ML Assignment No - 1

Title: . Data preparation:

Name:Bindi Shah
Class: TE 09
Batch: K 09
Roll no.: 33110

Problem Statement: Download heart dataset from following link.
https://www.kaggle.com/zhaoyingzhu/heartcsv
Perform the following operation on a given dataset.

a) Find Shape of Data
b) Find Missing Values
c) Find data type of each column
d) Finding out Zero's
e) Find Mean age of patients
f) Now extract only Age, Sex, ChestPain, RestBP, Chol.

Randomly divide the dataset in training (75%) and testing (25%). Through the diagnosis test I predicted 100 reports as COVID positive, but only 45 of those were actually positive. Total 50 people in my sample were actually COVID positive. I have a total of 500 samples.
Create a confusion matrix based on the above data and find I. Accuracy II. Precision III. Recall IV. F-1 score

Objective: This assignment will help the students to realize what is need of data preparation

S/W Packages and H/W apparatus used: Linux OS: Ubuntu/Windows , Jupyter notebook.

Theory:

Data Preparation

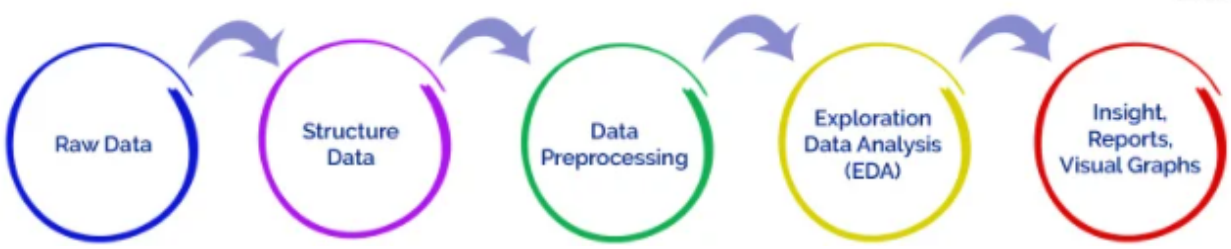
Data preparation also referred as "data preprocessing" is the process of cleaning and transforming raw data prior to processing and analysis. It is an important step prior to processing and often involves reformatting data, making corrections to data and the combining of data sets to enrich data.

Importance of Data preparation

- Because most machine learning algorithms require data to be structured in a specific way, datasets must be prepared before they can offer useful insights. a number of databases having missing, invalid, or otherwise difficult to process values for an algorithm. If you're looking for information, The algorithm will be unable to use it if it is missing. If the data is incorrect, the algorithm will produce inaccurate or even incorrect results. the outcomes are deceiving
- Some datasets are relatively clean but need to be shaped (e.g., aggregated or pivoted) and many datasets are just lacking useful business context (e.g., poorly defined ID values), hence the need for feature enrichment. Good data preparation produces clean and well curated data which leads to more practical, accurate model outcomes.
- Before entering the data into the machine learning model, this is the most important step. The reason for this is that the data set must be unique and specific to the model, thus we must identify the data's required characteristics. The data preparation process provides a mechanism for preparing data for project definition as well as project evaluation of machine learning algorithms.
- There are a variety of predicting machine learning models available, each with its own method. However, some processes are common to all models, and they allow us to identify the underlying business problem

and its solutions. The following are some of the data preparation procedures:

- 1. Determine the problems
- 2. Data cleaning
- 3. Feature selection
- 4. Data transformation
- 5. feature engineering
- 6. Dimensionality reduction



Conclusion: Data preparation is recognized for helping businesses and analytics to get ready and prepare the data for operations.

In [11]:
```
import pandas as pd
```

In [14]:
```
df=pd.read_csv('Heart.csv')
```

In [15]:
```
df
```

Out[15]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
0	1	63	1	typical	145	233	1	2	150	0	2.3	3	0.0	fixed	No
1	2	67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	normal	Yes
2	3	67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	reversable	Yes
3	4	37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	normal	No
4	5	41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	normal	No
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	299	45	1	typical	110	264	0	0	132	0	1.2	2	0.0	reversable	Yes
299	300	68	1	asymptomatic	144	193	1	0	141	0	3.4	2	2.0	reversable	Yes
300	301	57	1	asymptomatic	130	131	0	0	115	1	1.2	2	1.0	reversable	Yes
301	302	57	0	nontypical	130	236	0	2	174	0	0.0	2	1.0	normal	Yes
302	303	38	1	nonanginal	138	175	0	0	173	0	0.0	1	NaN	normal	No

303 rows × 15 columns

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

Out[16]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
0	1	63	1	typical	145	233	1	2	150	0	2.3	3	0.0	fixed	No
1	2	67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	normal	Yes
2	3	67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	reversable	Yes
3	4	37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	normal	No
4	5	41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	normal	No

In [17]:
```
#a. printing shape
df.shape
```

Out[17]:
```
(303, 15)
```

In [18]:
```
#view some basic statistical details
df.describe()
```

Out[18]:

	Unnamed: 0	Age	Sex	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	299.000000
mean	152.000000	54.438944	0.679868	131.689769	246.693069	0.148515	0.990099	149.607261	0.326733	1.039604	1.600660	0.672241
std	87.612784	9.038662	0.467299	17.599748	51.776918	0.356198	0.994971	22.875003	0.469794	1.161075	0.616226	0.937438
min	1.000000	29.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	1.000000	0.000000
25%	76.500000	48.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000
50%	152.000000	56.000000	1.000000	130.000000	241.000000	0.000000	1.000000	153.000000	0.000000	0.800000	2.000000	0.000000
75%	227.500000	61.000000	1.000000	140.000000	275.000000	0.000000	2.000000	166.000000	1.000000	1.600000	2.000000	1.000000
max	303.000000	77.000000	1.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	3.000000	3.000000

