COCSC17 MACHINE LEARNING – LAB FILE

AMOGH GARG – 2020UCO1688

COE - 3

Netaji Subhas University of Technology, New Delhi

INDEX

Experiment Number	Topic
1	Linear Regression
2	Logistic Regression
3	K Nearest Neighbours
4	K Means Clustering
5	Decision Tree
6	Bayesian Classification
7	Convolutional Neural Networks

EXPERIMENT – 1

TOPIC: Linear Regression

CODE AND OUTPUT:

jupyter Linear_Regression Last Checkpoint: 09/08/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Markdown

Problem Statement :- Predicts the selling price of second-hand cars Using LinearRegression Machine learning algorithm.

1) Import necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# %matplotlib inline
```

2) Load Dataset

In [2]:
```
data = pd.read_csv("car data.csv")
```

About Dataset

This dataset contains information about used cars.

This data can be used for a lot of purposes such as price prediction to exemplify the use of linear regression in Machine Learning. The columns in the given dataset are as follows:

1) name

2) year

3) selling_price

4) km_driven

jupyter Linear_Regression Last Checkpoint: 09/08/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Markdown

In [3]:
```
data.head() # checking the first five rows from the dataset
```

Out[3]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

In [4]:
```
data.shape ## printing the no. of columns and rows of the dataframe
```

Out[4]: (301, 9)

In [5]:
```
data.info() # printing the summary of the dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Car_Name       301 non-null    object
 1   Year           301 non-null    int64
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Kms_Driven     301 non-null    int64
 5   Fuel_Type      301 non-null    object
 6   Seller_Type    301 non-null    object
 7   Transmission   301 non-null    object
 8   Owner          301 non-null    int64
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

jupyter Linear_Regression Last Checkpoint: 09/08/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Markdown

In [6]:
```
data.isnull().sum() # finding the count of missing values from different columns
```

Out[6]:
```
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Kms_Driven       0
Fuel_Type        0
Seller_Type      0
Transmission     0
Owner            0
dtype: int64
```

In [7]:
```
data.describe() # Statistical Information
```

Out[7]:

	Year	Selling_Price	Present_Price	Kms_Driven	Owner
count	301.000000	301.000000	301.000000	301.000000	301.000000
mean	2013.627907	4.661296	7.628472	36947.205980	0.043189
std	2.891554	5.082812	8.644115	38886.883882	0.247915
min	2003.000000	0.100000	0.320000	500.000000	0.000000
25%	2012.000000	0.900000	1.200000	15000.000000	0.000000
50%	2014.000000	3.600000	6.400000	32000.000000	0.000000
75%	2016.000000	6.000000	9.900000	48767.000000	0.000000
max	2018.000000	35.000000	92.600000	500000.000000	3.000000

Now the data looks good and there are no missing values. Also, the first cloumn is just Car_Name, so we don' need that column. Let's drop it from data and make it more clean.

In [8]:
```
data= data.drop(columns = ['Car_Name'])
```

In [8]:
```
data= data.drop(columns = ['Car_Name'])
data.head()
```

Out[8]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

Let's visualize the data and analyze the relationship between independent and dependent variables:

In [9]:
```
Categorical_features = [col for col in data.columns if data[col].dtype == 'O']
Categorical_features
```

Out[9]:
```
['Fuel_Type', 'Seller_Type', 'Transmission']
```

In [10]:
```
numerical_features = [col for col in data.columns if data[col].dtype != 'O']
numerical_features
```

Out[10]:
```
['Year', 'Selling_Price', 'Present_Price', 'Kms_Driven', 'Owner']
```

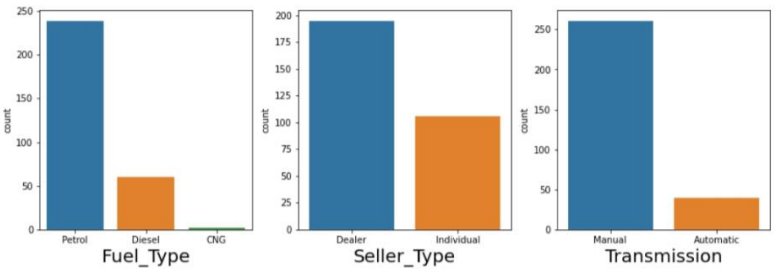
In [11]:
```
# let's see how numerical_features is distributed for every column
plt.figure(figsize=(20,25))
plotnumber = 1

for column in numerical_features:
    if plotnumber<=5 :
        ax = plt.subplot(4,5,plotnumber)
        sns.histplot(data[column],kde = True)
```

jupyter Linear_Regression Last Checkpoint: 09/08/2022 (autosaved) Logout
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)
Run Markdown

In [12]:
```
# let's see how data is distributed for every column
plt.figure(figsize=(20,20), facecolor='white')
plotnumber = 1

for column in Categorical_features:
    if plotnumber<=3 :
        ax = plt.subplot(5,5,plotnumber)
        sns.countplot(x = column,data = data)
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



Handling Categorical variables with More than Two Categories

Let's create a new column for Fuel_Type , Seller_Type , Transmission Columns.

jupyter Linear_Regression Last Checkpoint: 09/08/2022 (autosaved) Logout
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)
Run Markdown

In [13]:
```
data = pd.get_dummies(data)
data.head()
```

Out[13]:

	Year	Selling_Price	Present_Price	Kms_Driven	Owner	Fuel_Type_CNG	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Dealer	Seller_Type_Individual	Tran
0	2014	3.35	5.59	27000	0	0	0	1	1	0	
1	2013	4.75	9.54	43000	0	0	1	0	1	0	
2	2017	7.25	9.85	6900	0	0	0	1	1	0	
3	2011	2.85	4.15	5200	0	0	0	1	1	0	
4	2014	4.60	6.87	42450	0	0	1	0	1	0	

What does the encoding say?

1) For "Fuel_Type" Column

CNG is encoded as Fuel_Type_CNG = 1 , Fuel_Type_Diesel = 0 & Fuel_Type_Petrol = 0

Diesel is encoded as Fuel_Type_CNG = 0 , Fuel_Type_Diesel = 1 and Fuel_Type_Petrol = 0

Petrol is encoded as Fuel_Type_CNG = 0 , Fuel_Type_Diesel = 0 and Fuel_Type_Petrol = 1

2) For "Seller" Column

Dealer is encoded as Seller_Type_Dealer = 1 & Seller_Type_Individual = 0

Individual is encoded as Seller_Type_Dealer = 0 & Seller_Type_Individual = 1

3) For "Transmission" Column

Automatic is encoded as Transmission_Automatic = 1 & Transmission_Manual = 0

jupyter Linear_Regression Last Checkpoint: 09/08/2022 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Markdown

In [14]:
```
# create X(Independent data) and y(dependent data)
feature = ["Year","Present_Price","Kms_Driven","Owner","Fuel_Type_CNG","Fuel_Type_Diesel","Seller_Type_Dealer","Transmission_Auto
X = data[feature]
y = data.Selling_Price
```

In [15]:
```
X.head()
```

Out[15]:

	Year	Present_Price	Kms_Driven	Owner	Fuel_Type_CNG	Fuel_Type_Diesel	Seller_Type_Dealer	Transmission_Automatic
0	2014	5.59	27000	0	0	0	1	0
1	2013	9.54	43000	0	0	1	1	0
2	2017	9.85	6900	0	0	0	1	0
3	2011	4.15	5200	0	0	0	1	0
4	2014	6.87	42450	0	0	1	1	0

In [16]:
```
y.head()
```

Out[16]:
```
0    3.35
1    4.75
2    7.25
3    2.85
4    4.60
Name: Selling_Price, dtype: float64
```

Divide The Dataset into train and test data

In [17]:
```
from sklearn.model_selection import train_test_split
```

In [18]:
```
x_train, x_test, y_train, y_test = train_test_split(X,y,random_state=5)
```

jupyter Linear_Regression Last Checkpoint: 09/08/2022 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Markdown

Implementation of LinearRegression Model

In [20]:
```
from sklearn.linear_model import LinearRegression
```

In [21]:
```
lr = LinearRegression()
lr.fit(x_train,y_train)
```

Out[21]:
```
LinearRegression()
```

In [22]:
```
prediction = lr.predict(x_test)
```

In [23]:
```
lr.score(x_train,y_train) ## training score
```

Out[23]:
```
0.8799908666232626
```

In [24]:
```
lr.score(x_test,y_test) ## test score
```

Out[24]:
```
0.8608525898496743
```

In [25]:
```
plt.scatter(y_test,prediction) ## plot y_test v/s prediction value of model
```

Out[25]:
```
<matplotlib.collections.PathCollection at 0x1f2cef79f60>
```

jupyter Linear_Regression Last Checkpoint: 09/08/2022 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Markdown

Evolution Metrics of model

In [26]:
```
from sklearn import metrics
```

In [27]:
```
print('MAE : ',metrics.mean_absolute_error(y_test,prediction))
print('MSE : ',metrics.mean_squared_error(y_test,prediction))
print('RMSE : ',np.sqrt(metrics.mean_squared_error(y_test,prediction)))
print('R squared : ',metrics.r2_score(y_test,prediction))
```

