```
import pandas as pd
# reading csv files
data=pd.read_csv('/content/german.csv')
df=pd.DataFrame(data)
print(df)
```

```
     Status of existing checking account  Duration in month Credit history  \
0                                    A11                  6            A34   
1                                    A12                 48            A32   
2                                    A14                 12            A34   
3                                    A11                 42            A32   
4                                    A11                 24            A33   
..                                   ...                ...            ...   
995                                  A14                 12            A32   
996                                  A11                 30            A32   
997                                  A14                 12            A32   
998                                  A11                 45            A32   
999                                  A12                 45            A34   

    Purpose  Credit amount Savings account/bonds Present employment since  \
0       A43           1169                   A65                      A75   
1       A43           5951                   A61                      A73   
2       A46           2096                   A61                      A74   
3       A42           7882                   A61                      A74   
4       A40           4870                   A61                      A73   
..      ...            ...                   ...                      ...   
995     A42           1736                   A61                      A74   
996     A41           3857                   A61                      A73   
997     A43            804                   A61                      A75   
998     A43           1845                   A61                      A73   
999     A41           4576                   A62                      A71   

     Installment rate in percentage of disposable income  \
0                                                    4     
1                                                    2     
2                                                    2     
3                                                    2     
4                                                    3     
..                                                 ...     
995                                                  3     
996                                                  4     
997                                                  4     
998                                                  4     
999                                                  3     

    Personal status and sex Other debtors / guarantors  ...  Property  \
0                       A93                       A101  ...      A121   
1                       A92                       A101  ...      A121   
2                       A93                       A101  ...      A121   
3                       A93                       A103  ...      A122   
4                       A93                       A101  ...      A124   
..                      ...                        ...  ...       ...   
995                     A92                       A101  ...      A121   
996                     A91                       A101  ...      A122   
997                     A93                       A101  ...      A123   
998                     A93                       A101  ...      A124   
999                     A93                       A101  ...      A123   

    Age in years        Other installment plans  Housing  \
0             67                           A143     A152   
1             22                           A143     A152   
2             49                           A143     A152   
3             45                           A143     A153   
4             53                           A143     A153   
```

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

```
df.isnull().sum()
```

```
Status of existing checking account                        0
Duration in month                                          0
Credit history                                             0
Purpose                                                    0
Credit amount                                              0
Savings account/bonds                                      0
Present employment since                                   0
Installment rate in percentage of disposable income        0
Personal status and sex                                    0
Other debtors / guarantors                                 0
Present residence since                                    0
Property                                                   0
```

```
Age in years                                        0
      Other installment plans                       0
Housing                                             0
Number of existing credits at this bank             0
Job                                                 0
Number of people being liable to provide maintenance for    0
Telephone                                           0
foreign worker                                      0
Cost Matrix                                         0
dtype: int64
```

```
import statistics
df["Status of existing checking account"].mode()
```

```
0    A14
dtype: object
```

```
df["Credit history"].mode()
```

```
0    A32
dtype: object
```

```
df["Present employment since"].mode()
```

```
0    A73
dtype: object
```

```
df["Purpose"].mode()
```

```
0    A43
dtype: object
```

```
df["Savings account/bonds"].mode()
```

```
0    A61
dtype: object
```

```
df["Present employment since"].mode()
```

```
0    A73
dtype: object
```

```
df["Personal status and sex"].mode()
```

```
0    A93
dtype: object
```

```
df["Other debtors / guarantors"].mode()
```

```
0    A101
dtype: object
```

```
df["Property"].mode()
```

```
0    A123
dtype: object
```

```
df["Housing"].mode()
```

```
0    A152
dtype: object
```

```
df["Job"].mode()
```

```
0    A173
dtype: object
```

```
df["Telephone"].mode()
```

```
0    A191
dtype: object
```

```
df["foreign worker"].mode()
```

```
0    A201
dtype: object
```

```
df["Savings account/bonds"].mode()
```

```
0    A61
dtype: object
```

```
df["Personal status and sex"].mode()
```

```
0    A93
dtype: object
```

```
df["Duration in month"].median()
```

```
18.0
```

```
df["Credit amount"].median()
```

```
2319.5
```

```
df["Installment rate in percentage of disposable income"].median()
```

```
3.0
```

```
df["Present residence since"].median()
```

```
3.0
```

```
df["Age in years"].median()
```

```
33.0
```

```
df["Number of existing credits at this bank"].median()
```

```
1.0
```

```
df["Number of people being liable to provide maintenance for"].median()
```

```
1.0
```

