ML-MAJOR-MAY-ML-05-MLB1

1 Major Project By Leeladhar Issar

2 Importing Basic Libraries

[193]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

3 Importing the dataset

[194]:
```
df=pd.read_csv("archive\loan-predictionUC.csv.csv")
```

4 Analizing the dataset

[195]:
```
df.head()
```

[195]:
```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
```

```
3              1.0          Urban            Y
4              1.0          Urban            Y
```

4.0.1 Dividing the dataset to X(features) and Y(Labels)

[196]:
```
X=df.iloc[:,1:-1].values  #since Loan Status doesn't depend on Loan_ID
Y=df['Loan_Status']
```

[197]:
```
print(X)
print(Y)
```

```
[['Male' 'No' '0' … 360.0 1.0 'Urban']
 ['Male' 'Yes' '1' … 360.0 1.0 'Rural']
 ['Male' 'Yes' '0' … 360.0 1.0 'Urban']
 …
 ['Male' 'Yes' '1' … 360.0 1.0 'Urban']
 ['Male' 'Yes' '2' … 360.0 1.0 'Urban']
 ['Female' 'No' '0' … 360.0 0.0 'Semiurban']]
0      Y
1      N
2      Y
3      Y
4      Y
      ..
609    Y
610    Y
611    Y
612    Y
613    N
Name: Loan_Status, Length: 614, dtype: object
```

[198]:
```
df.isnull().sum()
```

[198]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

[199]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

5 Dealing with missing values

[200]:
```
from sklearn.impute import SimpleImputer
imputer_freq=SimpleImputer(missing_values=np.nan,strategy='most_frequent')
imputer_freq.fit(X[:,:5])  #using most frequent strategy as this is categorical
X[:,:5]=imputer_freq.transform(X[:,:5])
```

[201]:
```
imputer=SimpleImputer(missing_values=np.nan,strategy='mean')
imputer.fit(X[:,7:10])  #using the mean strategy as this isn't category based
X[:,7:10]=imputer.transform(X[:,7:10])
```

Converting all Semi-urban and semiurban to Semiurban to reduce number of labels

[202]:
```
for i,prop in enumerate(X[:,-1]):
    if prop in ['Semi-urban','semiurban']:
        X[:,-1][i]='Semiurban'
```

[203]:
```
X
```

[203]:
```
array([['Male', 'No', '0', …, 360.0, 1.0, 'Urban'],
       ['Male', 'Yes', '1', …, 360.0, 1.0, 'Rural'],
       ['Male', 'Yes', '0', …, 360.0, 1.0, 'Urban'],
       …,
       ['Male', 'Yes', '1', …, 360.0, 1.0, 'Urban'],
```

```
       ['Male', 'Yes', '2', …, 360.0, 1.0, 'Urban'],
       ['Female', 'No', '0', …, 360.0, 0.0, 'Semiurban']], dtype=object)
```

5.0.1 Encoding categorical data using LabelEncoder

[204]:
```python
from sklearn.preprocessing import LabelEncoder
enc=LabelEncoder()
```

[205]:
```python
for i in range(5):
    X[:,i]=enc.fit_transform(X[:,i])

X[:,-1]=enc.fit_transform(X[:,-1])
```

[206]:
```python
X
```

[206]:
```
array([[1, 0, 0, …, 360.0, 1.0, 2],
       [1, 1, 1, …, 360.0, 1.0, 0],
       [1, 1, 0, …, 360.0, 1.0, 2],
       …,
       [1, 1, 1, …, 360.0, 1.0, 2],
       [1, 1, 2, …, 360.0, 1.0, 2],
       [0, 0, 0, …, 360.0, 0.0, 1]], dtype=object)
```