In [19]:
```
#b. returns a DataFrame object where all the values are replaced with a Boolean value True for NULL values, and otherwise Fal
df.isnull()
```

Out[19]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
299	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
300	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
301	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
302	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False

303 rows × 15 columns

In [20]:
```
#b . separate count for each column
df.isna().sum()
```

Out[20]:
```
Unnamed: 0    0
Age           0
Sex           0
ChestPain     0
RestBP        0
Chol          0
Fbs           0
RestECG       0
MaxHR         0
ExAng         0
Oldpeak       0
Slope         0
Ca            4
Thal          2
AHD           0
dtype: int64
```

In [21]:
```
#c. printing datatype of each column
df.dtypes
```

Out[21]:
```
Unnamed: 0      int64
Age             int64
Sex             int64
ChestPain      object
RestBP          int64
Chol            int64
Fbs             int64
RestECG         int64
MaxHR           int64
ExAng           int64
Oldpeak       float64
Slope           int64
Ca            float64
Thal           object
AHD            object
dtype: object
```

In [23]:
```
#d. finding out zeroes
df==0
```

Out[23]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
0	False	False	False	False	False	False	False	False	False	True	False	False	True	False	False
1	False	False	False	False	False	False	True	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	True	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	True	True	False	True	False	False	True	False	False
4	False	False	True	False	False	False	True	False	False	True	False	False	True	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	False	False	False	False	False	False	True	True	False	True	False	False	True	False	False
299	False	False	False	False	False	False	False	True	False	True	False	False	False	False	False
300	False	False	False	False	False	False	True	True	False	False	False	False	False	False	False
301	False	False	True	False	False	False	True	False	False	True	True	False	False	False	False
302	False	False	False	False	False	False	True	True	False	True	True	False	False	False	False

303 rows × 15 columns

In [24]:
```
#d
(df == 0).sum()
```

Out[24]:
```
Unnamed: 0      0
Age             0
Sex            97
ChestPain       0
RestBP          0
Chol            0
Fbs           258
RestECG       151
MaxHR           0
ExAng         204
Oldpeak        99
Slope           0
Ca            176
Thal            0
AHD             0
dtype: int64
```

In [12]:
```
#e. find mean age of patients
df["Age"].mean()
```

Out[12]:
```
54.43894389438944
```

In [26]:
```
#f.
from sklearn.model_selection import train_test_split
```

In [27]:
```
train, test = train_test_split(df[["Age", "Sex", "ChestPain", "RestBP", "Chol"]], test_size = 0.25)
```

In [28]:
```
train
```

Out[28]:

	Age	Sex	ChestPain	RestBP	Chol

249	62	1	nontypical	128	208
298	45	1	typical	110	264
25	50	0	nonanginal	120	219
230	52	0	nonanginal	136	196
223	53	1	asymptomatic	123	282
...	...	...	...	...	...
276	66	0	nonanginal	146	278
289	56	1	nontypical	120	240
136	70	1	asymptomatic	145	174
168	35	1	asymptomatic	126	282
300	57	1	asymptomatic	130	131

227 rows × 5 columns

In [29]:
```
test
```

Out[29]:

	Age	Sex	ChestPain	RestBP	Chol
125	45	0	nontypical	130	234
180	48	1	asymptomatic	124	274
4	41	0	nontypical	130	204
163	58	0	asymptomatic	100	248
299	68	1	asymptomatic	144	193
...	...	...	...	...	...
115	41	1	nontypical	135	203
128	44	1	nontypical	120	220
55	54	1	asymptomatic	124	266
190	50	1	nonanginal	129	196
102	57	0	asymptomatic	128	303

76 rows × 5 columns

In [16]:
```
#duplicate rows
df.duplicated()
```

Out[16]:
```
0      False
1      False
2      False
3      False
4      False
       ...
298    False
299    False
300    False
301    False
302    False
Length: 303, dtype: bool
```

In [30]:
```
df.duplicated().sum()
```

Out[30]:
```
0
```

In [19]:
```
df[(df.AHD=='Yes')].Age.max()
```

Out[19]:
```
77
```

In [20]:
```
df[(df.AHD=='Yes')].Age.min()
```

Out[20]:
```
35
```

In [1]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [3]:
```
y_act = [1] * 50 + [0] * 450
y_pred = [1] * 45 + [0] * 400 + [1] * 55
```

In [5]:
```
print(confusion_matrix(y_act, y_pred))
```

```
[[395  55]
 [  5  45]]
```

In [6]:
```
print(classification_report(y_act, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.88      0.93       450
           1       0.45      0.90      0.60        50

    accuracy                           0.88       500
   macro avg       0.72      0.89      0.76       500
weighted avg       0.93      0.88      0.90       500
```

ML Assignment No - 2

Title : Regression technique

Name:Bindi Shah
Class: TE 09
Batch: K 09
Roll no.: 33110

Problem Statement: This data consists of temperatures of INDIA averaging the temperatures of all places month-wise.
Temperature values are recorded in CELSIUS
a) Apply Linear Regression using a suitable library function and predict the Month-wise temperature.
b) Assess the performance of regression models using MSE, MAE and R-Square metrics
c) Visualize a simple regression model.

Objective: This assignment will help the students to realize how Linear Regression can be used and predictions using the same can be performed.

S/W Packages and H/W apparatus used: Linux OS: Ubuntu/Windows , Jupyter notebook.

