Aim : Implementation of Linear Regression, Logistic regression, KNN- classification.

Linear Regression:

Linear regression is a statistical method that is used to model the relationship between a dependent variable and one or more independent variables. It is a popular technique for predictive modeling and is widely used in various fields, including:

- Machine learning
- Economics
- Finance
- Science

> y = mx + b

Where:

- y is the dependent variable (the variable we are trying to predict).
- x is the independent variable (the variable used to make predictions).
- m is the slope of the line (the change in y for a one-unit change in x).
- b is the y-intercept (the value of y when x is 0).

In multiple linear regression, there are multiple independent variables, and the relationship between the independent variables and the dependent variable is modeled using the equation:

> $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$

Linear regression is widely used in various fields, including economics, finance, biology, and engineering, for tasks such as prediction, forecasting, and understanding the relationships between variables.

Import necessary libraries:

- numpy for numerical operations.
- pandas for data manipulation.
- scikit-learn for machine learning algorithms.

Collab Link - https://colab.research.google.com/drive/1-k7uu8e1zihBxvGTFK-WQXg0R6ywEc2L?authuser=6#scrollTo=ZWo96q-yVqlx

Linear Regression on video's dataset

Video link: https://www.youtube.com/watch?v=UNv0Ao6ItJ0

- **Importing Libraries**

```
import pandas as pd #for making dataframes
import numpy as np #for arrays
import matplotlib.pyplot as plt #for plotting
%matplotlib inline
```

Double-click (or enter) to edit

- **Loading Dataset**

```
df_regression = pd.read_csv("/content/score.csv")
print("Data imported successfully")
df_regression.head(11)
```

```
Data imported successfully
```

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25
10	7.7	85

- **Understanding data**

```
df_regression.shape
```

```
(25, 2)
```

```
df_regression.info()
```

```
df_regression.describe()
```

	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000

- **Counts NA values under entire dataframe**

```
df_regression.isna().sum()
```

```
Hours     0
Scores    0
dtype: int64
```

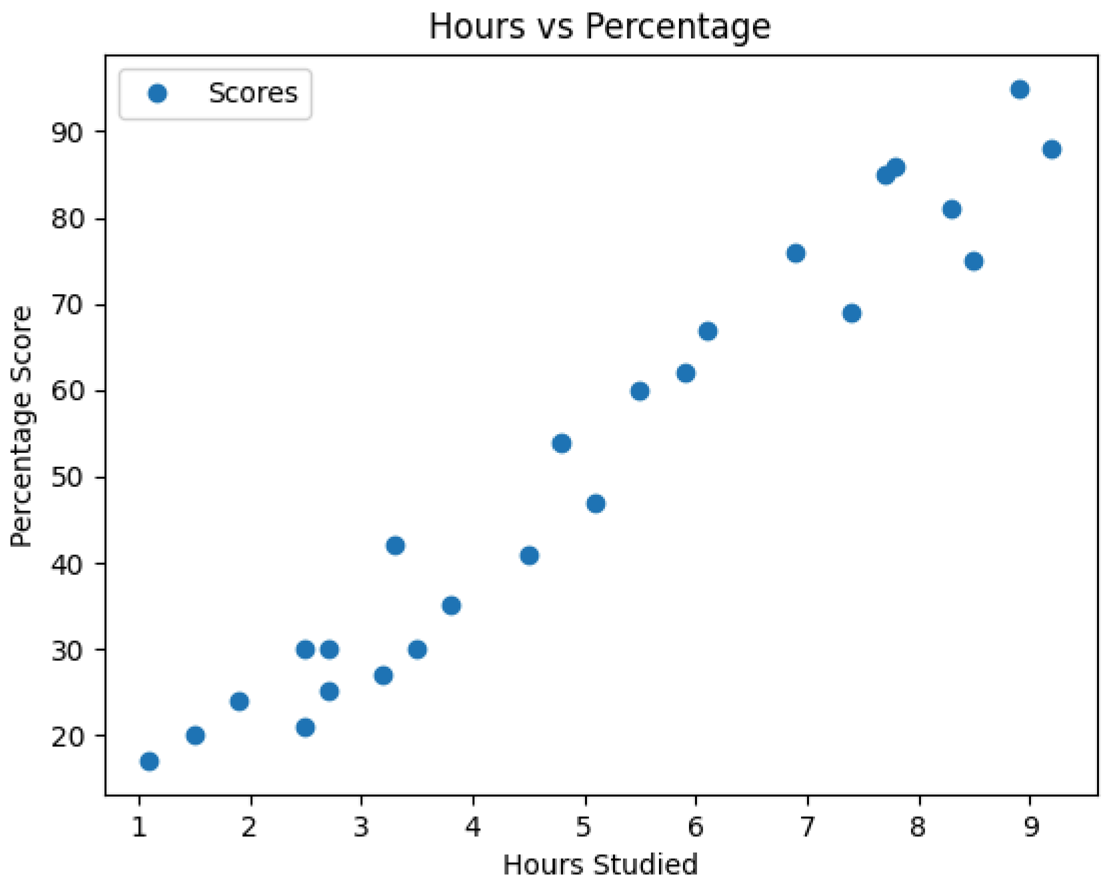
- **Finding Correlation of Dependent and independent variables**

```
df_regression.corr()
```