[207]:
```python
Y=enc.fit_transform(Y)
```

[208]:
```python
Y
```

[208]:
```
array([1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
```

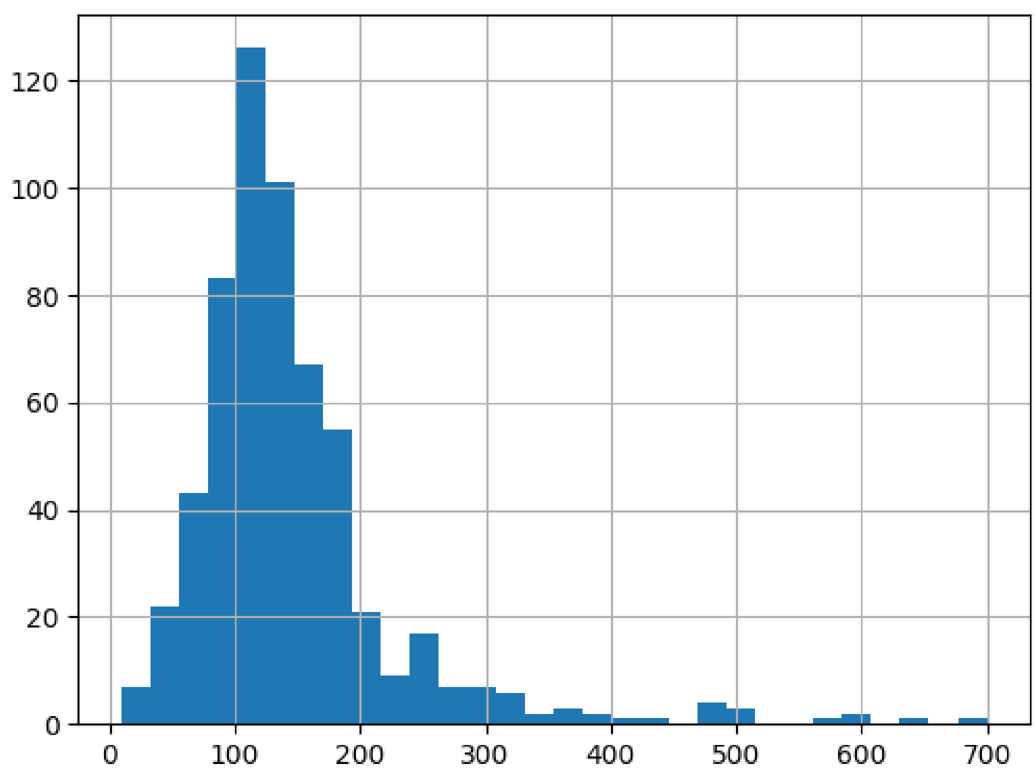
```
1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1,
1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,
1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0,
1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1,
1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0])
```

