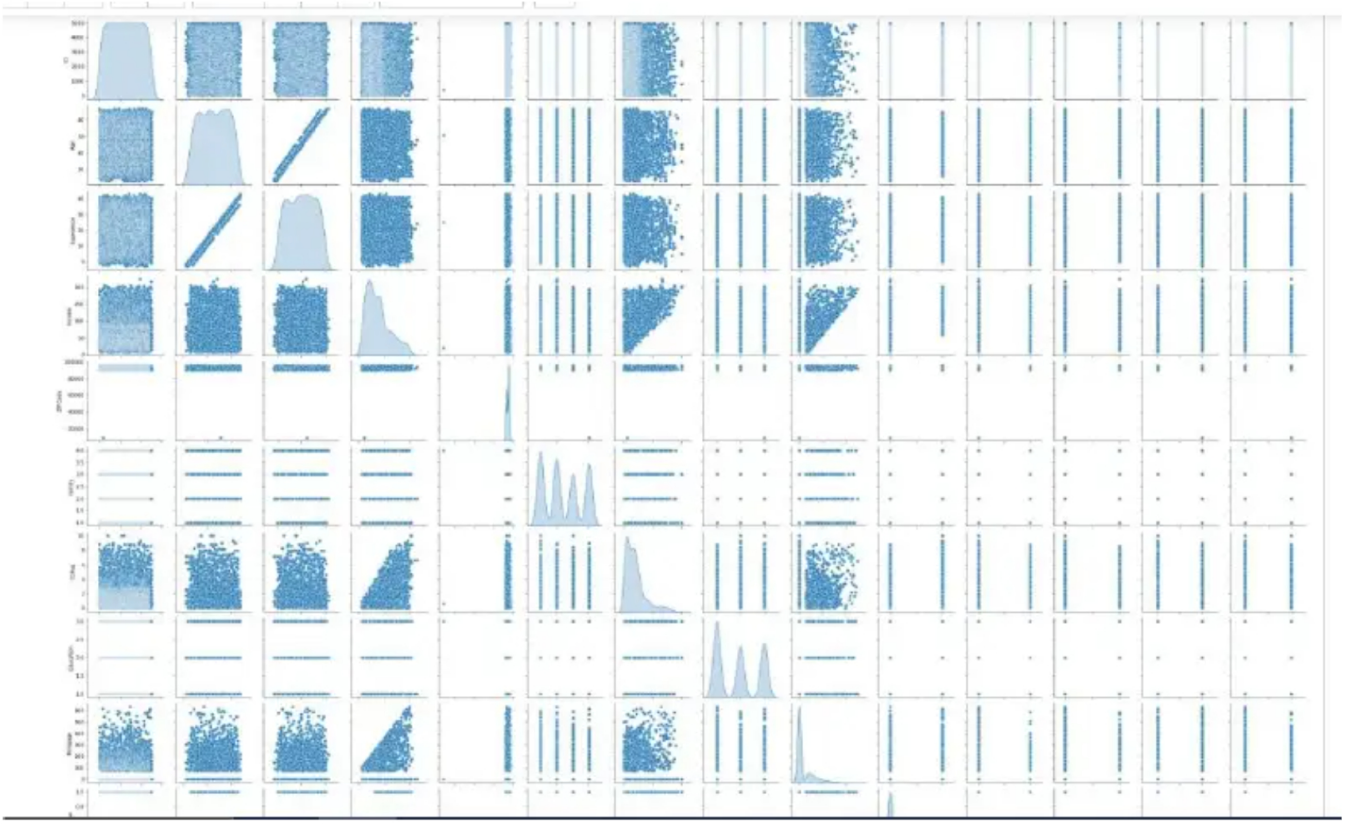
3. Perform correlation analysis among all the variables - you can use Pairplot and Correlation coefficients of every attribute with every other attribute

```
datacor = data.corr()
datacor
```