```
MAE :  1.1409961087854525
MSE :  3.281596341663872
RMSE :  1.8115176901327439
R squared :  0.8608525898496743
```

Implementation of Regularization Model

1) LASSO (L1-Norm)

In [28]:
```
from sklearn.linear_model  import Ridge,Lasso,RidgeCV, LassoCV, ElasticNet, ElasticNetCV
```

In [29]:
```
# LassoCV will return best alpha and coefficients after performing 10 cross validations
lassocv = LassoCV(alphas = None,cv =10, max_iter = 100000,normalize = True)
lassocv.fit(x_train, y_train)
```

```
C:\Users\AMOGH GARG\AppData\Roaming\Python\Python310\site-packages\sklearn\linear_model\_base.py:141: FutureWarning: 'normaliz
e' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), Lasso())
```

jupyter Linear_Regression Last Checkpoint: 09/08/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3 (ipykernel)

Run | Markdown

In [31]:
```
#now that we have best parameter, let's use Lasso regression and see how well our data has fitted before
lasso_reg = Lasso(alpha)
lasso_reg.fit(x_train, y_train)
```

Out[31]:
```
Lasso(alpha=0.014905096676743052)
```

In [32]:
```
prediction = lasso_reg.predict(x_test)
```

In [33]:
```
print('MAE : ',metrics.mean_absolute_error(y_test,prediction))
print('MSE : ',metrics.mean_squared_error(y_test,prediction))
print('RMSE : ',np.sqrt(metrics.mean_squared_error(y_test,prediction)))
print('R squared : ',metrics.r2_score(y_test,prediction))
```

```
MAE :  1.1318151217096615
MSE :  3.3386758590846632
RMSE :  1.827204383500834
R squared :  0.8584322839391348
```

2) Ridge(L2-Norm)

In [34]:
```
# RidgeCV will return best alpha and coefficients after performing 10 cross validations. We will pass an array of random numbers

alphas = np.random.uniform(low=0, high=10, size=(50,))
ridgecv = RidgeCV(alphas = alphas,cv=10,normalize = True)
ridgecv.fit(x_train, y_train)
```

```
C:\Users\AMOGH GARG\AppData\Roaming\Python\Python310\site-packages\sklearn\linear_model\_base.py:141: FutureWarning: 'normali
ze' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing stage. To reproduce the previous behavio
```

jupyter Linear_Regression Last Checkpoint: 09/08/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3 (ipykernel)

Run | Markdown

Prediction

In [45]:
```
Year = int(input("Enter Year of Purchase : "))
Present_Price = int(input("Enter Showroom Price of your car : "))
Kms_Driven = int(input("How many kilometer drive : "))
Owner = int(input("How much owners previously the car had? (0,1 or 2) : "))
Fuel_Type = str(input("What is fuel type? (CNG , DIESEL , PETROL) : "))
Seller_Type = str(input("Are u Dealer or Individual ? : "))
Transmission = str(input("Enter Transmission Type : (Automatic , Manual Car) : "))
```

```
Enter Year of Purchase : 2015
Enter Showroom Price of your car : 8
How many kilometer drive : 6000
How much owners previously the car had? (0,1 or 2) : 1
What is fuel type? (CNG , DIESEL , PETROL) : CNG
Are u Dealer or Individual ? : Individual
Enter Transmission Type : (Automatic , Manual Car) : Manual
```

In [46]:
```
if Fuel_Type == "CNG" :
    Fuel_Type_CNG , Fuel_Type_Diesel = 1 , 0
elif Fuel_Type == "DIESEL":
    Fuel_Type_CNG , Fuel_Type_Diesel = 0 , 1
else:
    Fuel_Type_CNG , Fuel_Type_Diesel = 0 , 0
```

In [47]:
```
if Seller_Type == "Dealer":
    Seller_Type_Dealer = 1
else:
    Seller_Type_Dealer = 0
```

In [48]:
```
if Transmission == "Automatic":
    Transmission_Automatic = 1
else:
    Transmission_Automatic = 0
```

jupyter Linear_Regression Last Checkpoint: 09/08/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3 (ipykernel)

Run | Markdown

What is fuel type? (CNG , DIESEL , PETROL) : CNG
Are u Dealer or Individual ? : Individual
Enter Transmission Type : (Automatic , Manual Car) : Manual

In [46]:
```
if Fuel_Type == "CNG" :
    Fuel_Type_CNG , Fuel_Type_Diesel = 1 , 0
elif Fuel_Type == "DIESEL":
    Fuel_Type_CNG , Fuel_Type_Diesel = 0 , 1
else:
    Fuel_Type_CNG , Fuel_Type_Diesel = 0 , 0
```

In [47]:
```
if Seller_Type == "Dealer":
    Seller_Type_Dealer = 1
else:
    Seller_Type_Dealer = 0
```

In [48]:
```
if Transmission == "Automatic":
    Transmission_Automatic = 1
else:
    Transmission_Automatic = 0
```

In [49]:
```
prediction_value = lr.predict([[Year,Present_Price,Kms_Driven,Owner,Fuel_Type_CNG,Fuel_Type_Diesel,Seller_Type_Dealer,Transmissio
print("Predicted value : ",prediction_value[0])
```

```
Predicted value :  3.115405892397007

C:\Users\AMOGH GARG\AppData\Roaming\Python\Python310\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feat
ure names, but LinearRegression was fitted with feature names
  warnings.warn(
```

In [ ]:

EXPERIMENT – 2

TOPIC: Logistic Regression

CODE AND OUTPUT:

jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Python Implementation

In [1]:
```
#Let's start with importing necessary libraries

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

In [2]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
```

In [4]:
```
data = pd.read_csv("diabetes.csv") # Reading the Data
data.head() # top 5 rows
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [5]:
```
data.tail() ## last 5 rows
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0

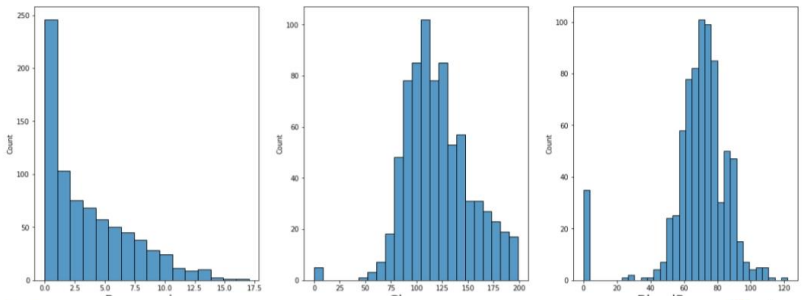
jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Explotary Data Analysis :-

In [12]:
```
# let's see how data is distributed for every column
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in data:
    if plotnumber<=9 :     # as there are 9 columns in the data
        ax = plt.subplot(3,3,plotnumber)
        sns.histplot(data[column])
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.show()
```



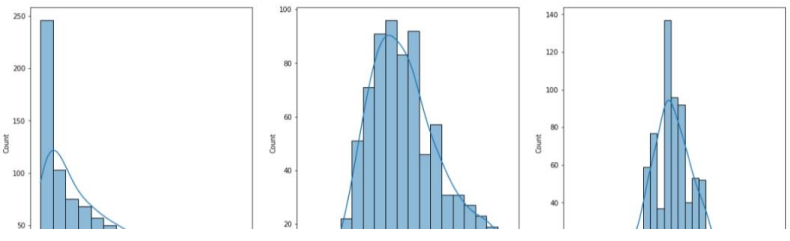
jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [13]:
```
# replacing zero values with the mean of the column
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure'].mean())
data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness'].mean())
```

In [14]:
```
# let's see how data is distributed for every column
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in data:
    if plotnumber<=9 :
        ax = plt.subplot(3,3,plotnumber)
        sns.histplot(data[column],kde=True)
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.show()
```



jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)

Logout

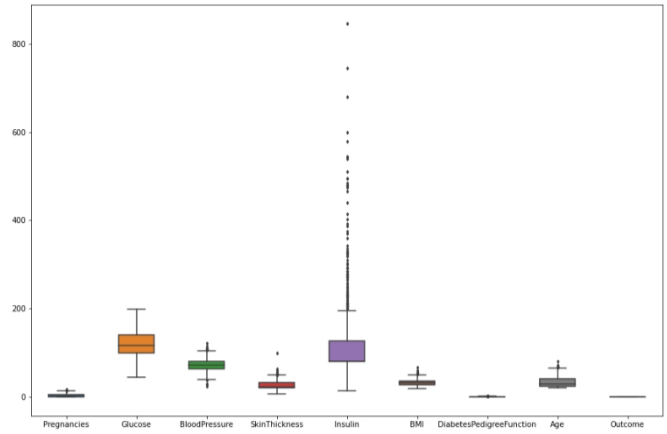
File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run | Markdown

Now we have dealt with the 0 values and data looks better. But, there still are outliers present in some columns. Let's deal with them.

In [15]:
```
fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=data, width= 0.5,ax=ax,  fliersize=3)
```

Out[15]: <AxesSubplot:>



"BMI","Pregnancies","BloodPressure","SkinThickness","Insulin","Age" columns contain Outilers.

jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run | Markdown

In [16]:
```
## Display percentage of Outilers
for k, v in data.items():
        q1 = v.quantile(0.25)
        q3 = v.quantile(0.75)
        inter_q = q3 - q1
        v_col = v[(v <= q1 - 1.5 * inter_q) | (v >= q3 + 1.5 * inter_q)]
        perc = np.shape(v_col)[0] * 100.0 / np.shape(data)[0]
        print("Column %s outliers = %.2f%%" % (k, perc))
```