6 Visualisation

6.0.1 Histogram showing relation between loan amount and number of applicants to that amount

[209]:
```
df['LoanAmount'].hist(bins=30)
```

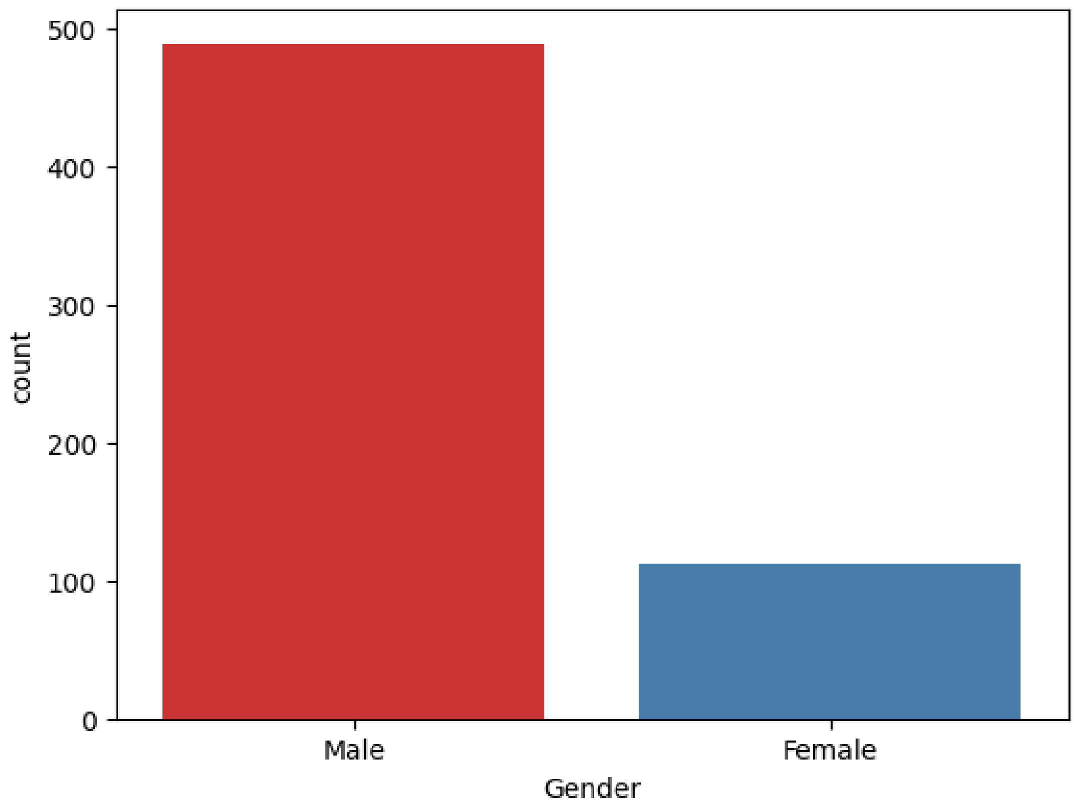
[209]: <Axes: >



6.0.2 Countplot describing the count of males and females applying for loan

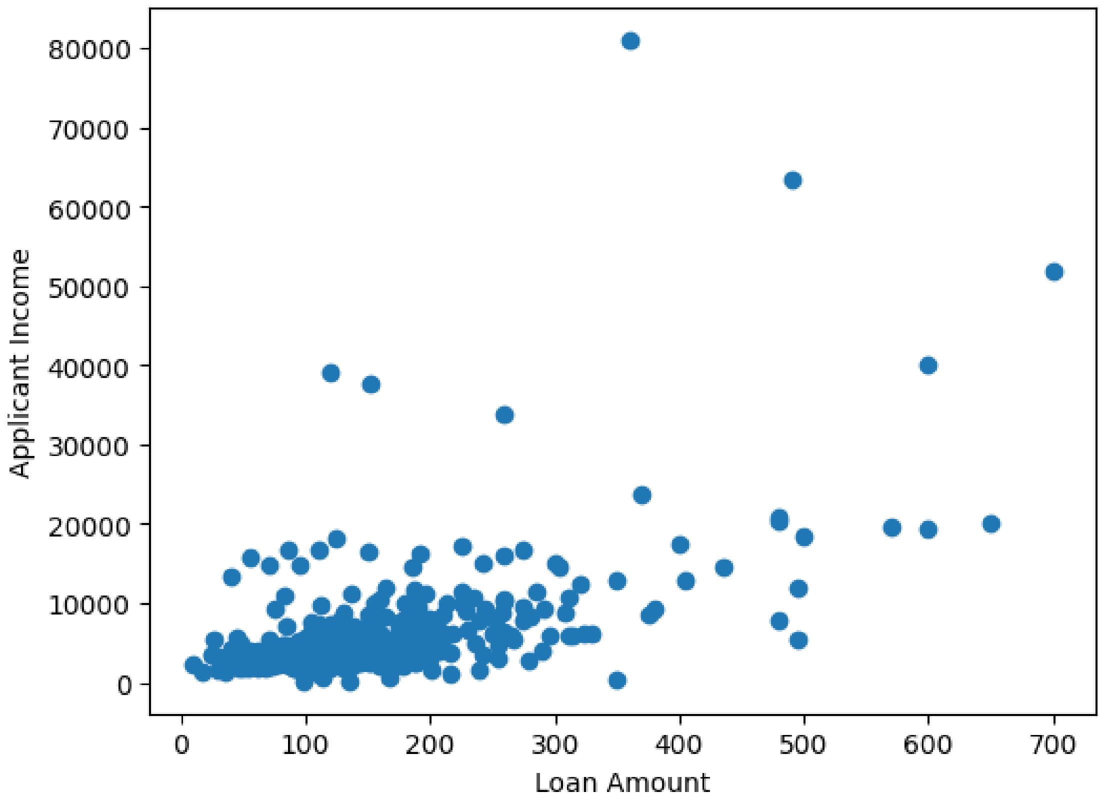
[210]:
```
sns.countplot(x='Gender',data=df,palette='Set1')
```