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	
ID	1.000000	-0.008473	-0.008326	-0.017695	0.013432	-0.016797	-0.024675	0.021463	-0.013920	-0.024801	-0.016972	-0.006909	-0.002528	0.017
Age	-0.008473	1.000000	0.994215	-0.055269	-0.029216	-0.046418	-0.052012	0.041334	-0.012539	-0.007726	-0.000436	0.008043	0.013702	0.007
Experience	-0.008326	0.994215	1.000000	-0.046574	-0.028626	-0.052563	-0.050077	0.013152	-0.010582	-0.007413	-0.001232	0.010353	0.013898	0.008
Income	-0.017695	-0.055269	-0.046574	1.000000	-0.016410	-0.157501	0.645984	-0.187524	0.206806	0.502462	-0.002616	0.169738	0.014206	-0.002
ZIP Code	0.013432	-0.029216	-0.028626	-0.016410	1.000000	0.011778	-0.004061	-0.017377	0.007383	0.000107	0.004704	0.019972	0.016990	0.007
Family	-0.016797	-0.046418	-0.052563	-0.157501	0.011778	1.000000	-0.109275	0.064929	-0.020445	0.061367	0.019994	0.014110	0.010354	0.011
CCAvg	-0.024675	-0.052012	-0.050077	0.645984	-0.004061	-0.109275	1.000000	-0.136124	0.109905	0.366889	0.015086	0.136534	-0.003611	-0.006
Education	0.021463	0.041334	0.013152	-0.187524	-0.017377	0.064929	-0.136124	1.000000	-0.033327	0.136722	-0.010812	0.013934	-0.015004	-0.011
Mortgage	-0.013920	-0.012539	-0.010582	0.206806	0.007383	-0.020445	0.109905	-0.033327	1.000000	0.142095	-0.005411	0.089311	-0.005995	-0.007
Personal Loan	-0.024801	-0.007726	-0.007413	0.502462	0.000107	0.061367	0.366889	0.136722	0.142095	1.000000	0.021954	0.316355	0.006278	0.002
Securities Account	-0.016972	-0.000436	-0.001232	-0.002616	0.004704	0.019994	0.015086	-0.010812	-0.005411	0.021954	1.000000	0.317034	0.012627	-0.015

CD Account	-0.006909	0.008043	0.010353	0.169738	0.019972	0.014110	0.136534	0.013934	0.089311	0.316355	0.317034	1.000000	0.175880
Online	-0.002528	0.013702	0.013898	0.014206	0.016990	0.010354	-0.003611	-0.015004	-0.005995	0.006278	0.012627	0.175880	1.000000
CreditCard	0.017028	0.007681	0.008967	-0.002385	0.007691	0.011588	-0.006689	-0.011014	-0.007231	0.002802	-0.015028	0.278644	0.004210

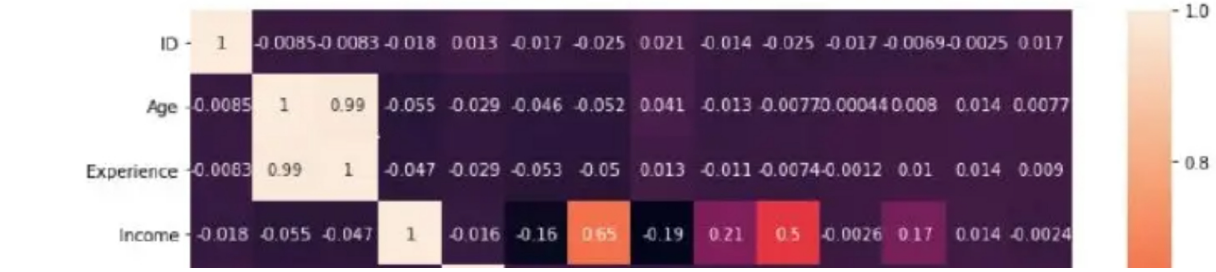
```
sns.pairplot(data, diag_kind='kde')
```