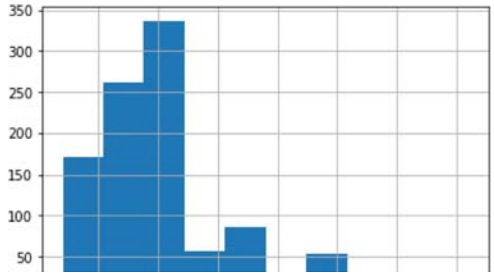
```
df.describe()
```

	Duration in month	Credit amount	Installment rate in percentage of disposable income	Present residence since	Age in years	Number of existing credits at this bank
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	20.903000	3271.258000	2.973000	2.845000	35.546000	1.407000
std	12.058814	2822.736876	1.118715	1.103718	11.375469	0.577654
min	4.000000	250.000000	1.000000	1.000000	19.000000	1.000000
25%	12.000000	1365.500000	2.000000	2.000000	27.000000	1.000000

```
import matplotlib.pyplot as plt
```

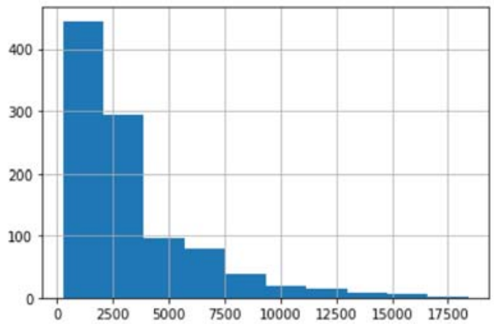
```
hist = df['Duration in month'].hist()
```

```
plt.savefig("pandas_hist_01.png", bbox_inches='tight', dpi=100)
```



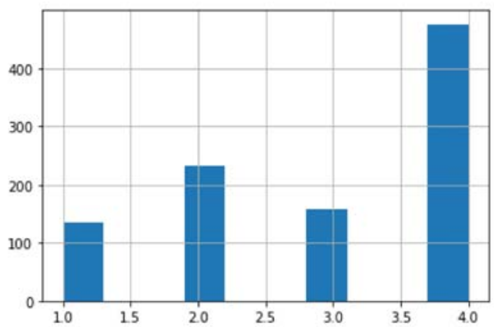

```
hist = df['Credit amount'].hist()

plt.savefig("pandas_hist_01.png", bbox_inches='tight', dpi=100)
```



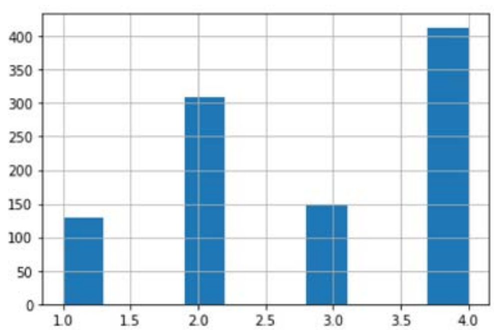

```
hist = df['Installment rate in percentage of disposable income'].hist()

plt.savefig("pandas_hist_01.png", bbox_inches='tight', dpi=100)
```



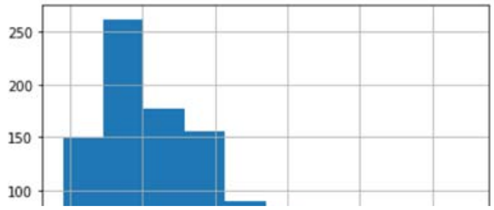

```
hist = df['Present residence since'].hist()

plt.savefig("pandas_hist_01.png", bbox_inches='tight', dpi=100)
```



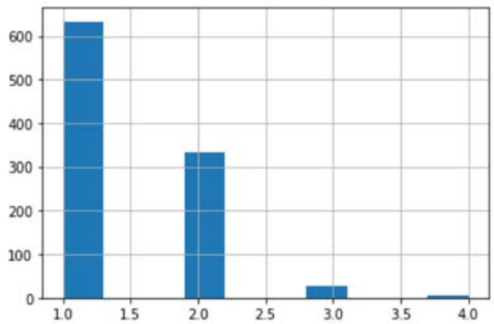

```
hist = df['Age in years'].hist()

plt.savefig("pandas_hist_01.png", bbox_inches='tight', dpi=100)
```



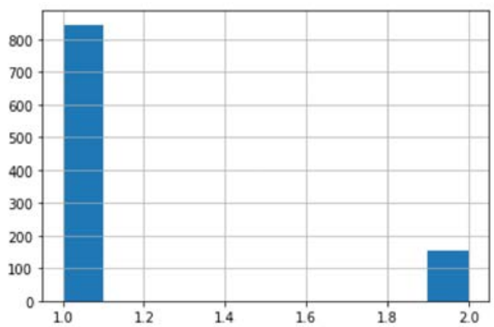
```
hist = df['Number of existing credits at this bank'].hist()

plt.savefig("pandas_hist_01.png", bbox_inches='tight', dpi=100)
```