[210]: <Axes: xlabel='Gender', ylabel='count'>



6.0.3 Shows us the relation between Loan Amount requested and the applicant income

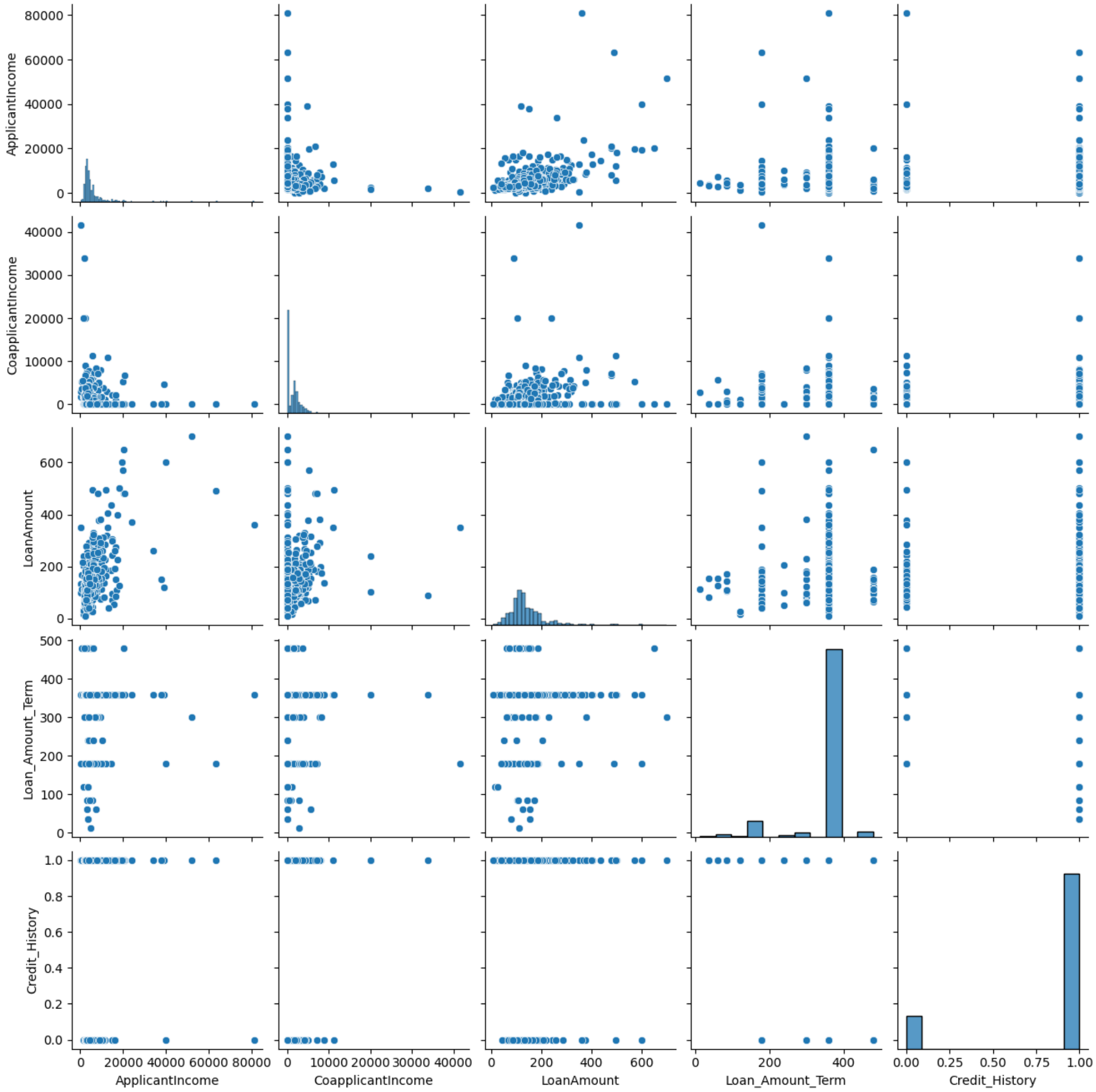
[211]:
```
plt.scatter(df['LoanAmount'],df['ApplicantIncome'])
plt.xlabel("Loan Amount")
plt.ylabel("Applicant Income")
plt.show()
```



6.0.4 Shows us the relation between different pairs of features

[212]:
```
sns.pairplot(df)
```

[212]: <seaborn.axisgrid.PairGrid at 0x2053e5b2740>



7 Test Train Split

[213]:
```python
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest= train_test_split(X,Y,test_size=0.2,random_state=1)
```

[214]:
```python
print(xtrain)
print(xtrain.size)
```

```
[[1 1 2 … 360.0 0.0 1]
 [1 0 0 … 360.0 1.0 2]
 [0 1 0 … 360.0 1.0 2]
```

```
 …
 [1 0 0 … 300.0 1.0 1]
 [1 1 1 … 360.0 1.0 0]
 [0 1 0 … 360.0 1.0 1]]
5401
```

