BANK CREDIT PREDICTION USING SOUTH GERMAN CREDIT DATA

Loading Data and Libraries

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,ConfusionMatrixDisplay,classification_report
df=pd.read_csv('/content/drive/MyDrive/Pandas dataset/german_credit.csv')
df
```

	status	duration	credit_history	purpose	amount	savin
0	no checking account	18	all credits at this bank paid back duly	car (used)	1049	unknown/ savin accou
1	no checking account	9	all credits at this bank paid back duly	others	2799	unknown/ savin accou
2	... < 0 DM	12	no credits taken/all credits paid back duly	retraining	841	... < 1 [
3	no checking account	12	all credits at this bank paid back duly	others	2122	unknown/ savin accou
4	no checking account	12	all credits at this bank paid back duly	others	2171	unknown/ savin accou
...	...	...	...	...	...	
995	no checking account	24	no credits taken/all credits paid back duly	furniture/equipment	1987	unknown/ savin accou
996	no checking account	24	no credits taken/all credits paid back duly	others	2303	unknown/ savin accou
997	... >= 200 DM / salary for at least 1 year	21	all credits at this bank paid back duly	others	12680	... >= 10 [
998	... < 0 DM	12	no credits taken/all credits paid back duly	furniture/equipment	6468	... >= 10 [
	no		no credits			>= 10

```
df.dtypes
```

```
status                   object
duration                  int64
credit_history           object
purpose                  object
amount                    int64
savings                  object
employment_duration      object
installment_rate         object
personal_status_sex      object
other_debtors            object
```

```
present_residence          object
property                   object
age                         int64
other_installment_plans    object
housing                    object
number_credits             object
job                        object
people_liable              object
telephone                  object
foreign_worker             object
credit_risk                object
dtype: object
```

```
df.shape
```

```
(1000, 21)
```

```
df['credit_history'].value_counts()
```

```
no credits taken/all credits paid back duly    530
all credits at this bank paid back duly        293
existing credits paid back duly till now        88
critical account/other credits elsewhere        49
delay in paying off in the past                 40
Name: credit_history, dtype: int64
```

```
df['purpose'].value_counts()
```

```
furniture/equipment    280
others                 234
car (used)             181
car (new)              103
retraining              97
repairs                 50
domestic appliances     22
business                12
radio/television        12
vacation                 9
Name: purpose, dtype: int64
```

```
df['employment_duration'].value_counts()
```

```
1 <= ... < 4 yrs    339
>= 7 yrs            253
4 <= ... < 7 yrs    174
< 1 yr              172
unemployed           62
Name: employment_duration, dtype: int64
```

```
df['personal_status_sex'].value_counts()
```

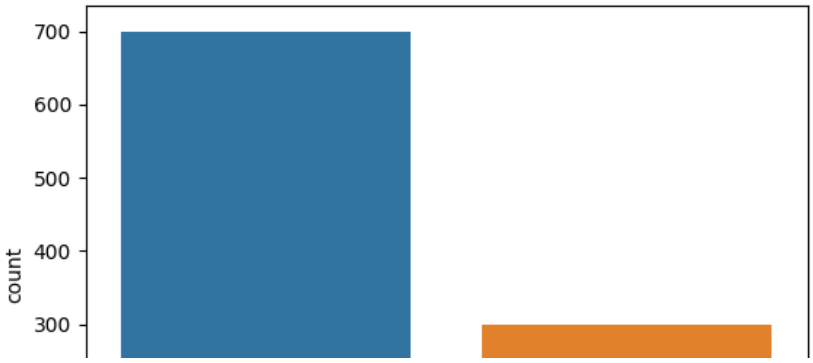
```
male : married/widowed                  548
female : non-single or male : single    310
female : single                          92
male : divorced/separated                50
Name: personal_status_sex, dtype: int64
```

```
df['credit_risk'].value_counts()
```

```
good    700
bad     300
Name: credit_risk, dtype: int64
```

```
sns.countplot(x=df['credit_risk'])
```

```
<Axes: xlabel='credit_risk', ylabel='count'>
```



DATA CLEANING AND DATA PREPROCESSING

1. Check for missing values
2. Handeling Catogerical Values

good bad

```
df.isna().sum()
```

```
status                    0
duration                  0
credit_history            0
purpose                   0
amount                    0
savings                   0
employment_duration       0
installment_rate          0
personal_status_sex       0
other_debtors             0
present_residence         0
property                  0
age                       0
other_installment_plans   0
housing                   0
number_credits            0
job                       0
people_liable             0
telephone                 0
foreign_worker            0
credit_risk               0
dtype: int64
```

```
le=LabelEncoder()
df['status']=le.fit_transform(df['status'])
df['credit_history']=le.fit_transform(df['credit_history'])
df['purpose']=le.fit_transform(df['purpose'])
df['savings']=le.fit_transform(df['savings'])
df['employment_duration']=le.fit_transform(df['employment_duration'])
df['installment_rate']=le.fit_transform(df['installment_rate'])
df['personal_status_sex']=le.fit_transform(df['personal_status_sex'])
df['other_debtors']=le.fit_transform(df['other_debtors'])
df['present_residence']=le.fit_transform(df['present_residence'])
df['property']=le.fit_transform(df['property'])
df['other_installment_plans']=le.fit_transform(df['other_installment_plans'])
df['housing']=le.fit_transform(df['housing'])
df['number_credits']=le.fit_transform(df['number_credits'])
df['job']=le.fit_transform(df['job'])
df['people_liable']=le.fit_transform(df['people_liable'])
df['telephone']=le.fit_transform(df['telephone'])
df['foreign_worker']=le.fit_transform(df['foreign_worker'])
df['credit_risk']=le.fit_transform(df['credit_risk'])
```