```
Column Pregnancies outliers = 0.52%
Column Glucose outliers = 0.00%
Column BloodPressure outliers = 2.21%
Column SkinThickness outliers = 1.56%
Column Insulin outliers = 11.59%
Column BMI outliers = 1.04%
Column DiabetesPedigreeFunction outliers = 3.78%
Column Age outliers = 1.17%
Column Outcome outliers = 0.00%
```

In [17]:
```
q = data['Pregnancies'].quantile(0.99)
# we are removing the top 1% data from the Pregnancies column
data_cleaned = data[data['Pregnancies']<q]

q = data_cleaned['BMI'].quantile(0.99)
# we are removing the top 1% data from the BMI column
data_cleaned  = data_cleaned[data_cleaned['BMI']<q]

q = data_cleaned['SkinThickness'].quantile(0.99)
# we are removing the top 1% data from the SkinThickness column
data_cleaned  = data_cleaned[data_cleaned['SkinThickness']<q]

q = data_cleaned['Insulin'].quantile(0.95)
# we are removing the top 5% data from the Insulin column
data_cleaned  = data_cleaned[data_cleaned['Insulin']<q]
```

jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)

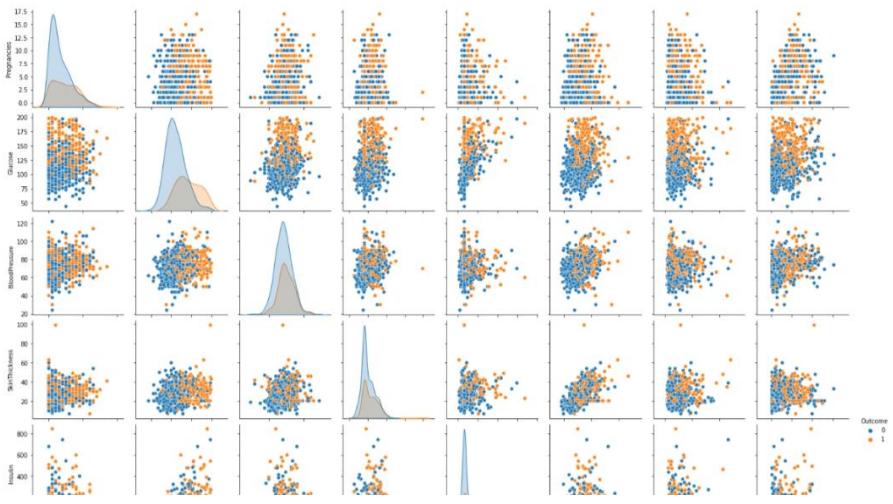
Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run | Markdown

In [19]:
```
sns.pairplot(data = data,hue = "Outcome")
```

Out[19]: <seaborn.axisgrid.PairGrid at 0x21d2f15d9c0>



In [26]:
```
plt.figure(figsize=(15,10), facecolor='white')
sns.heatmap(data=data.corr(), annot=True)
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x2adb5b86e80>

jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)
Run Markdown

Normalization

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. There are many types of Normalization techniques :-

1) Guassian Transformation

2) Min-Max Scaling

Normalization is good to use when you know that the distribution of your data does not follow a Gaussian distribution. This can be useful in algorithms that do not assume any distribution of the data like K-Nearest Neighbors and Neural Networks.

In [21]:
```
import scipy.stats as stat
import pylab
import matplotlib.pyplot as plt
```

In [22]:
```
## Function to check normal distribution
def plot_data(d,feature) :
    plt.figure(figsize = (10,6))
    plt.subplot(1,2,1)
    d[feature].hist()
    plt.subplot(1,2,2)
    stat.probplot(d[feature],dist = 'norm',plot=pylab)
    plt.show()
```

If our data is not normally distributed then we transform it.

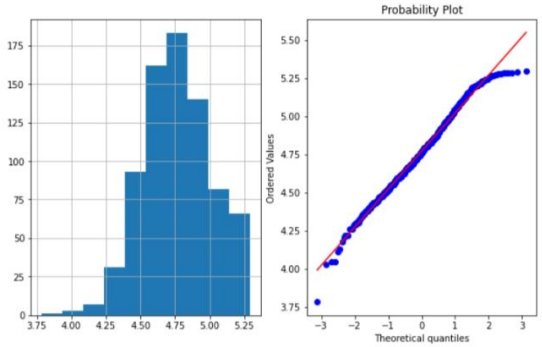
1) for "Glucose" Column

In [23]:
```
plot_data(data, "Glucose")
```

Probability Plot

jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)
Run Markdown

In [25]:
```
# Using Logarithmic Transformation
data['log_Glucose'] = np.log(data['Glucose'])
plot_data(data,'log_Glucose')
```



Observation : logarithmic Transformation is best fit for "Glucose" Column

2) for "BMI" Column

In [26]:
```
plot_data(data, "BMI")
```

Data set after Normalization

In [45]:
```
data.head()
```

Out[45]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	log_Glucose	...	Reci_Age	Exp_Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50	1	4.997212	...	0.020000	15415.622286
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31	0	4.442651	...	0.032258	4744.676155
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32	1	5.209486	...	0.031250	5130.913541
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21	0	4.488636	...	0.047619	1816.647065
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33	1	4.919981	...	0.030303	5535.245172

5 rows × 22 columns

In [46]:
```
dataset = data[["log_BMI","log_Glucose","log_Age","Sq_Pregnancies","Reci_Insulin","log_SkinThickness","log_DiabetesPedigreeFunct
dataset.head()
```

Out[46]:

	log_BMI	log_Glucose	log_Age	Sq_Pregnancies	Reci_Insulin	log_SkinThickness	log_DiabetesPedigreeFunction	log_BloodPressure	Outcome
0	3.514526	4.997212	3.912023	2.449490	0.012531	3.555348	-0.466809	4.276666	1
1	3.280911	4.442651	3.433987	1.000000	0.012531	3.367296	-1.046969	4.189655	0
2	3.148453	5.209486	3.465736	2.828427	0.012531	3.022202	-0.397497	4.158883	1
3	3.335770	4.488636	3.044522	1.000000	0.010638	3.135494	-1.789761	4.189655	0
4	3.763523	4.919981	3.496508	0.000000	0.005952	3.555348	0.827678	3.688879	1

Standardization

jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted | Python 3 (ipykernel)
Run Markdown

In [47]:
```
X = dataset.drop(columns = ['Outcome'])
y = dataset['Outcome']
```

In [48]:
```
scalar = StandardScaler()
X_scaled = scalar.fit_transform(X)
```

In [49]:
```
X_scaled
```

Out[49]:
```
array([[ 0.27009922,  0.90856252,  1.43637931, ...,  0.95720591,
         0.76584846,  0.06379277],
       [-0.83876157, -1.31367612, -0.04593931, ...,  0.42067547,
        -0.13515874, -0.43414786],
       [-1.46747726,  1.75918707,  0.05250873, ..., -0.56390989,
         0.87349186, -0.61024505],
       ...,
       [-0.91068013,  0.10142474, -0.14761572, ..., -0.24067602,
        -0.69351704,  0.06379277],
       [-0.25202375,  0.26368185,  1.2445128 , ..., -0.56390989,
        -0.14403323, -0.97958011],
       [-0.2049503 , -0.95323526, -0.97152211, ...,  0.61095214,
        -0.30321784, -0.09742091]])
```

In [50]:
```
dataset_scaled = pd.DataFrame(X_scaled,columns=dataset.columns[:-1])
dataset_scaled.head()
```

Out[50]:

	log_BMI	log_Glucose	log_Age	Sq_Pregnancies	Reci_Insulin	log_SkinThickness	log_DiabetesPedigreeFunction	log_BloodPressure
0	0.270099	0.908563	1.436379	0.765422	0.158418	0.957206	0.765848	0.063793
1	-0.838762	-1.313676	-0.045939	-0.706001	0.158418	0.420675	-0.135159	-0.434148
2	-1.467477	1.759187	0.052509	1.150094	0.158418	-0.563910	0.873492	-0.610245
3	-0.578375	-1.129404	-1.253612	-0.706001	-0.131039	-0.240676	-1.288739	-0.434148
4	1.451972	0.599081	0.147927	-1.721134	-0.847514	0.957206	2.776227	-3.299938

jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted | Python 3 (ipykernel)
Run Markdown

Splitting Of Data Into Train And test

In [51]:
```
x_train,x_test,y_train,y_test = train_test_split(dataset_scaled,y, test_size= 0.25, random_state = 355)
```

In [52]:
```
x_train.shape , x_test.shape
```

Out[52]:
```
((576, 8), (192, 8))
```

Implementation of Logistic Regression Algorithm

In [53]:
```
log_reg = LogisticRegression()

log_reg.fit(x_train,y_train)
```

Out[53]:
```
LogisticRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Let's see how well our model performs on the test data set.

In [54]:
```
y_pred = log_reg.predict(x_test)
```

In [55]:
```
accuracy = accuracy_score(y_test,y_pred)
accuracy
```

Out[55]:
```
0.75
```

In [56]:
```
# Confusion Matrix
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

In [57]:
```
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
```

In [58]:
```
# Breaking down the formula for Accuracy
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negative + true_negative)
Accuracy
```

Out[58]: 0.75

In [59]:
```
# Precison
Precision = true_positive/(true_positive+false_positive)
Precision
```

Out[59]: 0.856

In [60]:
```
# Recall
Recall = true_positive/(true_positive+false_negative)
Recall
```

Out[60]: 0.781021897810219

In [61]:
```
# F1 Score
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

Out[61]: 0.816793893129771

In [62]:
```
# Area Under Curve
auc = roc_auc_score(y_test, y_pred)
auc
```

Out[62]: 0.7041194029850747

jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)

Logout

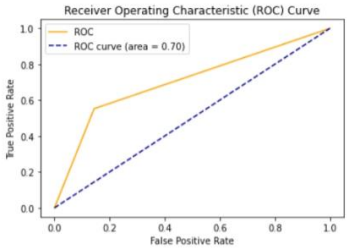
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

In [63]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

In [64]:
```
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f)' % auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



In [65]:
```
import pickle
# Writing different model files to file
with open( 'modelForPrediction.sav', 'wb') as f:
    pickle.dump(log_reg,f)

with open('sandardScalar.sav', 'wb') as f:
    pickle.dump(scalar,f)
```

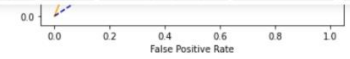
jupyter Logistic Regression Last Checkpoint: 09/15/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown



In [65]:
```
import pickle
# Writing different model files to file
with open( 'modelForPrediction.sav', 'wb') as f:
    pickle.dump(log_reg,f)

with open('sandardScalar.sav', 'wb') as f:
    pickle.dump(scalar,f)
```

Advantages of Logisitic Regression

- It is very simple and easy to implement.
- The output is more informative than other classification algorithms
- It expresses the relationship between independent and dependent variables
- Very effective with linearly seperable data

Disadvantages of Logisitic Regression

- Not effective with data which are not linearly seperable
- Not as powerful as other classification models
- Multiclass classifications are much easier to do with other algorithms than logisitic regression
- It can only predict categorical outcomes

In [ ]:

EXPERIMENT – 3

TOPIC: K Nearest Neighbours

CODE AND OUTPUT:

jupyter K Nearest Neighbors Last Checkpoint: 09/29/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

Import Libraries

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

Import the Data

In [7]:
```
df = pd.read_csv("D:/NSUT Work/Machine Learning/Decision Tree and KNN/KNN/Classified Data",index_col=0)
```

In [8]:
```
df.head() # display top 5 observation
```

Out[8]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1
4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.419167	1

In [4]:
```
df.tail() #display last 5 observation
```

Out[4]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
995	1.010953	1.034006	0.853116	0.622460	1.036610	0.586240	0.746811	0.319752	1.117340	1.348517	1
996	0.575529	0.955786	0.941835	0.792882	1.414277	1.269540	1.055928	0.713193	0.958684	1.663489	0
997	1.135470	0.982462	0.781905	0.916738	0.901031	0.884738	0.386802	0.389584	0.919191	1.385504	1

jupyter K Nearest Neighbors Last Checkpoint: 09/29/2022 (autosaved)

Logout

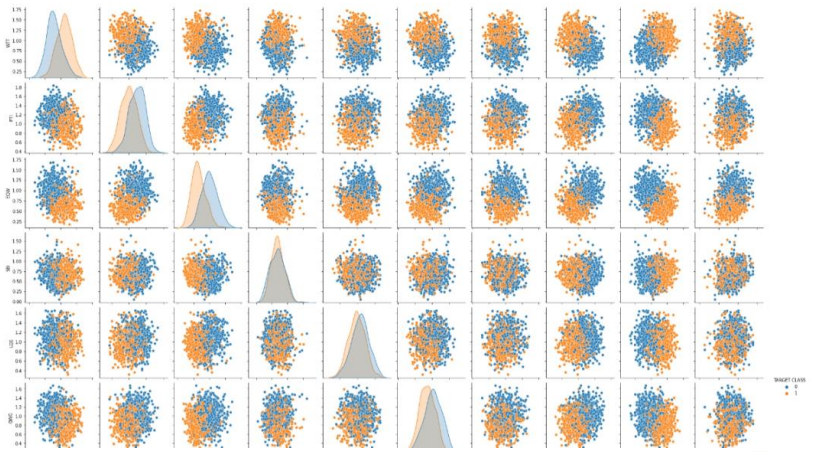
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

In [8]:
```
sns.pairplot(df,hue='TARGET CLASS')
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x283cd0c54c0>
```



jupyter K Nearest Neighbors Last Checkpoint: 09/29/2022 (autosaved)

Logout

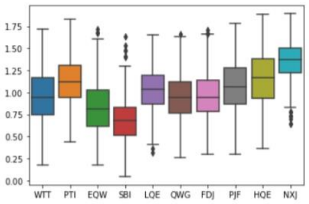
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

In [9]:
```
sns.boxplot(data = df.drop('TARGET CLASS',axis=1))
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x283d58a1d90>
```



Students themselves try to visualize the dataset using another graph.

Standardize the Variables

Since the KNN classifier predicts the class of a given experimental observation by determining the observations closest to it, the scaling of the variables is important. All large-scale variables will have a much larger effect on the interval between observations, and therefore on the KNN classifier, than small-scale variables.

In [10]:
```
from sklearn.preprocessing import StandardScaler
```

In [11]:
```
scaler = StandardScaler()
```

In [12]:
```
scaler.fit(df.drop('TARGET CLASS',axis=1)) # drop the target class from the dataset
```

Out[12]:
```
StandardScaler()
```

In [13]:
```
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))
```

In [14]:
```
## create the dataframe of scaled features
data = pd.DataFrame(scaled_features,columns=df.columns[:-1])
data.head()
```

Out[14]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772	0.276510

Split the dataset into train and test

In [15]:
```
from sklearn.model_selection import train_test_split
```

In [16]:
```
## divided the dataset into 70% of train data and 30% of test data
X_train, X_test, y_train, y_test = train_test_split(data,df['TARGET CLASS'],test_size=0.30)
```

In [17]:
```
X_train.shape , X_test.shape , y_train.shape , y_test.shape
```

Out[17]:
```
((700, 10), (300, 10), (700,), (300,))
```

jupyter K Nearest Neighbors Last Checkpoint: 09/29/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run Markdown

Implement KNN Model

In [18]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [19]:
```
knn = KNeighborsClassifier(n_neighbors=3) ## choose k = 3
```

In [20]:
```
knn.fit(X_train,y_train)
```

Out[20]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [21]:
```
pred = knn.predict(X_test) ## prediction of test data
```

In [22]:
```
pred
```

Out[22]:
```
array([1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1,
       1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0], dtype=int64)
```

Predictions and Evaluations of KNN model

jupyter K Nearest Neighbors Last Checkpoint: 09/29/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run Markdown

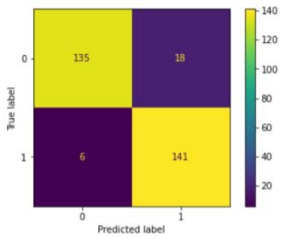
In [23]:
```
from sklearn.metrics import classification_report,confusion_matrix , plot_confusion_matrix
from sklearn.model_selection import cross_val_score
```

In [24]:
```
print(confusion_matrix(y_test,pred)) ## confusion matrix
```

```
[[135  18]
 [  6 141]]
```

In [25]:
```
plot_confusion_matrix(knn,X_test,y_test)
```

Out[25]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x283d0c325e0>
```



In [26]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.88      0.92       153
           1       0.89      0.96      0.92       147
```

Choosing a K Value by using the elbow method to pick a good K Value

In [27]:
```
accuracy_rate = []
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,data,df['TARGET CLASS'],cv=10)
    accuracy_rate.append(score.mean())
```

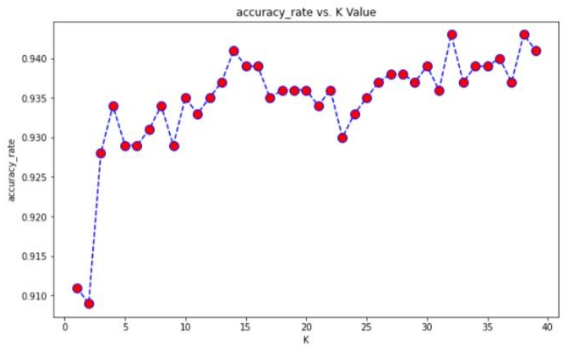
In [28]:
```
accuracy_rate
```

Out[28]:
```
[0.9109999999999999,
 0.909,
 0.9280000000000002,
 0.9339999999999999,
 0.9289999999999999,
 0.929,
 0.9310000000000003,
 0.9340000000000002,
 0.9289999999999999,
 0.9350000000000002,
 0.9329999999999998,
 0.9350000000000002,
 0.937,
 0.9410000000000001,
 0.9390000000000001,
 0.9390000000000001,
 0.9349999999999999,
 0.9360000000000002,
 0.9360000000000002,
 0.9360000000000002,
 0.9339999999999999,
 0.9359999999999999,
 0.93,
```

jupyter K Nearest Neighbors Last Checkpoint: 09/29/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted | Python 3 (ipykernel)
Run Markdown

In [29]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),accuracy_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('accuracy_rate vs. K Value')
plt.xlabel('K')
plt.ylabel('accuracy_rate')
```

Out[29]:
```
Text(0, 0.5, 'accuracy_rate')
```



In [30]:
```
error_rate = []
for i in range(1,40):
```

jupyter K Nearest Neighbors Last Checkpoint: 09/29/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted | Python 3 (ipykernel)
Run Markdown

In [30]:
```
error_rate = []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,data,df['TARGET CLASS'],cv=10)
    error_rate.append(1-score.mean())
```

In [31]:
```
error_rate
```

Out[31]:
```
[0.08900000000000008,
 0.09099999999999997,
 0.07199999999999984,
 0.06600000000000006,
 0.07100000000000006,
 0.07099999999999995,
 0.06899999999999973,
 0.06599999999999984,
 0.07100000000000006,
 0.06499999999999984,
 0.06700000000000017,
 0.06499999999999984,
 0.06299999999999994,
 0.05899999999999994,
 0.06099999999999994,
 0.06099999999999994,
 0.06500000000000006,
 0.06399999999999983,
 0.06399999999999983,
 0.06399999999999983,
 0.06600000000000006,
 0.06400000000000006,
 0.06999999999999995,
 0.06699999999999995,
 0.06500000000000006,
 0.06299999999999994,
```

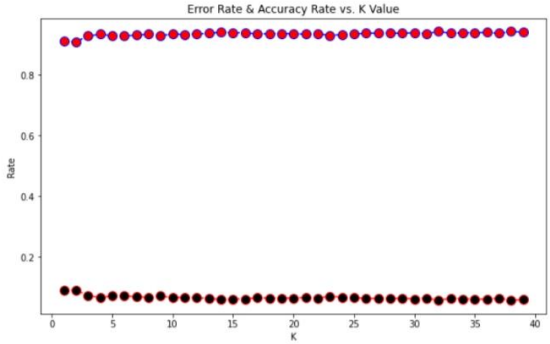
jupyter K Nearest Neighbors Last Checkpoint: 09/29/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run Markdown

In [33]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='red', linestyle='dashed', marker='o',markerfacecolor='black', markersize=10)
plt.plot(range(1,40),accuracy_rate,color='blue', linestyle='dashed', marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate & Accuracy Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Rate')
plt.show()
```



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

jupyter K Nearest Neighbors Last Checkpoint: 09/29/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run Markdown

In [34]:
```
# WITH K=23
knn = KNeighborsClassifier(n_neighbors=23)
knn.fit(X_train,y_train)
pred = knn.predict(X_test)
```

In [35]:
```
print('WITH K=23')
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=23


              precision    recall  f1-score   support

           0       0.95      0.91      0.93       153
           1       0.91      0.95      0.93       147

    accuracy                           0.93       300
   macro avg       0.93      0.93      0.93       300
weighted avg       0.93      0.93      0.93       300
```

In [36]:
```
print(confusion_matrix(y_test,pred))
```

```
[[139  14]
 [  7 140]]
```

In [37]:
```
plot_confusion_matrix(knn,X_test,y_test)
```

Out[37]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x283d68184f0>

jupyter K Nearest Neighbors Last Checkpoint: 09/29/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run Markdown

We can also implement KNN as a regression Model by using from sklearn.neighbors import KNeighborsRegressor.

Save the model

In [39]:
```
import pickle
filename = 'knn_model.pickle'
pickle.dump(knn, open(filename, 'wb'))
```

In [40]:
```
# Save the scaler model
filename_scaler = 'scaler_model.pickle'
pickle.dump(scaler, open(filename_scaler, 'wb'))
```

In [41]:
```
## load the model and predict the new feature
model = pickle.load(open(filename, 'rb'))
scaler = pickle.load(open(filename_scaler, 'rb'))
```

In [42]:
```
a = scaler.transform([[1.23 , 1.38 , 0.65 , 0.82 , 0.14 , 0.87 , 1.40 , 1.38 , 1.52 , 1.153]])
a
```

Out[42]:
```
array([[ 0.96831704,  1.03401652, -0.6318525 ,  0.60079521, -3.66777042,
        -0.28725193,  1.7121353 ,  1.06648171,  1.23215524, -1.02744284]])
```

In [43]:
```
output = model.predict(a)
output
```

Out[43]:
```
array([1], dtype=int64)
```

In [ ]:

EXPERIMENT – 4

TOPIC: K Means Clustering

CODE AND OUTPUT:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

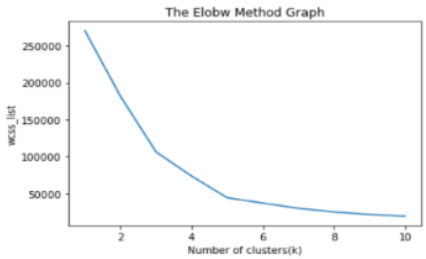
```
# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
```

```
x = dataset.iloc[:, [3, 4]].values
x
```

```
array([[ 15,  39],
       [ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
       [ 20,  77],
       [ 20,  13],
       [ 20,  79],
       [ 21,  35],
       [ 21,  66],
       [ 23,  29],
       [ 23,  98]
```

```
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```




```
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
```

```
#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

EXPERIMENT – 5

TOPIC: Decision Tree

CODE AND OUTPUT:

jupyter Decision_tree Last Checkpoint: 09/29/2022 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Implementation of Decision tree using sklearn library

Short Description About Project

Wine quality is a very important part for consumers as well as for the manufacturing industry. Every human has his or her own opinion about testing, so determining wine quality based on human experts is a daunting task. There are several characteristics for predicting wine quality, but not all are suitable for better prediction. Intended to implement machine learning classifier algorithms such as decision trees, using wine quality dataset. The red wine dataset contains 1599 cases and the white wine dataset contains Input characteristics based on physicochemical tests and output variables based on sensory data are classified into 11 quality categories from 0 to 10 (0 Very Bad to 10 Very Good).

Import libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Load dataset

In [2]:
```
data = pd.read_csv("winequality_red.csv")
data
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5

jupyter Decision_tree Last Checkpoint: 09/29/2022 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Split dataset into train and test data

In [4]:
```
X = data.drop(columns = 'quality')
y = data['quality']
```

In [5]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.30, random_state= 355)
```

In [6]:
```
x_train.shape ,x_test.shape ,y_train.shape ,y_test.shape
```

Out[6]: ((1119, 11), (480, 11), (1119,), (480,))

Implement decision tree on complete dataset

In [7]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(x_train,y_train)
```

Out[7]: DecisionTreeClassifier()

In [8]:
```
# accuracy of our classification tree for train data
model.score(x_train,y_train)
```

Out[8]: 1.0

In [9]:
```
# accuracy of our classification tree for test data
model.score(x_test,y_test)
```

Out[9]: 0.63125

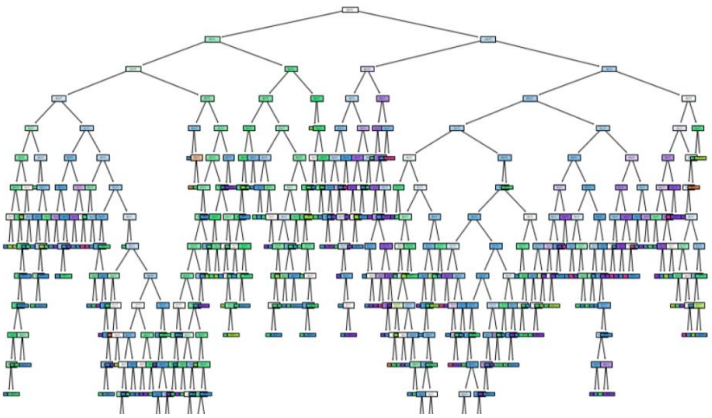
As u clearly seen our model is overfitted So, we need to remove this.

jupyter Decision_tree Last Checkpoint: 09/29/2022 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

In [10]:
```
## Draw decision tree
from sklearn import tree
fig = plt.figure(figsize = (15,15))
tree.plot_tree(model,filled = True)
plt.show()
```



jupyter Decision_tree Last Checkpoint: 09/29/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run | Markdown

In [11]:
```
# we are tuning three hyperparameters right now, we are passing the different values for both parameters
grid_param = {
    'criterion': ['gini', 'entropy'],
    'max_depth' : range(2,32,1),
    'min_samples_leaf' : range(1,10,1),
    'min_samples_split': range(2,10,1),
    'splitter' : ['best', 'random']
}
```

In [12]:
```
from sklearn.model_selection import GridSearchCV
```

In [21]:
```
grid_search = GridSearchCV(estimator=model,param_grid=grid_param,cv=7,n_jobs =-1)
```

In [22]:
```
grid_search.fit(x_train,y_train)
```

Out[22]:
```
GridSearchCV(cv=7,
             estimator=DecisionTreeClassifier(max_depth=8, min_samples_leaf=7,
                                              min_samples_split=6,
                                              random_state=89,
                                              splitter='random'),
             n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 32),
                         'min_samples_leaf': range(1, 10),
                         'min_samples_split': range(2, 10),
                         'splitter': ['best', 'random']})
```

In [23]:
```
best_parameters = grid_search.best_params_
print(best_parameters)
```

```
{'criterion': 'gini', 'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 3, 'splitter': 'best'}
```

In [24]:
```
grid_search.best_score_
```

jupyter Decision_tree Last Checkpoint: 09/29/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run | Markdown

In [25]:
```
model = DecisionTreeClassifier(criterion = 'gini', max_depth =15, min_samples_leaf= 1, min_samples_split= 3, splitter ='best',ran
model.fit(x_train,y_train)
```

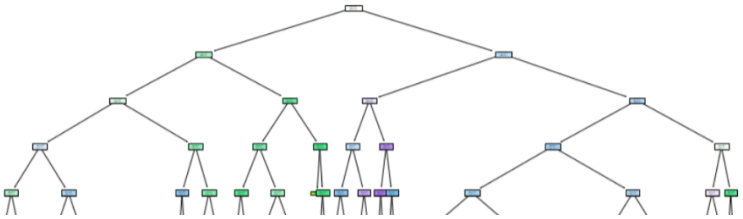
Out[25]:
```
DecisionTreeClassifier(max_depth=15, min_samples_split=3, random_state=89)
```

In [26]:
```
model.score(x_test,y_test)
```

Out[26]:
```
0.6229166666666667
```

Note : we must understand that giving all the hyperparameters in the gridSearch doesn't gurantee the best result. We have to do hit and trial with parameters to get the perfect score.

In [27]:
```
## Draw decision tree
from sklearn import tree
fig = plt.figure(figsize = (15,15))
tree.plot_tree(model,filled = True)
plt.show()
```

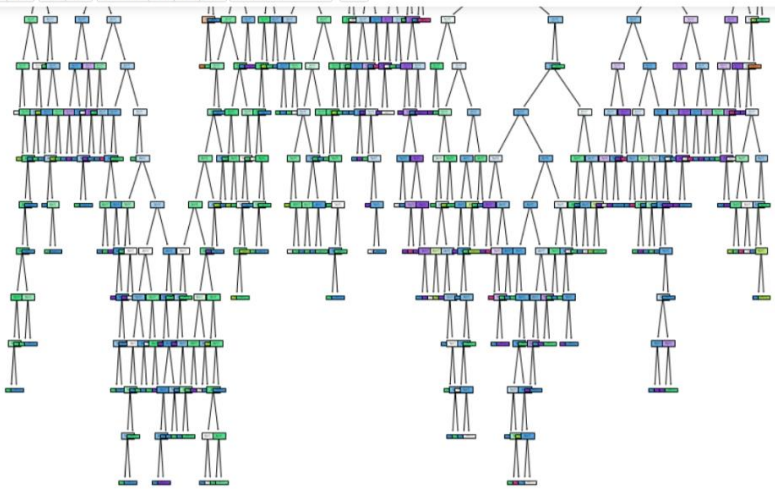


jupyter Decision_tree Last Checkpoint: 09/29/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run | Markdown



In [28]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score , classification_report
```

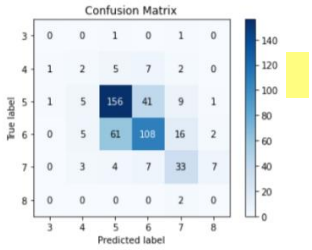
In [28]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score , classification_report
```

In [29]:
```
y_pred = model.predict(x_test)
confusion_matrix(y_test , y_pred)
```

Out[29]:
```
array([[  0,   0,   1,   0,   1,   0],
       [  1,   2,   5,   7,   2,   0],
       [  1,   5, 156,  41,   9,   1],
       [  0,   5,  61, 108,  16,   2],
       [  0,   3,   4,   7,  33,   7],
       [  0,   0,   0,   0,   2,   0]], dtype=int64)
```

In [30]:
```
import scikitplot
plt.figure(figsize = (7,7))
scikitplot.metrics.plot_confusion_matrix(y_test,y_pred)
plt.show()
```

```
<Figure size 504x504 with 0 Axes>
```



jupyter Decision_tree Last Checkpoint: 09/29/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted | Python 3 (ipykernel)
Run Markdown

In [31]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.13      0.12      0.12        17
           5       0.69      0.73      0.71       213
           6       0.66      0.56      0.61       192
           7       0.52      0.61      0.56        54
           8       0.00      0.00      0.00         2

    accuracy                           0.62       480
   macro avg       0.33      0.34      0.33       480
weighted avg       0.63      0.62      0.63       480
```

Feature selection on the bases of Information gain

Checking The Information Gain for the features :-

IG calculates the importance of each feature by measuring the increase in entropy when the feature is given vs. absent. Algorithm: IG(S, a) = H(S) — H(S | a) Where IG(S, a) is the information for the dataset S for the variable a for a random variable, H(S) is the entropy for the dataset before any change (described above) and H(S | a) is the conditional entropy for the dataset in the presence of variable a.