[215]:
```python
print(xtest)
print(xtest.size)
```

```
[[1 0 1 … 360.0 0.8421985815602837 1]
 [0 1 0 … 360.0 0.8421985815602837 1]
 [1 0 0 … 360.0 1.0 2]
 …
 [1 1 1 … 180.0 1.0 2]
 [1 0 0 … 360.0 0.0 1]
 [0 0 0 … 360.0 1.0 1]]
1353
```

[216]:
```python
print(ytrain)
print(ytrain.size)
```

```
[0 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 0 0 1 1 1 1 1 1 1 1
 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 1 1 0 1
 0 0 1 1 1 0 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 0 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1
 1 1 0 1 1 0 1 1 1 1 1 1 0 0 0 1 0 1 0 1 1 1 1 1 1 0 0 0 0 1 1 0 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 0 1 0 1 1 1 1 0 0 1 1 0 1 1 1 0
 0 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 0 1 0 1 0 0 1
 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 1 0 0
 1 1 1 1 0 0 1 1 1 1 0 0 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1
 1 0 1 1 0 1 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1
 1 1 1 1 0 1 0 0 1 0 0 1 1 1 1 0 1 1 1 0 0 1 0 1 1 1 1 1 0 1 0 0 0 1 1 1 1
 1 1 0 0 1 0 1 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 0 1 0 0 1 1 1 0 1 1
 1 0 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 0 0 1 1 0 1 0
 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 0 1 0 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1
 1 1 1 1 0 0 1 1 1 1]
491
```

[217]:
```python
print(ytest)
print(ytest.size)
```

```
[0 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 1 0 1 0 1 1 1
 1 1 1 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0
 1 1 0 1 0 0 0 1 1 0 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1
 1 1 0 0 1 1 1 0 0 1 0 1]
123
```

8 Feature Scaling

Feature Scaling is done to optimize the gradient descent and make it smoother

[218]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
xtrain[:,5:9]=sc.fit_transform(xtrain[:,5:9])
xtest[:,5:9]=sc.transform(xtest[:,5:9])
```

[219]:
```
print(xtrain)
```

```
[[1 1 2 … 0.2969290648905011 0.0 1]
 [1 0 0 … 0.2969290648905011 1.0 2]
 [0 1 0 … 0.2969290648905011 1.0 2]
 …
 [1 0 0 … -0.614272002992225 1.0 1]
 [1 1 1 … 0.2969290648905011 1.0 0]
 [0 1 0 … 0.2969290648905011 1.0 1]]
```

[220]:
```
print(xtest)
```

```
[[1 0 1 … 0.2969290648905011 0.8421985815602837 1]
 [0 1 0 … 0.2969290648905011 0.8421985815602837 1]
 [1 0 0 … 0.2969290648905011 1.0 2]
 …
 [1 1 1 … -2.436674138757677 1.0 2]
 [1 0 0 … 0.2969290648905011 0.0 1]
 [0 0 0 … 0.2969290648905011 1.0 1]]
```

9 Creating Different Models

Using classification models as this is a classification problem

[221]:
```
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