```
df
```

	status	duration	credit_history	purpose	amount	savings	employmen
0	3	18	0	2	1049	4	
1	3	9	0	5	2799	4	
2	0	12	4	8	841	0	
3	3	12	0	5	2122	4	
4	3	12	0	5	2171	4	
...	...	...	...	...	...	...	
995	3	24	4	4	1987	4	
996	3	24	4	5	2303	4	
997	1	21	0	5	12680	1	
998	0	12	4	4	6468	1	

```
df.describe()
```

	status	duration	credit_history	purpose	amount
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	1.342000	20.903000	2.513000	4.100000	3271.24800
std	1.145578	12.058814	1.780169	2.090064	2822.75176
min	0.000000	4.000000	0.000000	0.000000	250.00000
25%	0.000000	12.000000	0.000000	2.000000	1365.50000
50%	1.000000	18.000000	4.000000	4.000000	2319.50000
75%	3.000000	24.000000	4.000000	5.000000	3972.25000
max	3.000000	72.000000	4.000000	9.000000	18424.00000

```
df.loc[df['duration']>50]
```

	status	duration	credit_history	purpose	amount	savings	employmen
241	1	60	4	5	10366	4	
309	1	54	2	1	9436	1	
382	1	60	0	5	13756	1	
455	0	60	3	4	7418	1	
562	0	60	4	5	7408	0	
690	1	60	4	4	10144	0	
695	1	60	3	4	15653	4	
719	0	72	4	4	5595	0	
729	0	60	3	4	9157	1	
731	1	60	4	5	6527	1	
763	0	60	4	7	6288	4	
807	3	60	3	8	6836	4	
811	3	60	4	8	7297	4	
847	0	60	1	0	14782	0	
890	0	60	4	5	14027	4	

```
df.columns
```

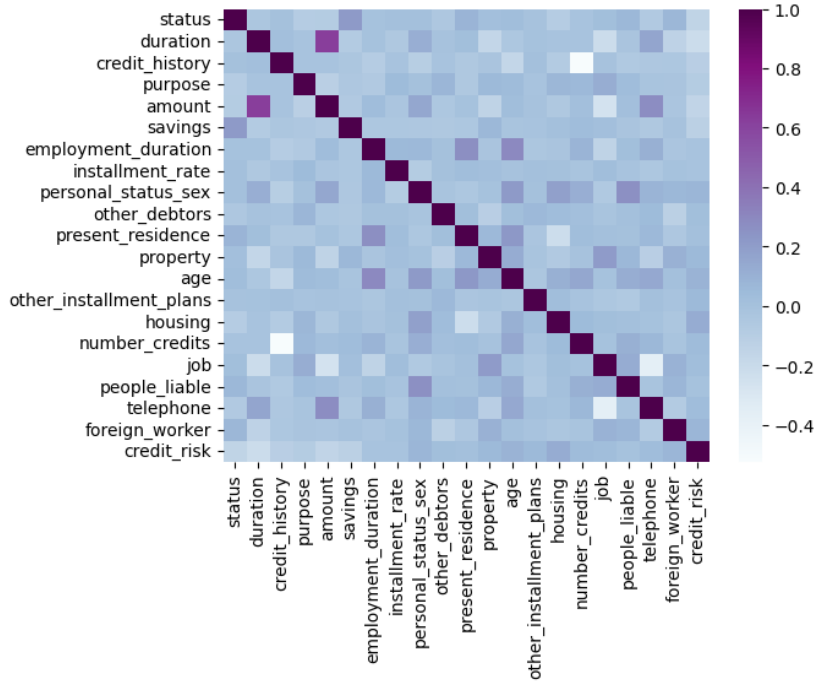
```
Index(['status', 'duration', 'credit_history', 'purpose', 'amount', 'savings',
       'employment_duration', 'installment_rate', 'personal_status_sex',
       'other_debtors', 'present_residence', 'property', 'age',
       'other_installment_plans', 'housing', 'number_credits', 'job',
       'people_liable', 'telephone', 'foreign_worker', 'credit_risk'],
      dtype='object')
```

DATA VISUALIZATION

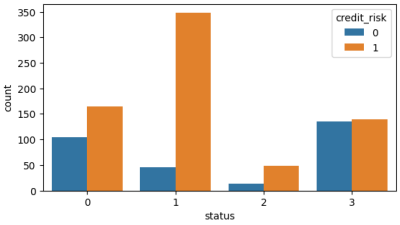
```
feauture=['status', 'duration', 'credit_history', 'purpose', 'amount', 'savings',
       'employment_duration', 'installment_rate', 'personal_status_sex',
       'other_debtors', 'present_residence', 'property', 'age',
       'other_installment_plans', 'housing', 'number_credits', 'job',
       'people_liable', 'telephone', 'foreign_worker']
```

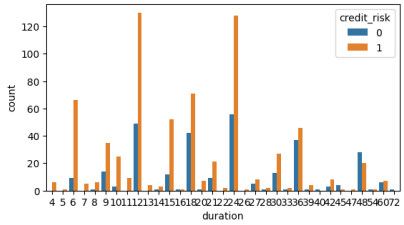
```
sns.heatmap(df.corr(),cmap='BuPu')
```

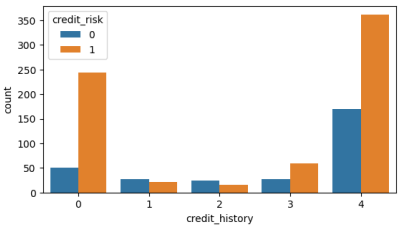
```
<Axes: >
```

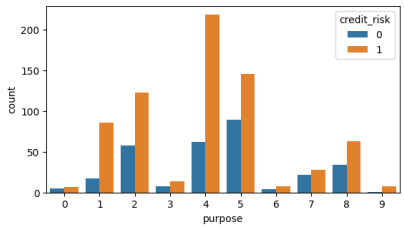


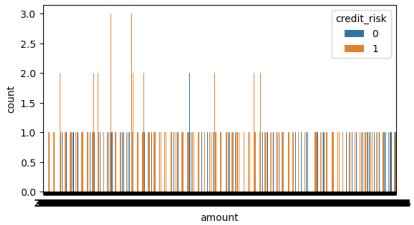