Theory

Linear Regression

It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

Types of Linear Regression

- Simple Linear Regression:
  If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.
- Multiple Linear regression:
  If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Assumptions of Linear Regression

To conduct a simple linear regression, one has to make certain assumptions about the data. This is because it is a parametric test. The assumptions used while performing a simple linear regression are as follows:

- Homogeneity of variance (homoscedasticity)- One of the main predictions in a simple linear regression method is that the size of the error stays constant. This simply means that in the value of the independent variable, the error size never changes significantly.
- Independence of observations- All the relationships between the observations are transparent, which means that nothing is hidden, and only valid sampling methods are used during the collection of data.
- Normality- There is a normal rate of flow in the data. These three are the assumptions of regression methods.

However, there is one additional assumption that has to be taken into consideration while specifically conducting a linear regression.

- The line is always a straight line- There is no curve or grouping factor during the conduction of a linear regression. There is a linear relationship between the variables (dependent variable and independent variable). If the data fails the assumptions of homoscedasticity or normality, a nonparametric test might be used. (For example, the Spearman rank test)

Applications of Simple Linear Regression

- 1. Marks scored by students based on number of hours studied (ideally)- Here marks scored in exams are dependent and the number of hours studied is independent.
- 2. Predicting crop yields based on the amount of rainfall- Yield is a dependent variable while the measure of precipitation is an independent variable.
- 3. Predicting the Salary of a person based on years of experience- Therefore, Experience becomes the independent variable while Salary turns into the dependent variable.

Limitations of Simple Linear Regression

Indeed, even the best information doesn't recount a total story. Regression investigation is ordinarily utilized in examinations to establish that a relationship exists between variables. However, correlation isn't equivalent to causation: a connection between two variables doesn't mean one causes the other to occur. Indeed, even a line in a simple linear regression that fits the information focuses well may not ensure a circumstances and logical results relationship.

Utilizing a linear regression model will permit you to find whether a connection between variables exists by any means. To see precisely what

that relationship is and whether one variable causes another, you will require extra examination and statistical analysis.

Conclusion

Simple linear regression is a regression model that figures out the relationship between one independent variable and one dependent variable using a straight line.

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

In [2]:
```
#Read csv file
df = pd.read_csv('temperatures.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	JAN-FEB	MAR-MAY	JUN-SEP	OCT-DEC
0	1901	22.40	24.14	29.07	31.91	33.41	33.18	31.21	30.39	30.47	29.97	27.31	24.49	28.96	23.27	31.46	31.27	27.25
1	1902	24.93	26.58	29.77	31.78	33.73	32.91	30.92	30.73	29.80	29.12	26.31	24.04	29.22	25.75	31.76	31.09	26.49
2	1903	23.44	25.03	27.83	31.39	32.91	33.00	31.34	29.98	29.85	29.04	26.08	23.65	28.47	24.24	30.71	30.92	26.26
3	1904	22.50	24.73	28.21	32.02	32.64	32.07	30.36	30.09	30.04	29.20	26.36	23.63	28.49	23.62	30.95	30.66	26.40
4	1905	22.00	22.83	26.68	30.01	33.32	33.25	31.44	30.68	30.12	30.67	27.52	23.82	28.30	22.25	30.00	31.33	26.57

In [4]:
```
#Check shape of data
df.shape
```

Out[4]: (117, 18)

In [5]:
```
#Check for null values
df.isna().sum()
```

Out[5]:
```
YEAR       0
JAN        0
FEB        0
MAR        0
APR        0
MAY        0
JUN        0
JUL        0
AUG        0
SEP        0
OCT        0
NOV        0
DEC        0
ANNUAL     0
JAN-FEB    0
MAR-MAY    0
JUN-SEP    0
OCT-DEC    0
dtype: int64
```

In [8]:
```
#Check info of data
df.info() ;
df.describe()
```

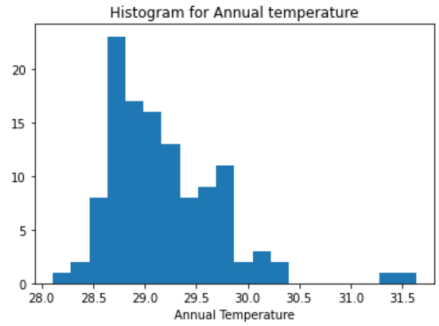
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 117 entries, 0 to 116
Data columns (total 18 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   YEAR     117 non-null    int64
 1   JAN      117 non-null    float64
 2   FEB      117 non-null    float64
 3   MAR      117 non-null    float64
 4   APR      117 non-null    float64
 5   MAY      117 non-null    float64
 6   JUN      117 non-null    float64
 7   JUL      117 non-null    float64
 8   AUG      117 non-null    float64
 9   SEP      117 non-null    float64
 10  OCT      117 non-null    float64
 11  NOV      117 non-null    float64
 12  DEC      117 non-null    float64
 13  ANNUAL   117 non-null    float64
 14  JAN-FEB  117 non-null    float64
 15  MAR-MAY  117 non-null    float64
 16  JUN-SEP  117 non-null    float64
 17  OCT-DEC  117 non-null    float64
dtypes: float64(17), int64(1)
memory usage: 16.6 KB
```

Out[8]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV
count	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000
mean	1959.000000	23.687436	25.597863	29.085983	31.975812	33.565299	32.774274	31.035897	30.507692	30.486752	29.766581	27.285470
std	33.919021	0.834588	1.150757	1.068451	0.889478	0.724905	0.633132	0.468818	0.476312	0.544295	0.705492	0.714518
min	1901.000000	22.000000	22.830000	26.680000	30.010000	31.930000	31.100000	29.760000	29.310000	29.070000	27.900000	25.700000