model_log = LogisticRegression()
model_svc = SVC()
model_rfc=RandomForestClassifier()
model_nb=GaussianNB()
```

9.1 KNN classifier

[222]:
```
error_rate=[]
for i in range (1,40):
    model_knn=KNeighborsClassifier(n_neighbors=i)
    model_knn.fit(xtrain,ytrain)
    pred_i=model_knn.predict(xtest)
    error_rate.append(np.mean(pred_i!=ytest))
```

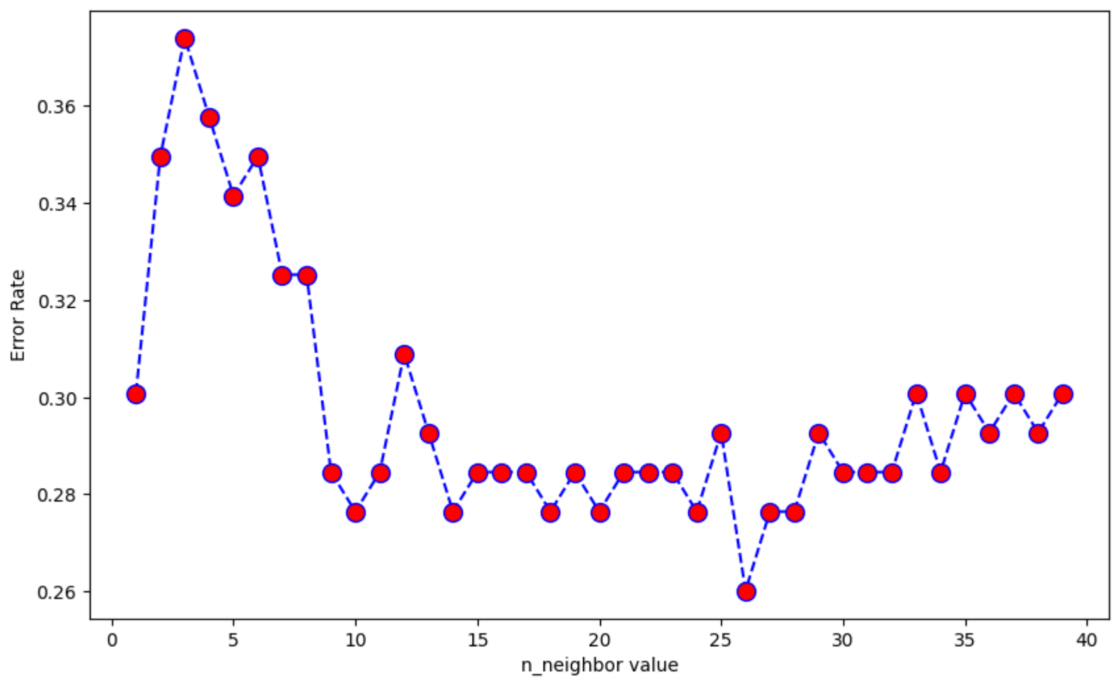
[223]:
```
error_rate
```

[223]:
```
[0.3008130081300813,
 0.34959349593495936,
 0.37398373983739835,
 0.35772357723577236,
 0.34146341463414637,
 0.34959349593495936,
 0.3252032520325203,
 0.3252032520325203,
 0.2845528455284553,
 0.2764227642276423,
 0.2845528455284553,
 0.3089430894308943,
 0.2926829268292683,
 0.2764227642276423,
 0.2845528455284553,
 0.2845528455284553,
 0.2845528455284553,
 0.2764227642276423,
 0.2845528455284553,
 0.2764227642276423,
 0.2845528455284553,
 0.2845528455284553,
 0.2845528455284553,
 0.2764227642276423,
 0.2926829268292683,
 0.2601626016260163,
 0.2764227642276423,
 0.2764227642276423,
 0.2926829268292683,
 0.2845528455284553,
 0.2845528455284553,
 0.2845528455284553,
 0.3008130081300813,
 0.2845528455284553,
 0.3008130081300813,
 0.2926829268292683,
```

```
 0.3008130081300813,
 0.2926829268292683,
 0.3008130081300813]
```

Elbow method to find the best value for the amount of neighbours

[224]:
```python
plt.figure(figsize=(10,6))
plt.
 ↪plot(range(1,40),error_rate,color='blue',linestyle='--',markersize=10,markerfacecolor='red'
plt.xlabel('n_neighbor value')
plt.ylabel('Error Rate')
plt.show()
```



[225]:
```python
best_k_ar,=np.where(error_rate == np.min(error_rate))
best_k=best_k_ar[0]
print(best_k)
```

```
25
```

[226]:
```python
model_knn=KNeighborsClassifier(n_neighbors=best_k)
model_knn.fit(xtrain,ytrain)
```

[226]:
```
KNeighborsClassifier(n_neighbors=25)
```

[227]:
```python
ypred=model_knn.predict(xtest)
ypred
```

[227]:
```
array([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, 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, 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, 1, 1, 1, 1])
```

9.1.1 The performance of KNN

[228]:
```python
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score

accuracy_score(ytest,ypred)
```

[228]:
```
0.7073170731707317
```

[229]:
```python
print(classification_report(ytest,ypred))
print(confusion_matrix(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.08      0.14        39
           1       0.70      1.00      0.82        84

    accuracy                           0.71       123
   macro avg       0.85      0.54      0.48       123
weighted avg       0.80      0.71      0.61       123

[[ 3 36]
 [ 0 84]]
```

9.2 Logistic Regressor

[230]:
```python
model_log.fit(xtrain,ytrain)
```

[230]:
```
LogisticRegression()
```

[231]:
```python
ypred_log=model_log.predict(xtest)
print(ypred_log)
```