	Hours	Scores
Hours	1.000000	0.976191
Scores	0.976191	1.000000

- **Plotting the data to check if relationship is linear**

```
df_regression.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```



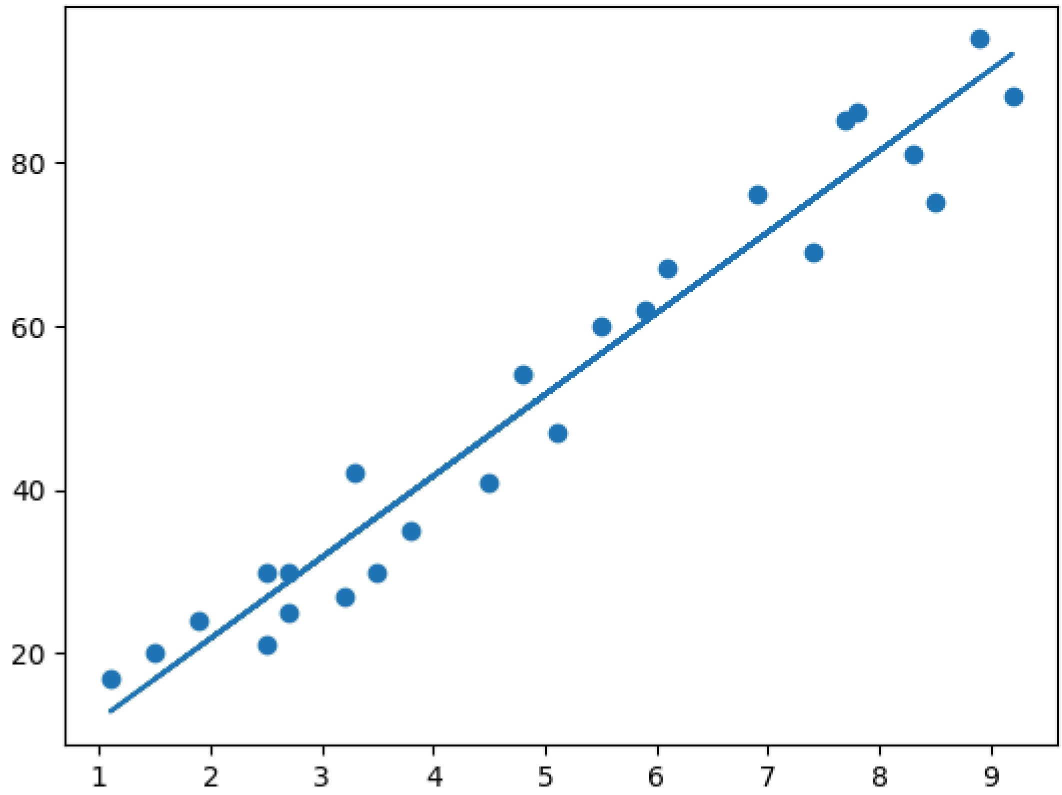
- **Subsetting of the data**

```
x_regression = df_regression.iloc[:, :-1].values #integer location 0 to -1
y_regression = df_regression.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_regression, y_regression, test_size
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()

regressor.fit(X_train, y_train)
print("Training complete.")
```

```
Training complete.
```

- **Plotting the data**

```
line = regressor.coef_*x_regression+regressor.intercept_

plt.scatter(x_regression, y_regression)
plt.plot(x_regression, line);
plt.show()
```



- **Checking the predicted values**

```
print(X_test) # Testing data - In Hours
y_pred = regressor.predict(X_test) # Predicting the scores
```

```
[[1.5]
 [3.2]
 [7.4]
 [2.5]
 [5.9]]
```

```
# Comparing Actual vs Predicted
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

	Actual	Predicted
0	20	16.884145
1	27	33.732261
2	69	75.357018
3	30	26.794801
4	62	60.491033

- **Check the scores**

```
regressor. score (X_train, y_train) # Score of our trained model
```

```
0.9515510725211552
```

- **Calculate Error in Model**

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

```
Mean Absolute Error: 4.183859899002982
```

```
print('r2 Score: ',metrics.r2_score (y_test, y_pred))
```