25%	1930.000000	23.100000	24.780000	28.370000	31.460000	33.110000	32.340000	30.740000	30.180000	30.120000	29.380000	26.790000
50%	1959.000000	23.680000	25.480000	29.040000	31.950000	33.510000	32.730000	31.000000	30.540000	30.520000	29.780000	27.300000
75%	1988.000000	24.180000	26.310000	29.610000	32.420000	34.030000	33.180000	31.330000	30.760000	30.810000	30.170000	27.720000
max	2017.000000	26.940000	29.720000	32.620000	35.380000	35.840000	34.480000	32.760000	31.840000	32.220000	32.290000	30.110000

In [9]:
```
plt.hist(df['ANNUAL'],bins=20 )
plt.xlabel('Annual Temperature')
plt.title('Histogram for Annual temperature')
plt.show()
```



In [11]:
```
#january
#Train data and test data
X = df['YEAR'].values[:,None]
Y = df['JAN']
```

In [12]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.3, random_state=0)
```

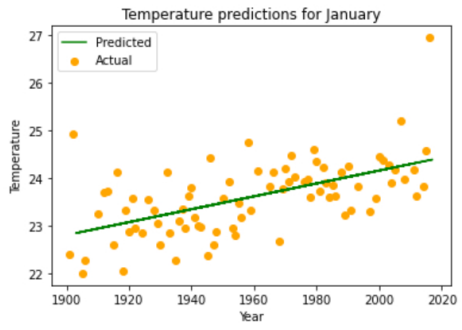
In [13]:
```
#Linear regression
from sklearn.linear_model import LinearRegression
model1 = LinearRegression()
model1.fit(x_train, y_train)
```

Out[13]:
```
LinearRegression()
```

In [14]:
```
y_test_predict = model1.predict(x_test)
```

In [15]:
```
y_train_predict = model1.predict(x_train)
```

In [16]:
```
plt.scatter(x_train, y_train, color='orange', label='Actual')
plt.plot(x_test, y_test_predict, color='green', label='Predicted')
plt.xlabel('Year')
plt.ylabel('Temperature')
plt.title('Temperature predictions for January')
plt.legend()
plt.show()
```



In [17]:
```
#Accuracy
from sklearn import metrics
r2_square = metrics.r2_score(y_test,y_test_predict)
mse = metrics.mean_squared_error(y_test,y_test_predict)
mae = metrics.mean_absolute_error(y_test, y_test_predict)
print("R2 score: {0}\nMSE: {1}\nMAE: {2}".format(r2_square,mse,mae))
```

```
R2 score: 0.2792172351531238
MSE: 0.6080338203121168
MAE: 0.6231302838065338
```

In [18]:
```
r2_square = metrics.r2_score(y_train,y_train_predict)
mse = metrics.mean_squared_error(y_train,y_train_predict)
mae = metrics.mean_absolute_error(y_train,y_train_predict)
print("R2 score: {0}\nMSE: {1}\nMAE: {2}".format(r2_square,mse,mae))
```

```
R2 score: 0.3192812012518227
MSE: 0.4085691037958135
MAE: 0.48235911219829086
```

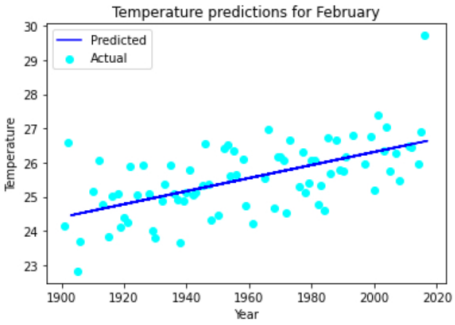
In [19]:
```
#February
#Train data and test data
Y = df['FEB']
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.3, random_state=0)
```

In [20]:
```
#Linear regression
model1.fit(x_train, y_train)
```

Out[20]:
```
LinearRegression()
```

In [21]:
```
y_test_predict = model1.predict(x_test)
y_train_predict = model1.predict(x_train)
```

In [22]:
```
plt.scatter(x_train, y_train, color='cyan', label='Actual')
plt.plot(x_test, y_test_predict, color='blue', label='Predicted')
plt.xlabel('Year')
plt.ylabel('Temperature')
plt.title('Temperature predictions for February')
plt.legend()
plt.show()
```



In [23]:
```
#accuracy
from sklearn import metrics
r2_square = metrics.r2_score(y_test,y_test_predict)
mse = metrics.mean_squared_error(y_test,y_test_predict)
mae = metrics.mean_absolute_error(y_test, y_test_predict)
print("R2 score: {0}\nMSE: {1}\nMAE: {2}".format(r2_square,mse,mae))
```

```
R2 score: 0.473733386001327
MSE: 0.9505646859799862
MAE: 0.7252754949018981
```

In [24]:
```
r2_square = metrics.r2_score(y_train,y_train_predict)
mse = metrics.mean_squared_error(y_train,y_train_predict)
mae = metrics.mean_absolute_error(y_train,y_train_predict)
print("R2 score: {0}\nMSE: {1}\nMAE: {2}".format(r2_square,mse,mae))
```