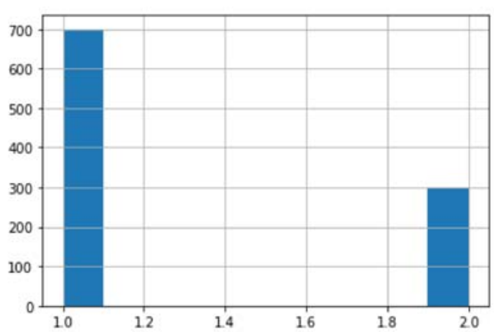
```
hist = df['Number of people being liable to provide maintenance for'].hist()

plt.savefig("pandas_hist_01.png", bbox_inches='tight', dpi=100)
```



```
hist = df['Cost Matrix'].hist()

plt.savefig("pandas_hist_01.png", bbox_inches='tight', dpi=100)
```



```
df = df[['Duration in month','Credit amount','Installment rate in percentage of disposable income','Present residence since','Age in years','
df
```

	Duration in month	Credit amount	Installment rate in percentage of disposable income	Present residence since	Age in years	Number of existing credits at this bank	Number of people being liable to provide maintenance for
0	6	1169	4	4	67	2	1
1	48	5951	2	2	22	1	1
2	12	2096	2	3	49	1	2
3	42	7882	2	4	45	1	2
4	24	4870	3	4	53	2	2

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.pairplot(df['Duration in month'], kind="scatter")
plt.show()
```

997	12	804	4	4	38	1	1

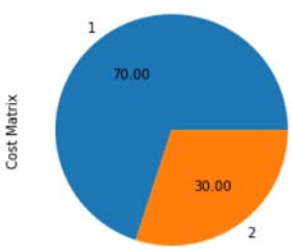
```
import numpy as np
corr = data.corr()
corr
```

	Duration in month	Credit amount	Installment rate in percentage of disposable income	Present residence since	Age in years	Number of existing credits at this bank	Nu li main
Duration in month	1.000000	0.624984	0.074749	0.034067	-0.036136	-0.011284	-0
Credit amount	0.624984	1.000000	-0.271316	0.028926	0.032716	0.020795	0
Installment rate in percentage of disposable	0.074749	-0.271316	1.000000	0.049302	0.058266	0.021669	-0

```
df['Cost Matrix'].value_counts()
```

```
# dataset is imbalanced
df['Cost Matrix'].value_counts().plot.pie(autopct='%.2f')
```

<matplotlib.axes._subplots.AxesSubplot at 0x26ac4ef4c70>



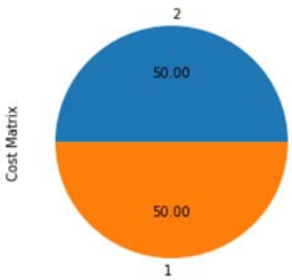

```
x = df.drop(['Cost Matrix'],axis =1)
y = df['Cost Matrix']
```

```
!pip install -U imbalanced-learn
```

```
Requirement already up-to-date: imbalanced-learn in c:\users\mohsen\anaconda3\lib\site-packages (0.8.1)
Requirement already satisfied, skipping upgrade: numpy>=1.13.3 in c:\users\mohsen\anaconda3\lib\site-packages (from imbalanced-learn) (1
Requirement already satisfied, skipping upgrade: joblib>=0.11 in c:\users\mohsen\anaconda3\lib\site-packages (from imbalanced-learn) (0.
Requirement already satisfied, skipping upgrade: scikit-learn>=0.24 in c:\users\mohsen\anaconda3\lib\site-packages (from imbalanced-lear
```

```
Requirement already satisfied, skipping upgrade: scipy>=0.19.1 in c:\users\mohsen\anaconda3\lib\site-packages (from imbalanced-learn) (1
Requirement already satisfied, skipping upgrade: threadpoolctl>=2.0.0 in c:\users\mohsen\anaconda3\lib\site-packages (from scikit-learn>
```

```
# dataset becomes balanced
import imblearn
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(sampling_strategy=1)
x_res,y_res=rus.fit_resample(x,y)
ax= y_res.value_counts().plot.pie(autopct='%.2f')
```




```
X = x_res[['Duration in month','Credit amount','Installment rate in percentage of disposable income','Present residence since','Age in years'
```

```
X
```

	Duration in month	Credit amount	Installment rate in percentage of disposable income	Present residence since	Age in years	Number of existing credits at this bank	Number of people being liable to provide maintenance for
0	12	640	4	2	49	1	1
1	12	841	2	4	23	1	1
2	27	5190	4	4	48	4	2
3	11	2142	1	2	28	1	1
4	12	701	4	2	40	1	1
...	...	...	...	...	...	...	...
595	15	1264	2	2	25	1	1
596	30	8386	2	2	49	1	1
597	48	4844	3	2	33	1	1
598	36	8229	2	2	26	1	2

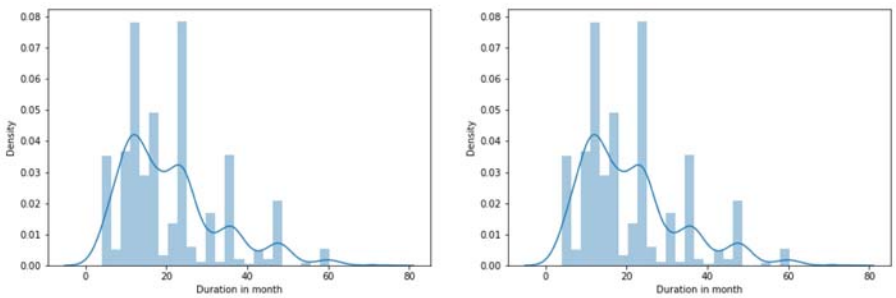
```
# Normalize data set
from sklearn import preprocessing
X_normalised = preprocessing.normalize(X)
print(X_normalised)
```