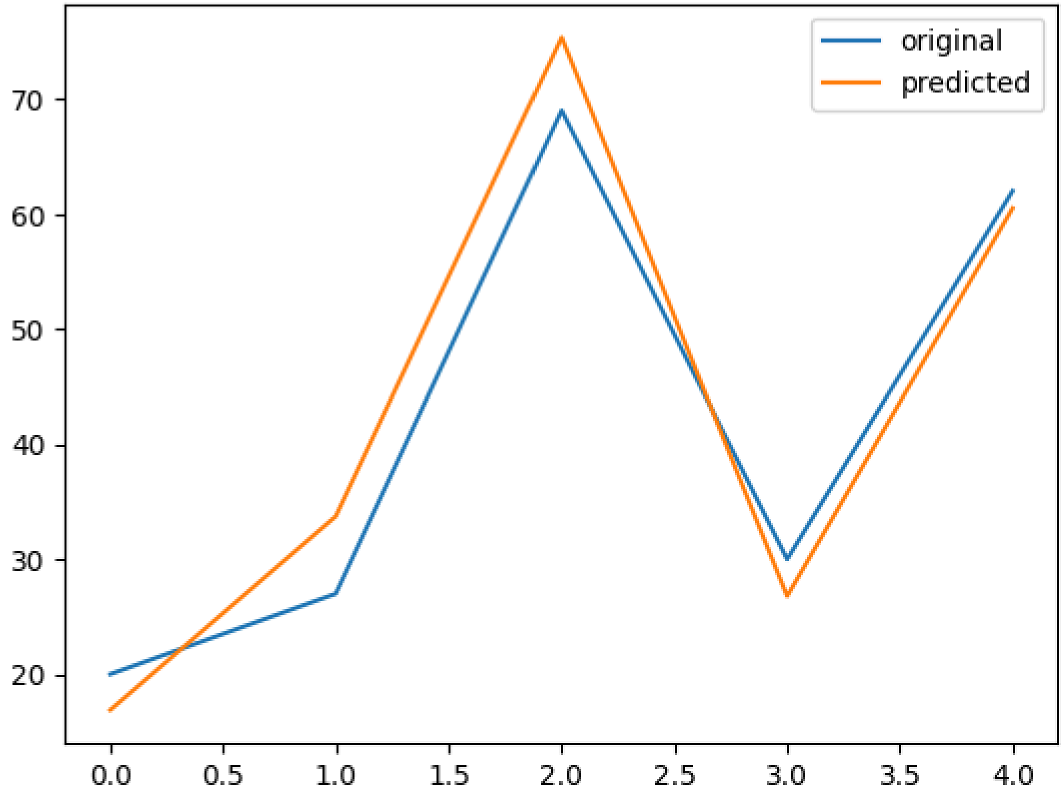
```
r2 Score:  0.9454906892105354
```

```
x_axis = range(len(y_test))
x_axis
```

```
range(0, 5)
```

- **Plotting the values to visualize how well our model works.**

```
plt.plot(x_axis, y_test, label='original')
plt.plot(x_axis, y_pred, label='predicted')
plt.legend()
plt.show()
```



Linear Regression on own dataset

- **Importing Libraries**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

- **Loading Dataset**

```
df_regression = pd.read_csv("/content/Ice Cream Sales - temperatures.csv")
print("Data imported successfully")
df_regression.head(10)
```

```
Data imported successfully
```

	Temperature	Ice Cream Profits
0	39	13.17
1	40	11.88
2	41	18.82
3	42	18.65
4	43	17.02
5	43	15.88
6	44	19.07
7	44	19.57
8	45	21.62
9	45	22.34

- **Understanding data**

```
df_regression.shape
```

```
(365, 2)
```

```
df_regression.columns
```

```
Index(['Temperature', 'Ice Cream Profits'], dtype='object')
```

```
df_regression.describe()
```

	Temperature	Ice Cream Profits
count	365.000000	365.000000
mean	71.980822	52.103616
std	13.258510	15.989004
min	39.000000	11.880000
25%	63.000000	40.650000
50%	73.000000	53.620000
75%	82.000000	63.630000
max	101.000000	89.290000

- **Counts NA values under entire dataframe**

```
df_regression.isna().sum()
```

```
Temperature          0
Ice Cream Profits    0
dtype: int64
```

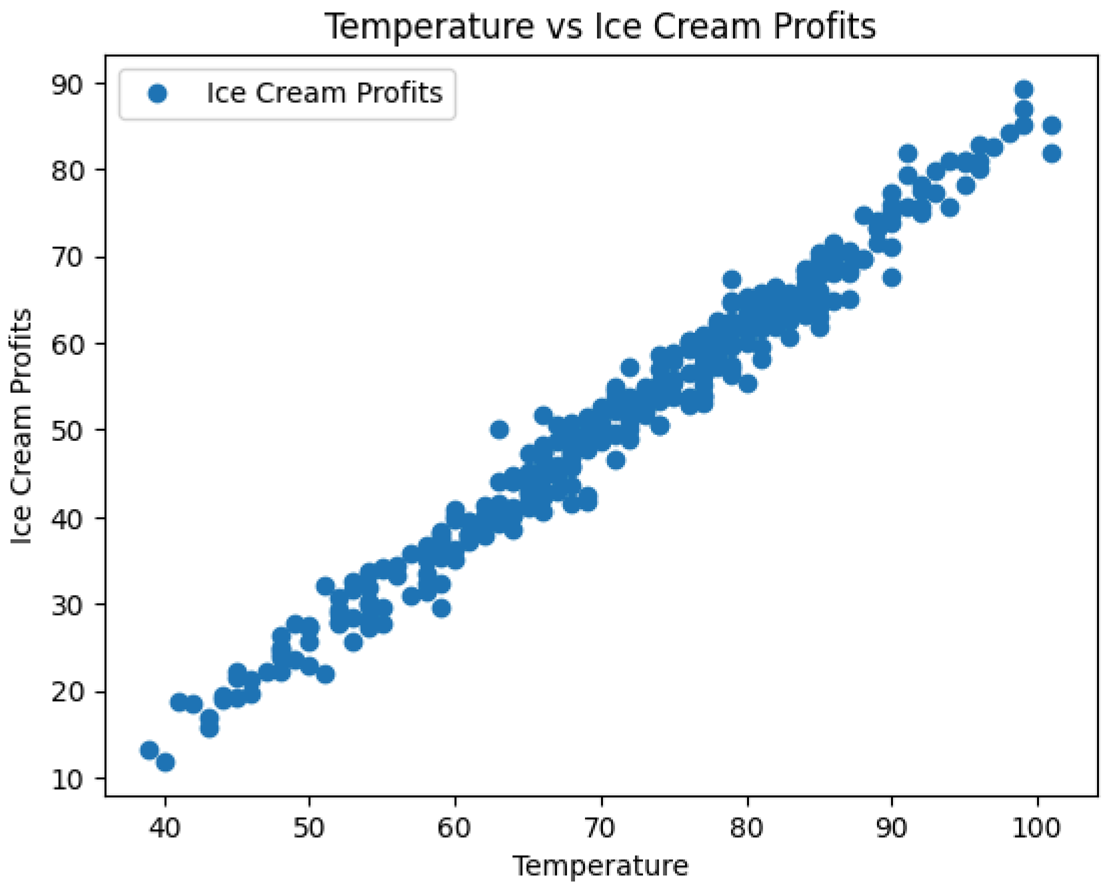
- **Finding Correlation of Dependent and independent variables**

```
df_regression.corr()
```