```
plt.figure(figsize=(14,40))
for i,col in enumerate(feauture):
  plt.subplot(10,2,i+1)
  sns.countplot(x=df[col],hue='credit_risk',data=df)
  plt.xlabel(col)
```

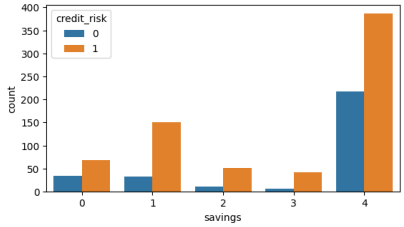

credit_risk
0
1
count
status

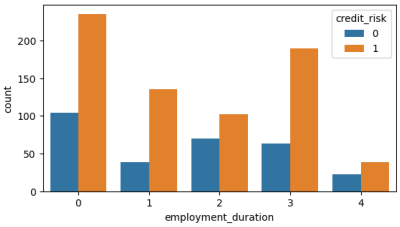

credit_risk
0
1
count
duration

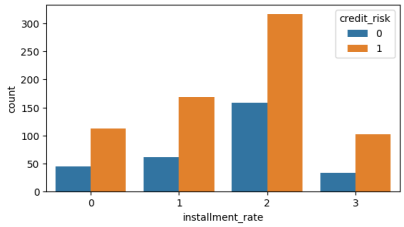

credit_risk
0
1
count
credit_history

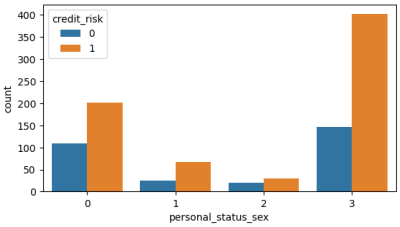

credit_risk
0
1
count
purpose

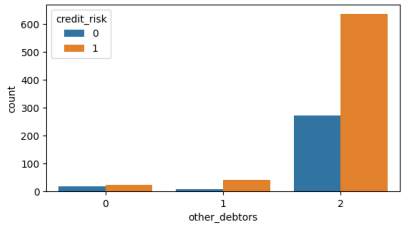

credit_risk
0
1
count
amount

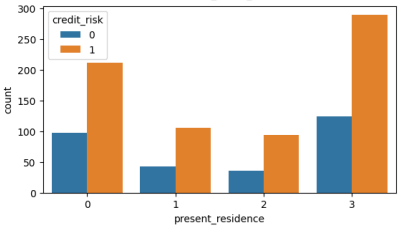

credit_risk
0
1
count
savings

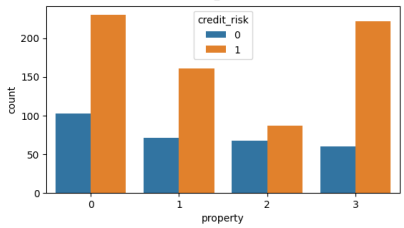

credit_risk
0
1
count
employment_duration

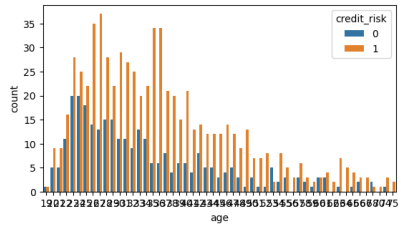

credit_risk
0
1
count
installment_rate

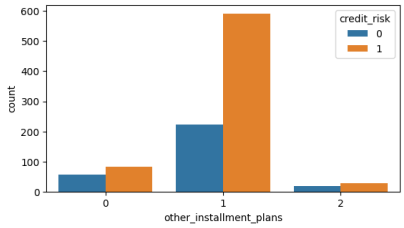

credit_risk
0
1
count
personal_status_sex

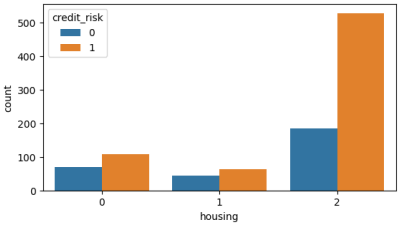

credit_risk
0
1
count
other_debtors

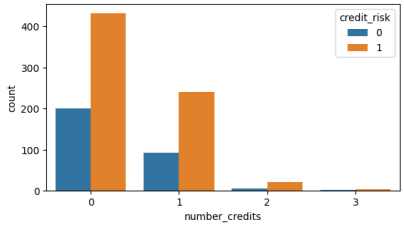

credit_risk
0
1
count
present_residence

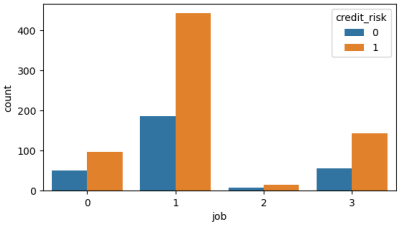

credit_risk
0
1
count
property

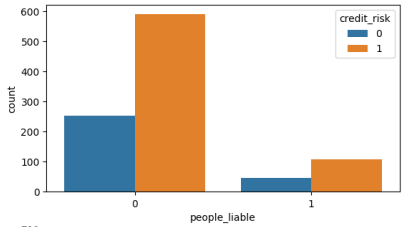

credit_risk
0
1
count
age


credit_risk
0
1
count
other_installment_plans


credit_risk
0
1
count
housing

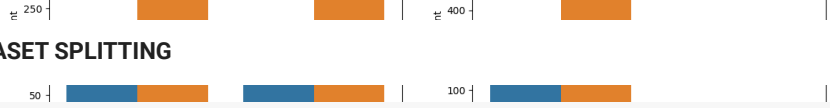

credit_risk
0
1
count
number_credits


credit_risk
0
1
count
job


credit_risk
0
1
count
people_liable


credit_risk
0
1


credit_risk
0
1



DATASET SPLITTING