```
R2 score: 0.35290726380537074
MSE: 0.699893333670794
MAE: 0.6468901850808766
```

In [26]:
```
#Seasonal data
#Jun-Sep
x_year = df['YEAR'].values[:,None]
y_seasonal = df['JUN-SEP']
```

In [30]:
```
#Train and test data
x_train, x_test, y_train, y_test = train_test_split(x_year,y_seasonal,test_size=0.3, random_state=42);
```

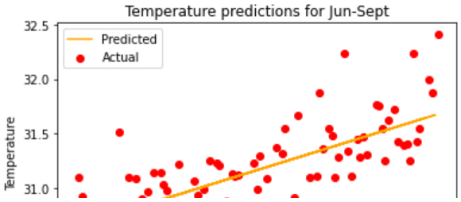
In [31]:
```
#Linear regression
model1.fit(x_train, y_train)
```

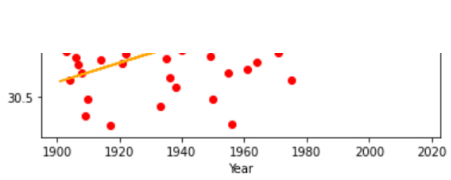
Out[31]:
```
LinearRegression()
```

In [32]:
```
y_test_predict = model1.predict(x_test)
```

In [33]:
```
y_train_predict = model1.predict(x_train)
```

In [34]:
```
plt.scatter(x_train, y_train, color='red', label='Actual')
plt.plot(x_test, y_test_predict, color='orange', label='Predicted')
plt.xlabel('Year')
plt.ylabel('Temperature')
plt.title('Temperature predictions for Jun-Sept')
plt.legend()
plt.show()
```





In [35]:
```
#Accuracy
from sklearn import metrics
r2_square = metrics.r2_score(y_test,y_test_predict)
mse = metrics.mean_squared_error(y_test,y_test_predict)
mae = metrics.mean_absolute_error(y_test, y_test_predict)
print("R2 score: {0}\nMSE: {1}\nMAE: {2}".format(r2_square,mse,mae))
```

```
R2 score: 0.3409534287452929
MSE: 0.09316091126387105
MAE: 0.2476746018806843
```

In [36]:
```
r2_square = metrics.r2_score(y_train,y_train_predict)
mse = metrics.mean_squared_error(y_train,y_train_predict)
mae = metrics.mean_absolute_error(y_train,y_train_predict)
print("R2 score: {0}\nMSE: {1}\nMAE: {2}".format(r2_square,mse,mae))
```