	Temperature	Ice Cream Profits
Temperature	1.000000	0.988446
Ice Cream Profits	0.988446	1.000000

- **Plotting the data to check if relationship is linear**

```
df_regression.plot(x='Temperature', y='Ice Cream Profits', style='o')
plt.title('Temperature vs Ice Cream Profits')
plt.xlabel('Temperature')
plt.ylabel('Ice Cream Profits')
plt.show()
```



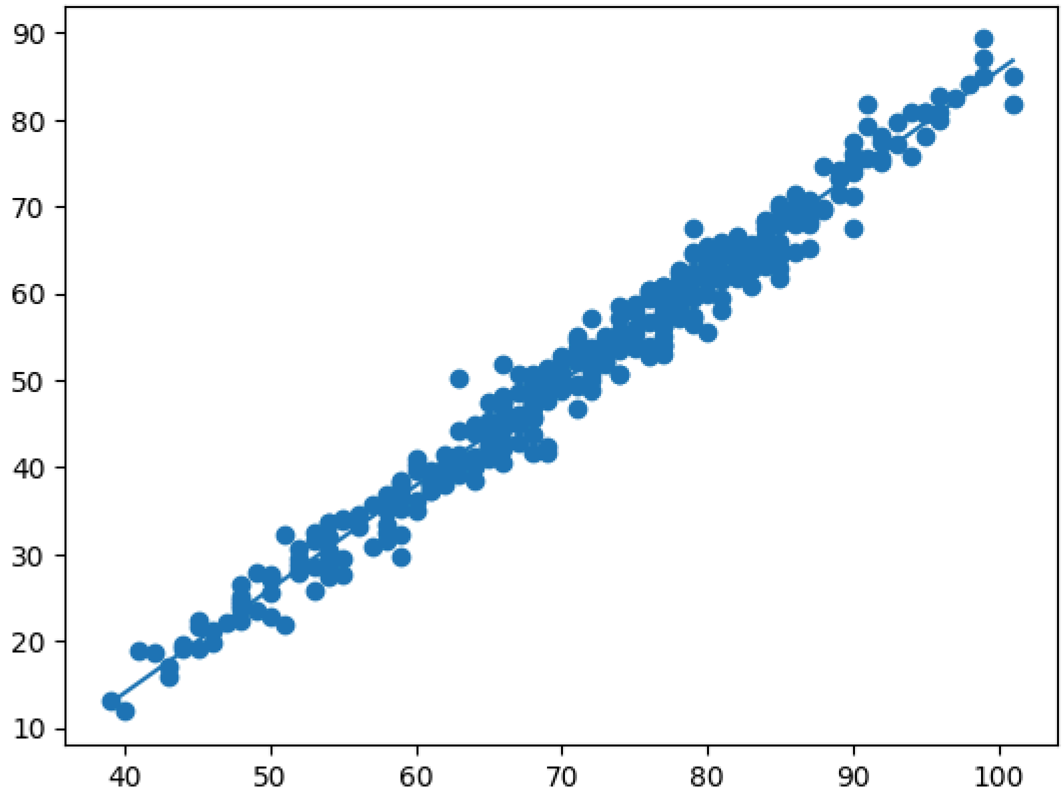
- **Subsetting of the data**

```
x_regression = df_regression.iloc[:, :-1].values #integer location 0 to -1
y_regression = df_regression.iloc[:, 1].values

# Splitting the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_regression, y_regression, test_size
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()

# Fitting the data
regressor.fit(X_train, y_train)
print("Training complete.")
```

```
Training complete.
```

- **Plotting the data**

```
# Plotting the regression line y=mx+c
line = regressor.coef_*x_regression+regressor.intercept_
# Plotting for the test data
plt.scatter(x_regression, y_regression)
plt.plot(x_regression, line);
plt.show()
```