```
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

```
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)
```

```
print(X_train.shape,X_test.shape)
```

```
(700, 20) (300, 20)
```

```
#Scaling the Dataset using StanderdScaler

from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)

X_train
```

```
array([[-0.28096006,  3.2566844 ,  0.85693866, ...,  2.36722909,
         1.22110737, -0.20412415],
       [ 1.48291475,  2.26198769,  0.85693866, ..., -0.42243482,
         1.22110737, -0.20412415],
       [ 1.48291475,  1.26729098, -1.39393211, ..., -0.42243482,
         1.22110737, -0.20412415],
       ...,
       [ 1.48291475,  0.27259427,  0.29422096, ...,  2.36722909,
        -0.81892881, -0.20412415],
       [ 1.48291475,  0.27259427,  0.85693866, ..., -0.42243482,
        -0.81892881, -0.20412415],
       [-0.28096006, -0.72210245, -1.39393211, ..., -0.42243482,
        -0.81892881, -0.20412415]])
```

MODEL CREATION

- KNN
- SVM
- Naive Bayes
- Random Forest
- AdaBoost Classifier

```
#1 KNN


knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train,y_train)
y_pred=knn.predict(X_test)
y_pred
```

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

```
print('Accuracy score of KNN:')
knn_score=accuracy_score(y_test,y_pred)*100
knn_score
```

```
Accuracy score of KNN:
69.0
```

```
print(classification_report(y_test,y_pred))
```

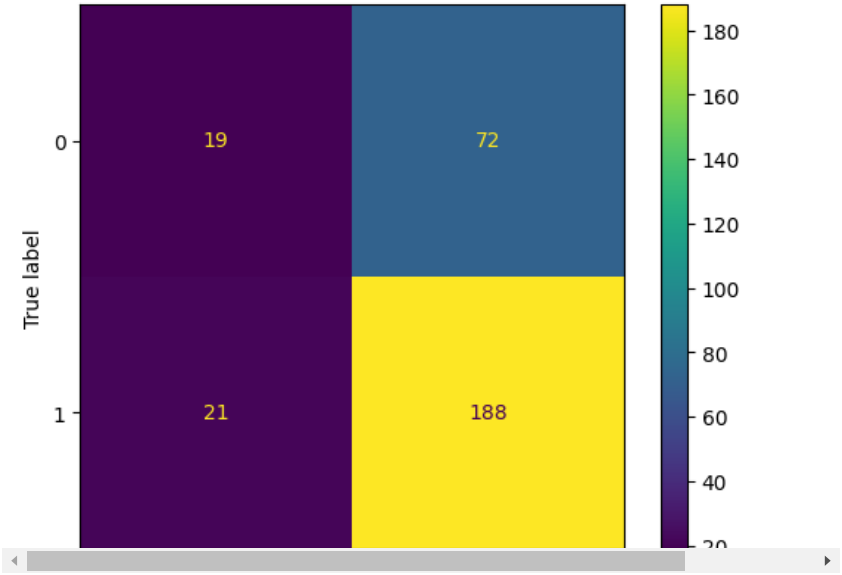
```
                precision    recall  f1-score   support

             0       0.47      0.21      0.29        91
             1       0.72      0.90      0.80       209

      accuracy                           0.69       300
     macro avg       0.60      0.55      0.55       300
  weighted avg       0.65      0.69      0.65       300
```

```
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0
```




```
#2 SVM

sv=SVC()
sv.fit(X_train,y_train)
y_pred=sv.predict(X_test)
y_pred
```

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

```
#accuracy score
print('Accuracy score of SVM:')
sv_score=accuracy_score(y_test,y_pred)*100
print(sv_score)
```

```
Accuracy score of SVM:
70.66666666666667
```

```
print(classification_report(y_test,y_pred))
```

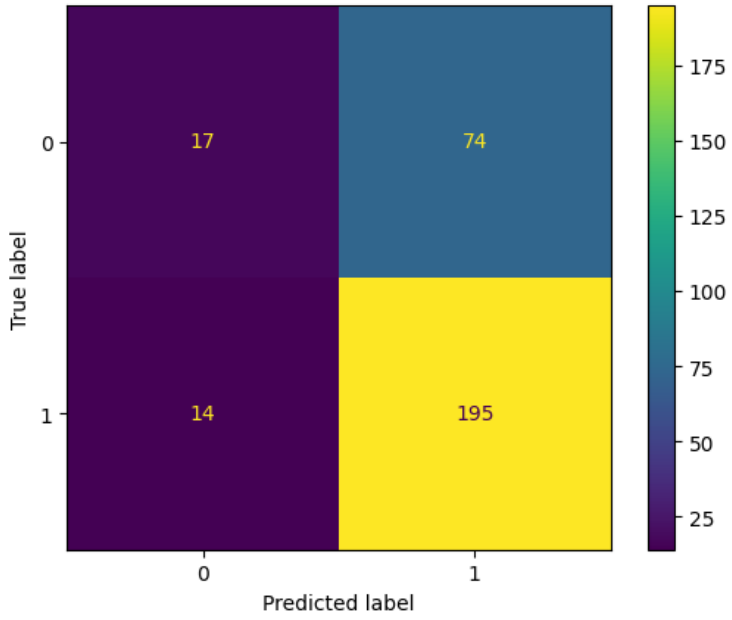
```
                precision    recall  f1-score   support

           0       0.55      0.19      0.28        91
           1       0.72      0.93      0.82       209

    accuracy                           0.71       300
   macro avg       0.64      0.56      0.55       300
weighted avg       0.67      0.71      0.65       300
```

```
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f4a947c5b20>
```




```
#3 Random Forest Classifier
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
y_pred=rf.predict(X_test)
y_pred
```