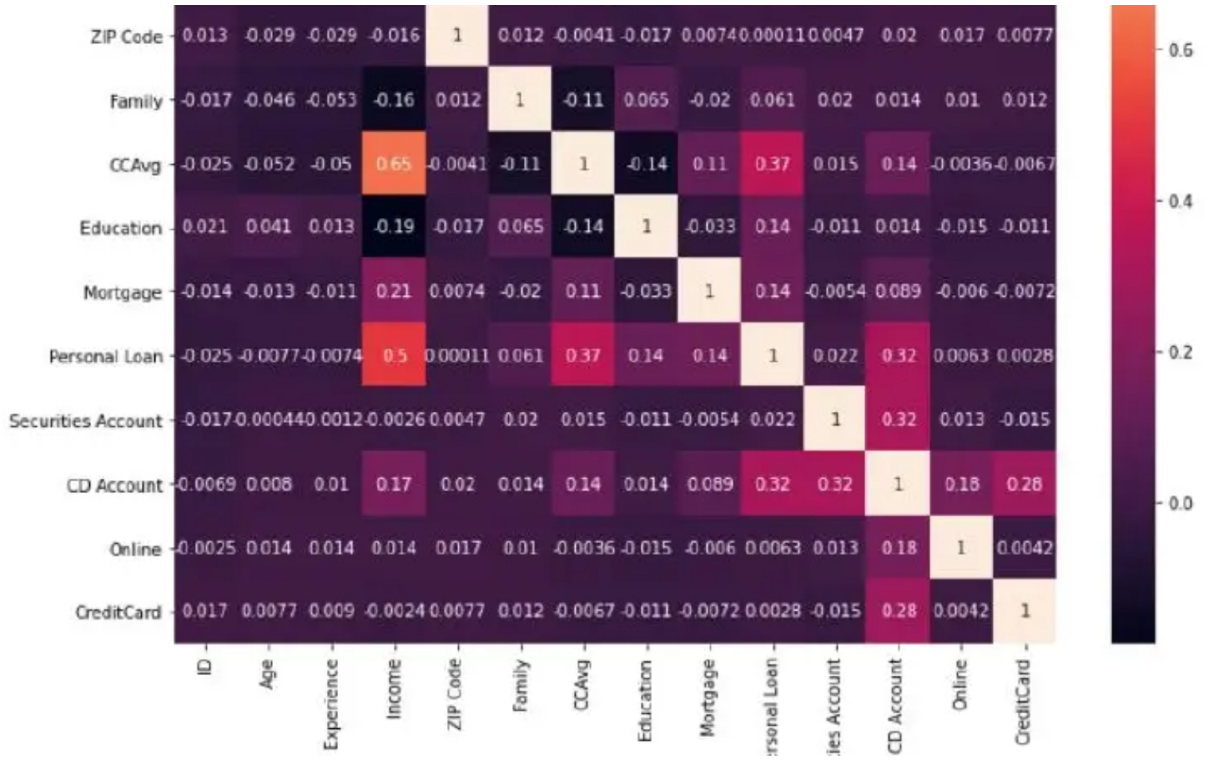


```
plt.subplots(figsize=(12,10))
sns.heatmap(datacor,annot=True)
```

```
plt.subplots(figsize=(12,10))
sns.heatmap(datacor,annot=True)
```

```
<AxesSubplot:>
```





4. One hot encode the Education variable (3 points)

OHED = pd.get_dummies(data, columns =['Education'])
OHED

```
In [35]: OHED = pd.get_dummies(data, columns =['Education'])
         OHED
```

Out[35]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard	Education_1	Education_2	Education_3
0	1	25	1	49	91107	4	1.6	0	0	1	0	0	0	1	0	0
1	2	45	19	34	90089	3	1.5	0	0	1	0	0	0	1	0	0