```
[[1.86915208e-02 9.96881107e-01 6.23050692e-03 ... 7.63237098e-02
  1.55762673e-03 1.55762673e-03]
 [1.42617224e-02 9.99509044e-01 2.37695373e-03 ... 2.73349679e-02
  1.18847687e-03 1.18847687e-03]
 [5.20201425e-03 9.99942740e-01 7.70668778e-04 ... 9.24802534e-03
  7.70668778e-04 3.85334389e-04]
 ...
 [9.90844644e-03 9.99927387e-01 6.19277903e-04 ... 6.81205693e-03
  2.06425968e-04 2.06425968e-04]
 [4.37470803e-03 9.99985344e-01 2.43039335e-04 ... 3.15951135e-03
  1.21519667e-04 2.43039335e-04]
 [2.43809775e-02 9.99620078e-01 2.16719800e-03 ... 1.24613885e-02
  5.41799500e-04 5.41799500e-04]]
```

```
import numpy as np
import pandas as pd
```

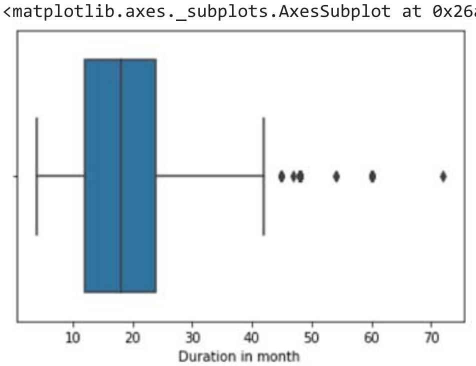
```
import matplotlib.pyplot as plt
import seaborn as sns


import warnings
warnings.filterwarnings('ignore')
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['Duration in month'])
plt.subplot(1,2,2)
sns.distplot(df['Duration in month'])
plt.show()
```

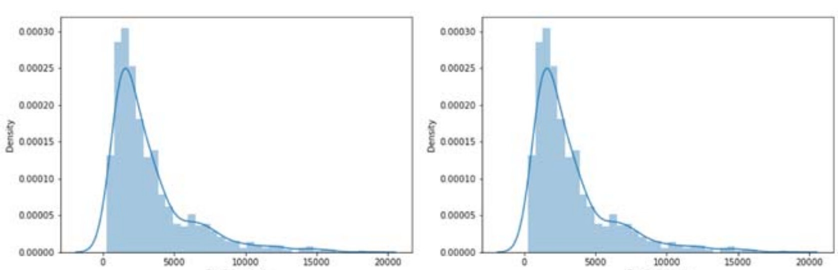



```
sns.boxplot(df['Duration in month'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x26ac4fabf10>
```

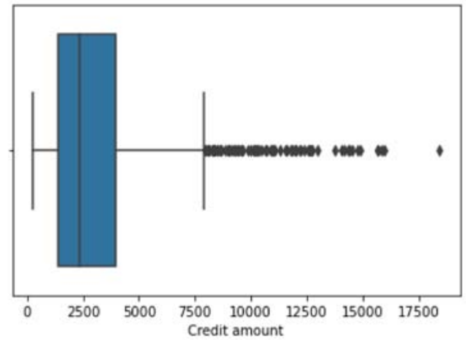



```
import warnings
warnings.filterwarnings('ignore')
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['Credit amount'])
plt.subplot(1,2,2)
sns.distplot(df['Credit amount'])
plt.show()
```

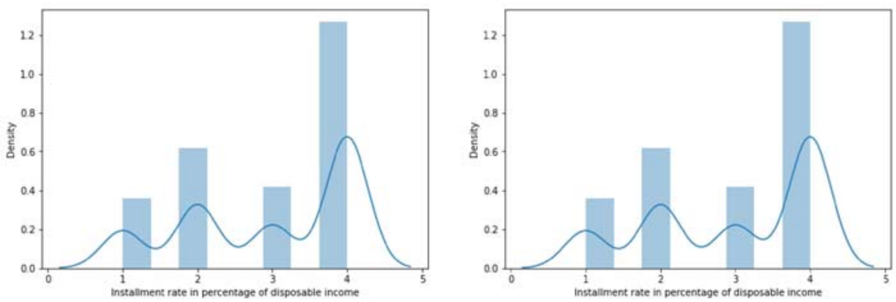