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

```
rf_score=accuracy_score(y_test,y_pred)*100
print('Accuracy of Random Forest :',rf_score)
```

```
Accuracy of Random Forest : 75.0
```

```
print(classification_report(y_test,y_pred))
```

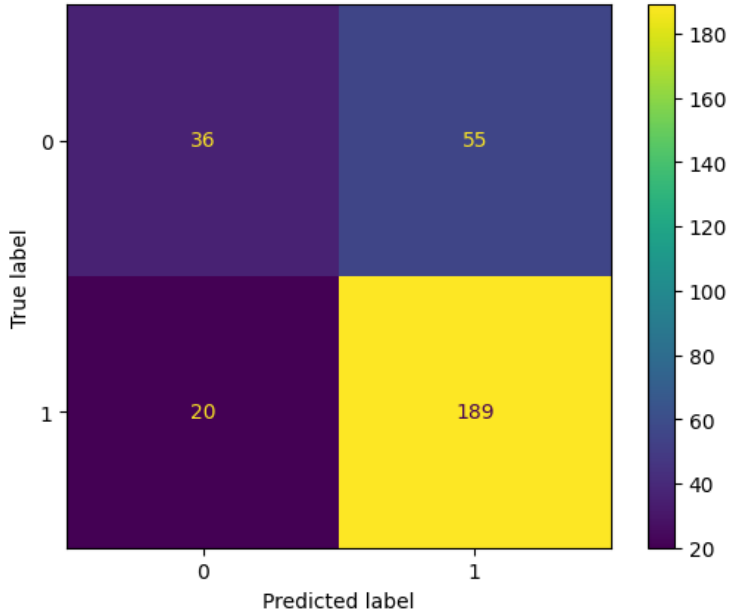
```
                precision    recall  f1-score   support

           0       0.64      0.40      0.49        91
           1       0.77      0.90      0.83       209

    accuracy                           0.75       300
   macro avg       0.71      0.65      0.66       300
weighted avg       0.73      0.75      0.73       300
```

```
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f4a9cb8fc40>
```




```
#Ada Boost Classifier
ab=AdaBoostClassifier(n_estimators=50,random_state=2)
ab.fit(X_train,y_train)
y_pred=ab.predict(X_test)
y_pred
```

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

```
ada_score=accuracy_score(y_test,y_pred)*100
print('Accuracy of AdaBoost :',ada_score)
```

```
Accuracy of AdaBoost : 74.66666666666667
```

```
#  Decisio tree Classifier
dt=DecisionTreeClassifier(criterion='entropy')
dt.fit(X_train,y_train)
y_pred=dt.predict(X_test)
dt_score=accuracy_score(y_test,y_pred)*100
print('Accuracy of Decision Tree:',dt_score)
```

```
Accuracy of Decision Tree: 68.33333333333333
```

COMPARING ACCURACY

```
alg=['KNN','SVM','Random Forest','AdaBoost','Decision Tree']
acc=[knn_score,sv_score,rf_score,ada_score,dt_score]
Accuracy=pd.DataFrame({'Algorithms':alg,'Accuracy':acc})
Accuracy
```

	Algorithms	Accuracy
0	KNN	69.000000
1	SVM	70.666667

IMBALANCE DATA HANDELLING

```
from imblearn.over_sampling import SMOTE
oversampling=SMOTE(random_state=3)
X_os,y_os=oversampling.fit_resample(X,y)
```

```
from collections import Counter
print(Counter(y_os))
```

```
Counter({1: 700, 0: 700})
```

```
X_os.shape
```

```
(1400, 20)
```

```
from sklearn.model_selection import train_test_split
X_trainos,X_testos,y_trainos,y_testos=train_test_split(X_os,y_os,test_size=0.3,random_state=2)
```

```
#KNN
knn.fit(X_trainos,y_trainos)
y_predos=knn.predict(X_testos)
knn_os=accuracy_score(y_testos,y_predos)*100
print('Accuracy of KNN after Over Sampling')
knn_os
```

```
Accuracy of KNN after Over Sampling
65.0
```

```
#SVM
sv.fit(X_trainos,y_trainos)
y_predos=sv.predict(X_testos)
sv_os=accuracy_score(y_testos,y_predos)*100
print('Accuracy of SVM after Over Sampling')
sv_os
```

```
Accuracy of SVM after Over Sampling
56.42857142857143
```

```
#Random Forest
rf.fit(X_trainos,y_trainos)
y_predos=rf.predict(X_testos)
rf_os=accuracy_score(y_testos,y_predos)*100
print('Accuracy of Random Forest after Over Sampling')
rf_os
```

```
Accuracy of Random Forest after Over Sampling
81.42857142857143
```

```
#AdaBoost
ab.fit(X_trainos,y_trainos)
y_predos=ab.predict(X_testos)
ab_os=accuracy_score(y_testos,y_predos)*100
print('Accuracy of AdaBoost after Over Sampling')
ab_os
```

```
Accuracy of AdaBoost after Over Sampling
75.95238095238095
```

```
#DecisionTree
dt.fit(X_trainos,y_trainos)
y_predos=dt.predict(X_testos)
dt_os=accuracy_score(y_testos,y_predos)*100
print('Accuracy of DecisionTree after Over Sampling')
dt_os
```

```
Accuracy of DecisionTree after Over Sampling
73.09523809523809
```