In [32]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif

def FeatureInfoGain(X, y):
    bestfeatures = SelectKBest(score_func=mutual_info_classif, k='all') # k is number of features you want to select
    fit = bestfeatures.fit(X,y)
    dfscores = pd.DataFrame(fit.scores_)
    dfcolumns = pd.DataFrame(X.columns)
    featureScores = pd.concat([dfcolumns,dfscores],axis=1)
```

jupyter Decision_tree Last Checkpoint: 09/29/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted | Python 3 (ipykernel)
Run Markdown

In [32]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif

def FeatureInfoGain(X, y):
    bestfeatures = SelectKBest(score_func=mutual_info_classif, k='all') # k is number of features you want to select
    fit = bestfeatures.fit(X,y)
    dfscores = pd.DataFrame(fit.scores_)
    dfcolumns = pd.DataFrame(X.columns)
    featureScores = pd.concat([dfcolumns,dfscores],axis=1)
    featureScores.columns = ['Specs','Score']
    return featureScores
```

In [33]:
```
featureScores = FeatureInfoGain(X, y)
featureScores
```

Out[33]:

	Specs	Score
0	fixed acidity	0.065815
1	volatile acidity	0.104627
2	citric acid	0.066601
3	residual sugar	0.019222
4	chlorides	0.059358
5	free sulfur dioxide	0.021889
6	total sulfur dioxide	0.081038
7	density	0.078900
8	pH	0.021998
9	sulphates	0.099010
10	alcohol	0.172690

EXPERIMENT – 6

TOPIC: Bayesian Classification

CODE AND OUTPUT:

Creating a toy dataset

The function I'm referring to resides within scikit-learn's `datasets` module. Let's create 100 data points, each belonging to one of two possible classes, and group them into two Gaussian blobs. To make the experiment reproducible, we specify an integer to pick a seed for the `random_state`. You can again pick whatever number you prefer. Here I went with Thomas Bayes' year of birth (just for kicks):

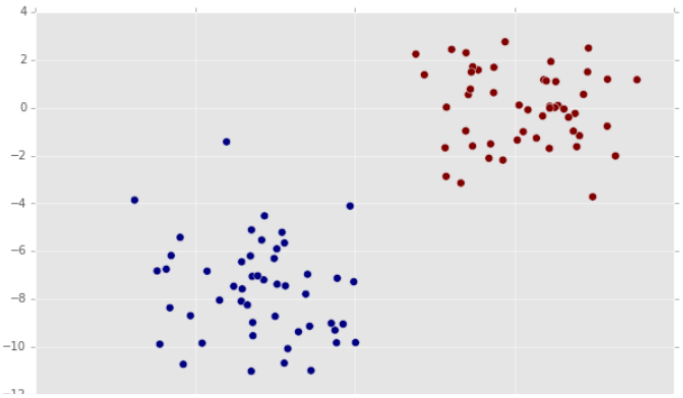
In [1]:
```
from sklearn import datasets
X, y = datasets.make_blobs(100, 2, centers=2, random_state=1701, cluster_std=2)
```

Let's have a look at the dataset we just created using our trusty friend, Matplotlib:

In [2]:
```
import matplotlib.pyplot as plt
plt.style.use('ggplot')
%matplotlib inline
```

I'm sure this is getting easier every time. We use scatter to create a scatter plot of all x values (`X[:, 0]`) and y values (`X[:, 1]`), which will result in the following output:

In [3]:
```
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], c=y, s=50);
```



In [4]:
```
import numpy as np
from sklearn import model_selection as ms
X_train, X_test, y_train, y_test = ms.train_test_split(
    X.astype(np.float32), y, test_size=0.1
)
```

Classifying the data with a normal Bayes classifier

We will then use the same procedure as in earlier chapters to train a **normal Bayes classifier**. Wait, why not a naive Bayes classifier? Well, it turns out OpenCV doesn't really provide a true naive Bayes classifier... Instead, it comes with a Bayesian classifier that doesn't necessarily expect features to be independent, but rather expects the data to be clustered into Gaussian blobs. This is exactly the kind of dataset we created earlier!

We can create a new classifier using the following function:

In [5]:
```
import cv2
model_norm = cv2.ml.NormalBayesClassifier_create()
```

Then, training is done via the `train` method:

In [6]:
```
model_norm.train(X_train, cv2.ml.ROW_SAMPLE, y_train)
```

Out[6]: True

Once the classifier has been trained successfully, it will return True. We go through the motions of predicting and scoring the classifier, just like we have done a million times before:

In [7]:
```
_, y_pred = model_norm.predict(X_test)
```

In [8]:
```
from sklearn import metrics
metrics.accuracy_score(y_test, y_pred)
```

Out[8]: 1.0

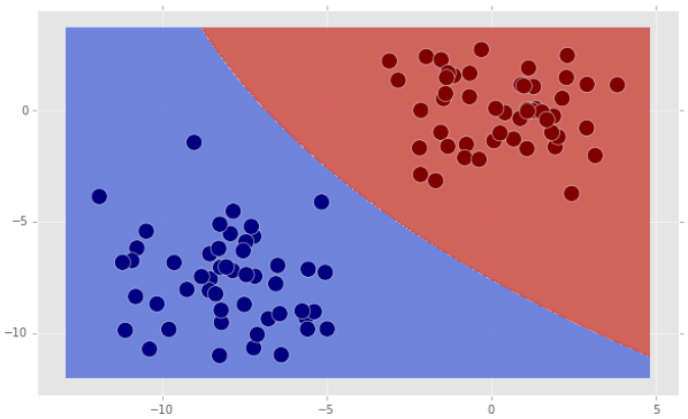
Even better—we can reuse the plotting function from the last chapter to inspect the decision boundary! If you recall, the idea was to create a mesh grid that would encompass all data points and then classify every point on the grid. The mesh grid is created via the NumPy function of the same name:

In [9]:
```
def plot_decision_boundary(model, X_test, y_test):
    # create a mesh to plot in
    h = 0.02  # step size in mesh
    x_min, x_max = X_test[:, 0].min() - 1, X_test[:, 0].max() + 1
    y_min, y_max = X_test[:, 1].min() - 1, X_test[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    X_hypo = np.column_stack((xx.ravel().astype(np.float32),
                              yy.ravel().astype(np.float32)))
    ret = model.predict(X_hypo)
    if isinstance(ret, tuple):
        zz = ret[1]
    else:
        zz = ret
    zz = zz.reshape(xx.shape)

    plt.contourf(xx, yy, zz, cmap=plt.cm.coolwarm, alpha=0.8)
    plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, s=200)
```

In [10]:
```
plt.figure(figsize=(10, 6))
plot_decision_boundary(model_norm, X, y)
```



Classifying the data with a naive Bayes classifier

We can compare the result to a true naïve Bayes classifier by asking scikit-learn for help:

In [13]:
```
from sklearn import naive_bayes
model_naive = naive_bayes.GaussianNB()
```

As usual, training the classifier is done via the `fit` method:

In [14]:
```
model_naive.fit(X_train, y_train)
```

Out[14]:
```
GaussianNB(priors=None)
```

Scoring the classifier is built in:

In [15]:
```
model_naive.score(X_test, y_test)
```

Out[15]:
```
1.0
```

Again a perfect score! However, in contrast to OpenCV, this classifier's `predict_proba` method returns true probability values, because all values are between 0 and 1, and because all rows add up to 1:

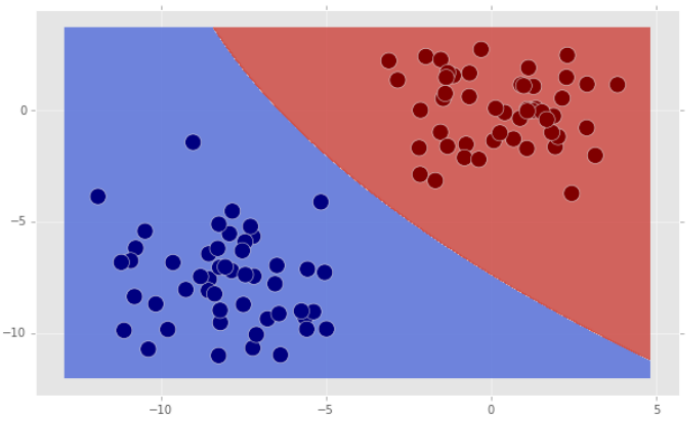
In [16]:
```
yprob = model_naive.predict_proba(X_test)
yprob.round(2)
```

Out[16]:
```
array([[ 1.,  0.],
       [ 0.,  1.],
       [ 0.,  1.],
       [ 0.,  1.],
       [ 1.,  0.],
       [ 0.,  1.],
       [ 1.,  0.],
       [ 1.,  0.],
       [ 0.,  1.],
       [ 1.,  0.]])
```

You might have noticed something else: This classifier has absolutely no doubt about the target label of each and every data point. It's all or nothing.