- **Checking the predicted values**

```
print(X_test) # Testing data - In Age
y_pred = regressor.predict(X_test) # Predicting the Premium
```

```
[[ 65]
 [ 80]
 [ 54]
 [ 50]
 [ 61]
 [ 92]
 [ 63]
 [ 85]
 [ 78]
 [ 44]
 [ 66]
 [ 68]
 [ 81]
 [ 83]
 [ 84]
 [ 76]
 [ 84]
 [ 65]
 [ 77]
 [ 68]
 [ 69]
 [ 75]
 [ 58]
 [ 84]
 [ 85]
 [ 80]
 [ 59]
```

```
 [ 89]
 [ 66]
 [ 74]
 [ 67]
 [ 65]
 [ 48]
 [ 53]
 [ 57]
 [ 43]
 [ 58]
 [ 68]
 [ 81]
 [ 59]
 [ 76]
 [ 85]
 [ 76]
 [ 80]
 [ 49]
 [ 85]
 [ 76]
 [ 77]
 [ 78]
 [ 59]
 [ 72]
 [ 77]
 [ 78]
 [ 71]
 [ 64]
 [ 58]
 [ 56]
```

```
# Comparing Actual vs Predicted
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

	Actual	Predicted
0	42.10	43.862528
1	64.45	61.754719
2	27.99	30.741588
3	27.31	25.970337
4	39.53	39.091277
...	...	...
141	54.36	54.597843
142	53.78	52.212217
143	44.31	43.862528
144	30.37	30.741588
145	64.22	67.718783

146 rows × 2 columns

- **Check the Premium**

```
regressor.score (X_train, y_train) # Score of our trained model
```

```
0.9764169859322894
```

- **Calculate Error in Model**

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

```
Mean Absolute Error: 1.8876536707403655
```

```
print('r2 Score: ',metrics.r2_score (y_test, y_pred))
```

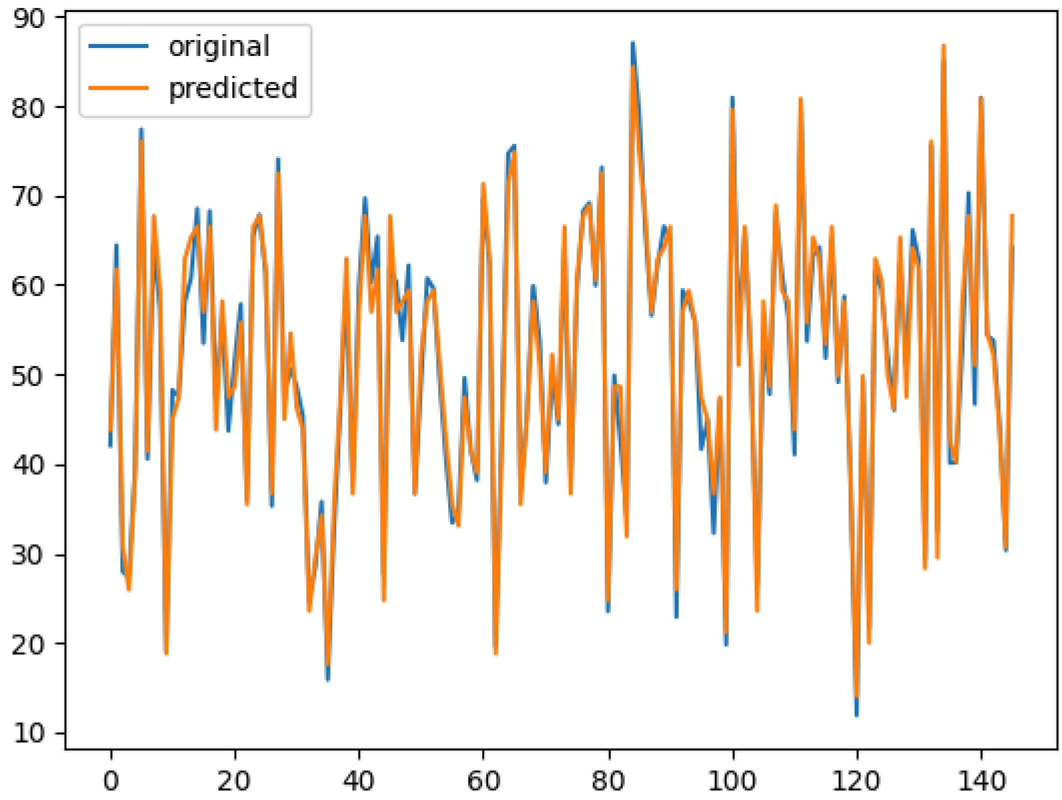
```
r2 Score:  0.9779175387273723
```

```
x_axis = range(len(y_test))
x_axis
```

```
range(0, 146)
```

- **Plotting the values to visualize how well our model works.**

```
plt.plot(x_axis, y_test, label='original')
plt.plot(x_axis, y_pred, label='predicted')
plt.legend()
plt.show()
```



Logistic Regression on video's dataset

Video link: https://www.youtube.com/watch?v=TT_njLsB7-0

Logistic regression is a statistical method commonly used in machine learning for classification problems. It is a powerful tool for predicting the probability of an event occurring, such as whether an email is spam or not, whether a customer will churn or not, or whether a loan will be repaid or not.

Logistic regression is not the same as linear regression, although they share some similarities. It is a powerful tool for classification tasks, especially when dealing with probabilities. It is interpretable, meaning you can understand the impact of each independent variable on the predicted probability.

Application:-

- Spam filtering: Email providers use logistic regression to classify incoming emails as spam or legitimate based on various features like sender information and keywords.
- Sentiment analysis: Analyzing text data, logistic regression can be used to classify sentiments (positive, negative, neutral) expressed in reviews, social media posts, etc.
- Targeted advertising: Based on user data, companies can leverage logistic regression to determine which users are more likely to click on an advertisement, optimizing marketing strategies.