```
[1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1
 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0
 1 1 1 1 0 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 1 0 1 1
 1 1 1 0 1 1 1 1 1 1 0 1]
```

9.2.1 Logistic Regressor Performance

[232]:
```
accuracy_score(ytest,ypred_log)
```

[232]:
```
0.7967479674796748
```

[233]:
```
print(classification_report(ytest,ypred_log))
print(confusion_matrix(ytest,ypred_log))
```

```
              precision    recall  f1-score   support

           0       0.89      0.41      0.56        39
           1       0.78      0.98      0.87        84

    accuracy                           0.80       123
   macro avg       0.83      0.69      0.71       123
weighted avg       0.82      0.80      0.77       123

[[16 23]
 [ 2 82]]
```

9.3 Support Vector Classifier

[234]:
```
model_svc.fit(xtrain,ytrain)
```

[234]:
```
SVC()
```

[235]:
```
ypred_svc=model_svc.predict(xtest)
print(ypred_svc)
```

```
[1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 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 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0
 1 1 1 1 0 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 1 0 1 1
 1 1 1 0 1 1 1 1 1 1 0 1]
```

9.3.1 SVC performance

[236]:
```
accuracy_score(ytest,ypred_svc)
```

[236]:
```
0.8048780487804879
```

[237]:
```
print(classification_report(ytest,ypred_svc))
print(confusion_matrix(ytest,ypred_svc))
```

```
              precision    recall  f1-score   support

           0       0.94      0.41      0.57        39
           1       0.78      0.99      0.87        84
```

```
    accuracy                           0.80       123
   macro avg       0.86      0.70      0.72       123
weighted avg       0.83      0.80      0.78       123

[[16 23]
 [ 1 83]]
```

9.4 Random Forest Classifier

[238]:
```
model_rfc.fit(xtrain,ytrain)
```

[238]:
```
RandomForestClassifier()
```

[239]:
```
ypred_rfc=model_rfc.predict(xtest)
print(ypred_rfc)
```

```
[1 1 1 1 1 0 1 1 1 0 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 1 1
 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0
 1 1 1 1 0 0 0 1 0 0 0 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 1
 1 1 1 0 1 1 1 1 1 1 0 1]
```

9.4.1 RFC performance

[240]:
```
accuracy_score(ytest,ypred_rfc)
```

[240]:
```
0.7886178861788617
```

[241]:
```
print(classification_report(ytest,ypred_rfc))
print(confusion_matrix(ytest,ypred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.81      0.44      0.57        39
           1       0.78      0.95      0.86        84

    accuracy                           0.79       123
   macro avg       0.80      0.69      0.71       123
weighted avg       0.79      0.79      0.77       123

[[17 22]
 [ 4 80]]
```

9.5 Naive Bayes Classifier

[242]:
```
model_nb.fit(xtrain,ytrain)
```

[242]:
```
GaussianNB()
```

[243]:
```
ypred_nb=model_nb.predict(xtest)
print(ypred_nb)
```

```
[1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1
 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0
 1 1 1 1 0 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 1 0 1 1
 1 1 0 0 1 1 1 1 1 1 0 1]
```

9.5.1 Naive Bayes performance

[244]:
```
accuracy_score(ytest,ypred_nb)
```

[244]: 0.8048780487804879

[245]:
```
print(classification_report(ytest,ypred_nb))
print(confusion_matrix(ytest,ypred_nb))
```

```
              precision    recall  f1-score   support

           0       0.86      0.46      0.60        39
           1       0.79      0.96      0.87        84

    accuracy                           0.80       123
   macro avg       0.83      0.71      0.74       123
weighted avg       0.81      0.80      0.79       123

[[18 21]
 [ 3 81]]
```

10 Conclusion

NaiveBayesClassifier(NBC) and SupportVectorClassifier(SVC) gives the best accuracy trained on the given dataset with an accuracy of 80.48%

LogisticRegressor has a 79.67% accuracy score

RandomForestClassifier has a 78.04% accuracy score

KNN had a 70.73% accuracy score

So , thereby the model gave best performance with NaiveBayes and SupportVectorClassifier

The accuracy of the models can be increased by taking a larger dataset , as for now NaiveBayes or SupportVectorClassifier to be used

[ ]:
```
```