2	3	39	15	11	94720	1	1.0	0	0	0	0	0	0	1	0	0
3	4	35	9	100	94112	1	2.7	0	0	0	0	0	0	0	1	0
4	5	35	8	45	91330	4	1.0	0	0	0	0	0	1	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5	4996	29	3	40	92697	1	1.9	0	0	0	0	1	0	0	0	1
6	4997	30	4	15	92037	4	0.4	85	0	0	0	1	0	1	0	0
7	4998	63	39	24	93023	2	0.3	0	0	0	0	0	0	0	0	1
8	4999	65	40	49	90034	3	0.5	0	0	0	0	1	0	0	1	0
9	5000	28	4	83	92612	3	0.8	0	0	0	0	1	1	1	0	0

) rows × 16 columns

5. **Separate the data into dependant and independent variables and create training and test sets out of them (X_train, y_train, X_test, y_test) (2 points)**

   Before separating, we need to drop the redundant columns and append the education one hot encoded data into the main data.

   Here, the dependent variable and the variable of interest is Personal loan, therefore, Personal Loan column would be y axis, the rest of the columns would be counted as independent variables.

```
data['Experience']=abs(data['Experience'])
dt = OHED
dt = dt.drop(['ID'], axis =1)
dt = dt.drop(['ZIP Code'], axis=1)


X = dt.drop(['Personal Loan'], axis=1)
Y = dt['Personal Loan']

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, train_size = 0.7, test_size=0.3,
random_state = 100)

X_train
Y_train
```

1890	52	27	184	1	8.1	0	0	0	0	0	1	0	0
350	39	14	113	1	1.0	0	1	0	1	0	0	0	1
79	50	26	19	2	0.4	118	0	0	1	0	1	0	0
3927	59	34	38	4	1.7	0	0	0	0	1	1	0	0

3500 rows × 13 columns

```
In [61]: Y_train

Out[61]: 4966    0
         4086    0
         2881    0
         1237    1
         1429    0
                ..
         4149    0
         1890    0
         350     0
         79      0
         3927    0
         Name: Personal Loan, Length: 3500, dtype: int64
```

print("Training set: {0:0.2f}% ".format((len(X_train)/len(data.index)) * 100))
print("Test set: {0:0.2f}% ".format((len(X_test)/len(data.index)) * 100))

```
print("Training set: {0:0.2f}% ".format((len(X_train)/len(data.index)) * 100))
print("Test set: {0:0.2f}% ".format((len(X_test)/len(data.index)) * 100))
```

```
Training set: 70.00%
Test set: 30.00%
```

6. **Use StandardScaler( ) from sklearn, to transform the training and test data into scaled values (fit the StandardScaler object to the train data and transform train and test data using this object, making sure that the test set does not influence the values of the train set)**

```
from sklearn.preprocessing import StandardScaler
Std_Scalar = StandardScaler()
X_train = Std_Scalar.fit_transform(X_train)
X_test = Std_Scalar.fit_transform(X_test)
X_train
```

X_test

```
In [23]: from sklearn.preprocessing import StandardScaler

In [39]: Std_Scalar = StandardScaler()
         X_train = Std_Scalar.fit_transform(X_train)
         X_test = Std_Scalar.fit_transform(X_test)
         X_train
         X_test

Out[39]: array([[-1.16811195, -1.39630254, -1.46723006, ..., -0.85446112,
                  1.60356745, -0.65153774],
                [ 0.02088097,  1.18976218,  1.20463984, ...,  1.17032826,
                 -0.62360956, -0.65153774],
                [-0.06978147,  1.10356002,  1.11845048, ...,  1.17032826,
                 -0.62360956, -0.65153774],
                ...,
                [ 1.23340459,  0.58634707,  0.51512502, ..., -0.85446112,
                 -0.62360956,  1.53483051],
                [ 0.83684293,  1.27596433,  1.29082919, ...,  1.17032826,
                 -0.62360956, -0.65153774],
                [-0.76462949, -0.53428097, -0.60533655, ...,  1.17032826,
                 -0.62360956, -0.65153774]])
```

7. **Write a function which takes a model, X_train, X_test, y_train and y_test as input and returns the accuracy, recall, precision, specificity, f1_score of the model trained on the train set and evaluated on the test set**

```
def funct(confusion_matrix):

    total=sum(sum(confusion_matrix))
    accuracy=  (confusion_matrix[0,0]+confusion_matrix[1,1])/total
    print ('Accuracy :{:.2%}'.format(accuracy))

    specificity =
confusion_matrix[0,0]/(confusion_matrix[0,0]+confusion_matrix[0,1])
    print('Specificity : {:.2%}'.format(specificity) )

    sensitivity = confusion_matrix[1,1]/(confusion_matrix[1,0]+confusion_matrix[1,1])
    print('Sensitivity : {:.2%}'.format(sensitivity))

    precision = confusion_matrix[1,1]/(confusion_matrix[1,1]+confusion_matrix[0,1])
    print('Precision : {:.2%}'.format(precision))

    F1 = 2*((precision*sensitivity)/(precision+sensitivity))
    print('F1 Score : {:.2%}'.format(F1))

        return accuracy, sensitivity, specificity, precision, F1
```

8. **Employ multiple Classification models (Logistic, K-NN, Naïve Bayes etc) and use the function from step 7 to train and get the metrics of the model**

Logistic Regression:

 # Import libraries
from sklearn import metrics
from sklearn.linear_model import LogisticRegression

 # Model
Log_model = LogisticRegression(solver="liblinear")
Log_model.fit(X_train, Y_train)

predict = Log_model.predict(X_test)
coef_df = pd.DataFrame(Log_model.coef_)
coef_df['intercept'] = Log_model.intercept_
print(coef_df)

```
In [88]: from sklearn import metrics
         from sklearn.linear_model import LogisticRegression

In [89]: Log_model = LogisticRegression(solver="liblinear")
         Log_model.fit(X_train, Y_train)


         predict = Log_model.predict(X_test)

In [90]: coef_df = pd.DataFrame(Log_model.coef_)
         coef_df['intercept'] = Log_model.intercept_
         print(coef_df)
```

```
          0         1         2         3         4         5         6  \
0  0.021017  0.173965  2.686426  0.651084  0.291498  0.096053 -0.16255

          7         8         9        10        11        12  intercept
0  0.757086 -0.470049 -0.34435 -1.248857  0.635596  0.720049  -4.999253
```

 # Score
score = Log_model.score(X_test, Y_test)
print(score)

```
score = Log_model.score(X_test, Y_test)
print(score)
```

```
0.9546666666666667
```

 # Confusion matrix
from sklearn.metrics import confusion_matrix
conf_mat_Log = confusion_matrix(Y_test, predict)
print(conf_mat_Log)

print(conf_mat_Log)

test_mat = funct(conf_mat_Log)

```
test_mat = funct(conf_mat_Log)
```

```
Accuracy :95.47%
Specificity : 99.25%
Sensitivity : 63.29%
Precision : 90.91%
F1 Score : 74.63%
```

Naïve Bayes

```
# Import required libraries
from sklearn.naive_bayes import GaussianNB

# Model
GNB1 = GaussianNB()
GNB1.fit(X_train, Y_train)
predicted_labels_GNB = GNB1.predict(X_test)

GNB1.score(X_test, Y_test)

# Confusion matrix
con_mat = metrics.confusion_matrix(Y_test, predicted_labels_GNB)
print(con_mat)
u = funct(con_mat)
```

```
GNB1.fit(X_train, Y_train)
predicted_labels_GNB = GNB1.predict(X_test)

GNB1.score(X_test, Y_test)
con_mat_Naive = metrics.confusion_matrix(Y_test, predicted_labels_GNB)
print(con_mat_Naive)
```

```
[[1260   82]
 [  60   98]]
```

```
In [159]: u = funct(con_mat_Naive)
```

```
Accuracy :90.53%
Specificity : 93.89%
Sensitivity : 62.03%
Precision : 54.44%
F1 Score : 57.99%
```