```
#reading the dataset using pandas
df=pd.read_csv('/content/User_Data.csv')
print(df)
```

```
      User ID  Gender  Age  EstimatedSalary  Purchased
0    15624510    Male   19            19000          0
1    15810944    Male   35            20000          0
2    15668575  Female   26            43000          0
3    15603246  Female   27            57000          0
4    15804002    Male   19            76000          0
..        ...     ...  ...              ...        ...
395  15691863  Female   46            41000          1
396  15706071    Male   51            23000          1
397  15654296  Female   50            20000          1
398  15755018    Male   36            33000          0
399  15594041  Female   49            36000          1

[400 rows x 5 columns]
```

```
from sklearn.model_selection import train_test_split
X=df[['Age', 'EstimatedSalary']].values
Y=df[['Purchased']].values
x_train, x_test, y_train, y_test= train_test_split(X,Y, test_size=0.25, random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
st_x=StandardScaler()
x_train=st_x.fit_transform(x_train)
x_test=st_x.fit_transform(x_test)
```

```
from sklearn.linear_model import LogisticRegression
lm=LogisticRegression (random_state=0)
lm.fit(x_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConvers
  y = column_or_1d(y, warn=True)
▾       LogisticRegression
LogisticRegression(random_state=0)
```

```
y_pred=lm.predict(x_test)
print(y_pred)
```

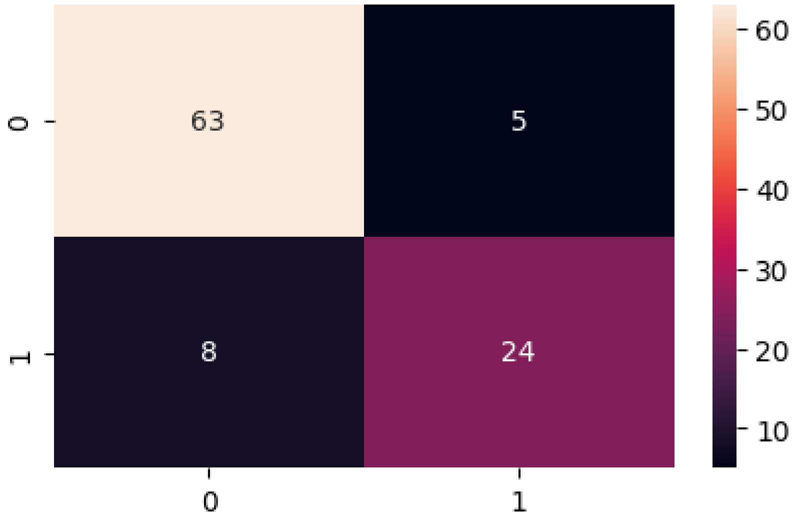
```
[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0
 0 0 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1]
```

```
from sklearn.metrics import confusion_matrix as cm, accuracy_score
print(accuracy_score (y_test,y_pred))
df_cm=cm(y_test,y_pred)
print(df_cm)
```

```
0.87
[[63  5]
 [ 8 24]]
```

```
import seaborn as sn
import matplotlib.pyplot as plt
plt.figure(figsize = (5,3))
sn.heatmap(df_cm, annot=True)
```

```
<Axes: >
```



Logistic Regression on own dataset

```
#reading the dataset using pandas
df=pd.read_csv('/content/SBI.csv')
print(df)
```

```
      Unnamed: 0      id  fever_hours   age sex   wcc prevAB            sbi  \
0              1   57906         24.0  0.79   M   3.8     No            UTI
1              2   58031         48.0  1.91   F  25.3    Yes            UTI
2              3   58148         24.0  0.07   F  20.0     No            UTI
3              4   58169         72.0  0.95   M   6.0     No            UTI
4              5   58517          1.0  0.11   F  15.6     No            UTI
...          ...     ...          ...   ...  ..   ...    ...            ...
2343        2344  229318         48.0  1.06   M  14.1     No  NotApplicable
2344        2345  229506         24.0  3.05   M  14.6     No  NotApplicable
2345        2346  229794         48.0  1.81   M   6.0     No  NotApplicable
2346        2347  229962         24.0  1.24   M  16.3    Yes  NotApplicable
2347        2348  229985         24.0  3.56   F  13.0     No  NotApplicable

           pct         crp
0     0.090000   17.700000
1     4.400000  150.400000
2     0.548136   47.359279
3     0.310000    4.900000
4     0.936872   31.394860
...        ...         ...
2343  0.160000   16.700000
```