The decision boundary returned by the naive Bayes classifier looks slightly different, but can be considered identical to the previous command for the purpose of this

In [17]:
```
plt.figure(figsize=(10, 6))
plot_decision_boundary(model_naive, X, y)
```



Visualizing conditional probabilities

Similarly, we can also visualize probabilities. For this, we slightly modify the plot function from the previous example. We start out by creating a mesh grid between (`x_min`, `x_max`) and (`y_min`, `y_max`):

In [18]:
```
def plot_proba(model, X_test, y_test):
    # create a mesh to plot in
    h = 0.02  # step size in mesh
    x_min, x_max = X_test[:, 0].min() - 1, X_test[:, 0].max() + 1
    y_min, y_max = X_test[:, 1].min() - 1, X_test[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    X_hypo = np.column_stack((xx.ravel().astype(np.float32),
                              yy.ravel().astype(np.float32)))
    if hasattr(model, 'predictProb'):
        _, _, y_proba = model.predictProb(X_hypo)
```

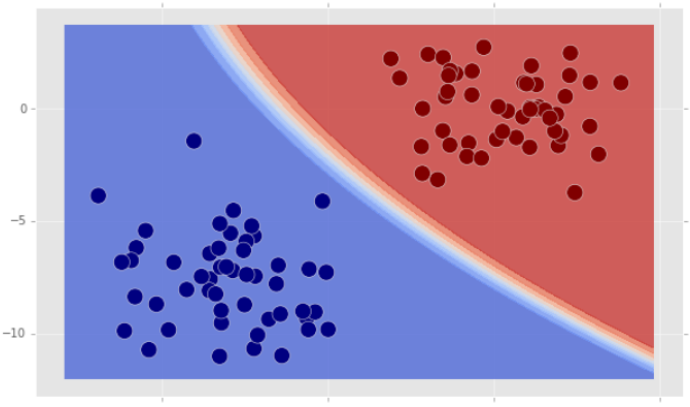
In [18]:
```
def plot_proba(model, X_test, y_test):
    # create a mesh to plot in
    h = 0.02  # step size in mesh
    x_min, x_max = X_test[:, 0].min() - 1, X_test[:, 0].max() + 1
    y_min, y_max = X_test[:, 1].min() - 1, X_test[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    X_hypo = np.column_stack((xx.ravel().astype(np.float32),
                              yy.ravel().astype(np.float32)))
    if hasattr(model, 'predictProb'):
        _, _, y_proba = model.predictProb(X_hypo)
    else:
        y_proba = model.predict_proba(X_hypo)

    zz = y_proba[:, 1] - y_proba[:, 0]
    zz = zz.reshape(xx.shape)

    plt.contourf(xx, yy, zz, cmap=plt.cm.coolwarm, alpha=0.8)
    plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, s=200)
```

In [19]:
```
plt.figure(figsize=(10, 6))
plot_proba(model_naive, X, y)
```



EXPERIMENT – 7

TOPIC: CNN

CODE AND OUTPUT:

jupyter CNN (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

```
In [1]: import numpy as np
        import pandas as pd
        %matplotlib inline
        import matplotlib as mpl
        import matplotlib.pyplot as plt
        import tensorflow as tf
        from tensorflow import keras
```

```
In [2]: #Importing Dataset
        fashion_mnist = keras.datasets.fashion_mnist
        (x_train_full,y_train_full),(x_test,y_test) = fashion_mnist.load_data()
```

```
In [3]: class_names=["T-shirt/top","Trouser","Pullover","Dress","Coat","Sandal","Shirt","Sneaker","Bag","Ankle boot"]
```

```
In [4]: #Data Reshape, for cnn we need 3D array
        x_train_full = x_train_full.reshape((60000,28,28,1)) #(1 is channel, 60000 is number of items)
        x_test = x_test.reshape((10000,28,28,1))
```

```
In [5]: #Data Normalisation
        x_train_n=x_train_full/255.    #Normalisisng pixel intensity
        x_test_n=x_test/255.
```

```
In [6]: x_valid,x_train=x_train_n[:5000],x_train_n[5000:]    #0-5000 in validation set,5000-60000 in train set
        y_valid,y_train=y_train_full[:5000],y_train_full[5000:]
        x_test=x_test_n
```

```
In [7]: #Model Architecture->Replicate same answer every time
        np.random.seed(42)
        tf.random.set_seed(42)
```

```
In [8]: model = keras.models.Sequential()
```

jupyter CNN (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

```
In [8]: model = keras.models.Sequential()
        model.add(keras.layers.Conv2D(filters=32,kernel_size =(3,3), strides=1,padding='valid',activation="relu",input_shape=(28,28,1)))
        model.add(keras.layers.MaxPooling2D((2,2)))
        model.add(keras.layers.Flatten())
        model.add(keras.layers.Dense(300,activation="relu")) #RELU is prefered in hidden layers in classification model
        model.add(keras.layers.Dense(100,activation="relu")) #2 hidden layers, 100 and 300 are number of neurons
        model.add(keras.layers.Dense(10,activation="softmax")) #Softmax activation in o/p
```

```
In [9]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 5408)              0
_________________________________________________________________
dense (Dense)                (None, 300)               1622700
_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 1,654,130
Trainable params: 1,654,130
Non-trainable params: 0
_________________________________________________________________
```

```
In [10]: model.compile(loss="sparse_categorical_crossentropy",optimizer="sgd",metrics=["accuracy"])
```

jupyter CNN (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

```
In [12]: model_history = model.fit(x_train,y_train,epochs=30,batch_size=64,validation_data=(x_valid,y_valid))
```

```
Epoch 1/30
860/860 [==============================] - 28s 32ms/step - loss: 0.8766 - accuracy: 0.6990 - val_loss: 0.7514 - val_accuracy:
0.7340
Epoch 2/30
860/860 [==============================] - 28s 32ms/step - loss: 0.5390 - accuracy: 0.8047 - val_loss: 0.4929 - val_accuracy:
0.8184
Epoch 3/30
860/860 [==============================] - 28s 33ms/step - loss: 0.4807 - accuracy: 0.8256 - val_loss: 0.9232 - val_accuracy:
0.7088
Epoch 4/30
860/860 [==============================] - 28s 32ms/step - loss: 0.4436 - accuracy: 0.8416 - val_loss: 0.4176 - val_accuracy:
0.8542
Epoch 5/30
860/860 [==============================] - 29s 33ms/step - loss: 0.4196 - accuracy: 0.8502 - val_loss: 0.4318 - val_accuracy:
0.8496
Epoch 6/30
860/860 [==============================] - 28s 33ms/step - loss: 0.3955 - accuracy: 0.8583 - val_loss: 0.3760 - val_accuracy:
0.8714
Epoch 7/30
860/860 [==============================] - 28s 33ms/step - loss: 0.3780 - accuracy: 0.8659 - val_loss: 0.4047 - val_accuracy:
0.8562
Epoch 8/30
860/860 [==============================] - 28s 32ms/step - loss: 0.3627 - accuracy: 0.8697 - val_loss: 0.4999 - val_accuracy:
0.8156
Epoch 9/30
860/860 [==============================] - 28s 33ms/step - loss: 0.3489 - accuracy: 0.8737 - val_loss: 0.3700 - val_accuracy:
0.8692
Epoch 10/30
860/860 [==============================] - 28s 33ms/step - loss: 0.3368 - accuracy: 0.8787 - val_loss: 0.3816 - val_accuracy:
0.8602
Epoch 11/30
860/860 [==============================] - 28s 33ms/step - loss: 0.3249 - accuracy: 0.8816 - val_loss: 0.3457 - val_accuracy:
0.8770
Epoch 12/30
```

jupyter CNN (autosaved)

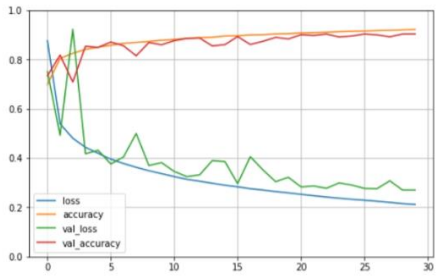
Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

In [13]:
```
pd.DataFrame(model_history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```



In [14]:
```
ev = model.evaluate(x_test_n,y_test)
ev
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.2812 - accuracy: 0.8987
```

Out[14]:
```
[0.28120481967926025, 0.8986999988555908]
```

In [15]:
```
x_new = x_test[:3]
```

In [16]:
```
y_test[:3]
```

jupyter CNN (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

In [14]:
```
ev = model.evaluate(x_test_n,y_test)
ev
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.2812 - accuracy: 0.8987
```

Out[14]:
```
[0.28120481967926025, 0.8986999988555908]
```

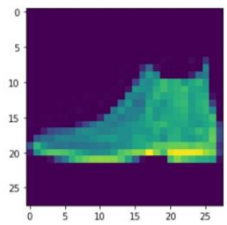
In [15]:
```
x_new = x_test[:3]
```

In [16]:
```
y_test[:3]
```

Out[16]:
```
array([9, 2, 1], dtype=uint8)
```

In [17]:
```
print(plt.imshow(x_test[0].reshape((28,28)))) #9 is equivalent to ankle boot (x_test[0])
```

```
AxesImage(54,36;334.8x217.44)
```



In [18]:
```
keras.backend.clear_session
del model
```