```
R2 score: 0.4788803742892933
MSE: 0.09810370436250214
MAE: 0.2411774728499514
```

ML Assignment No - 3

Title: Classification using Machine Learning

Name:Bindi Shah
Class: TE 09
Batch: K 09
Roll no.: 33110

Problem Statement:

Perform following operations on given dataset:

a. Apply Data pre-processing (Label Encoding, Data Transformation....) techniques if necessary.

b. Perform data-preparation (Train-Test Split)

c. Apply Decision tree classification Algorithm

d. Evaluate Model.

THEORY:

Classification: Classification is a process of categorizing a given set of data into classes, It can be performed on both structured or unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories.

What is a Decision Tree?

It uses a flowchart like a tree structure to show the predictions that result from a series of feature-based splits. It starts with a root node and ends with a decision made by leaves.

Root Nodes – It is the node present at the beginning of a decision tree. From this node the population starts dividing according to various features.

Decision Nodes – the nodes we get after splitting the root nodes are called Decision Node

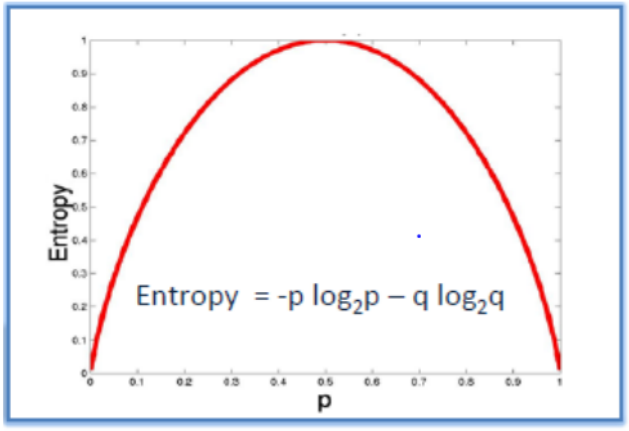
Leaf Nodes – the nodes where further splitting is not possible are called leaf nodes or terminal nodes

Sub-tree – just like a small portion of a graph is called sub-graph similarly a subsection of this the decision tree is called a sub-tree.

Pruning – It is cutting down some nodes to stop overfitting

Entropy:

Entropy is used to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is equally divided it has entropy of one.



Entropy = $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$

a) Entropy using the frequency table of one attribute:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

b) Entropy using the frequency table of two attributes:

$$E(T,X) = \sum_{c \in X} P(c)E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)

= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971

= 0.693

Information Gain

The information gain is based on the decrease in entropy after a dataset is split on an attribute.

Constructing a decision tree

IS all about finding attributes that return the highest information gain (i.e., the most homogeneous branches)

Step 1: Calculate entropy of the target.

Entropy(PlayGolf) = Entropy (5,9)

= Entropy (0.36, 0.64)

= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)

= 0.94

Step 2: The dataset is then split on the different attributes. The entropy for each branch is calculated.

Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the Information Gain, or decrease in entropy.

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

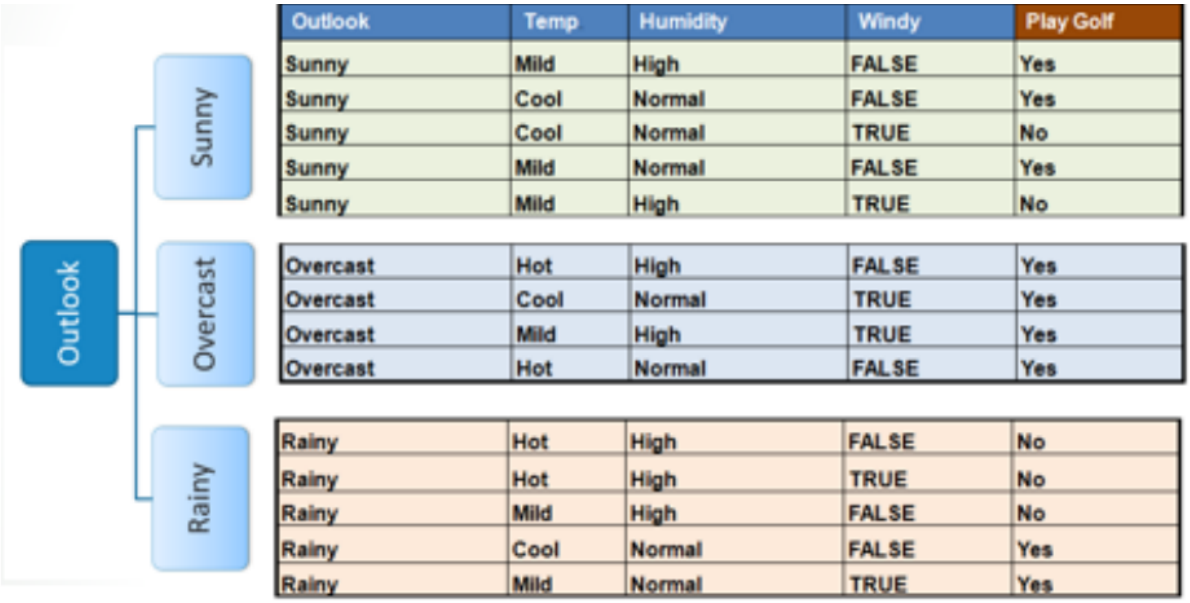
		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

$$Gain(T,X) = Entropy(T) - Entropy(T,X)$$

G(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

= 0.940 – 0.693 = 0.247

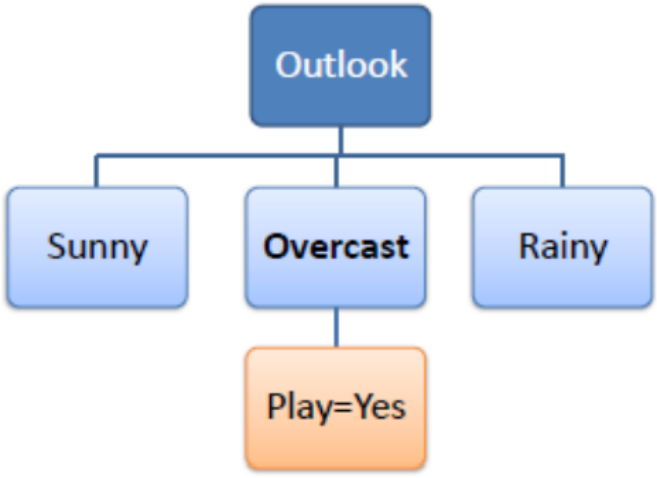
Step 3: Choose the attribute with the largest information gain as the decision node, divide the dataset by its branches and repeat the same process on every branch.



Outlook	Temp	Humidity	Windy	Play Golf
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Sunny	Mild	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Overcast	Cool	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes

Step 4a: A branch with entropy of 0 is a leaf node.

Temp	Humidity	Windy	Play Golf
Hot	High	FALSE	Yes
Cool	Normal	TRUE	Yes
Mild	High	TRUE	Yes
Hot	Normal	FALSE	Yes



Step 4b: A branch with entropy more than 0 needs further splitting.
Step 5: The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.

Decision Tree to Decision Rules

A decision tree can easily be transformed to a set of rules by mapping from the root node to the leaf nodes one by one.

Pruning:

It is another method that can help us avoid overfitting. It helps in improving the performance of the tree by cutting the nodes or sub-nodes which are not significant. It removes the branches which have very low importance.
There are mainly 2 ways for pruning:
(i) Pre-pruning – we can stop growing the tree earlier, which means we can prune/remove/cut a node if it has low importance while growing the tree.
(ii) Post-pruning – once our tree is built to its depth, we can start pruning the nodes based on their significance.

CONCLUSION:

Classification techniques help in classifying problems and helps figure out relationship between the various variables.

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

In [3]:
```
df = pd.read_csv('Admission.csv')
df.head()
```

Out[3]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

In [4]:
```
df.shape
```

Out[4]:
```
(500, 9)
```

In [7]:
```
#Information about data set
df.info() ;
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Serial No.         500 non-null    int64
 1   GRE Score          500 non-null    int64
 2   TOEFL Score        500 non-null    int64
 3   University Rating  500 non-null    int64
 4   SOP                500 non-null    float64
 5   LOR                500 non-null    float64
 6   CGPA               500 non-null    float64
 7   Research           500 non-null    int64
 8   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 35.3 KB
```

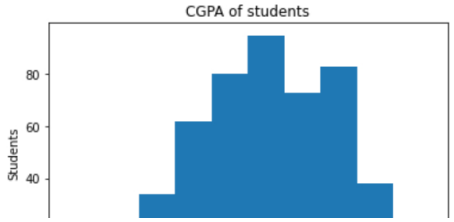
Out[7]:

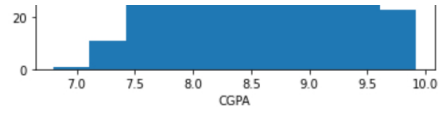
	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

In [8]:
```
#Check for null values
df.isna().sum()
```

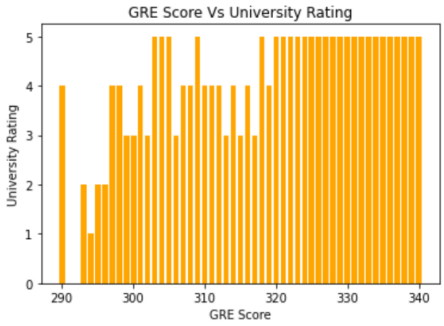
Out[8]:
```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [9]:
```
#Data visualization
plt.hist(df['CGPA'], bins=10)
plt.xlabel('CGPA')
plt.ylabel('Students')
plt.title('CGPA of students')
plt.show()
```