```
2344   1.080000   77.500000
2345   0.480000   75.300000
2346  20.280000   17.300000
2347   0.606293   18.181134

[2348 rows x 10 columns]
```

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

	Unnamed: 0	id	fever_hours	age	sex	wcc	prevAB	sbi	pct	crp
0	1	57906	24.0	0.79	M	3.8	No	UTI	0.090000	17.700000
1	2	58031	48.0	1.91	F	25.3	Yes	UTI	4.400000	150.400000
2	3	58148	24.0	0.07	F	20.0	No	UTI	0.548136	47.359279
3	4	58169	72.0	0.95	M	6.0	No	UTI	0.310000	4.900000
4	5	58517	1.0	0.11	F	15.6	No	UTI	0.936872	31.394860
5	6	58535	96.0	0.91	M	6.2	No	UTI	0.690000	9.000000
6	7	59139	48.0	1.56	F	13.0	No	UTI	2.680000	110.779789
7	8	59159	96.0	0.88	F	26.4	No	UTI	4.760000	163.495967
8	9	59560	96.0	0.42	F	8.2	No	UTI	5.050000	151.375166
9	10	60089	48.0	0.81	M	7.5	Yes	UTI	0.080000	9.300000

```
from sklearn.model_selection import train_test_split
X=df[['id', 'fever_hours', 'wcc']].values
Y=df[['prevAB']].values
x_train, x_test, y_train, y_test= train_test_split(X,Y, test_size=0.25, random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
st_x=StandardScaler()
x_train=st_x.fit_transform(x_train)
x_test=st_x.fit_transform(x_test)
```

```
from sklearn.linear_model import LogisticRegression
lm=LogisticRegression (random_state=0)
lm.fit(x_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConvers
  y = column_or_1d(y, warn=True)
▾        LogisticRegression
LogisticRegression(random_state=0)
```

```
y_pred=lm.predict(x_test)
print(y_pred)
```

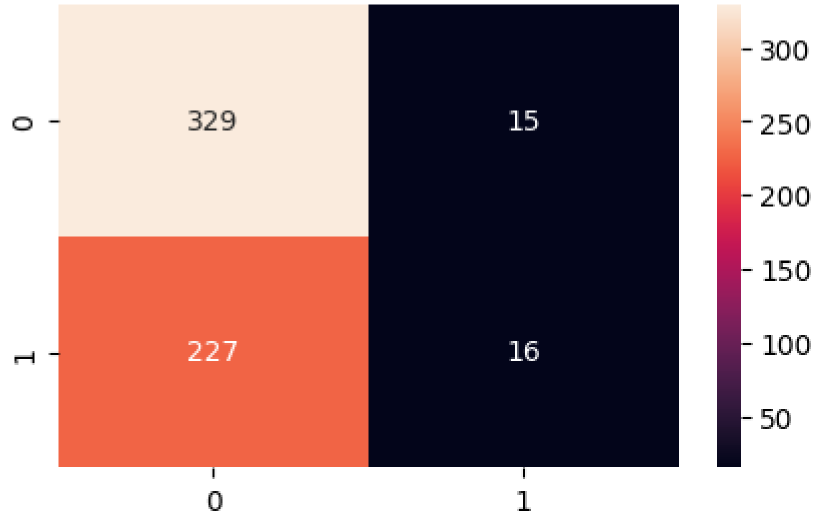
```
['No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'No'
 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes'
 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'Yes' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'Yes' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'Yes' 'Yes' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No'
 'Yes' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No'
 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No'
 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No']
```

```
from sklearn.metrics import confusion_matrix as cm, accuracy_score
print(accuracy_score (y_test,y_pred))
df_cm=cm(y_test,y_pred)
print(df_cm)
```

```
0.5877342419080068
[[329  15]
 [227  16]]
```

```
import seaborn as sn
import matplotlib.pyplot as plt
plt.figure(figsize = (5,3))
sn.heatmap(df_cm, annot=True, fmt='g') #fmt="g" cause annot turns fmt= ".2g" so it doesn'
```

```
<Axes: >
```



K Nearest Neighbors with Python

You've been given a classified data set from a company! They've hidden the feature column names but have given you the data and the target classes.

We'll try to use KNN to create a model that directly predicts a class for a new data point based off of the features.

Let's grab it and use it!

Import Libraries

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

Get the Data

Set index_col=0 to use the first column as the index.

```
df = pd.read_csv("/content/fake_bills_KNN.csv")
```

```
#df.drop('margin_low',axis=1)
```

```
df.head()
```

	is_genuine	diagonal	height_left	height_right	margin_up	length
0	1	171.81	104.86	104.95	2.89	112.83
1	1	171.46	103.36	103.66	2.99	113.09
2	1	172.69	104.48	103.50	2.94	113.16
3	1	171.36	103.91	103.94	3.01	113.51
4	1	171.73	104.28	103.46	3.48	112.54

Standardize the Variables

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
scaler.fit(df.drop('is_genuine',axis=1))
```

```
▾ StandardScaler
StandardScaler()
```

```
scaled_features = scaler.transform(df.drop('is_genuine',axis=1))
```

```
df_feat = pd.DataFrame(scaled_features,columns=df.columns[1:])
df_feat.head()
```

	diagonal	height_left	height_right	margin_up	length
0	-0.486540	2.774123	3.163240	-1.128325	0.173651
1	-1.633729	-2.236535	-0.799668	-0.696799	0.471666
2	2.397823	1.504756	-1.291191	-0.912562	0.551901
3	-1.961498	-0.399294	0.060498	-0.610494	0.953075
4	-0.748754	0.836669	-1.414072	1.417677	-0.158750