Principal component analysis (PCA).

```
from sklearn.decomposition import PCA
pca=PCA(n_components=15)
X_trainpca=pca.fit_transform(X_train)
X_testpca=pca.transform(X_test)
X_trainpca
```

```
array([[ 2.64646301, -0.34454832, -0.13110059, ..., -0.53446959,
         0.42516331,  0.38108122],
       [ 2.54464398, -0.28807274,  0.74018572, ...,  0.0987077 ,
         0.08189235, -0.54567621],
       [ 0.81246378,  0.35726454, -0.40983205, ...,  0.88383446,
         1.69228246,  0.17852579],
       ...,
       [-0.54290257,  2.1584127 ,  0.86213866, ..., -0.79305746,
        -1.15201708, -1.58067082],
       [-0.0075846 , -0.18304145,  1.52476331, ...,  0.19948575,
         0.52220862,  0.67513053],
       [-0.75110975,  1.0744922 , -0.92298996, ...,  0.32330432,
        -1.10088493, -1.45359515]])
```

```
#knn
knn.fit(X_trainpca,y_train)
y_predpca=knn.predict(X_testpca)
print('Accuracy knn after  PCA :')
knn_pca=accuracy_score(y_test,y_predpca)*100
knn_pca
```

```
Accuracy knn after  PCA :
71.33333333333334
```

```
#SVM
sv.fit(X_trainpca,y_train)
y_predpca=sv.predict(X_testpca)
print('Accuracy svm after  PCA :')
sv_pca=accuracy_score(y_test,y_predpca)*100
sv_pca
```

```
Accuracy svm after  PCA :
71.33333333333334
```

```
#AdABoost
ab.fit(X_trainpca,y_train)
y_predpca=ab.predict(X_testpca)
print('Accuracy Adaboost after  PCA :')
ab_pca=accuracy_score(y_test,y_predpca)*100
ab_pca
```

```
Accuracy Adaboost after  PCA :
68.0
```

```
#Random Forest
rf.fit(X_trainpca,y_train)
y_predpca=rf.predict(X_testpca)
print('Accuracy RandomForest after  PCA :')
rf_pca=accuracy_score(y_test,y_predpca)*100
rf_pca
```

```
Accuracy RandomForest after  PCA :
72.0
```

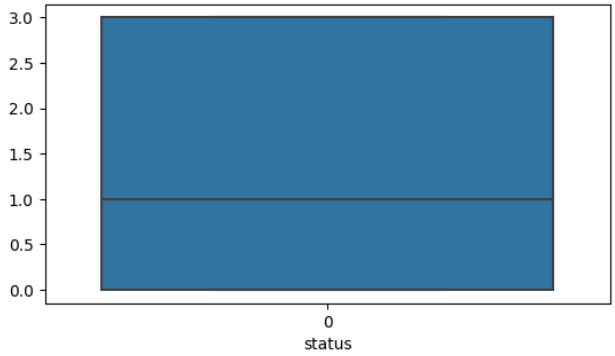
```
#Decision Tree
dt.fit(X_trainpca,y_train)
y_predpca=dt.predict(X_testpca)
print('Accuracy Decision Tree after  PCA :')
dt_pca=accuracy_score(y_test,y_predpca)*100
dt_pca
```

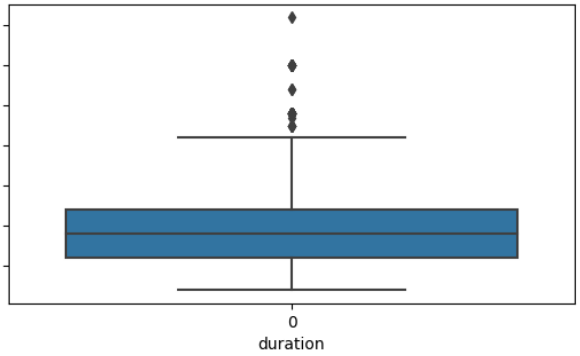
```
Accuracy Decision Tree after  PCA :
62.66666666666667
```

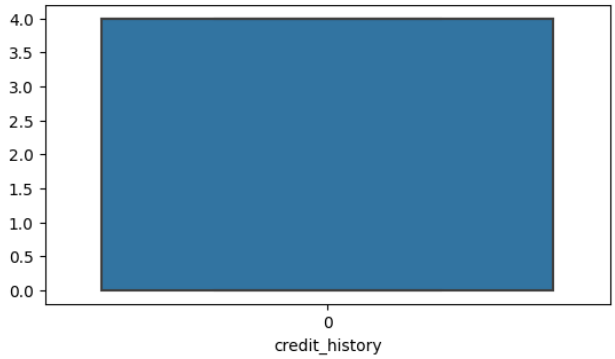
Removal OF Outlier

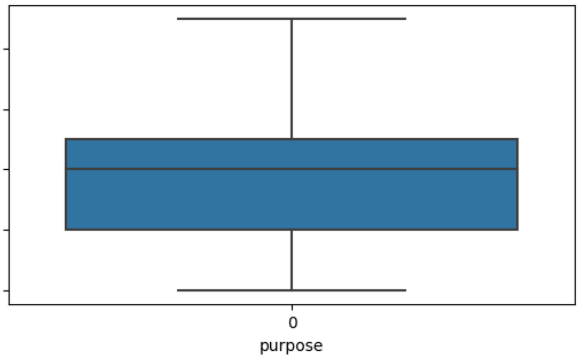
Check for Outlier

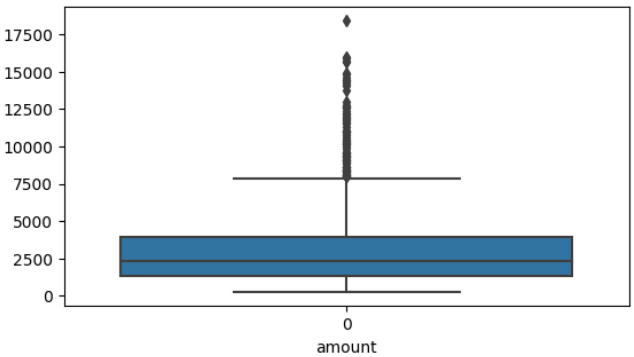
```
plt.figure(figsize=(14,40))
for i,col in enumerate(feauture):
  plt.subplot(10,2,i+1)
  sns.boxplot(df[col])
  plt.xlabel(col)
```

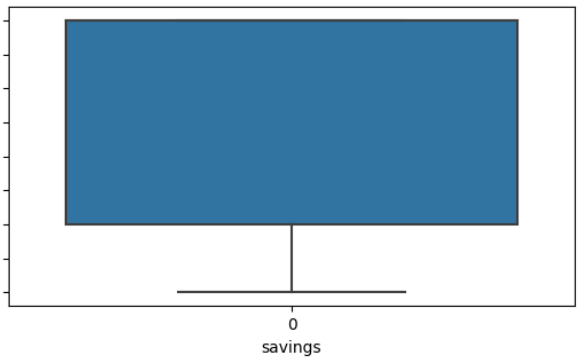

3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
status

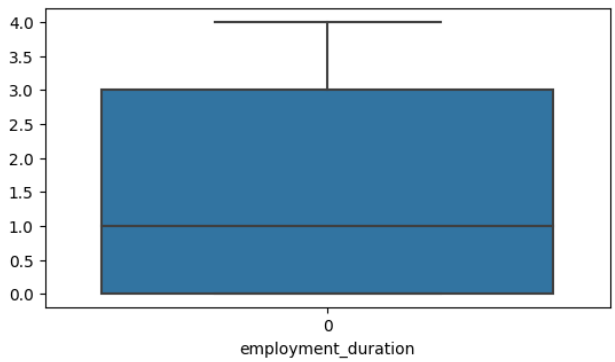

70
60
50
40
30
20
10
0
duration

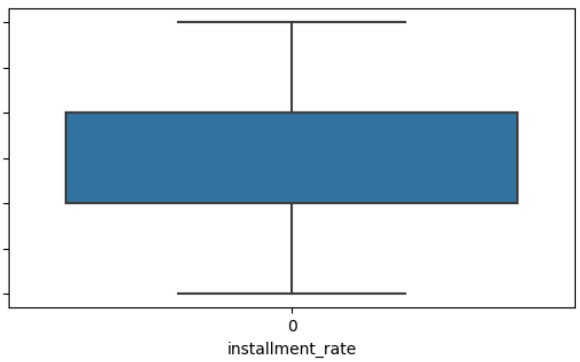

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
credit_history

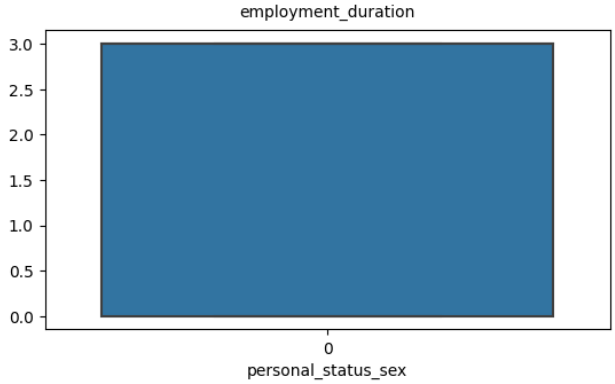

8
6
4
2
0
0
purpose

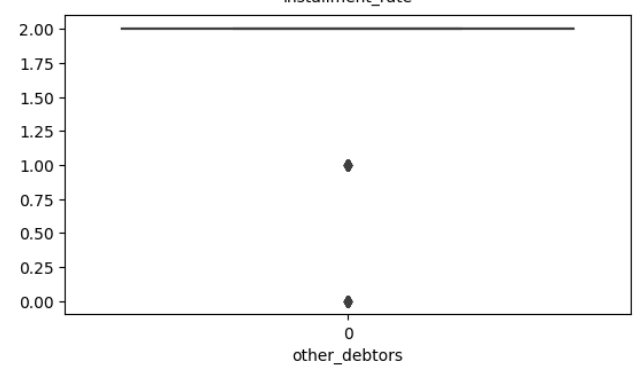

17500
15000
12500
10000
7500
5000
2500
0
0
amount

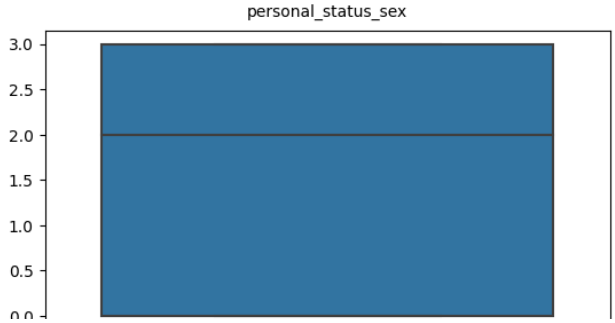

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
savings


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
employment_duration


3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
installment_rate


3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
personal_status_sex


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
other_debtors


3.0
2.5
2.0
1.5
1.0
0.5


3.0
2.5
2.0
1.5
1.0
0.5
0.0

```
upper_limit=df['amount'].mean()+3*df['amount'].std()
lower_limit=df['amount'].mean()-3*df['amount'].std()
print(upper_limit)
```

```
11739.503279686953
```

```
print(lower_limit)
```

```
-5197.007279686954
```

```
df.loc[(df['amount']>upper_limit)|(df['amount']<lower_limit)]
```