```
sns.boxplot(df['Credit amount'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x26ac530a820>
```

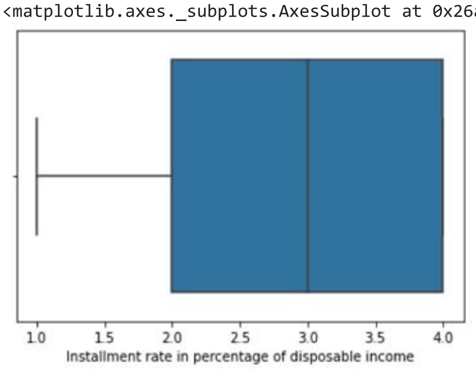



```
import warnings
warnings.filterwarnings('ignore')
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['Installment rate in percentage of disposable income'])
plt.subplot(1,2,2)
sns.distplot(df['Installment rate in percentage of disposable income'])
plt.show()
```

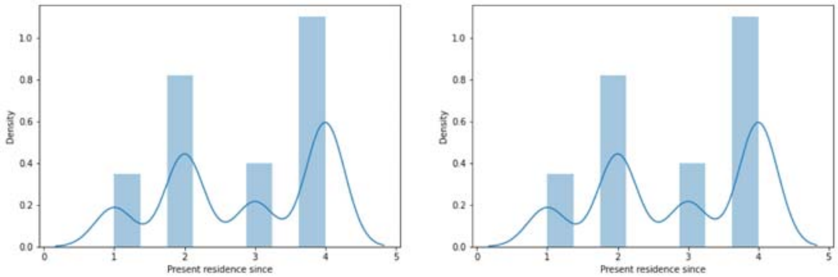



```
sns.boxplot(df['Installment rate in percentage of disposable income'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x26ac5459ee0>
```

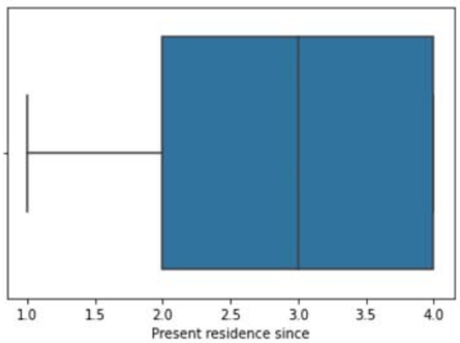



```
import warnings
warnings.filterwarnings('ignore')
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['Present residence since'])
plt.subplot(1,2,2)
sns.distplot(df['Present residence since'])
plt.show()
```

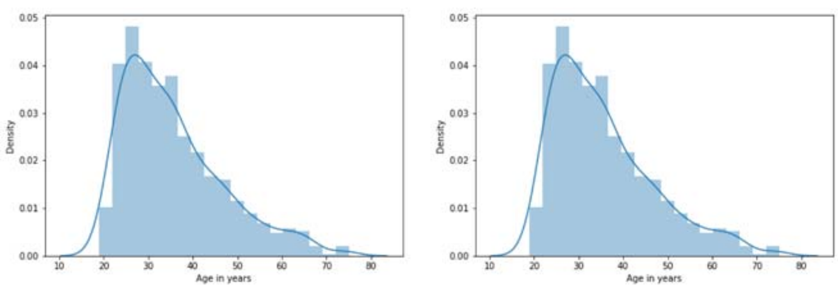



```
sns.boxplot(df['Present residence since'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x26ac6a38c40>

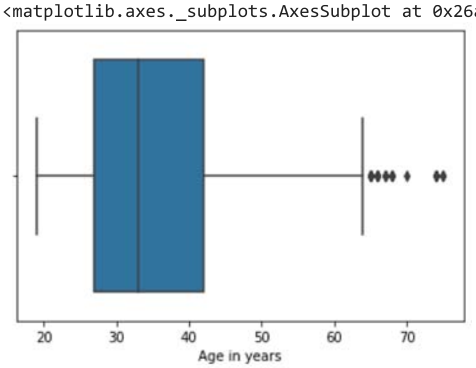



```
import warnings
warnings.filterwarnings('ignore')
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['Age in years'])
plt.subplot(1,2,2)
sns.distplot(df['Age in years'])
plt.show()
```

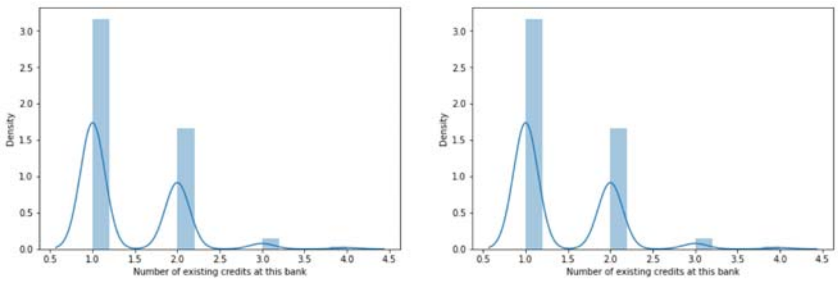



```
sns.boxplot(df['Age in years'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x26ac6d89df0>

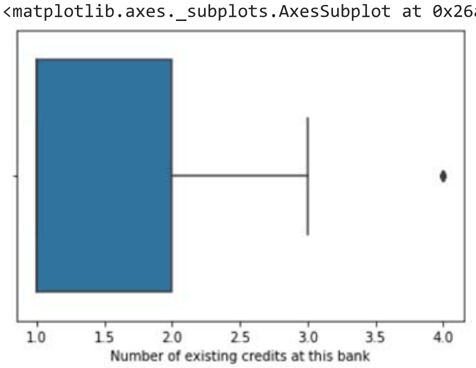


```
import warnings
warnings.filterwarnings('ignore')
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['Number of existing credits at this bank'])
plt.subplot(1,2,2)
sns.distplot(df['Number of existing credits at this bank'])
plt.show()
```




```
sns.boxplot(df['Number of existing credits at this bank'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x26ac6fb4850>
```

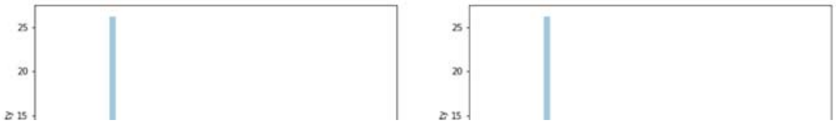



```
Number of people being liable to provide maintenance for
```

```
  File "<ipython-input-308-b2d39e46d0bf>", line 1
    Number of people being liable to provide maintenance for
           ^
SyntaxError: invalid syntax
```

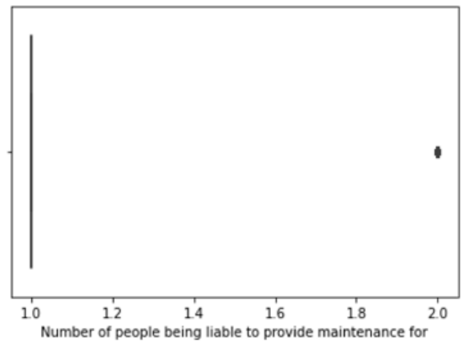
SEARCH STACK OVERFLOW

```
import warnings
warnings.filterwarnings('ignore')
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['Number of people being liable to provide maintenance for'])
plt.subplot(1,2,2)
sns.distplot(df['Number of people being liable to provide maintenance for'])
plt.show()
```



```
sns.boxplot(df['Number of people being liable to provide maintenance for'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x26ac72e4460>
```




```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                                                  Non-Null Count  Dtype 
---  ------                                                  --------------  ----- 
 0   Status of existing checking account                     1000 non-null   object
 1   Duration in month                                       1000 non-null   int64 
 2   Credit history                                          1000 non-null   object
 3   Purpose                                                 1000 non-null   object
 4   Credit amount                                           1000 non-null   int64 
 5   Savings account/bonds                                   1000 non-null   object
 6   Present employment since                                1000 non-null   object
 7   Installment rate in percentage of disposable income     1000 non-null   int64 
 8   Personal status and sex                                 1000 non-null   object
 9   Other debtors / guarantors                              1000 non-null   object
 10  Present residence since                                 1000 non-null   int64 
 11  Property                                                1000 non-null   object
 12  Age in years                                            1000 non-null   int64 
 13        Other installment plans                           1000 non-null   object
 14  Housing                                                 1000 non-null   object
 15  Number of existing credits at this bank                 1000 non-null   int64 
 16  Job                                                     1000 non-null   object
 17  Number of people being liable to provide maintenance for  1000 non-null   int64 
 18  Telephone                                               1000 non-null   object
 19  foreign worker                                          1000 non-null   object
 20  Cost Matrix                                             1000 non-null   int64 
dtypes: int64(8), object(13)
memory usage: 164.2+ KB
```

```
#Random Forests
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_normalised, y_res, test_size=0.2, random_state=0)
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
y_test.shape
y_pred.shape
```

```
(120,)
```

```
#Evaluating the Algorithm
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
```

```
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 0.04
Mean Squared Error: 0.04
Root Mean Squared Error: 0.2
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
```

```
forest = RandomForestClassifier()
forest.fit(X_train, y_train)
```

```
RandomForestClassifier()
```

```
# Make predictions for the test set
y_pred_test = forest.predict(X_test)
```

```
# View accuracy score
accuracy_score(y_test, y_pred_test)
```

```
0.96
```

```
# View confusion matrix for test data and predictions
confusion_matrix(y_test, y_pred_test)
```

```
array([[13,  0],
       [ 1, 11]], dtype=int64)
```

```
# View the classification report for test data and predictions
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.93      1.00      0.96        13
           1       1.00      0.92      0.96        12

    accuracy                           0.96        25
   macro avg       0.96      0.96      0.96        25
weighted avg       0.96      0.96      0.96        25
```

```
# Random forest
```

```
#Interpretation

#Due to more number of trees and with each tree having different random hypermparamater the random forest classifier generalize the training

#Due to more randomisation in the selectio of data we get the accuracy for the validation set more and hence reduces teh overfitting.

#We can also reduces teh overfitting doing some more changes in teh number of hyper-paramaters

#Tree classifier accuracy : Training set : 100 % | validation =0.27%

#Randome Forest accuracy : Training set : 97.55 % | validation =86.67%
```

```
# Normalize Data
```

```
# import scoring methods
import numpy as np
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
from sklearn.model_selection import cross_val_score
```

```
# our data for training (used '_train' just for improving readability)
X_train, y_train=X,y
```

```
X_train.shape, y_train.shape
```

```
((100, 20), (100,))
```

```
# a dictionary for keeping all scores of the classifiers
trainScores={}
```

```
#Classification
#Now, it is your turn, use the training set to build an accurate model. Then use the test set to report the accuracy of the model You should

#K Nearest Neighbor(KNN)
#Decision Tree
#Support Vector Machine
#Logistic Regression
```

```
#K Nearest Neighbor(KNN)
```

```
from sklearn.neighbors import KNeighborsClassifier

bestScore=0.0
accList=[]

for k in range(3,12):

    clf_knn = KNeighborsClassifier(n_neighbors=k,algorithm='auto')

    # using 10 fold cross validation for scoring the classifier's accuracy
    scores = cross_val_score(clf_knn, X, y, cv=10)
    score=scores.mean()
    accList.append(score)

    if score > bestScore:
        bestScore=score
        best_clf=clf_knn
        bestK=k

print("Best K is :",bestK,"| Cross validation Accuracy :",bestScore)
clf_knn=best_clf
```

```
Best K is : 8 | Cross validation Accuracy : 0.8800000000000001
```

```
clf_knn.fit(X_train,y_train)
y_pred=best_clf.predict(X_train)
```

```
!pip install scikit-learn==0.24
from sklearn.metrics import jaccard_score
trainScores['KNN-jaccard']=jaccard_score(y_train, y_pred)
trainScores['KNN-f1-score']=f1_score(y_train, y_pred, average='weighted')
```

```
Requirement already satisfied: scikit-learn==0.24 in c:\users\mohsen\anaconda3\lib\site-packages (0.24.0)
Requirement already satisfied: joblib>=0.11 in c:\users\mohsen\anaconda3\lib\site-packages (from scikit-learn==0.24) (0.16.0)
Requirement already satisfied: scipy>=0.19.1 in c:\users\mohsen\anaconda3\lib\site-packages (from scikit-learn==0.24) (1.5.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\mohsen\anaconda3\lib\site-packages (from scikit-learn==0.24) (2.1.0)
Requirement already satisfied: numpy>=1.13.3 in c:\users\mohsen\anaconda3\lib\site-packages (from scikit-learn==0.24) (1.18.5)
```

```
trainScores
```

```
{'KNN-jaccard': 0.8367346938775511, 'KNN-f1-score': 0.9193535353535354}
```

```
plt.plot(range(3,12),accList)
plt.xlabel('K')
plt.ylabel('CV Accuracy')
plt.show()
```


```
#Decision Tree
```

```
from sklearn import tree

clf_tree = tree.DecisionTreeClassifier()
clf_tree = clf_tree.fit(X_train, y_train)

y_pred=clf_tree.predict(X_train)
```

```
trainScores['Tree-jaccard']=jaccard_score(y_train, y_pred)
trainScores['Tree-f1-score']=f1_score(y_train, y_pred, average='weighted')
```

```
trainScores
```

```
{}
```

```
# Accuracy
clf_tree.score(X_test, y_test)
```

```
1.0
```

```
!pip install graphviz
!pip install pydotplus
import graphviz
import pydotplus

dot_data = tree.export_graphviz(clf_tree, out_file=None,
                     feature_names=['Status of existing checking account','Duration in month','Credit history '

                                    ,'Purpose ','Credit amount','Savings account/bonds ','Present employment since ',
                                    'Installment rate in percentage of disposable income ','Personal status and sex',
                                    'Other debtors / guarantors','Present residence since','Property ','Age in years',
                                    'Other installment plans','Housing','Number of existing credits at this bank',
                                    'Job','Number of people being liable to provide maintenance for','Telephone',
                                    'foreign worker'

                                   ],
                     class_names='Cost Matrix',
                     filled=True, rounded=True,
                     special_characters=True)

graph = pydotplus.graph_from_dot_data(dot_data)
graph.set_size('"8,8!"')
gvz_graph = graphviz.Source(graph.to_string())

gvz_graph
```

```
Requirement already satisfied: graphviz in c:\users\mohsen\anaconda3\lib\site-pack
Requirement already satisfied: pydotplus in c:\users\mohsen\anaconda3\lib\site-pac
Requirement already satisfied: pyparsing>=2.0.1 in c:\users\mohsen\anaconda3\lib\s
```

```
#Support Vector Machine
```

```
y_train=y_train.astype(float)
```

```
from sklearn import svm

clf_svm = svm.LinearSVC(random_state=7)
clf_svm.fit(X_train, y_train)

y_pred=clf_svm.predict(X_train)
```

```
trainScores['SVM-jaccard']=jaccard_score(y_train, y_pred)
trainScores['SVM-f1-score']=f1_score(y_train, y_pred, average='weighted')
```