Train Test Split

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(scaled_features,df['is_genuine'],
                                                    test_size=0.30)
```

Using KNN

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=1)
```

```
knn.fit(X_train,y_train)
```

```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

```
pred = knn.predict(X_test)
```

Predictions and Evaluations

Let's evaluate our KNN model!

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(confusion_matrix(y_test,pred))
```

```
[[143  10]
 [  8 289]]
```

```
print(classification_report(y_test,pred))
```

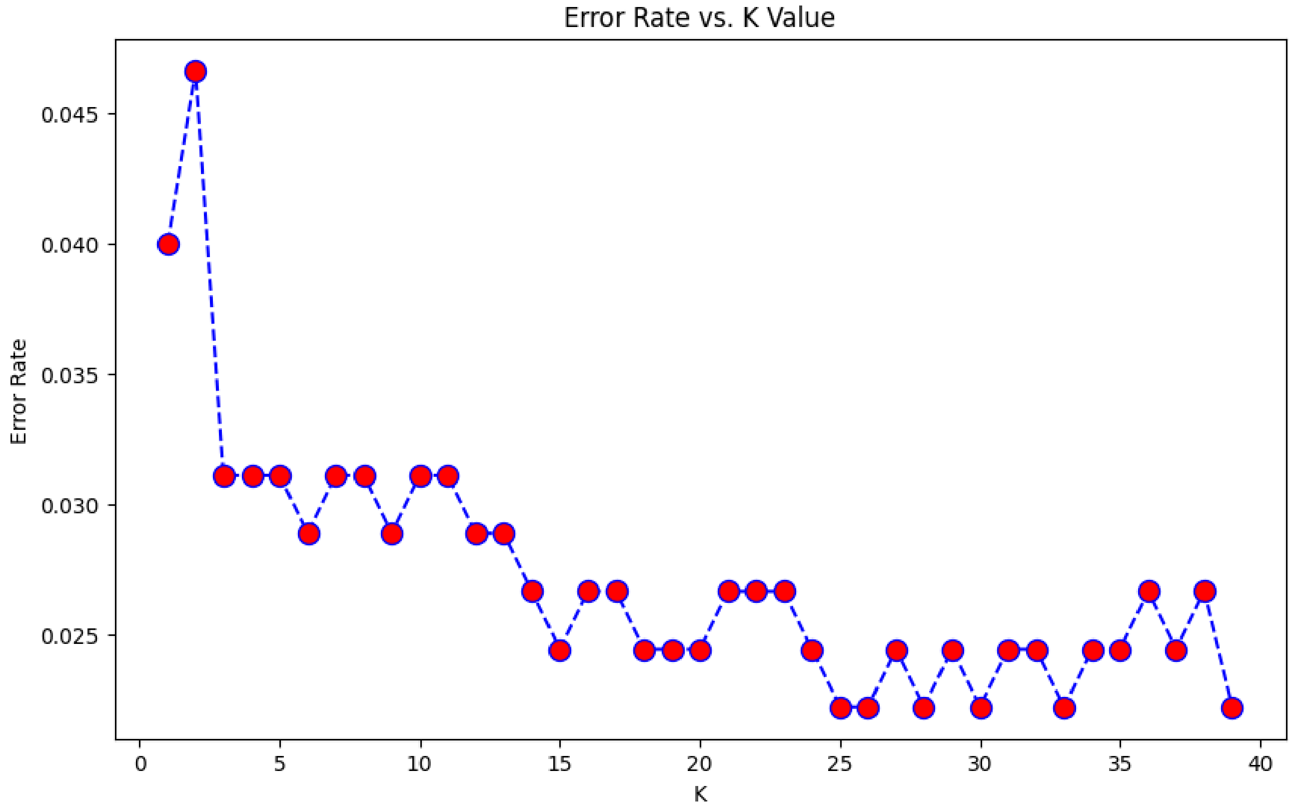
```
              precision    recall  f1-score   support

           0       0.95      0.93      0.94       153
           1       0.97      0.97      0.97       297

    accuracy                           0.96       450
   macro avg       0.96      0.95      0.96       450
weighted avg       0.96      0.96      0.96       450
```

Choosing a K Value

Let's go ahead and use the elbow method to pick a good K Value:

```
error_rate = []

# Will take some time
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))


plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

```
Text(0, 0.5, 'Error Rate')
```



Here we can see that that after arouns K>23 the error rate just tends to hover around 0.06-0.05 Let's retrain the model with that and check the classification report!

```
# K=1
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=1


[[143  10]
 [  8 289]]


              precision    recall  f1-score   support

           0       0.95      0.93      0.94       153
           1       0.97      0.97      0.97       297

    accuracy                           0.96       450
   macro avg       0.96      0.95      0.96       450
weighted avg       0.96      0.96      0.96       450
```