	status	duration	credit_history	purpose	amount	savings	employment_duration	installment_rate	personal_statu
179	1	48	3	4	12749	2	1	2	
208	0	48	1	5	12169	1	4	2	
382	1	60	0	5	13756	1	3	1	
426	0	39	3	7	11760	0	1	1	
442	3	39	0	2	14179	1	1	2	
575	0	48	2	8	12204	1	0	1	
645	3	36	4	0	15857	4	4	1	
695	1	60	3	4	15653	4	1	1	
754	0	6	4	5	14555	1	4	3	
782	0	36	4	5	14318	4	3	2	
810	0	48	4	8	15672	4	0	1	
844	0	24	0	0	11938	4	0	1	
847	0	60	1	0	14782	0	3	0	
849	0	36	4	7	12612	0	0	3	
890	0	60	4	5	14027	4	1	2	
895	0	18	4	1	12976	4	4	0	
929	0	48	2	8	14421	4	0	1	
943	0	36	4	5	12389	1	0	3	
962	0	54	2	8	15945	4	2	0	
973	3	45	2	8	11816	4	3	1	
975	3	30	4	3	11998	4	2	3	
976	0	48	2	0	18424	4	0	3	
977	3	6	4	5	14896	4	3	3	
991	0	24	4	1	12579	4	3	2	
997	1	21	0	5	12680	1	3	2	

25 rows × 21 columns

```
new_df=df.loc[(df['amount']<upper_limit)&(df['amount']>lower_limit)]
print('Before remove outliers',len(df))
print('After removing outliers',len(new_df))
```

```
Before remove outliers 1000
After removing outliers 975
```

```
X_new=new_df.iloc[:,:-1]
y_new=new_df.iloc[:,-1]
```

```
from sklearn.model_selection import train_test_split
X_train_new,X_test_new,y_train_new,y_test_new=train_test_split(X_new,y_new,test_size=0.3,random_state=1)
```

```
from sklearn.preprocessing import MinMaxScaler
sc=MinMaxScaler()
X_train_new=sc.fit_transform(X_train_new)
X_test_new=sc.transform(X_test_new)
```

```
#knn
knn.fit(X_train_new,y_train_new)
y_pred_new=knn.predict(X_test_new)
print('Accuracy after  outlier removal :')
knn_new=accuracy_score(y_test_new,y_pred_new)*100
knn_new
```

```
Accuracy after  outlier removal :
70.64846416382252
```

```
#svm
sv.fit(X_train_new,y_train_new)
y_pred_new=sv.predict(X_test_new)
print('Accuracy after  outlier removal :')
sv_new=accuracy_score(y_test_new,y_pred_new)*100
sv_new
```

```
Accuracy after  outlier removal :
73.72013651877133
```

```
#adaboost
ab.fit(X_train_new,y_train_new)
y_pred_new=ab.predict(X_test_new)
print('Accuracy after  outlier removal :')
ab_new=accuracy_score(y_test_new,y_pred_new)*100
ab_new
```

```
Accuracy after  outlier removal :
70.30716723549489
```

```
#RandomForest
rf.fit(X_train_new,y_train_new)
y_pred_new=rf.predict(X_test_new)
print('Accuracy after  outlier removal :')
rf_new=accuracy_score(y_test_new,y_pred_new)*100
rf_new
```

```
Accuracy after  outlier removal :
73.37883959044369
```

```
#DecisionTree
dt.fit(X_train_new,y_train_new)
y_pred_new=dt.predict(X_test_new)
print('Accuracy after  outlier removal :')
dt_new=accuracy_score(y_test_new,y_pred_new)*100
dt_new
```

```
Accuracy after  outlier removal :
64.84641638225256
```

```
#COMPARING ACCURACY
```

```
alg=['KNN','SVM','Random Forest','AdaBoost','Decision Tree']
acc=[knn_score,sv_score,rf_score,ada_score,dt_score]
acc_os=[knn_os,sv_os,rf_os,ab_os,dt_os]
acc_pca=[knn_pca,sv_pca,rf_pca,ab_pca,dt_pca]
acc_new=[knn_new,sv_new,rf_new,ab_new,dt_new]
Accuracy=pd.DataFrame({'Algorithms':alg,'Accuracy':acc,'Accuracy after over Sampling':acc_os,'Accuracy after PCA':acc_pca, 'Accuracy after outli
Accuracy
```

	Algorithms	Accuracy	Accuracy after over Sampling	Accuracy after PCA	Accuracy after outlierRemoval
0	KNN	69.000000	65.000000	71.333333	70.648464
1	SVM	70.666667	56.428571	71.333333	73.720137
2	Random Forest	75.000000	81.428571	72.000000	73.378840
3	AdaBoost	74.666667	75.952381	68.000000	70.307167
4	Decision Tree	68.333333	73.095238	62.666667	64.846416