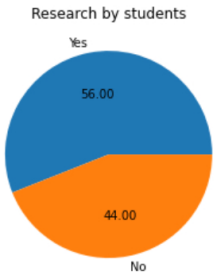




In [10]:
```
plt.bar(df['GRE Score'], df['University Rating'], color='orange')
plt.xlabel('GRE Score')
plt.ylabel('University Rating')
plt.title('GRE Score Vs University Rating')
plt.show()
```



In [11]:
```
research = df.Research.value_counts()
label1 = ['Yes', 'No']
plt.pie(research, autopct='%.2f',labels=label1)
plt.title('Research by students')
plt.show()
```



In [13]:
```
#Change value of output variable
#If chance is less than 0.8 change value to 0 else 1
df.loc[df['Chance of Admit '] < 0.8,'Chance of Admit'] = 0
df.loc[df['Chance of Admit '] >= 0.8,'Chance of Admit'] = 1
```

In [14]:
```
#Independant and Dependant variables
X = df[["GRE Score","CGPA"]]
Y = df['Chance of Admit']
```

In [15]:
```
#Split dataset into train and test data
from sklearn.model_selection import train_test_split
```

In [16]:
```
x_train, x_test, y_train, y_test = train_test_split(X,Y, train_size=0.75, random_state=123)
```

In [17]:
```
x_train.head()
```

Out[17]:

	GRE Score	CGPA
455	305	7.64
384	340	9.74
293	312	8.18
421	321	8.95
374	315	7.65

In [18]:
```
y_test.head()
```

Out[18]:
```
229    1.0
337    1.0
327    0.0
416    0.0
306    0.0
Name: Chance of Admit, dtype: float64
```

In [19]:
```
#Decision tree Algorithm
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
```

In [20]:
```
classifier = DecisionTreeClassifier()
```

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

In [22]:
```
y_predict = classifier.predict(x_test)
y_predict
```

Out[22]:
```
array([1., 1., 0., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 0., 1., 0.,
       0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0.,
       0., 0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 1., 1., 1.,
       1., 1., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0.,
       0., 1., 0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 1., 0., 1., 0., 0.,
       1., 1., 0., 0., 1., 0.])
```

In [24]:
```
#Accuracy of the mode
confusion_matrix = metrics.confusion_matrix(y_test,y_predict)
confusion_matrix
```

Out[24]:
```
array([[79,  3],
       [ 6, 37]], dtype=int64)
```

In [32]:
```
accuracy = metrics.accuracy_score(y_test,y_predict)
precision = metrics.precision_score(y_test,y_predict)
recall = metrics.recall_score(y_test,y_predict)
f1_score = metrics.f1_score(y_test,y_predict)
print("Accuracy: {0}\nPrecision: {1}\nRecall: {2}\nf1 score: {3}".format(accuracy,precision,recall,f1_score))
```

```
Accuracy: 0.928
Precision: 0.925
Recall: 0.8604651162790697
f1 score: 0.891566265060241
```

In [ ]:
```

```

ML Assignment No - 5

Title: K Means Clustering

Name:Bindi Shah
Class: TE 09
Batch: K 09
Roll no.: 33110

Problem Statement:

a) Apply Data pre-processing (Label Encoding, Data Transformation....) techniques if necessary.
b) Perform data-preparation (Train-Test Split)
c) Apply Machine Learning Algorithm
d) Evaluate Model.
e) Apply Cross-Validation and Evaluate Model

Objective: This assignment will help the students to realize how to do Clustering using the K- Means Clustering algorithm.

Theory:

- K-Means Clustering : K-Means Clustering is an unsupervised learning algorithm that is used to solve clustering problems in machine learning or data science

- What is the K-Means Algorithm? :
  K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of predefined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It allows us to cluster the data into different groups and is a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has data points with some commonalities, and it is away from other clusters.

Algorithm: The working of the K-Means algorithm is explained in the below steps:

- Step-1: Select the number K to decide the number of clusters.
- Step-2: Select random K points or centroids. (It can be other from the input dataset).
- Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.
- Step-4: Calculate the variance and place a new centroid of each cluster.

- Step-5: Repeat the third steps, which means reassigning each datapoint to the new closest centroid of each cluster.
- Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.
- Step-7: The model is ready

Some examples of use cases are:

1. Behavioral segmentation: Segment by purchase history, Segment by activities on application, website, or platform, Define personas based on interests , Create profiles based on activity monitoring
2. Inventory categorization: Group inventory by sales activity , Group inventory by manufacturing metrics.

Clustering

In [49]:
```
#Convert object column to numerical type
from sklearn import preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [50]:
```
#Read CSV File
df = pd.read_csv('Mall_Customers.csv')
df.head()
```

Out[50]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

In [51]:
```
#Shape of Dataset
df.shape
```

Out[51]: (200, 5)

In [52]:
```
#Information about data set
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Genre                   200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

Out[52]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

In [53]:
```
#Check for null values
df.isna().sum()
```

Out[53]:
```
CustomerID              0
Genre                   0
Age                     0
Annual Income (k$)      0
Spending Score (1-100)  0
dtype: int64
```

In [54]:
```
#Set index column
df = df.set_index('CustomerID')
df.head()
```

Out[54]:

CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40

In [55]:
```
label_encoder = preprocessing.LabelEncoder()
df['Genre'] = label_encoder.fit_transform(df['Genre'])
```

In [56]:
```
df.head()
```

Out[56]:

CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
1	1	19	15	39
2	1	21	15	81
3	0	20	16	6
4	0	23	16	77
5	0	31	17	40

In [57]:
```
#data preparation
from sklearn.model_selection import train_test_split
```

In [58]:
```
X = df.iloc[:,:4]
Y = df.index
```

In [59]:
```
x_train, x_test, y_train, y_test = train_test_split(X,Y, train_size=0.75, random_state=123)
y_train.shape
```