```
trainScores
```

```
{'KNN-jaccard': 0.8367346938775511,
 'KNN-f1-score': 0.9193535353535354,
 'Tree-jaccard': 1.0,
 'Tree-f1-score': 1.0,
 'SVM-jaccard': 1.0,
 'SVM-f1-score': 1.0}
```

```
#Logistic Regression
```

```
from sklearn.linear_model import LogisticRegression

clf_log = LogisticRegression(random_state=0, solver='lbfgs',
                        multi_class='multinomial')
clf_log.fit(X_train, y_train)

y_pred=clf_log.predict(X_train)
y_proba=clf_log.predict_proba(X_train)
```

```
trainScores['LogReg-jaccard']=jaccard_score(y_train, y_pred)
trainScores['LogReg-f1-score']=f1_score(y_train, y_pred, average='weighted')
trainScores['LogReg-logLoss']=log_loss(y_train, y_proba)
```

```
trainScores
```

```
{}
```

```
#Load Test set for evaluation
```

```
from sklearn import preprocessing
testy=y.astype(float)
testX= preprocessing.StandardScaler().fit_transform(X)
```

```
testScores={}
```

```
from sklearn.neighbors import KNeighborsClassifier
knn_pred=clf_knn.predict(testX)
testScores['KNN-jaccard']=jaccard_score(testy, knn_pred)
testScores['KNN-f1-score']=f1_score(testy, knn_pred, average='weighted')
```

```
tree_pred=clf_tree.predict(testX)
testScores['Tree-jaccard']=jaccard_score(testy, tree_pred)
testScores['Tree-f1-score']=f1_score(testy, tree_pred, average='weighted')
```

```
svm_pred=clf_svm.predict(testX)
testScores['SVM-jaccard']=jaccard_score(testy, svm_pred)
testScores['SVM-f1-score']=f1_score(testy, svm_pred, average='weighted')
```

```
log_pred=clf_log.predict(testX)
proba=clf_log.predict_proba(testX)
testScores['LogReg-jaccard']=jaccard_score(testy, log_pred)
testScores['LogReg-f1-score']=f1_score(testy, log_pred, average='weighted')
testScores['LogReg-logLoss']=log_loss(testy, proba)
```

```
trainScores
```

```
{'KNN-jaccard': 0.8367346938775511,
 'KNN-f1-score': 0.9193535353535354,
 'Tree-jaccard': 1.0,
 'Tree-f1-score': 1.0,
 'SVM-jaccard': 1.0,
 'SVM-f1-score': 1.0,
 'LogReg-jaccard': 0.98,
 'LogReg-f1-score': 0.9900010001000099,
 'LogReg-logLoss': 0.056382280983385755}
```

```
#Report
#You should be able to report the accuracy of the built model using different evaluation metrics:
# Table
```

```
Algorithm   Jaccard F1-score    LogLoss
KNN ?   ?   NA
Decision Tree   ?   ?   NA
SVM ?   ?   NA
LogisticRegression  ?   ?   ?
```

```
  File "<ipython-input-409-b374babb37b5>", line 4
    Algorithm   Jaccard F1-score        LogLoss
                ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

```
#the Naive Bayes Algorithm
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

```
#Feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
#Training the Naive Bayes model on the training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

```
GaussianNB()
```

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

```
 y_pred
```

```
array([1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 1])
```

```
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test,y_pred)
```

```
# Acuracy
ac
```

```
0.96
```

```
cm
```

```
array([[13,  0],
       [ 1, 11]], dtype=int64)
```

```
# Neural Network
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
X, y = make_classification(n_samples=100, random_state=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
                                                    random_state=1)
```

```
clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
```

```
clf.predict_proba(X_test[:1])
```

```
array([[0.03838405, 0.96161595]])
```

```
clf.predict(X_test[:10, :])
```

```
array([1, 0, 1, 0, 1, 0, 0, 1, 0, 0])
```

```
# Accuracy
clf.score(X_test, y_test)
```

```
0.88
```

```
# Evaluate Gradient Boosting Models with XGBoost
```

```
# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=7)
```

```
# train-test split evaluation of xgboost model
from numpy import loadtxt
!pip install xgboost
import xgboost
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# load data
dataset = df

# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=7)
# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)
# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Requirement already satisfied: xgboost in c:\users\mohsen\anaconda3\lib\site-packages (1.5.0)
Requirement already satisfied: scipy in c:\users\mohsen\anaconda3\lib\site-packages (from xgboost) (1.5.0)
Requirement already satisfied: numpy in c:\users\mohsen\anaconda3\lib\site-packages (from xgboost) (1.18.5)
[23:22:17] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting in XGBoost 1.3.0, the def
C:\Users\Mohsen\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier is deprecate
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
Accuracy: 93.94%
```