K-NN

 # Import required libraries
from sklearn.neighbors import KNeighborsClassifier

 # Model
NNH = KNeighborsClassifier(n_neighbors= 5 , weights = 'uniform')
NNH.fit(X_train, Y_train)

predicted_labels_KNN = NNH.predict(X_test)

NNH.score(X_test, Y_test)

```
In [152]: from sklearn.neighbors import KNeighborsClassifier

In [153]: NNH = KNeighborsClassifier(n_neighbors= 5 , weights = 'uniform' )
          NNH.fit(X_train, Y_train)

Out[153]: KNeighborsClassifier()

In [154]: predicted_labels_KNN = NNH.predict(X_test)

In [155]: NNH.score(X_test, Y_test)

Out[155]: 0.956
```

 # Create confusion matrix

```
# Create confusion matrix
Conf_matrix_KNN = metrics.confusion_matrix(Y_test, predicted_labels_KNN)
print(Conf_matrix_KNN)
```

```
In [157]: Conf_matrix_KNN = metrics.confusion_matrix(Y_test, predicted_labels_KNN)
          print(Conf_matrix_KNN)

          [[1336    6]
           [  60   98]]
```

```
tests = funct(Conf_matrix_KNN)
```

```
Precision : 94.23%
F1 Score : 74.81%
```

9. Create a dataframe with the columns - "Model", "accuracy", "recall", "precision", "specificity", "f1_score". Populate the dataframe accordingly

```
df = pd.DataFrame({'model': ['Logistic', 'Naive','KNN'],
      'Accuracy':[95.47,90.53,95.60],
      'Specificity':[99.25,93.89,99.55],
      'sensitivity':[63.29,62.03,62.03],
      'precision':[90.91,54.44,94.23],
      'F1 Score':[74.63,57.99,74.81]

   })
df
```

```
In [55]: df = pd.DataFrame({'model': ['Logistic', 'Naive','KNN'],
                'Accuracy':[95.47,90.53,95.60],
                'Specificity':[99.25,93.89,99.55],
                'sensitivity':[63.29,62.03,62.03],
                'precision':[90.91,54.44,94.23],
                'F1 Score':[74.63,57.99,74.81]
```

```
    })
df
```

Out[55]:

	model	Accuracy	Specificity	sensitivity	precision	F1 Score
0	Logistic	95.47	99.25	63.29	90.91	74.63
1	Naive	90.53	93.89	62.03	54.44	57.99
2	KNN	95.60	99.55	62.03	94.23	74.81

10. Give your reasoning on which is the best model in this case

In this case, Naïve Bayes is the worst fit model with lowest accuracy, F1 score, Precision and specificity.
K-NN has the highest accuracy (95.60%), specificity (99.55%), precision (94.23%) and F1 score (74.81%). Although the sensitivity of logistic regression is higher than K-NN (63.29%), K-NN showcases an acceptable level of sensitivity at 62.03%. Therefore, from the above data, it can be seen that **K-NN is the best fit model** for the given bank data.

Share this document

You might also like

CIPAC 2010 Thailand Nawaporn-Poster
Idon Wahidin

sigfignotes
api-295116239

Significant Figures
mecdinsue

Sensitivity And Specificity | Accuracy And Precision | Statistics | Data Analysis

3UG46161CR20 Datasheet En
engmnf

Bike Share
SUBIRMITRA

Assignment on DWDM
Venkat Nani

C - spine
Charly Montiel

THEFUNCTIONALCAPACITYEVALUATIONama-fce.pdf
Daniela

Tutorial 6 - Accuracy Assessment Tool.pdf
Fabiano Belém

ECO EN DR
Zarella Ramírez Borrero

Sesion-1 (UoM & Decision Rule)
Subrat Das

base paper
Giri Prasad

Show more