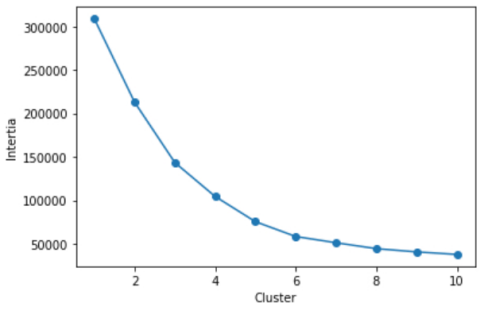
Out[59]:
```
(150,)
```

In [60]:
```
#Kmeans clustering
from sklearn.cluster import KMeans

import warnings
warnings.filterwarnings("ignore")
```

In [61]:
```
cluster = []
for k in range(1,11):
    kmean = KMeans(n_clusters=k).fit(df)
    cluster.append(kmean.inertia_)
```

In [62]:
```
plt.plot(range(1,11), cluster, marker="o" )
plt.xlabel('Cluster')
plt.ylabel('Intertia')
plt.show()
```



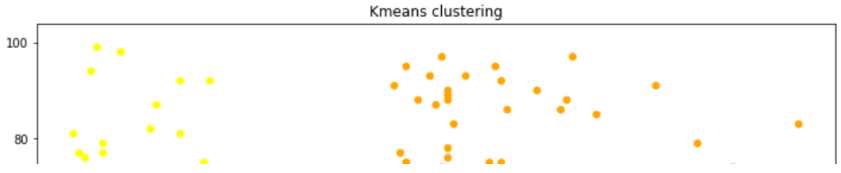
In [63]:
```
km = KMeans(n_clusters=5).fit(df)
df['Labels'] = km.labels_
df.head()
```

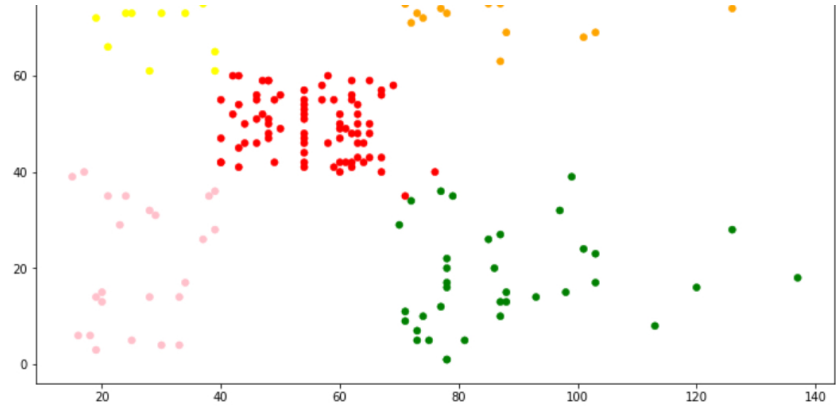
Out[63]:

CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)	Labels
1	1	19	15	39	4
2	1	21	15	81	3
3	0	20	16	6	4
4	0	23	16	77	3
5	0	31	17	40	4

In [64]:
```
centers = np.array(km.cluster_centers_)
```

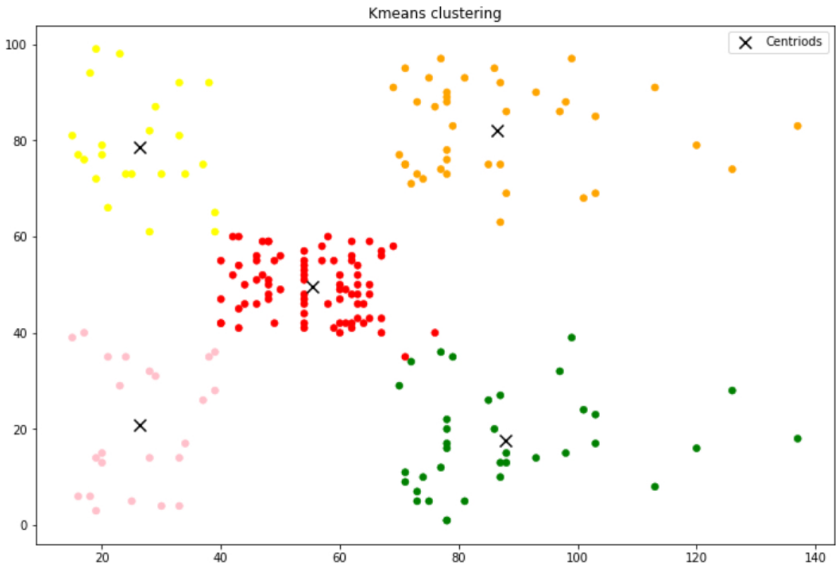
In [67]:
```
#Plotting clusters
colors = {0:'red', 1:'green', 2:'orange',3:'yellow',4:'pink'}
plt.figure(figsize=(12, 8))
plt.scatter(df['Annual Income (k$)'], df['Spending Score (1-100)'], linewidths=0.5, color =df['Labels'].map(colors))
plt.title('Kmeans clustering')
plt.show()
```





In [69]:

```
#Plotting centroids of the clusters
colors = {0:'red', 1:'green', 2:'orange',3:'yellow',4:'pink'}
plt.figure(figsize=(12, 8))
plt.scatter(df['Annual Income (k$)'], df['Spending Score (1-100)'], linewidths=0.5,color=df['Labels'].map(colors))
plt.scatter(km.cluster_centers_[:, 2], km.cluster_centers_[:, 3], s = 100, c = 'black', label='Centriods',marker='x')
plt.title('Kmeans clustering')
plt.legend()
plt.show()
```



In [ ]: