**Assignment -1**
**Assignment on Regression technique**
**Machine Learning LP-1**

**Name: Prasad Sanjay Khalkar**
**Roll No: 33138**
**TE-09 L-09**

**Problem Statement:**

Perform following operation on 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 dataset intraining (75%) and testing (25%).
g) Through the diagnosis test I predicted 100 report as COVID positive, but only 45 of those wereactually positive. Total 50 people in my sample were actually COVID positive. I have total 500 samples.

Create confusion matrix based on above data and find

i. Accuracy
ii. Precision
iii. Recall
iv. F-1 score

---

**Theory:**

## Data Preparation:

Data preparation (also referred to as "data preprocessing") is the process of transforming raw data so that data scientists and analysts can run it through machine learning algorithms to uncover insights or make predictions.

### Why is Data Preparation Important?

Most machine learning algorithms require data to be formatted in a very specific way, so datasets generally require some amount of preparation before they can yield useful insights. Some datasets have values that are missing, invalid, or otherwise difficult for an algorithm to process. If data is missing, the algorithm can't use it. If data is invalid, the algorithm produces less accurate or even misleading outcomes. Some datasets are relatively clean but need to be shaped (e.g., aggregated or pivoted). It is the most required process before feeding the data into the machine learning model. The reason behind that the data set needs to be different and specific according to the model so that wehave to find out the required features of that data.

Some of the data preparationprocesses are:

1. **Determine the problems:**

This step tells us about the learning method of the project to find out the results for future predictionor forecasting. For example, which ML model suitable for the data set regression or classificationor clustering algorithms. This includes data collection that is useful for predicting the result and also involving the communication to project stakeholders and domain expertise. We use classification and regression models for categorical and numerical data respectively.

2. **Data cleaning:**

After collecting the data, it is very necessary to clean that data and make it proper for the ML model. It includes solving problems like outliers, inconsistency, missing values, incorrect, skewed,and trends. Cleaning the data is very important as the model learning from that data only, so if wefeed inconsistent, appropriate data to model it will return garbage only, so it is required to make sure that the data does not contains any unseen problem. For example, if we have a data set of sales, it might be possible that it contains some features like height, age, that cannot help in the model building so we can remove it. We generally remove the null values columns, fill the missingvalues, make the data set consistent, and remove the outliers and skewed data in data cleaning.

3. **Feature selection:**

Sometimes we face the problem of identifying the related features from the set of data and deletingthe irrelevant and less important data without touching the target variables to get the better accuracy of the model. Features selection plays a wide role in building a machine learning modelthat impacts the performance and accuracy of the model. It is that process which contributes mostly to the predictions or output that we need by selecting the features automatically or manually. If wehave irrelevant data that would cause the model with overfitting and underfitting.

4. **Data transformation:**

Data transformation is the process that converts the data from one form to another. It is required for data integration and data management. In data transformation, we can change the types of data,clear the data removing the null values or duplicate values, and get enrich data that depends on therequirements of the model. It allows us to perform data mapping that determines how individual features are mapped, modified, filtered, aggregated, and joined.

5. **Feature engineering:**

Every ML algorithms use some input data for giving required output and this input required somefeatures which are in a structured form. To get the proper result the algorithms required features with some specific characteristics which we find out with feature engineering. we need to performdifferent feature engineering on different datasets and we can observe their effect on model performance.

6. **Dimensionality reduction:**

When we use the dataset for building an ML model, we need to work with 1000s of features that cause the curse of dimensionality, or we can say that it refers to the process to convert a set of data.For the ML model, we have to access a large amount of data and that large amount of data can lead us in a situation where we can take possible data that can be available to feed it into a forecasting model to predict and give the result of the target variable. It reduced the time that is required for training and testing our machine learning model and also helps to eliminate over-fitting.

**Conclusion:**

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

**Implementation:**

**Implementation is as shown below:**

# Name: Prasad Sanjay Khalkar

# Roll No: 33138

# TE-09 (L-09)

In [9]:

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

In [6]:

```
df = pd.read_csv("/home/prasadkhalkar/Desktop/ML/Datasets/Heart.csv")
heart = df.copy()
heart
```

Out[6]:

| | 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 [9]:

```
heart.head(10)
```

Out[9]:

| | 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 |
| **5** | 6 | 56 | 1 | nontypical | 120 | 236 | 0 | 0 | 178 | 0 | 0.8 | 1 | 0.0 | normal | No |
| **6** | 7 | 62 | 0 | asymptomatic | 140 | 268 | 0 | 2 | 160 | 0 | 3.6 | 3 | 2.0 | normal | Yes |
| **7** | 8 | 57 | 0 | asymptomatic | 120 | 354 | 0 | 0 | 163 | 1 | 0.6 | 1 | 0.0 | normal | No |
| **8** | 9 | 63 | 1 | asymptomatic | 130 | 254 | 0 | 2 | 147 | 0 | 1.4 | 2 | 1.0 | reversable | Yes |
| **9** | 10 | 53 | 1 | asymptomatic | 140 | 203 | 1 | 2 | 155 | 1 | 3.1 | 3 | 0.0 | reversable | Yes |

In [10]:

```
heart.describe()
```

Out[10]:

| | 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 |

## Find shape of data

In [11]:

In [11]:

```
heart.shape
```

Out[11]:

```
(303, 15)
```

#### Find Missing Values

In [12]:

```
heart.isnull().sum()
```

Out[12]:

```
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
```

#### Find data type of each column

In [13]:

```
heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  303 non-null    int64
 1   Age         303 non-null    int64
 2   Sex         303 non-null    int64
 3   ChestPain   303 non-null    object
 4   RestBP      303 non-null    int64
 5   Chol        303 non-null    int64
 6   Fbs         303 non-null    int64
 7   RestECG     303 non-null    int64
 8   MaxHR       303 non-null    int64
 9   ExAng       303 non-null    int64
 10  Oldpeak     303 non-null    float64
 11  Slope       303 non-null    int64
 12  Ca          299 non-null    float64
 13  Thal        301 non-null    object
 14  AHD         303 non-null    object
dtypes: float64(2), int64(10), object(3)
memory usage: 35.6+ KB
```

#### Finding out Zero's

In [14]:

```
(heart==0).sum()
```

Out[14]:

```
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
```

#### Find Mean Age of Patients

In [15]:

```
age = heart['Age']
np.mean(age)
```

Out[15]:

```
54.43894389438944
```

### Replacing Null values

In [16]:

```
heart['Ca'].unique()
```

Out[16]:

```
array([ 0.,  3.,  2.,  1., nan])
```

In [17]:

```
heart['Ca'] = heart.Ca.astype(object)
heart['Ca'].unique()
```

Out[17]:

```
array([0.0, 3.0, 2.0, 1.0, nan], dtype=object)
```

In [18]:

```
heart['Ca'].fillna(heart['Ca'].mode()[0],inplace=True)
```

In [19]:

```
heart['Ca'].isnull().sum()
```

Out[19]:

```
0
```

In [20]:

```
heart['Thal'].unique()
```

Out[20]:

```
array(['fixed', 'normal', 'reversable', nan], dtype=object)
```

In [21]:

```
heart['Thal'].mode()
```

Out[21]:

```
0    normal
dtype: object
```

In [22]:

```
heart['Thal'].fillna(heart['Thal'].mode()[0],inplace=True)
heart['Thal'].isnull().sum()
```

Out[22]:

```
0
```

In [23]:

```
heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  303 non-null    int64
 1   Age         303 non-null    int64
 2   Sex         303 non-null    int64
 3   ChestPain   303 non-null    object
 4   RestBP      303 non-null    int64
 5   Chol        303 non-null    int64
 6   Fbs         303 non-null    int64
 7   RestECG     303 non-null    int64
 8   MaxHR       303 non-null    int64
 9   ExAng       303 non-null    int64
 10  Oldpeak     303 non-null    float64
 11  Slope       303 non-null    int64
 12  Ca          303 non-null    float64
 13  Thal        303 non-null    object
 14  AHD         303 non-null    object
dtypes: float64(2), int64(10), object(3)
memory usage: 35.6+ KB
```

### Few Plots

In [24]:

```
heart
```

Out[24]:

| | 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 |

| | Unnamed: 0 | Age | Sex | ChestPain | RestBP | Chol | Fbs | RestECG | MaxHR | ExAng | Oldpeak | Slope | Ca | Thal | AHD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 | 0.0 | normal | No |

303 rows × 15 columns

In [25]:

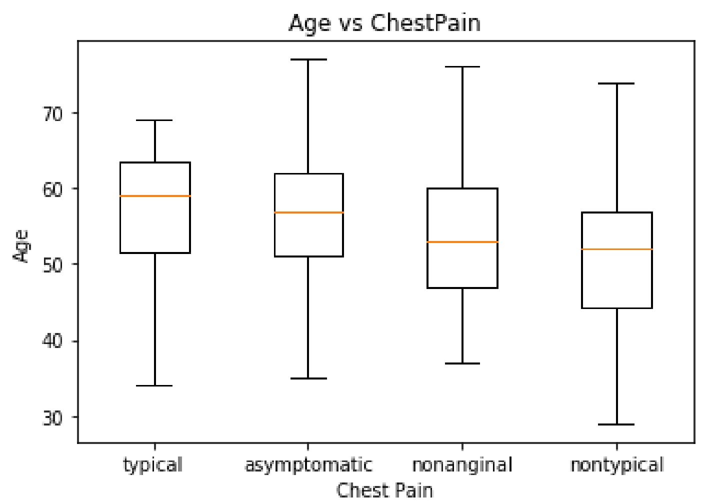
```
heart['ChestPain'].unique()
```

Out[25]:

```
array(['typical', 'asymptomatic', 'nonanginal', 'nontypical'],
      dtype=object)
```

In [26]:

```
data1 = heart[heart['ChestPain']=='typical']['Age']
data2 = heart[heart['ChestPain']=='asymptomatic']['Age']
data3 = heart[heart['ChestPain']=='nonanginal']['Age']
data4 = heart[heart['ChestPain']=='nontypical']['Age']
data = [data1,data2,data3,data4]
plt.xlabel('Chest Pain')
plt.ylabel('Age')
plt.title('Age vs ChestPain')
plt.boxplot(data)
plt.xticks([1,2,3,4],['typical', 'asymptomatic', 'nonanginal', 'nontypical'])
plt.show()
```




In [27]:

```
heart['Thal'].unique()
```

Out[27]:

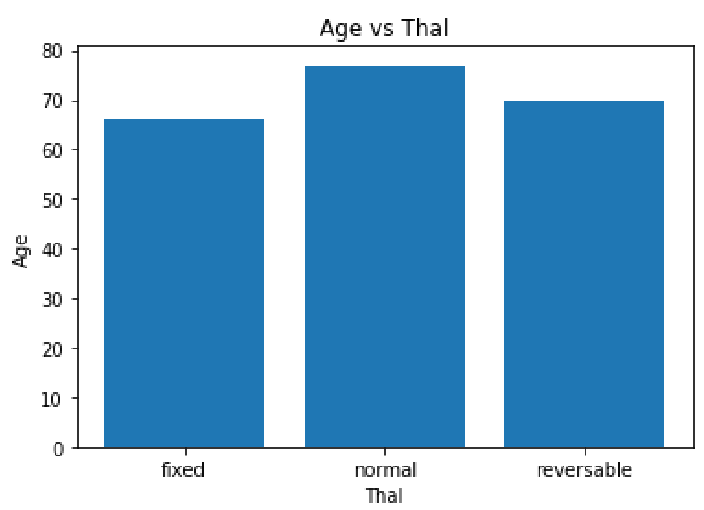
```
array(['fixed', 'normal', 'reversable'], dtype=object)
```

In [28]:

```
x = heart['Thal']
y = heart['Age']
plt.xlabel('Thal')
plt.ylabel('Age')
plt.title('Age vs Thal')
plt.bar(x,y)
```

Out[28]:

```
<BarContainer object of 303 artists>
```

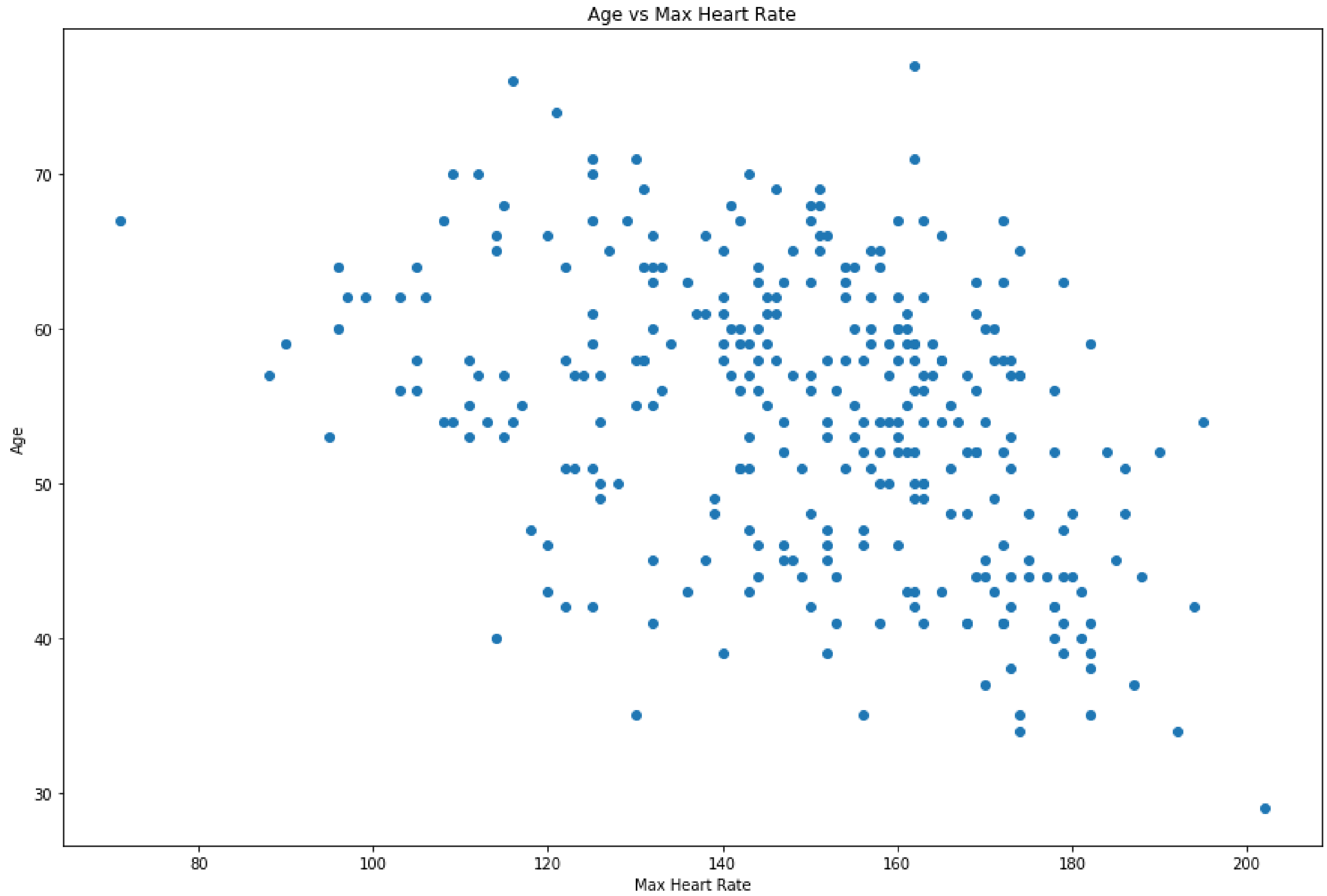



In [29]:

```
x = heart['MaxHR']
y = heart['Age']
plt.figure(figsize=(15,10))
plt.xlabel("Max Heart Rate")
plt.ylabel('Age')
plt.title('Age vs Max Heart Rate')
plt.scatter(x,y)
```

Out[29]:

<matplotlib.collections.PathCollection at 0x7f08b2228750>




**Now extract only Age, Sex, ChestPain, RestBP, Chol. Randomly divide dataset in training (75%) and testing (25%).**

In [30]:

```
heart
```

Out[30]:

| | 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 | 0.0 | normal | No |

303 rows × 15 columns

In [7]:

```
x = heart[['Age','Sex','ChestPain','RestBP','Chol']]
y = heart['AHD']
```

In [11]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=1)
```

In [12]:

```
x_train
```

Out[12]:

| | Age | Sex | ChestPain | RestBP | Chol |
|---|---|---|---|---|---|
| 170 | 70 | 1 | nonanginal | 160 | 269 |
| 192 | 43 | 1 | asymptomatic | 132 | 247 |
| 168 | 35 | 1 | asymptomatic | 126 | 282 |
| 42 | 71 | 0 | nontypical | 160 | 302 |
| 90 | 66 | 1 | asymptomatic | 120 | 302 |
| ... | ... | ... | ... | ... | ... |
| 203 | 64 | 0 | nonanginal | 140 | 313 |

| | Age | Sex | ChestPain | RestBP | Chol |
|---|---|---|---|---|---|
| 255 | 42 | 0 | nonanginal | 120 | 209 |
| 72 | 62 | 1 | asymptomatic | 120 | 267 |
| 235 | 54 | 1 | asymptomatic | 122 | 286 |
| 37 | 57 | 1 | asymptomatic | 150 | 276 |

227 rows × 5 columns

In [13]:

```
x_test
```

Out[13]:

| | Age | Sex | ChestPain | RestBP | Chol |
|---|---|---|---|---|---|
| 204 | 43 | 1 | asymptomatic | 110 | 211 |
| 159 | 68 | 1 | nonanginal | 118 | 277 |
| 219 | 59 | 1 | asymptomatic | 138 | 271 |
| 174 | 64 | 1 | asymptomatic | 145 | 212 |
| 184 | 60 | 0 | asymptomatic | 158 | 305 |
| ... | ... | ... | ... | ... | ... |
| 131 | 51 | 1 | nonanginal | 94 | 227 |
| 234 | 54 | 0 | nonanginal | 160 | 201 |
| 107 | 57 | 1 | nonanginal | 128 | 229 |
| 285 | 58 | 1 | asymptomatic | 114 | 318 |
| 17 | 54 | 1 | asymptomatic | 140 | 239 |

76 rows × 5 columns

In [14]:

```
y_train
```

Out[14]:

```
170    Yes
192    Yes
168    Yes
42      No
90      No
      ...
203     No
255     No
72     Yes
235    Yes
37     Yes
Name: AHD, Length: 227, dtype: object
```

In [15]:

```
y_test
```

Out[15]:

```
204     No
159     No
219     No
174    Yes
184    Yes
      ...
131     No
234     No
107    Yes
285    Yes
17      No
Name: AHD, Length: 76, dtype: object
```

In [ ]:

```

```

**Assignment -2**
**Machine Learning LP-1**

**Name: Prasad Sanjay Khalkar**
**Roll No: 33138**
**TE-09 L-09**

**Problem Statement:**
Download temperature data from below link. https://www.kaggle.com/venky73/temperatures-of-india?select=temperatures.csv
This data consists of temperatures of INDIA averaging the temperatures of all places month wise. Temperatures values are recorded in CELSIUS
A. Apply Linear Regression using 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 simple regression model.

---

**Theory:**

**Definition of Linear Regression**

In layman terms, we can define linear regression as it is used for learning the linear relationship between the target and one or more forecasters, and it is probably one of the most popular andwell inferential algorithms in statistics. Linear regression endeavours to demonstrate the connection between two variables by fitting a linear equation to observed information. One variable is viewed as an explanatory variable, and the other is viewed as a dependent variable.

**Types of Linear Regression**

Normally, linear regression is divided into two types: Multiple linear regression and Simple linear regression.

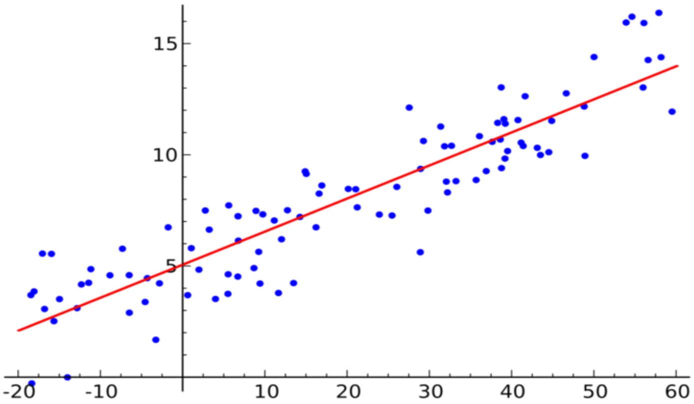
1. **Multiple Linear Regression**

   In this type of linear regression, we always attempt to discover the relationship between twoor more independent variables or inputs and the corresponding dependent variable or outputand the independent variables can be either continuous or categorical.
   This linear regression analysis is very helpful in several ways like it helps in foreseeing trends, future values, and moreover predict the impacts of changes.

2. **Simple Linear Regression**

   In simple linear regression, we aim to reveal the relationship between a single independent variable or you can say input, and a corresponding dependent variable or output. We can discuss this in a simple line as $y = \beta 0 + \beta 1x + \varepsilon$

Here, Y speaks to the output or dependent variable, β0 and β1 are two obscure constants thatspeak to the intercept and coefficient that is slope separately, and the error term is ε Epsilon.We can also discuss this in the form of a graph and here is a sample simple linear regression model graph.




Simple Linear Regression graph [3]

**<u>What Actually is Simple Linear Regression?</u>**

It can be described as a method of statistical analysis that can be used to study the relationship between two quantitative variables.

Primarily, there are two things which can be found out by using the method of simple linear regression:

1. **Strength of the relationship between the given duo of variables.** (For example, the relationship between global warming and the melting of glaciers)
2. **How much the value of the dependent variable is at a given value of the independent variable.** (For example, the amount of melting of a glacier at a certain level of global warming or temperature)

Regression models are used for the elaborated explanation of the relationship between two given variables. There are certain types of regression models like <u>logistic regression models</u>, nonlinear regression models, and linear regression models. The linear regression model fits a straight line into the summarized data to establish the relationship between two variables.

**<u>Assumptions of Linear Regression</u>**

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

- **<u>Homogeneity of variance (homoscedasticity)-</u>** One of the main predictions in a simple linear regression method is that the size of the error stays constant. This simply means thatin the value of the independent variable, the error size never changes significantly.
- **<u>Independence of observations-</u>** All the relationships between the observations are transparent, which means that nothing is hidden, and only valid sampling methods are usedduring the collection of data.

- **Normality-** There is a normal rate of flow in the data.

**Conclusion:** Implemented and understood simple linear regression on the temperature.csv files and also visualised the model.

**Implementation:**
**Implementation is as shown below:**

## Name: Prasad Sanjay Khalkar

## Roll No: 33138

## TE-09 (L-09)

In [48]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [3]:

```
df = pd.read_csv("/home/prasadkhalkar/Desktop/ML/Datasets/temperatures.csv")
temp = df.copy()
temp
```

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 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **112** | 2013 | 24.56 | 26.59 | 30.62 | 32.66 | 34.46 | 32.44 | 31.07 | 30.76 | 31.04 | 30.27 | 27.83 | 25.37 | 29.81 | 25.58 | 32.58 | 31.33 | 27.83 |
| **113** | 2014 | 23.83 | 25.97 | 28.95 | 32.74 | 33.77 | 34.15 | 31.85 | 31.32 | 30.68 | 30.29 | 28.05 | 25.08 | 29.72 | 24.90 | 31.82 | 32.00 | 27.81 |
| **114** | 2015 | 24.58 | 26.89 | 29.07 | 31.87 | 34.09 | 32.48 | 31.88 | 31.52 | 31.55 | 31.04 | 28.10 | 25.67 | 29.90 | 25.74 | 31.68 | 31.87 | 28.27 |
| **115** | 2016 | 26.94 | 29.72 | 32.62 | 35.38 | 35.72 | 34.03 | 31.64 | 31.79 | 31.66 | 31.98 | 30.11 | 28.01 | 31.63 | 28.33 | 34.57 | 32.28 | 30.03 |
| **116** | 2017 | 26.45 | 29.46 | 31.60 | 34.95 | 35.84 | 33.82 | 31.88 | 31.72 | 32.22 | 32.29 | 29.60 | 27.18 | 31.42 | 27.95 | 34.13 | 32.41 | 29.69 |

117 rows × 18 columns

In [4]:

```
temp.describe()
```

Out[4]:

| | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC | ANNUAL | JA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 117.000000 | 117.000000 | 117. |
| **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 | 24.608291 | 29.181368 | 24. |
| **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 | 0.782644 | 0.555555 | 0. |
| **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 | 23.020000 | 28.110000 | 22. |
| **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 | 24.040000 | 28.760000 | 24. |
| **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 | 24.660000 | 29.090000 | 24. |
| **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 | 25.110000 | 29.470000 | 25. |
| **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 | 28.010000 | 31.630000 | 28. |

In [5]:

```
temp.isnull().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 [6]:

```
temp.info()
```

```
<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
```

In [16]:

```
temp.corr()
```

Out[16]:

| | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC | ANNUAL | JAN-FEB | MAR-MAY | JUN-SEP | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **YEAR** | 1.000000 | 0.575499 | 0.647066 | 0.553886 | 0.540662 | 0.407648 | 0.371840 | 0.478512 | 0.654138 | 0.664008 | 0.589073 | 0.697887 | 0.732222 | 0.801129 | 0.679869 | 0.640438 | 0.677061 | 0.7 |
| **JAN** | 0.575499 | 1.000000 | 0.647017 | 0.457081 | 0.594674 | 0.365236 | 0.292855 | 0.339337 | 0.459944 | 0.499764 | 0.480695 | 0.526615 | 0.595902 | 0.749880 | 0.874226 | 0.575734 | 0.496515 | 0.6 |
| **FEB** | 0.647066 | 0.647017 | 1.000000 | 0.589088 | 0.548803 | 0.377722 | 0.341302 | 0.418956 | 0.503188 | 0.472755 | 0.466916 | 0.519595 | 0.619320 | 0.792541 | 0.928731 | 0.635904 | 0.544527 | 0.6 |
| **MAR** | 0.553886 | 0.457081 | 0.589088 | 1.000000 | 0.618621 | 0.387756 | 0.228349 | 0.232647 | 0.382344 | 0.370066 | 0.312226 | 0.498202 | 0.523316 | 0.689205 | 0.584612 | 0.848637 | 0.380640 | 0.5 |
| **APR** | 0.540662 | 0.594674 | 0.548803 | 0.618621 | 1.000000 | 0.563317 | 0.299866 | 0.286052 | 0.490668 | 0.437970 | 0.473873 | 0.538037 | 0.579775 | 0.770596 | 0.643942 | 0.878402 | 0.474542 | 0.5 |
| **MAY** | 0.407648 | 0.365236 | 0.377722 | 0.387756 | 0.563317 | 1.000000 | 0.274521 | 0.299072 | 0.473171 | 0.347289 | 0.468993 | 0.482822 | 0.444695 | 0.609015 | 0.403316 | 0.708221 | 0.431314 | 0.5 |
| **JUN** | 0.371840 | 0.292855 | 0.341302 | 0.228349 | 0.299866 | 0.274521 | 1.000000 | 0.480925 | 0.504354 | 0.305761 | 0.380782 | 0.419968 | 0.366242 | 0.520189 | 0.351115 | 0.341301 | 0.749132 | 0.4 |
| **JUL** | 0.478512 | 0.339337 | 0.418956 | 0.232647 | 0.286052 | 0.299072 | 0.480925 | 1.000000 | 0.622985 | 0.531865 | 0.568341 | 0.535413 | 0.440813 | 0.588454 | 0.423876 | 0.321388 | 0.799602 | 0.5 |
| **AUG** | 0.654138 | 0.459944 | 0.503188 | 0.382344 | 0.490668 | 0.473171 | 0.504354 | 0.622985 | 1.000000 | 0.680212 | 0.661177 | 0.588961 | 0.595330 | 0.755384 | 0.534818 | 0.560118 | 0.866202 | 0.6 |
| **SEP** | 0.664008 | 0.499764 | 0.472755 | 0.370066 | 0.437970 | 0.347289 | 0.305761 | 0.531865 | 0.680212 | 1.000000 | 0.680744 | 0.683866 | 0.629223 | 0.730756 | 0.529533 | 0.485397 | 0.778875 | 0.7 |
| **OCT** | 0.589073 | 0.480695 | 0.466916 | 0.312226 | 0.473873 | 0.468993 | 0.380782 | 0.568341 | 0.661177 | 0.680744 | 1.000000 | 0.770277 | 0.719305 | 0.768170 | 0.506640 | 0.522917 | 0.705733 | 0.8 |
| **NOV** | 0.697887 | 0.526615 | 0.519595 | 0.498202 | 0.538037 | 0.482822 | 0.419968 | 0.535413 | 0.588961 | 0.683866 | 0.770277 | 1.000000 | 0.782075 | 0.812868 | 0.568893 | 0.620161 | 0.692585 | 0.9 |
| **DEC** | 0.732222 | 0.595902 | 0.619320 | 0.523316 | 0.579775 | 0.444695 | 0.366242 | 0.440813 | 0.595330 | 0.629223 | 0.719305 | 0.782075 | 1.000000 | 0.843660 | 0.663719 | 0.643015 | 0.634747 | 0.9 |
| **ANNUAL** | 0.801129 | 0.749880 | 0.792541 | 0.689205 | 0.770596 | 0.609015 | 0.520189 | 0.588454 | 0.755384 | 0.730756 | 0.768170 | 0.812868 | 0.843660 | 1.000000 | 0.849828 | 0.853277 | 0.810786 | 0.8 |
| **JAN-FEB** | 0.679869 | 0.874226 | 0.928731 | 0.584612 | 0.643942 | 0.403316 | 0.351115 | 0.423876 | 0.534818 | 0.529533 | 0.506640 | 0.568893 | 0.663719 | 0.849828 | 1.000000 | 0.675918 | 0.575513 | 0.6 |
| **MAR-MAY** | 0.640438 | 0.575734 | 0.635904 | 0.848637 | 0.878402 | 0.708221 | 0.341301 | 0.321388 | 0.560118 | 0.485397 | 0.522917 | 0.620161 | 0.643015 | 0.853277 | 0.675918 | 1.000000 | 0.534279 | 0.6 |
| **JUN-SEP** | 0.677061 | 0.496515 | 0.544527 | 0.380640 | 0.474542 | 0.431314 | 0.749132 | 0.799602 | 0.866202 | 0.778875 | 0.705733 | 0.692585 | 0.634747 | 0.810786 | 0.575513 | 0.534279 | 1.000000 | 0.7 |
| **OCT-DEC** | 0.749792 | 0.607752 | 0.609839 | 0.505879 | 0.596943 | 0.503445 | 0.409325 | 0.541023 | 0.665040 | 0.734650 | 0.888144 | 0.913522 | 0.922692 | 0.897046 | 0.661805 | 0.664773 | 0.730397 | 1.0 |

In [8]:

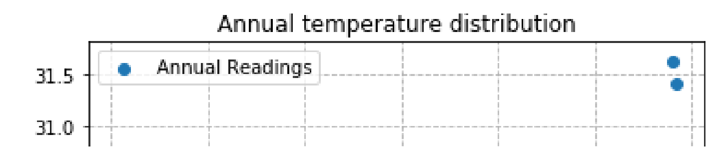
```
temp.shape
```

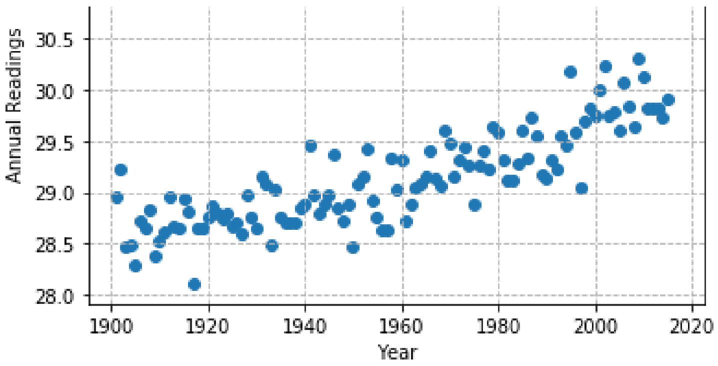
Out[8]:

```
(117, 18)
```

In [9]:

```
x = temp["YEAR"]
y = temp["ANNUAL"]

plt.scatter(x,y)
plt.xlabel("Year")
plt.ylabel("Annual Readings")
plt.title("Annual temperature distribution")
plt.grid(ls='--')
plt.legend(['Annual Readings'],loc=2)
plt.show()
```






# Linear Regression for December

## Splitting data into train and test

In [30]:

```
x = temp[['YEAR']]
y = temp[['DEC']]
```

In [31]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=1)
```

## Training the model

In [35]:

```
lm = LinearRegression()
```

In [36]:

```
model = lm.fit(x_train,y_train)
```

In [38]:

```
b1 = model.coef_
b0 = model.intercept_
```

In [39]:

```
print(b1)
print(b0)
```

```
[[0.01706226]]
[-8.77738666]
```

## Predicting

In [41]:

```
pred = model.predict(x_test)
```

In [44]:

```
d = pd.DataFrame(x_test)
```

In [45]:

```
d['Actual_temp'] = y_test
d['Predicted_temp'] = pred
```

In [46]:

```
d
```

Out[46]:

| | YEAR | Actual_temp | Predicted_temp |
|---|---|---|---|
| **69** | 1970 | 25.07 | 24.835263 |
| **46** | 1947 | 24.31 | 24.442831 |
| **58** | 1959 | 25.00 | 24.647578 |
| **114** | 2015 | 25.67 | 25.603065 |
| **73** | 1974 | 23.63 | 24.903512 |
| **98** | 1999 | 25.72 | 25.330069 |
| **31** | 1932 | 24.52 | 24.186897 |
| **53** | 1954 | 24.42 | 24.562267 |
| **65** | 1966 | 24.70 | 24.767014 |
| **96** | 1997 | 23.92 | 25.295944 |
| **95** | 1996 | 24.83 | 25.278882 |
| **97** | 1998 | 25.15 | 25.313006 |
| **2** | 1903 | 23.65 | 23.692092 |

| | YEAR | Actual_temp | Predicted_temp |
|---|---|---|---|
| **62** | 1963 | 24.33 | 24.715827 |
| **110** | 2011 | 25.60 | 25.534816 |
| **55** | 1956 | 24.30 | 24.596391 |
| **103** | 2004 | 25.54 | 25.415380 |
| **100** | 2001 | 25.33 | 25.364193 |
| **66** | 1967 | 24.10 | 24.784076 |
| **44** | 1945 | 23.96 | 24.408707 |
| **77** | 1978 | 25.21 | 24.971761 |
| **17** | 1918 | 23.32 | 23.948026 |
| **81** | 1982 | 24.69 | 25.040010 |
| **74** | 1975 | 24.66 | 24.920574 |
| **56** | 1957 | 24.52 | 24.613454 |
| **94** | 1995 | 26.21 | 25.261819 |
| **35** | 1936 | 23.72 | 24.255146 |
| **38** | 1939 | 24.85 | 24.306333 |
| **93** | 1994 | 24.98 | 25.244757 |
| **48** | 1949 | 24.21 | 24.476956 |

## Finding Errors

In [57]:

```
MAE = mean_absolute_error(y_test,pred)
MSE = mean_squared_error(y_test,pred)
RSE = r2_score(y_test,pred)
```

In [56]:

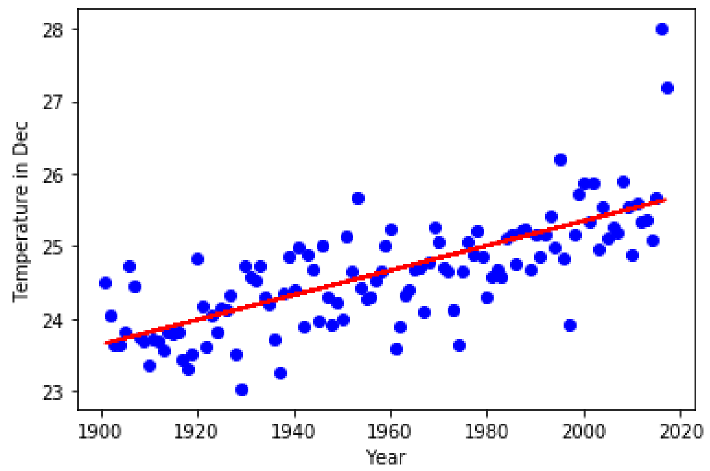
```
print('Mean Absolute Error = %.3f'%MAE)
print('Mean Squared Error = %.3f'%MSE)
print('RSE = %.3f'%RSE)
```

```
Mean Absolute Error = 0.373
Mean Squared Error = 0.248
RSE = 0.484
```

## Plotting Line

In [58]:

```
plt.scatter(temp['YEAR'],temp['DEC'],c='blue')
plt.xlabel('Year')
plt.ylabel('Temperature in Dec')
plt.plot(x_train,b0+b1*x_train,c='red')
plt.show()
```




# Linear Regression for September

## Splitting data into train and test

In [60]:

```
y1 = temp[['SEP']]
```

In [61]:

```
x1_train,x1_test,y1_train,y1_test = train_test_split(x,y1,test_size=0.25,random_state=1)
```

## Training the model

In [62]:

```
model1 = lm.fit(x1_train,y1_train)
```

In [63]:

```
slope1 = model1.coef_
inter1 = model1.intercept_
```

In [64]:

```
print(slope1,inter1)
```

```
[[0.0102336]] [10.42872908]
```

## Predicting

In [65]:

```
pred1 = model1.predict(x1_test)
d1 = pd.DataFrame(x1_test)
```

In [66]:

```
d1['Actual_temp'] = y1_test
d1['Predicted_temp'] = pred1
```

In [67]:

```
d1
```

Out[67]:

| | YEAR | Actual_temp | Predicted_temp |
|---|---|---|---|
| 69 | 1970 | 30.41 | 30.588919 |
| 46 | 1947 | 29.70 | 30.353546 |
| 58 | 1959 | 30.39 | 30.476350 |
| 114 | 2015 | 31.55 | 31.049431 |
| 73 | 1974 | 30.87 | 30.629854 |
| 98 | 1999 | 31.22 | 30.885694 |
| 31 | 1932 | 30.83 | 30.200042 |
| 53 | 1954 | 29.87 | 30.425182 |
| 65 | 1966 | 30.25 | 30.547985 |
| 96 | 1997 | 31.11 | 30.865226 |
| 95 | 1996 | 30.86 | 30.854993 |
| 97 | 1998 | 30.73 | 30.875460 |
| 2 | 1903 | 29.85 | 29.903268 |
| 62 | 1963 | 30.51 | 30.517284 |
| 110 | 2011 | 30.81 | 31.008497 |
| 55 | 1956 | 30.29 | 30.445649 |
| 103 | 2004 | 31.20 | 30.936862 |
| 100 | 2001 | 31.66 | 30.906161 |
| 66 | 1967 | 30.64 | 30.558218 |
| 44 | 1945 | 30.55 | 30.333079 |
| 77 | 1978 | 30.60 | 30.670788 |
| 17 | 1918 | 30.27 | 30.056772 |
| 81 | 1982 | 31.01 | 30.711722 |
| 74 | 1975 | 29.62 | 30.640087 |
| 56 | 1957 | 30.56 | 30.455882 |
| 94 | 1995 | 31.24 | 30.844759 |
| 35 | 1936 | 30.40 | 30.240977 |
| 38 | 1939 | 30.54 | 30.271678 |
| 93 | 1994 | 30.78 | 30.834526 |
| 48 | 1949 | 30.36 | 30.374014 |

## Finding Errors

In [68]:

```
MAE1 = mean_absolute_error(y1_test,pred1)
MSE1 = mean_squared_error(y1_test,pred1)
RSE1 = r2_score(y1_test,pred1)
```

In [69]:

```
print('Mean Absolute Error = %.3f'%MAE1)
print('Mean Squared Error = %.3f'%MSE1)
print('RSE = %.3f'%RSE1)
```

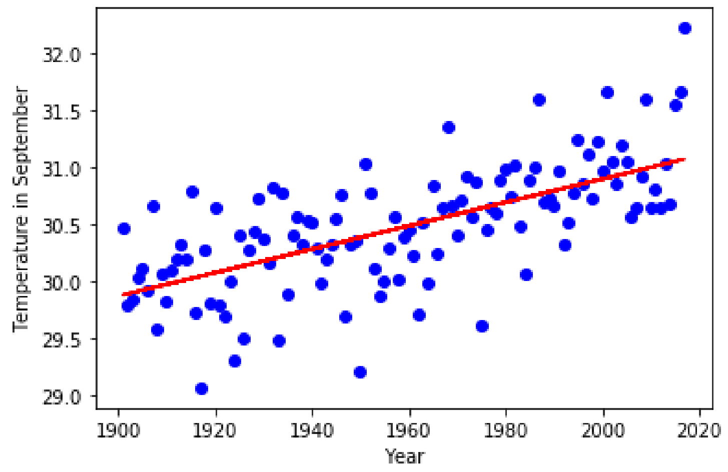
```
Mean Absolute Error = 0.273
Mean Squared Error = 0.132
RSE = 0.454
```

## Plotting Line

### Plotting Line

In [76]:

```
plt.scatter(temp['YEAR'],temp['SEP'],c='blue')
plt.xlabel('Year')
plt.ylabel('Temperature in September')
plt.plot(x1_train,inter1+slope1*x1_train,c='red')
plt.show()
```




## Linear Regression for Annual temp

### Splitting the data into train and test

In [73]:

```
y2 = temp[['ANNUAL']]
```

In [74]:

```
x2_train,x2_test,y2_train,y2_test = train_test_split(x,y2,test_size=0.25,random_state=1)
```

### Training the model

In [77]:

```
model2 = lm.fit(x2_train,y2_train)
```

In [78]:

```
slope2 = model2.coef_
inter2 = model2.intercept_
```

In [79]:

```
print(slope2,inter2)
```

```
[[0.0134026]] [2.9609047]
```

### Predicting

In [81]:

```
pred2 = model2.predict(x2_test)
d2 = pd.DataFrame(x2_test)
```

In [82]:

```
d2['Actual_temp'] = y2_test
d2['Predicted_temp'] = pred2
```

In [83]:

```
d2
```

Out[83]:

| | YEAR | Actual_temp | Predicted_temp |
|---|---|---|---|
| 69 | 1970 | 29.47 | 29.364036 |
| 46 | 1947 | 28.84 | 29.055776 |
| 58 | 1959 | 29.02 | 29.216607 |
| 114 | 2015 | 29.90 | 29.967153 |
| 73 | 1974 | 29.26 | 29.417646 |
| 98 | 1999 | 29.81 | 29.752711 |
| 31 | 1932 | 29.09 | 28.854737 |
| 53 | 1954 | 28.92 | 29.149594 |
| 65 | 1966 | 29.41 | 29.310425 |
| 96 | 1997 | 29.05 | 29.725906 |
| 95 | 1996 | 29.58 | 29.712503 |
| 97 | 1998 | 29.70 | 29.739309 |

| | YEAR | Actual_temp | Predicted_temp |
|---|---|---|---|
| 97 | 1998 | 29.70 | 29.739309 |
| 2 | 1903 | 28.47 | 28.466061 |
| 62 | 1963 | 29.04 | 29.270217 |
| 110 | 2011 | 29.82 | 29.913542 |
| 55 | 1956 | 28.63 | 29.176399 |
| 103 | 2004 | 29.79 | 29.819724 |
| 100 | 2001 | 29.99 | 29.779516 |
| 66 | 1967 | 29.14 | 29.323828 |
| 44 | 1945 | 28.97 | 29.028971 |
| 77 | 1978 | 29.23 | 29.471256 |
| 17 | 1918 | 28.66 | 28.667100 |
| 81 | 1982 | 29.12 | 29.524867 |
| 74 | 1975 | 28.89 | 29.431049 |
| 56 | 1957 | 28.64 | 29.189802 |
| 94 | 1995 | 30.18 | 29.699101 |
| 35 | 1936 | 28.71 | 28.908347 |
| 38 | 1939 | 28.85 | 28.948555 |
| 93 | 1994 | 29.46 | 29.685698 |
| 48 | 1949 | 28.89 | 29.082581 |

## Finding Errors

In [84]:

```
MAE2 = mean_absolute_error(y2_test,pred2)
MSE2 = mean_squared_error(y2_test,pred2)
RSE2 = r2_score(y2_test,pred2)
```

In [85]:

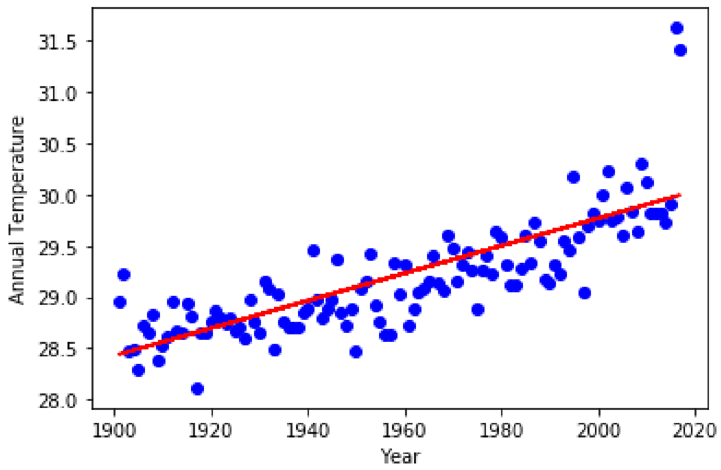
```
print('Mean Absolute Error = %.3f'%MAE2)
print('Mean Squared Error = %.3f'%MSE2)
print('RSE = %.3f'%RSE2)
```

```
Mean Absolute Error = 0.217
Mean Squared Error = 0.078
RSE = 0.617
```

## Plotting Line

In [98]:

```
plt.scatter(temp['YEAR'],temp['ANNUAL'],c='blue')
plt.xlabel('Year')
plt.ylabel('Annual Temperature')
plt.plot(x2_train,inter2+slope2*x2_train,c='red')
plt.show()
```




# Regression for Jan

## Splitting the data into train and test

In [88]:

```
y3 = temp[['JAN']]
```

In [89]:

```
x3_train,x3_test,y3_train,y3_test = train_test_split(x,y3,test_size=0.25,random_state=1)
```

## Training the model

In [90]:

```
model3 = lm.fit(x3_train,y3_train)
```

In [91]:

```
slope3 = model3.coef_
inter3 = model3.intercept_
```

In [92]:

```
print(slope3,inter3)
```

```
[[0.01468067]] [-5.00631942]
```

## Predicting

In [93]:

```
pred3 = model3.predict(x3_test)
d3 = pd.DataFrame(x3_test)
```

In [94]:

```
d3['Actual_temp'] = y3_test
d3['Predicted_temp'] = pred3
```

In [95]:

```
d3
```

Out[95]:

| | YEAR | Actual_temp | Predicted_temp |
|---|---|---|---|
| 69 | 1970 | 24.19 | 23.914602 |
| 46 | 1947 | 22.61 | 23.576947 |
| 58 | 1959 | 23.33 | 23.753115 |
| 114 | 2015 | 24.58 | 24.575233 |
| 73 | 1974 | 23.54 | 23.973325 |
| 98 | 1999 | 23.57 | 24.340342 |
| 31 | 1932 | 24.13 | 23.356737 |
| 53 | 1954 | 22.79 | 23.679712 |
| 65 | 1966 | 24.11 | 23.855880 |
| 96 | 1997 | 23.30 | 24.310981 |
| 95 | 1996 | 25.18 | 24.296300 |
| 97 | 1998 | 23.95 | 24.325661 |
| 2 | 1903 | 23.44 | 22.930997 |
| 62 | 1963 | 22.90 | 23.811838 |
| 110 | 2011 | 24.18 | 24.516510 |
| 55 | 1956 | 23.16 | 23.709073 |
| 103 | 2004 | 23.89 | 24.413745 |
| 100 | 2001 | 24.36 | 24.369703 |
| 66 | 1967 | 23.72 | 23.870560 |
| 44 | 1945 | 22.38 | 23.547586 |
| 77 | 1978 | 23.60 | 24.032048 |
| 17 | 1918 | 22.06 | 23.151208 |
| 81 | 1982 | 24.23 | 24.090770 |
| 74 | 1975 | 23.15 | 23.988006 |
| 56 | 1957 | 22.98 | 23.723754 |
| 94 | 1995 | 24.44 | 24.281619 |
| 35 | 1936 | 23.10 | 23.415460 |
| 38 | 1939 | 23.61 | 23.459502 |
| 93 | 1994 | 24.67 | 24.266939 |
| 48 | 1949 | 24.31 | 23.606308 |

## Finding errors

In [96]:

```
MAE3 = mean_absolute_error(y3_test,pred3)
MSE3 = mean_squared_error(y3_test,pred3)
RSE3 = r2_score(y3_test,pred3)
```

In [97]:

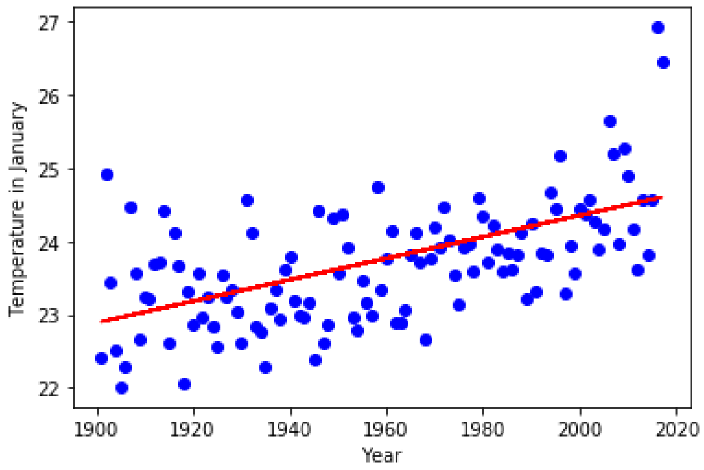
```
print('Mean Absolute Error = %.3f'%MAE3)
print('Mean Squared Error = %.3f'%MSE3)
print('RSE = %.3f'%RSE3)
```

```
Mean Absolute Error = 0.540
Mean Squared Error = 0.400
RSE = 0.220
```

## Plotting the line

In [99]:

```
plt.scatter(temp['YEAR'],temp['JAN'],c='blue')
plt.xlabel('Year')
plt.ylabel('Temperature in January')
plt.plot(x3_train,inter3+slope3*x3_train,c='red')
plt.show()
```




In [ ]:

**Assignment -3**

**Machine Learning LP-1**

**Name: Prasad Sanjay Khalkar**
**Roll No: 33138**
**TE-09 L-09**

**Problem Statement:**
Every year many students give the GRE exam to get admission in foreign Universities. The dataset contains GRE Scores (out of 340), TOEFL Scores (out of 120), University Rating (out of 5), Statement of Purpose strength (out of 5), Letter of Recommendation strength (out of 5), Undergraduate GPA (out of 10), Research Experience (0=no, 1=yes), Admitted (0=no, 1=yes). Admitted is the target variable.
Data Set Available on kaggle (The last column of the dataset needs to be changed to 0 or 1) Data Set: https://www.kaggle.com/mohansacharya/graduate-admissions.
The counselor of the firm is supposed check whether the student will get an admission or not based on his/her GRE score and Academic Score.
So, to help the counselor to take appropriate decisions build a machine learning model classifier using Decision tree to predict whether a student will get admission or not.
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.

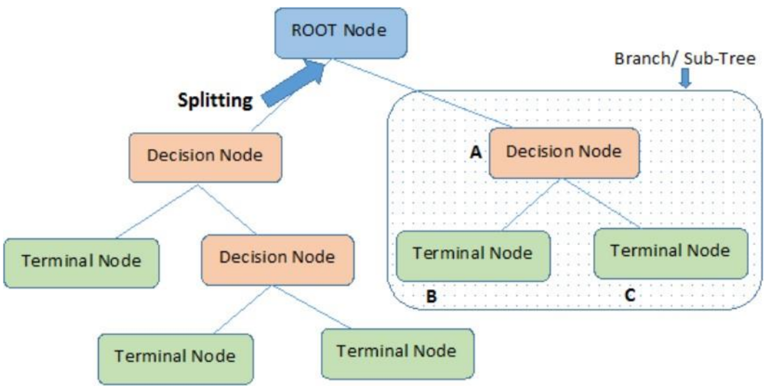
---

**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.[1]



[1]

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

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

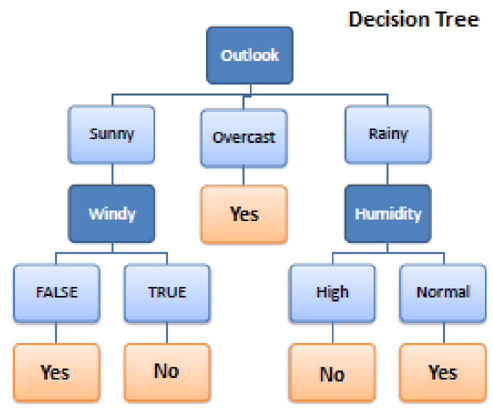
***Leaf Nodes*** – the nodes where further splitting is not possible are called leaf nodes or terminalnodes

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

***Pruning*** – It is cutting down some nodes to stop overfitting.

Predictors | Target

| Outlook | Temp | Humidity | Windy | Play Golf |
|---|---|---|---|---|
| Rainy | Hot | High | False | No |
| Rainy | Hot | High | True | No |
| Overcast | Hot | High | False | Yes |
| Sunny | Mild | High | False | Yes |
| Sunny | Cool | Normal | False | Yes |
| Sunny | Cool | Normal | True | No |
| Overcast | Cool | Normal | True | Yes |
| Rainy | Mild | High | False | No |
| Rainy | Cool | Normal | False | Yes |
| Sunny | Mild | Normal | False | Yes |
| Rainy | Mild | Normal | True | Yes |
| Overcast | Mild | High | True | Yes |
| Overcast | Hot | Normal | False | Yes |
| Sunny | Mild | High | True | No |




$$E(T,X) = \sum_{c \in X} P(c)E(c)$$ [2]

| | | 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).[2]

*Step 1*: Calculate entropy of the target.

$$\begin{aligned} \textbf{Entropy(PlayGolf)} &= \text{Entropy}(5,9) \\ &= \text{Entropy}(0.36, 0.64) \\ &= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64) \\ &= 0.94 \end{aligned}$$

*Step 2*: The dataset is then split on the different attributes. The entropy for each branch iscalculated.
Then it is added proportionally, to get total entropy for the split. The resulting entropy issubtracted
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 | | | |

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

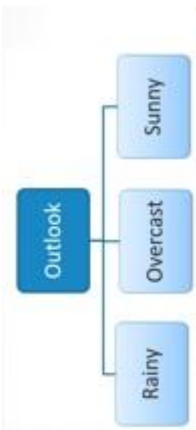
$$\begin{aligned} \mathbf{G}(\text{PlayGolf, Outlook}) &= \mathbf{E}(\text{PlayGolf}) - \mathbf{E}(\text{PlayGolf, Outlook}) \\ &= 0.940 - 0.693 = 0.247 \end{aligned}$$

[2]

*Step 3*: Choose attribute with the largest information gain as the decision node, divide the dataset by its

branches and repeat the same process on every branch.

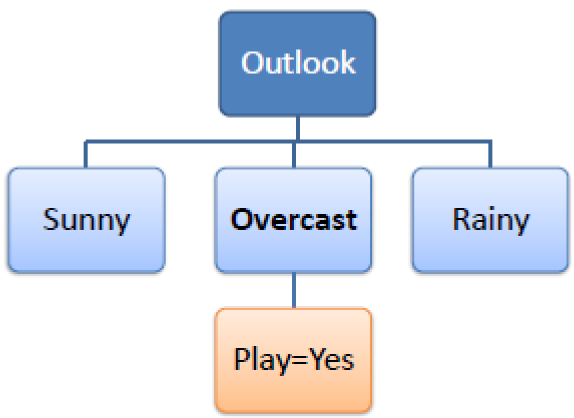
| ★ | | Play Golf | |
|---|---|---|---|
| | | Yes | No |
| Outlook | Sunny | 3 | 2 |
| | Overcast | 4 | 0 |
| | Rainy | 2 | 3 |
| Gain = 0.247 | | | |




| 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 |

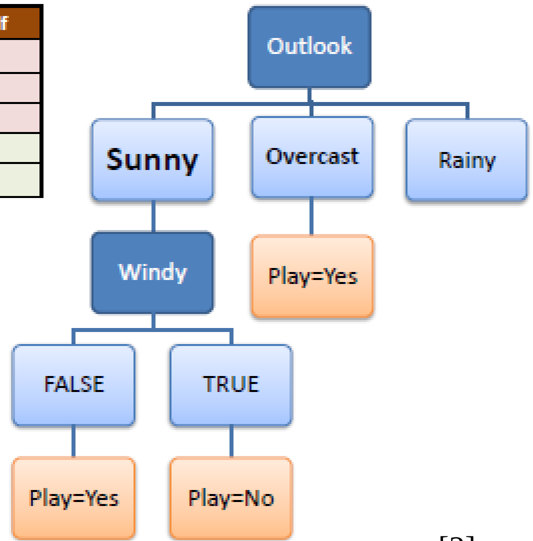
*Step 4a*: A branch with entropy of 0 is a leaf node.

| 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.

| Temp. | Humidity | Windy | Play Golf |
|---|---|---|---|
| Mild | High | FALSE | Yes |
| Cool | Normal | FALSE | Yes |
| Mild | Normal | FALSE | Yes |
| Cool | Normal | TRUE | No |
| Mild | High | TRUE | No |




[2]

*Step 5*: The ID3 algorithm is run recursively on the non-leaf branches, until all data isclassified.

**Decision Tree to Decision Rules**
A decision tree can easily be transformed to a set of rules by mapping fromthe root node to the leaf nodes one by one.

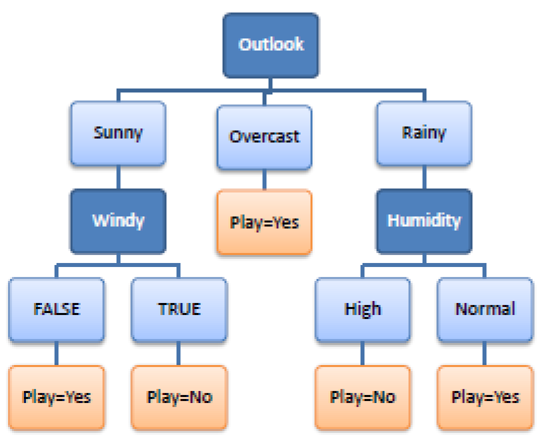
$R_1$: IF (Outlook=Sunny) AND (Windy=FALSE) THEN Play=Yes

$R_2$: IF (Outlook=Sunny) AND (Windy=TRUE) THEN Play=No

$R_3$: IF (Outlook=Overcast) THEN Play=Yes

$R_4$: IF (Outlook=Rainy) AND (Humidity=High) THEN Play=No

$R_5$: IF (Outlook=Rain) AND (Humidity=Normal) THEN Play=Yes




[2]

**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 ontheir significance.**

**Conclusion:**
Implemented and understood decision tree algorithms and evaluated the model.

**Implemetation:**
**Implemetation is as shown below:**

# Name: Prasad Sanjay Khalkar

# Roll No: 33138

# TE-09 L-09

In [57]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn import tree
```

## EDA

In [58]:

```
df = pd.read_csv("/home/prasadkhalkar/Desktop/ML/Datasets/Admission_Predict_Ver1.1.csv")
adm = df.copy()
adm
```

Out[58]:

| | 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 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **495** | 496 | 332 | 108 | 5 | 4.5 | 4.0 | 9.02 | 1 | 0.87 |
| **496** | 497 | 337 | 117 | 5 | 5.0 | 5.0 | 9.87 | 1 | 0.96 |
| **497** | 498 | 330 | 120 | 5 | 4.5 | 5.0 | 9.56 | 1 | 0.93 |
| **498** | 499 | 312 | 103 | 4 | 4.0 | 5.0 | 8.43 | 0 | 0.73 |
| **499** | 500 | 327 | 113 | 4 | 4.5 | 4.5 | 9.04 | 0 | 0.84 |

500 rows × 9 columns

In [59]:

```
adm.describe()
```

Out[59]:

| | 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 [60]:

```
adm.isnull().sum()
```

Out[60]:

```
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 [61]:

```
adm.info()
```

```
<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
```

In [62]:

```
adm.shape
```

Out[62]:

```
(500, 9)
```

In [63]:

```
adm.corr()
```

Out[63]:

| | Serial No. | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
|---|---|---|---|---|---|---|---|---|---|
| **Serial No.** | 1.000000 | -0.103839 | -0.141696 | -0.067641 | -0.137352 | -0.003694 | -0.074289 | -0.005332 | 0.008505 |
| **GRE Score** | -0.103839 | 1.000000 | 0.827200 | 0.635376 | 0.613498 | 0.524679 | 0.825878 | 0.563398 | 0.810351 |
| **TOEFL Score** | -0.141696 | 0.827200 | 1.000000 | 0.649799 | 0.644410 | 0.541563 | 0.810574 | 0.467012 | 0.792228 |
| **University Rating** | -0.067641 | 0.635376 | 0.649799 | 1.000000 | 0.728024 | 0.608651 | 0.705254 | 0.427047 | 0.690132 |
| **SOP** | -0.137352 | 0.613498 | 0.644410 | 0.728024 | 1.000000 | 0.663707 | 0.712154 | 0.408116 | 0.684137 |
| **LOR** | -0.003694 | 0.524679 | 0.541563 | 0.608651 | 0.663707 | 1.000000 | 0.637469 | 0.372526 | 0.645365 |
| **CGPA** | -0.074289 | 0.825878 | 0.810574 | 0.705254 | 0.712154 | 0.637469 | 1.000000 | 0.501311 | 0.882413 |
| **Research** | -0.005332 | 0.563398 | 0.467012 | 0.427047 | 0.408116 | 0.372526 | 0.501311 | 1.000000 | 0.545871 |
| **Chance of Admit** | 0.008505 | 0.810351 | 0.792228 | 0.690132 | 0.684137 | 0.645365 | 0.882413 | 0.545871 | 1.000000 |

In [64]:

```
adm
```

Out[64]:

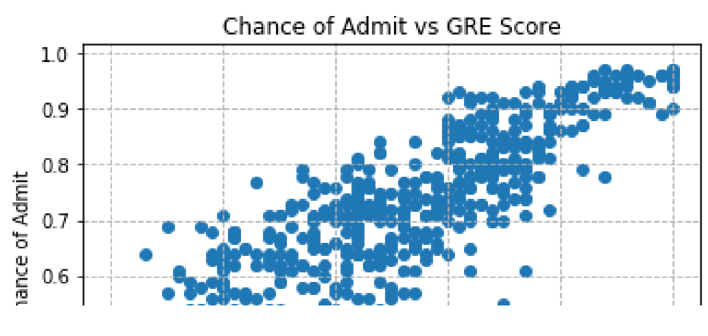
| | Serial No. | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 | 0.92 |
| **1** | 2 | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 | 0.76 |
| **2** | 3 | 316 | 104 | 3 | 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 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **495** | 496 | 332 | 108 | 5 | 4.5 | 4.0 | 9.02 | 1 | 0.87 |
| **496** | 497 | 337 | 117 | 5 | 5.0 | 5.0 | 9.87 | 1 | 0.96 |
| **497** | 498 | 330 | 120 | 5 | 4.5 | 5.0 | 9.56 | 1 | 0.93 |
| **498** | 499 | 312 | 103 | 4 | 4.0 | 5.0 | 8.43 | 0 | 0.73 |
| **499** | 500 | 327 | 113 | 4 | 4.5 | 4.5 | 9.04 | 0 | 0.84 |

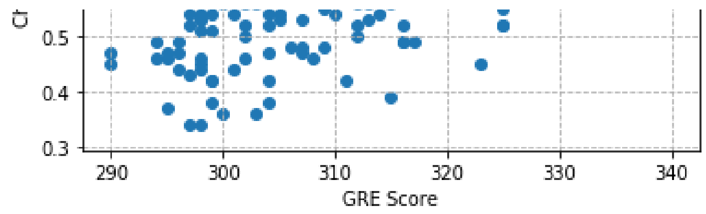
500 rows × 9 columns

## Data Visualisation

In [65]:

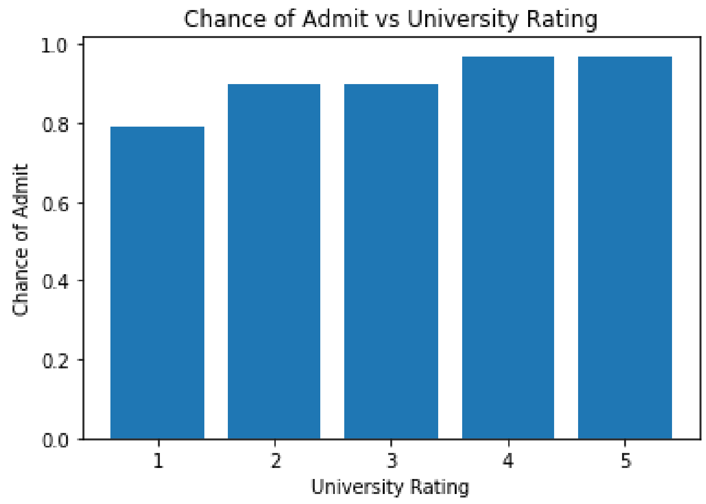
```
y = adm['Chance of Admit ']
x = adm['GRE Score']

plt.scatter(x,y)
plt.ylabel('Chance of Admit')
plt.xlabel('GRE Score')
plt.title('Chance of Admit vs GRE Score')
plt.grid(ls='--')
plt.show()
```





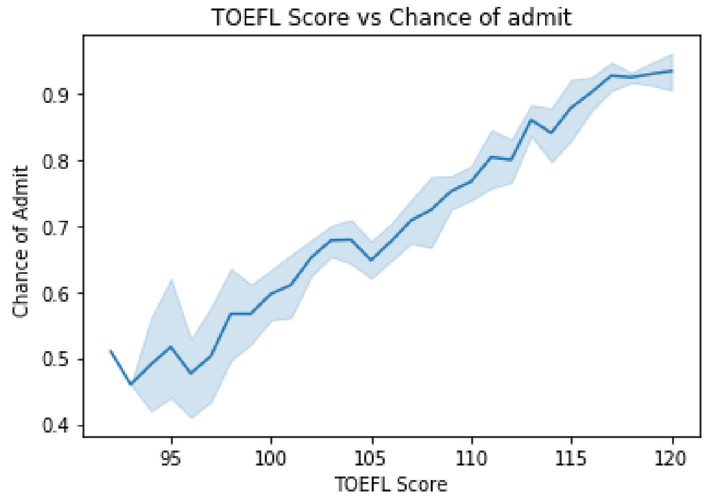

In [66]:

```
x = adm['University Rating']
plt.bar(x,y)
plt.ylabel('Chance of Admit')
plt.xlabel('University Rating')
plt.title('Chance of Admit vs University Rating')
plt.show()
```




In [67]:

```
x = adm["TOEFL Score"]
y = adm["Chance of Admit "]
plt.xlabel("GRE Score")
plt.ylabel("Chance of Admit")
plt.title("TOEFL Score vs Chance of admit")
sns.lineplot(x,y)
plt.show()
```




## Label Encoding and Data Transformation

In [68]:

```
adm['Chance of Admit '].unique()
```

Out[68]:

```
array([0.92, 0.76, 0.72, 0.8 , 0.65, 0.9 , 0.75, 0.68, 0.5 , 0.45, 0.52,
       0.84, 0.78, 0.62, 0.61, 0.54, 0.66, 0.63, 0.64, 0.7 , 0.94, 0.95,
       0.97, 0.44, 0.46, 0.74, 0.91, 0.88, 0.58, 0.48, 0.49, 0.53, 0.87,
       0.86, 0.89, 0.82, 0.56, 0.36, 0.42, 0.47, 0.55, 0.57, 0.96, 0.93,
       0.38, 0.34, 0.79, 0.71, 0.69, 0.59, 0.85, 0.77, 0.81, 0.83, 0.67,
       0.73, 0.6 , 0.43, 0.51, 0.39, 0.37])
```

In [69]:

```
adm.loc[adm['Chance of Admit ']>=0.8,'Chance of Admit '] = 1
adm.loc[adm['Chance of Admit ']<0.8,'Chance of Admit '] = 0
```

In [70]:

```
adm['Chance of Admit '].unique()
```

Out[70]:

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

In [71]:

```
adm['Chance of Admit '] = adm['Chance of Admit '].astype('int64')
```

In [72]:

```
adm
```

Out[72]:

| | 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 | 1 |
| **1** | 2 | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 | 0 |
| **2** | 3 | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 | 0 |
| **3** | 4 | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 | 1 |
| **4** | 5 | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **495** | 496 | 332 | 108 | 5 | 4.5 | 4.0 | 9.02 | 1 | 1 |
| **496** | 497 | 337 | 117 | 5 | 5.0 | 5.0 | 9.87 | 1 | 1 |
| **497** | 498 | 330 | 120 | 5 | 4.5 | 5.0 | 9.56 | 1 | 1 |
| **498** | 499 | 312 | 103 | 4 | 4.0 | 5.0 | 8.43 | 0 | 0 |
| **499** | 500 | 327 | 113 | 4 | 4.5 | 4.5 | 9.04 | 0 | 1 |

500 rows × 9 columns

## Data Preparation using Decision Tree Classifier

In [73]:

```
y = adm['Chance of Admit ']
x = adm.drop(columns=['Serial No.','Chance of Admit '],axis=1)
```

In [74]:

```
x
```

Out[74]:

| | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research |
|---|---|---|---|---|---|---|---|
| **0** | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 |
| **1** | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 |
| **2** | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 |
| **3** | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 |
| **4** | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **495** | 332 | 108 | 5 | 4.5 | 4.0 | 9.02 | 1 |
| **496** | 337 | 117 | 5 | 5.0 | 5.0 | 9.87 | 1 |
| **497** | 330 | 120 | 5 | 4.5 | 5.0 | 9.56 | 1 |
| **498** | 312 | 103 | 4 | 4.0 | 5.0 | 8.43 | 0 |
| **499** | 327 | 113 | 4 | 4.5 | 4.5 | 9.04 | 0 |

500 rows × 7 columns

In [75]:

```
y
```

Out[75]:

```
0      1
1      0
2      0
3      1
4      0
      ..
495    1
496    1
497    1
498    0
499    1
Name: Chance of Admit , Length: 500, dtype: int64
```

In [76]:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=42)
```

In [77]:

```
y_test
```

Out[77]:

```
361    1
73     1
374    0
155    0
104    0
      ..
220    0
176    1
320    0
153    0
231    0
Name: Chance of Admit , Length: 125, dtype: int64
```

In [78]:

In [78]:

```
model = DecisionTreeClassifier()
```

In [79]:

```
model.fit(x_train,y_train)
```

Out[79]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [80]:

```
y_pred = model.predict(x_test)
```

In [81]:

```
final = pd.DataFrame(y_test)
final['Predicted'] = y_pred
```

In [82]:

```
final
```

Out[82]:

| | Chance of Admit | Predicted |
|---|---|---|
| **361** | 1 | 1 |
| **73** | 1 | 0 |
| **374** | 0 | 0 |
| **155** | 0 | 0 |
| **104** | 0 | 1 |
| **...** | ... | ... |
| **220** | 0 | 0 |
| **176** | 1 | 1 |
| **320** | 0 | 0 |
| **153** | 0 | 0 |
| **231** | 0 | 0 |

125 rows × 2 columns

## Without tree pruning

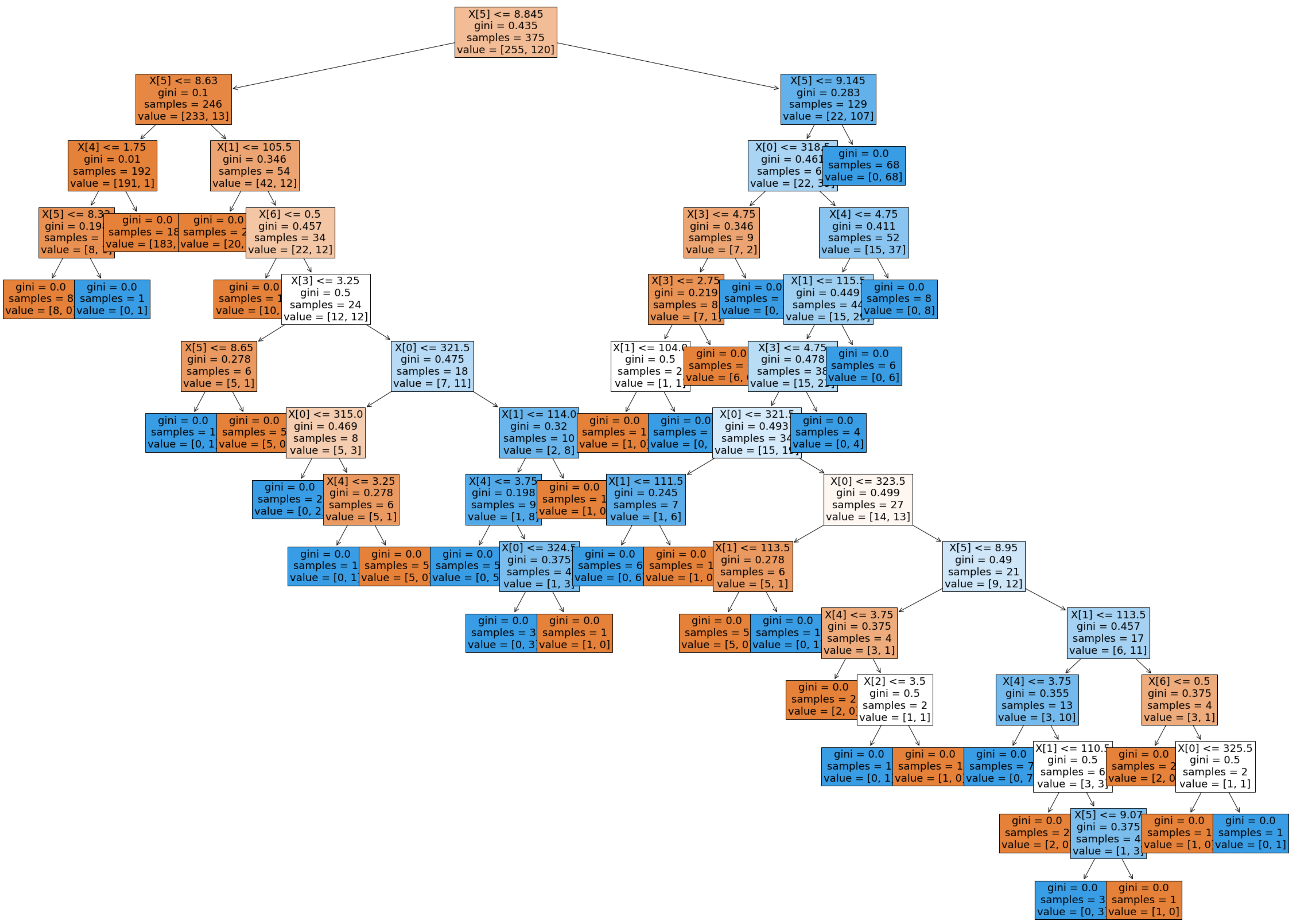
In [83]:

```
plt.figure(figsize=(40,30))
tree.plot_tree(model, filled=True, fontsize=18)
```

Out[83]:

```
[Text(871.875, 1572.5571428571427, 'X[5] <= 8.845\ngini = 0.435\nsamples = 375\nvalue = [255, 120]'),
 Text(310.0, 1456.0714285714284, 'X[5] <= 8.63\ngini = 0.1\nsamples = 246\nvalue = [233, 13]'),
 Text(186.0, 1339.5857142857142, 'X[4] <= 1.75\ngini = 0.01\nsamples = 192\nvalue = [191, 1]'),
 Text(124.0, 1223.1, 'X[5] <= 8.33\ngini = 0.198\nsamples = 9\nvalue = [8, 1]'),
 Text(62.0, 1106.6142857142856, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
 Text(186.0, 1106.6142857142856, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(248.0, 1223.1, 'gini = 0.0\nsamples = 183\nvalue = [183, 0]'),
 Text(434.0, 1339.5857142857142, 'X[1] <= 105.5\ngini = 0.346\nsamples = 54\nvalue = [42, 12]'),
 Text(372.0, 1223.1, 'gini = 0.0\nsamples = 20\nvalue = [20, 0]'),
 Text(496.0, 1223.1, 'X[6] <= 0.5\ngini = 0.457\nsamples = 34\nvalue = [22, 12]'),
 Text(434.0, 1106.6142857142856, 'gini = 0.0\nsamples = 10\nvalue = [10, 0]'),
 Text(558.0, 1106.6142857142856, 'X[3] <= 3.25\ngini = 0.5\nsamples = 24\nvalue = [12, 12]'),
 Text(372.0, 990.1285714285714, 'X[5] <= 8.65\ngini = 0.278\nsamples = 6\nvalue = [5, 1]'),
 Text(310.0, 873.6428571428571, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(434.0, 873.6428571428571, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(744.0, 990.1285714285714, 'X[0] <= 321.5\ngini = 0.475\nsamples = 18\nvalue = [7, 11]'),
 Text(558.0, 873.6428571428571, 'X[0] <= 315.0\ngini = 0.469\nsamples = 8\nvalue = [5, 3]'),
 Text(496.0, 757.1571428571428, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(620.0, 757.1571428571428, 'X[4] <= 3.25\ngini = 0.278\nsamples = 6\nvalue = [5, 1]'),
 Text(558.0, 640.6714285714286, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(682.0, 640.6714285714286, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(930.0, 873.6428571428571, 'X[1] <= 114.0\ngini = 0.32\nsamples = 10\nvalue = [2, 8]'),
 Text(868.0, 757.1571428571428, 'X[4] <= 3.75\ngini = 0.198\nsamples = 9\nvalue = [1, 8]'),
 Text(806.0, 640.6714285714286, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(930.0, 640.6714285714286, 'X[0] <= 324.5\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
 Text(868.0, 524.1857142857143, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(992.0, 524.1857142857143, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(992.0, 757.1571428571428, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(1433.75, 1456.0714285714284, 'X[5] <= 9.145\ngini = 0.283\nsamples = 129\nvalue = [22, 107]'),
 Text(1371.75, 1339.5857142857142, 'X[0] <= 318.5\ngini = 0.461\nsamples = 61\nvalue = [22, 39]'),
 Text(1247.75, 1223.1, 'X[3] <= 4.75\ngini = 0.346\nsamples = 9\nvalue = [7, 2]'),
 Text(1185.75, 1106.6142857142856, 'X[3] <= 2.75\ngini = 0.219\nsamples = 8\nvalue = [7, 1]'),
 Text(1123.75, 990.1285714285714, 'X[1] <= 104.0\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(1061.75, 873.6428571428571, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(1185.75, 873.6428571428571, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(1247.75, 990.1285714285714, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(1309.75, 1106.6142857142856, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(1495.75, 1223.1, 'X[4] <= 4.75\ngini = 0.411\nsamples = 52\nvalue = [15, 37]'),
```

Text(1495.75, 1223.1, 'X[4] <= 4.75\ngini = 0.411\nsamples = 52\nvalue = [15, 37]'),
Text(1433.75, 1106.6142857142856, 'X[1] <= 115.5\ngini = 0.449\nsamples = 44\nvalue = [15, 29]'),
Text(1371.75, 990.1285714285714, 'X[3] <= 4.75\ngini = 0.478\nsamples = 38\nvalue = [15, 23]'),
Text(1309.75, 873.6428571428571, 'X[0] <= 321.5\ngini = 0.493\nsamples = 34\nvalue = [15, 19]'),
Text(1116.0, 757.1571428571428, 'X[1] <= 111.5\ngini = 0.245\nsamples = 7\nvalue = [1, 6]'),
Text(1054.0, 640.6714285714286, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
Text(1178.0, 640.6714285714286, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(1503.5, 757.1571428571428, 'X[0] <= 323.5\ngini = 0.499\nsamples = 27\nvalue = [14, 13]'),
Text(1302.0, 640.6714285714286, 'X[1] <= 113.5\ngini = 0.278\nsamples = 6\nvalue = [5, 1]'),
Text(1240.0, 524.1857142857143, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(1364.0, 524.1857142857143, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(1705.0, 640.6714285714286, 'X[5] <= 8.95\ngini = 0.49\nsamples = 21\nvalue = [9, 12]'),
Text(1488.0, 524.1857142857143, 'X[4] <= 3.75\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
Text(1426.0, 407.70000000000005, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(1550.0, 407.70000000000005, 'X[2] <= 3.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(1488.0, 291.2142857142858, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(1612.0, 291.2142857142858, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(1922.0, 524.1857142857143, 'X[1] <= 113.5\ngini = 0.457\nsamples = 17\nvalue = [6, 11]'),
Text(1798.0, 407.70000000000005, 'X[4] <= 3.75\ngini = 0.355\nsamples = 13\nvalue = [3, 10]'),
Text(1736.0, 291.2142857142858, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
Text(1860.0, 291.2142857142858, 'X[1] <= 110.5\ngini = 0.5\nsamples = 6\nvalue = [3, 3]'),
Text(1798.0, 174.7285714285715, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(1922.0, 174.7285714285715, 'X[5] <= 9.07\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(1860.0, 58.24285714285725, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(1984.0, 58.24285714285725, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(2046.0, 407.70000000000005, 'X[6] <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
Text(1984.0, 291.2142857142858, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(2108.0, 291.2142857142858, 'X[0] <= 325.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(2046.0, 174.7285714285715, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(2170.0, 174.7285714285715, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(1433.75, 873.6428571428571, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(1495.75, 990.1285714285714, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
Text(1557.75, 1106.6142857142856, 'gini = 0.0\nsamples = 8\nvalue = [0, 8]'),
Text(1495.75, 1339.5857142857142, 'gini = 0.0\nsamples = 68\nvalue = [0, 68]')]




### Confusion Matrix

In [85]:

```
mat = confusion_matrix(y_test,y_pred)
mat
```

Out[85]:

```
array([[85,  5],
       [ 7, 28]])
```

In [94]:

```
from sklearn.metrics import mean_absolute_error,mean_squared_error,precision_score,recall_score,r2_score,accuracy_score
```

In [87]:

```
print('Accuracy: {:.2f}\n'.format(accuracy_score(y_test, y_pred)))
print('Mean Absolute Error: {:.2f}\n'.format(mean_absolute_error(y_test, y_pred)))
print('Mean Squared Error: {:.2f}\n'.format(mean_squared_error(y_test, y_pred)))
print('Precision: {:.2f}\n'.format(precision_score(y_test, y_pred)))
print('Recall: {:.2f}\n'.format(recall_score(y_test, y_pred)))
print('R2 Score: {:.2f}\n'.format(r2_score(y_test, y_pred)))
```

```
Accuracy: 0.90

Mean Absolute Error: 0.10

Mean Squared Error: 0.10

Precision: 0.85

Recall: 0.80

R2 Score: 0.52
```

### With tree pruning

In [88]:

```
model = DecisionTreeClassifier(criterion='gini', min_samples_split=10 , max_leaf_nodes = 5)
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
```

In [89]:

```
final = pd.DataFrame(y_test)
final['Predicted'] = y_pred
final
```

Out[89]:

| | Chance of Admit | Predicted |
|---|---|---|
| 361 | 1 | 1 |
| 73 | 1 | 0 |
| 374 | 0 | 0 |
| 155 | 0 | 0 |
| 104 | 0 | 1 |
| ... | ... | ... |
| 220 | 0 | 0 |
| 176 | 1 | 1 |
| 320 | 0 | 0 |
| 153 | 0 | 0 |
| 231 | 0 | 0 |

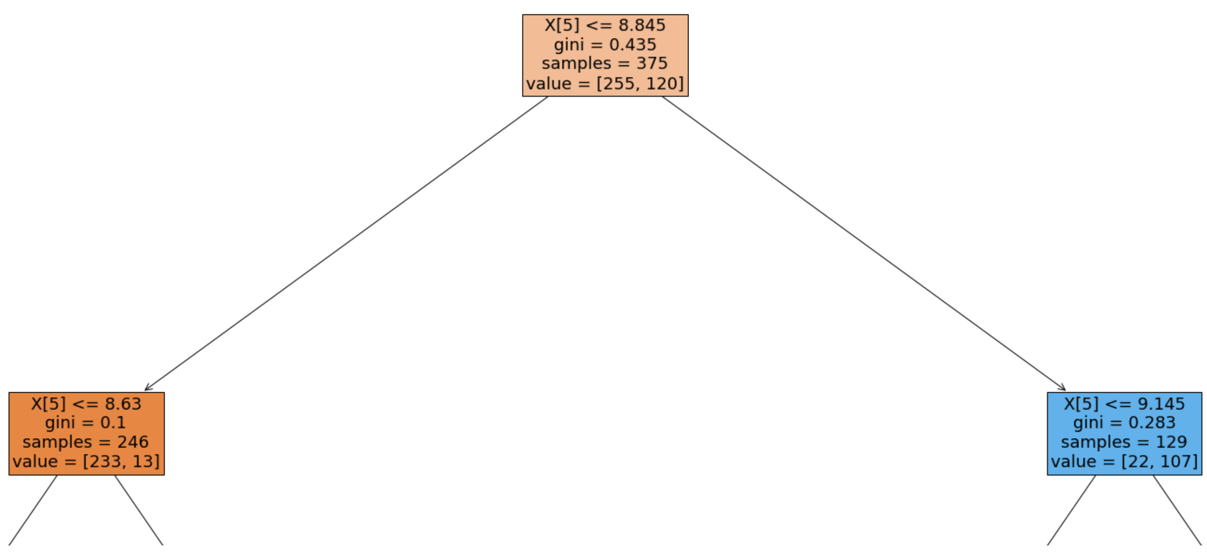
125 rows × 2 columns

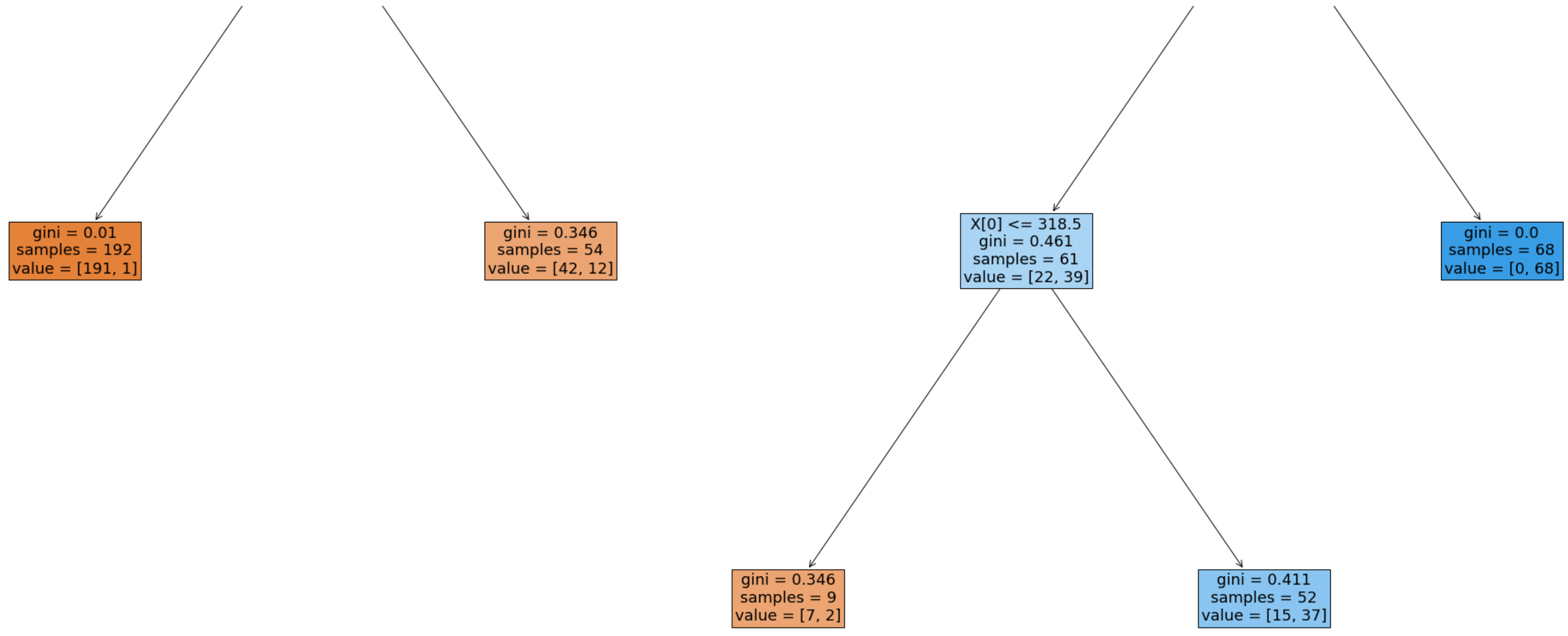
In [90]:

```
plt.figure(figsize=(40,30))
tree.plot_tree(model, filled=True, fontsize=18)
```

Out[90]:

```
[Text(1116.0, 1426.95, 'X[5] <= 8.845\ngini = 0.435\nsamples = 375\nvalue = [255, 120]'),
 Text(558.0, 1019.25, 'X[5] <= 8.63\ngini = 0.1\nsamples = 246\nvalue = [233, 13]'),
 Text(279.0, 611.55, 'gini = 0.01\nsamples = 192\nvalue = [191, 1]'),
 Text(837.0, 611.55, 'gini = 0.346\nsamples = 54\nvalue = [42, 12]'),
 Text(1674.0, 1019.25, 'X[5] <= 9.145\ngini = 0.283\nsamples = 129\nvalue = [22, 107]'),
 Text(1395.0, 611.55, 'X[0] <= 318.5\ngini = 0.461\nsamples = 61\nvalue = [22, 39]'),
 Text(1116.0, 203.8499999999999, 'gini = 0.346\nsamples = 9\nvalue = [7, 2]'),
 Text(1674.0, 203.8499999999999, 'gini = 0.411\nsamples = 52\nvalue = [15, 37]'),
 Text(1953.0, 611.55, 'gini = 0.0\nsamples = 68\nvalue = [0, 68]')]
```






### Confusion Matrix

In [91]:

```
mat = confusion_matrix(y_test,y_pred)
mat
```

Out[91]:

```
array([[87,  3],
       [ 2, 33]])
```

In [92]:

```
print('Accuracy: {:.2f}\n'.format(accuracy_score(y_test, y_pred)))
print('Mean Absolute Error: {:.2f}\n'.format(mean_absolute_error(y_test, y_pred)))
print('Mean Squared Error: {:.2f}\n'.format(mean_squared_error(y_test, y_pred)))
print('Precision: {:.2f}\n'.format(precision_score(y_test, y_pred)))
print('Recall: {:.2f}\n'.format(recall_score(y_test, y_pred)))
print('R2 Score: {:.2f}\n'.format(r2_score(y_test, y_pred)))
```

```
Accuracy: 0.96

Mean Absolute Error: 0.04

Mean Squared Error: 0.04

Precision: 0.92

Recall: 0.94

R2 Score: 0.80
```

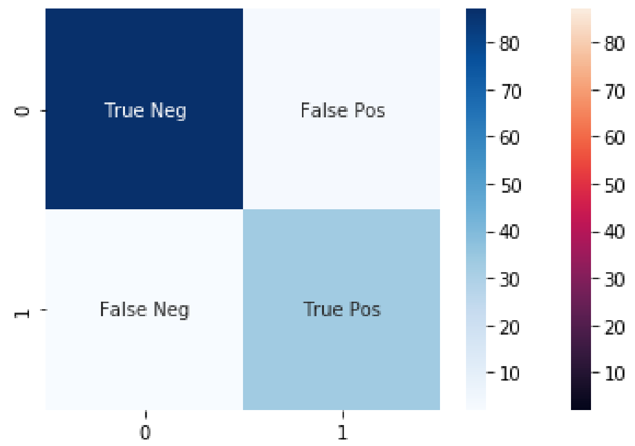
### HeatMap

In [93]:

```
sns.heatmap(mat)
labels = ['True Neg','False Pos','False Neg','True Pos']
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(mat, annot=labels, fmt='', cmap='Blues')
```

Out[93]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9553a53f90>
```




In [ ]:

**Assignment - 5**

**Machine Learning LP-1**

**Name: Prasad Sanjay Khalkar**
**Roll No: 33138**
**TE-09 L-09**

**Problem Statement:**
Download the following customer dataset from below link:
Data Set: https://www.kaggle.com/shwetabh123/mall-customers
This dataset gives the data of Income and money spent by the customers visiting a shopping mall.
The data set contains Customer ID, Gender, Age, Annual Income, Spending Score.
Therefore, as a mall owner you need to find the group of people who are the profitable customers for the mall owner. Apply at least two clustering algorithms (based on Spending Score) to find the group of customers.
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

---

**Theory:**

**<u>KMeans Clustering:</u>**

**Introduction to K-means Clustering**

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. [1]

The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity.

The results of the K-means clustering algorithm are:

1. The centroids of the K clusters, which can be used to label new data
2. Labels for the training data (each data point is assigned to a single cluster) Rather than defining groups before looking at the data, clustering allows you to find and analyze the groupsthat have formed organically.

The "Choosing K" section below describes how the number of groups can be determined. Each centroid of a cluster is a collection of feature values which define the resulting groups.

Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.

This introduction to the K-means clustering algorithm covers:

· Common business cases where K-means is used
· The steps involved in running the

algorithmSome examples of use cases are:

· **Behavioral segmentation:**
o Segment by purchase history
o Segment by activities on application, website, or platform.
o Define personas based on interests
o Create profiles based on activity monitoring
· **Inventory categorization:**
o Group inventory by sales activity
o Group inventory by manufacturing metrics
· **Sorting sensor measurements:**
o Detect activity types in motion sensors o Group images
o Separate audio o Identify groups in health monitoring
· **Detecting bots or anomalies:**
o Separate valid activity groups from bots
o Group valid activity to clean up outlier detection In addition, monitoring if a tracked data point switches between groups over time can be used to detect meaningful changesin the data.

**Algorithm**:

The K-means clustering algorithm uses iterative refinement to produce a final result. The algorithm inputs are the number of clusters K and the data set. The data set is a collection of features for each data point. The algorithms start with initial estimates for the K centroids, which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

1. **Data assignment step:**
   Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance. More formally, if ci is the collection of centroids in set C, then each data point x is assigned to a cluster based on

$$\underset{c_i \in C}{\operatorname{argmin}}\ dist(c_i, x)^2$$

   where dist( · ) is the standard (L2) Euclidean distance. Let the set of data point assignments for each i th cluster centroid be Si.

**2.Centroid update step:**

In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster.

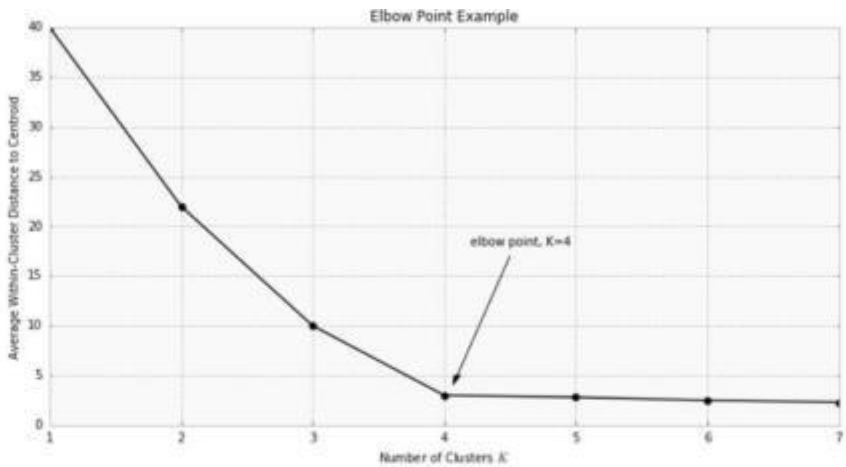
$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

The algorithm iterates between steps one and two until a stopping criteria is met (i.e., no data points change clusters, the sum of the distances is minimized, or some maximum number of iterations is reached). This algorithm is guaranteed to converge to a result. The result may be a local optimum (i.e. not necessarily the best possible outcome), meaning that assessing more than one run of the algorithm with randomized starting centroids may give a better outcome.

**Choosing K**

The algorithm described above finds the clusters and data set labels for a particular pre-chosen K. To find the number of clusters in the data, the user needs to run the K-means clustering algorithm for a range of K values and compare the results. In general, there is no method for determining exact value of K, but an accurate estimate can be obtained using the following techniques. One of the metrics that is commonly used to compare results across different values of K is the mean distance between data points and their cluster centroid. Since increasing the number of clusters will always reduce the distance to data points, increasing K will always decrease this metric, to the extreme of reaching zero when K is the same as the number of data points. Thus, this metric cannot be used as the sole target. Instead, mean distance to the centroid as a function of K is plotted and the "elbow point," where the rate of decrease sharply shifts, can be used to roughly determine K. A number of other techniques exist for validating K, including cross-validation, information criteria, the information theoretic jump method, the silhouette method, and the G-means algorithm.

In addition, monitoring the distribution of data points across groups provides insight into how the algorithm is splitting the data for each K.



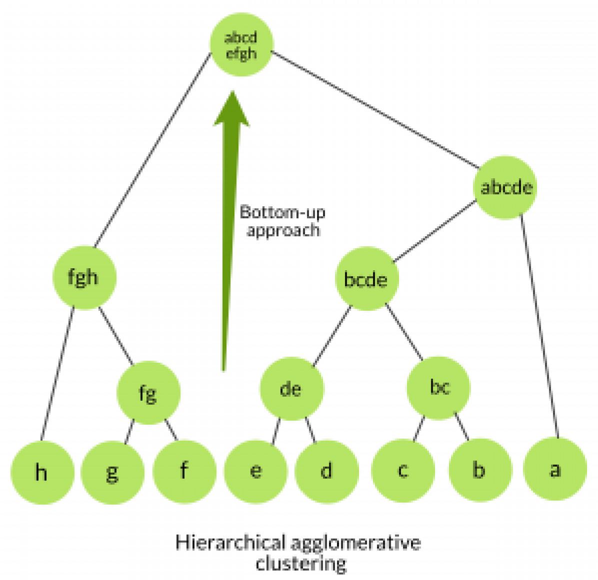

**Hierarchical Clustering:**

Hierarchical clustering analysis is a method of cluster analysis that seeks to build a hierarchy of clusters i.e., tree-type structure based on the hierarchy.

**Basically, there are two types of hierarchical cluster analysis strategies –**

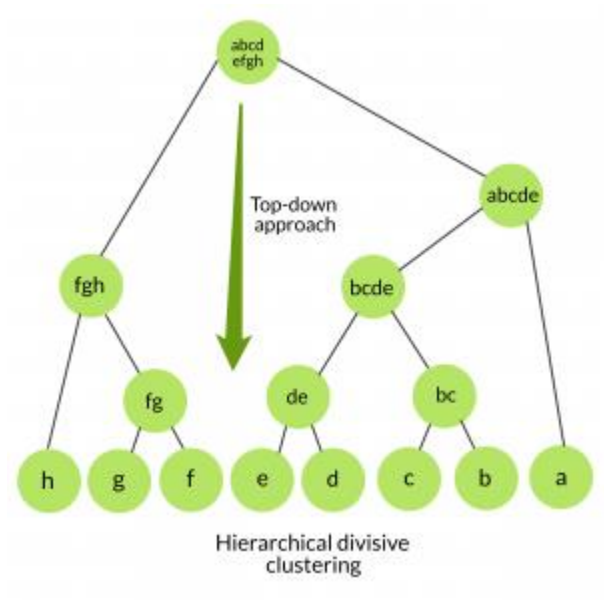
**1. Agglomerative Clustering:** Also known as bottom-up approach or hierarchical agglomerative clustering (HAC). A structure that is more informative than the unstructured set of clusters returned by flat clustering. This clustering algorithm does not require us to prespecify the number of clusters. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerates pairs of clusters until all clusters have been merged into a single cluster that contains all data.

**Algorithm :**
given a dataset ($d_1$, $d_2$, $d_3$, ....$d_N$) of size N
# compute the distance matrix
for i=1 to N:
  # as the distance matrix is symmetric about
  # the primary diagonal so we compute only lower
  # part of the primary diagonal
  for j=1 to i:
    dis_mat[i][j] = distance[$d_i$, $d_j$]
each data point is a singleton cluster
**repeat**
  merge the two cluster having minimum distance
  update the distance matrix
**until** only a single cluster remains




**2. Divisive clustering:** Also known as a top-down approach. This algorithm also does not require to prespecify the number of clusters. Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been split into singleton clusters.

**Algorithm:**
given a dataset ($d_1$, $d_2$, $d_3$, ....$d_N$) of size N
at the top we have all data in one cluster
the cluster is split using a flat clustering method eg. K-Means etc
**repeat**
choose the best cluster among all the clusters to split
split that cluster by the flat clustering algorithm
**until** each data is in its own singleton cluster




**Conclusion:**
Implemeted and studied K-means Clustering

**Implementation:**
**Implementation is as shown below:**

# Name: Prasad Sanjay Khalkar

# Roll No: 33138

# TE-09 L-09

In [49]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.cluster import KMeans
```

In [50]:

```
df = pd.read_csv("/home/prasadkhalkar/Desktop/ML/Datasets/Mall_Customers.csv")
mall = df.copy()
mall
```

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 |
| ... | ... | ... | ... | ... | ... |
| 195 | 196 | Female | 35 | 120 | 79 |
| 196 | 197 | Female | 45 | 126 | 28 |
| 197 | 198 | Male | 32 | 126 | 74 |
| 198 | 199 | Male | 32 | 137 | 18 |
| 199 | 200 | Male | 30 | 137 | 83 |

200 rows × 5 columns

In [51]:

```
mall.describe()
```

Out[51]:

| | 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 [52]:

```
mall.isnull().sum()
```

Out[52]:

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

In [53]:

```
mall.info()
```

```
<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

In [54]:

```
mall.shape
```

Out[54]:

```
(200, 5)
```

In [55]:

```
mall.head(5)
```

Out[55]:

| | 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 [56]:

```
mall.tail(5)
```

Out[56]:

| | CustomerID | Genre | Age | Annual Income (k$) | Spending Score (1-100) |
|---|---|---|---|---|---|
| **195** | 196 | Female | 35 | 120 | 79 |
| **196** | 197 | Female | 45 | 126 | 28 |
| **197** | 198 | Male | 32 | 126 | 74 |
| **198** | 199 | Male | 32 | 137 | 18 |
| **199** | 200 | Male | 30 | 137 | 83 |

In [57]:

```
encoder = preprocessing.LabelEncoder()
```

## Clustering for dataset1

In [58]:

```
temp = mall.drop(columns=['CustomerID','Age'],axis=1)
temp
```

Out[58]:

| | Genre | Annual Income (k$) | Spending Score (1-100) |
|---|---|---|---|
| **0** | Male | 15 | 39 |
| **1** | Male | 15 | 81 |
| **2** | Female | 16 | 6 |
| **3** | Female | 16 | 77 |
| **4** | Female | 17 | 40 |
| **...** | ... | ... | ... |
| **195** | Female | 120 | 79 |
| **196** | Female | 126 | 28 |
| **197** | Male | 126 | 74 |
| **198** | Male | 137 | 18 |
| **199** | Male | 137 | 83 |

200 rows × 3 columns

In [59]:

```
temp['Genre'] = encoder.fit_transform(temp['Genre'])
temp['Genre'].unique()
```

Out[59]:

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

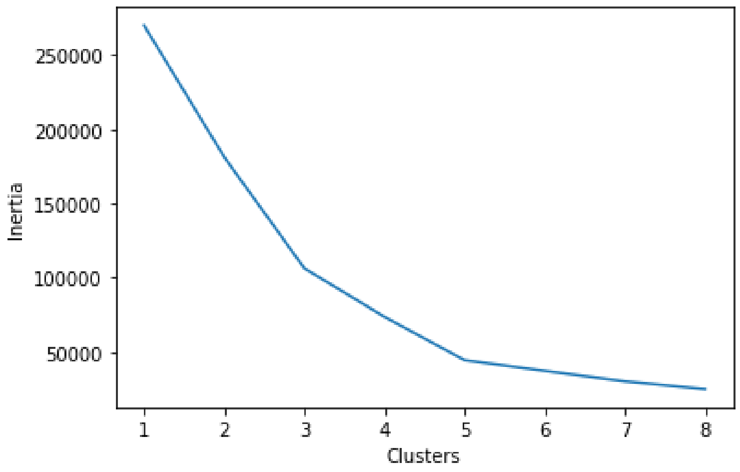
In [60]:

```
cluster=[]
for k in range(1,9):
    kmean = KMeans(n_clusters=k).fit(temp)
    cluster.append(kmean.inertia_)
```

In [61]:

```
plt.plot(range(1,9),cluster)
plt.xlabel('Clusters')
```

```
plt.ylabel('Inertia')
plt.xticks(range(1,9))
plt.show()
```




In [77]:

```
km = KMeans(n_clusters=5).fit(temp)
```

In [82]:

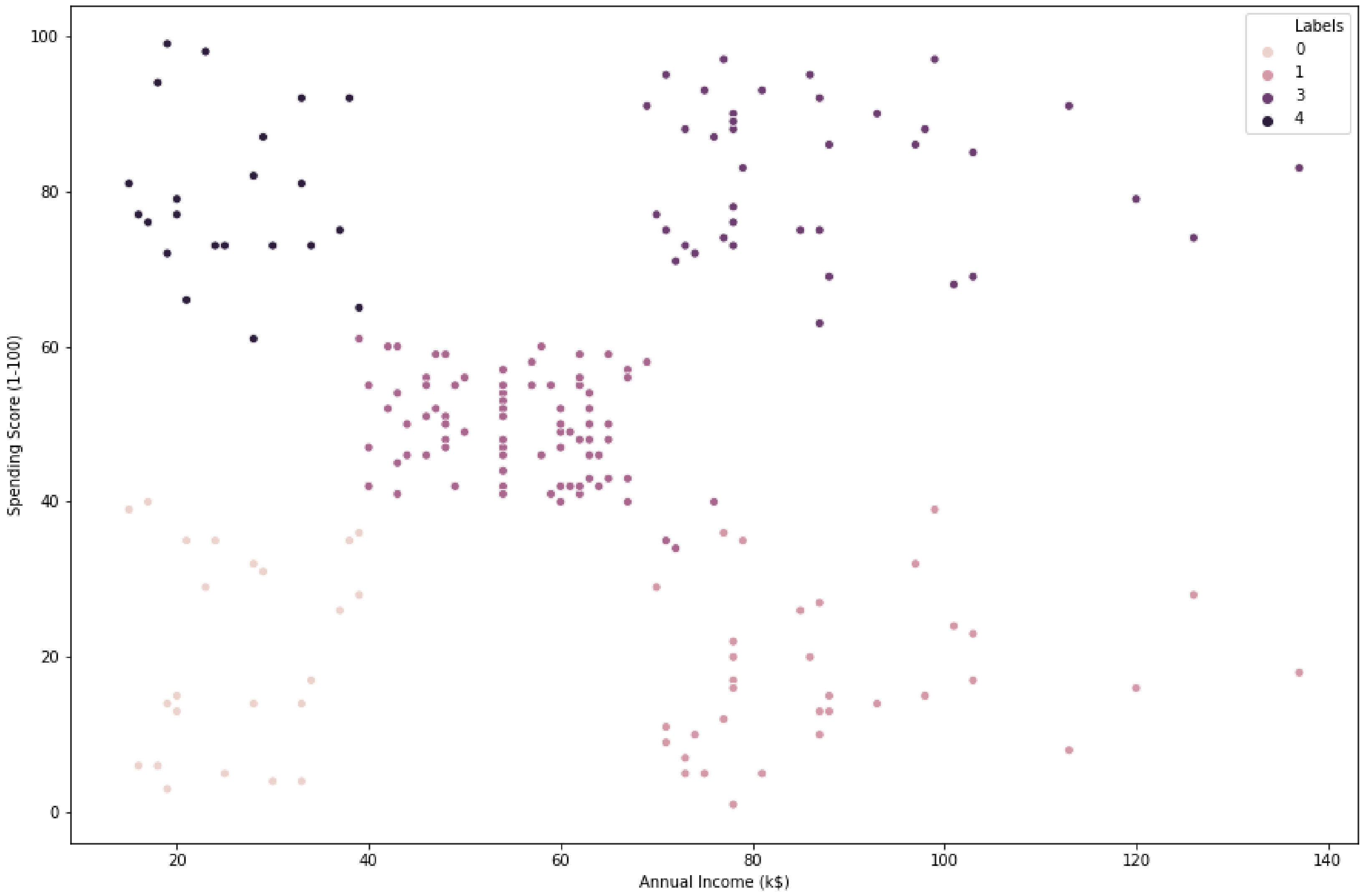
```
temp['Labels'] = km.labels_
temp.Labels.unique()
```

Out[82]:

```
array([0, 4, 2, 3, 1], dtype=int32)
```

In [83]:

```
plt.figure(figsize=(15,10))
sns.scatterplot(temp['Annual Income (k$)'],temp['Spending Score (1-100)'],hue=temp['Labels'])
plt.show()
```




In [84]:

```
km.cluster_centers_
```

Out[84]:

```
array([[ 3.91304348e-01,  2.63043478e+01,  2.09130435e+01,
         4.00000000e+00],
       [ 5.42857143e-01,  8.82000000e+01,  1.71142857e+01,
         1.00000000e+00],
       [ 4.07407407e-01,  5.52962963e+01,  4.95185185e+01,
        -8.88178420e-16],
       [ 4.61538462e-01,  8.65384615e+01,  8.21282051e+01,
         2.00000000e+00],
       [ 4.09090909e-01,  2.57272727e+01,  7.93636364e+01,
         3.00000000e+00]])
```

In [85]:

```
temp.Labels.value_counts()
```

Out[85]:

```
2    81
3    39
1    35
0    23
```

```
4    22
Name: Labels, dtype: int64
```

### Clustering for dataset 2

In [67]:

```
ds2 = mall.drop(columns=['CustomerID','Annual Income (k$)'],axis=1)
ds2
```

Out[67]:

| | Genre | Age | Spending Score (1-100) |
|---|---|---|---|
| **0** | Male | 19 | 39 |
| **1** | Male | 21 | 81 |
| **2** | Female | 20 | 6 |
| **3** | Female | 23 | 77 |
| **4** | Female | 31 | 40 |
| **...** | ... | ... | ... |
| **195** | Female | 35 | 79 |
| **196** | Female | 45 | 28 |
| **197** | Male | 32 | 74 |
| **198** | Male | 32 | 18 |
| **199** | Male | 30 | 83 |

200 rows × 3 columns

In [68]:

```
ds2['Genre'] = encoder.fit_transform(ds2['Genre'])
ds2['Genre'].unique()
```

Out[68]:

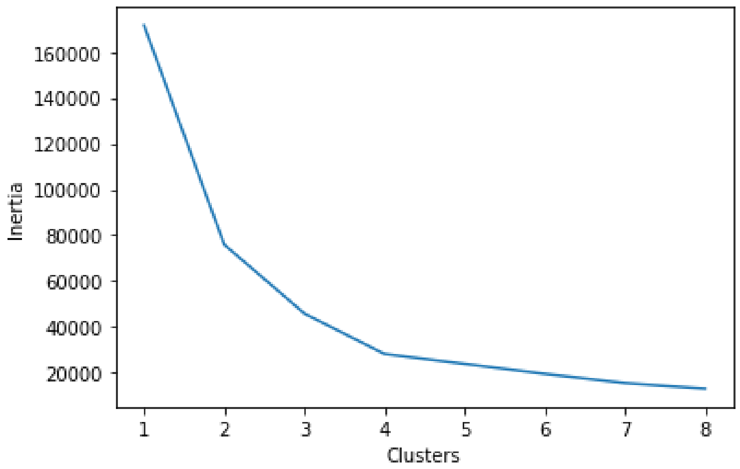
```
array([1, 0])
```

In [69]:

```
cluster=[]
for k in range(1,9):
    kmean = KMeans(n_clusters=k).fit(ds2)
    cluster.append(kmean.inertia_)
```

In [70]:

```
plt.plot(range(1,9),cluster)
plt.xlabel('Clusters')
plt.ylabel('Inertia')
plt.xticks(range(1,9))
plt.show()
```




In [71]:

```
km = KMeans(n_clusters=3).fit(ds2)
```

In [72]:

```
ds2['Labels'] = km.labels_
ds2
```

Out[72]:

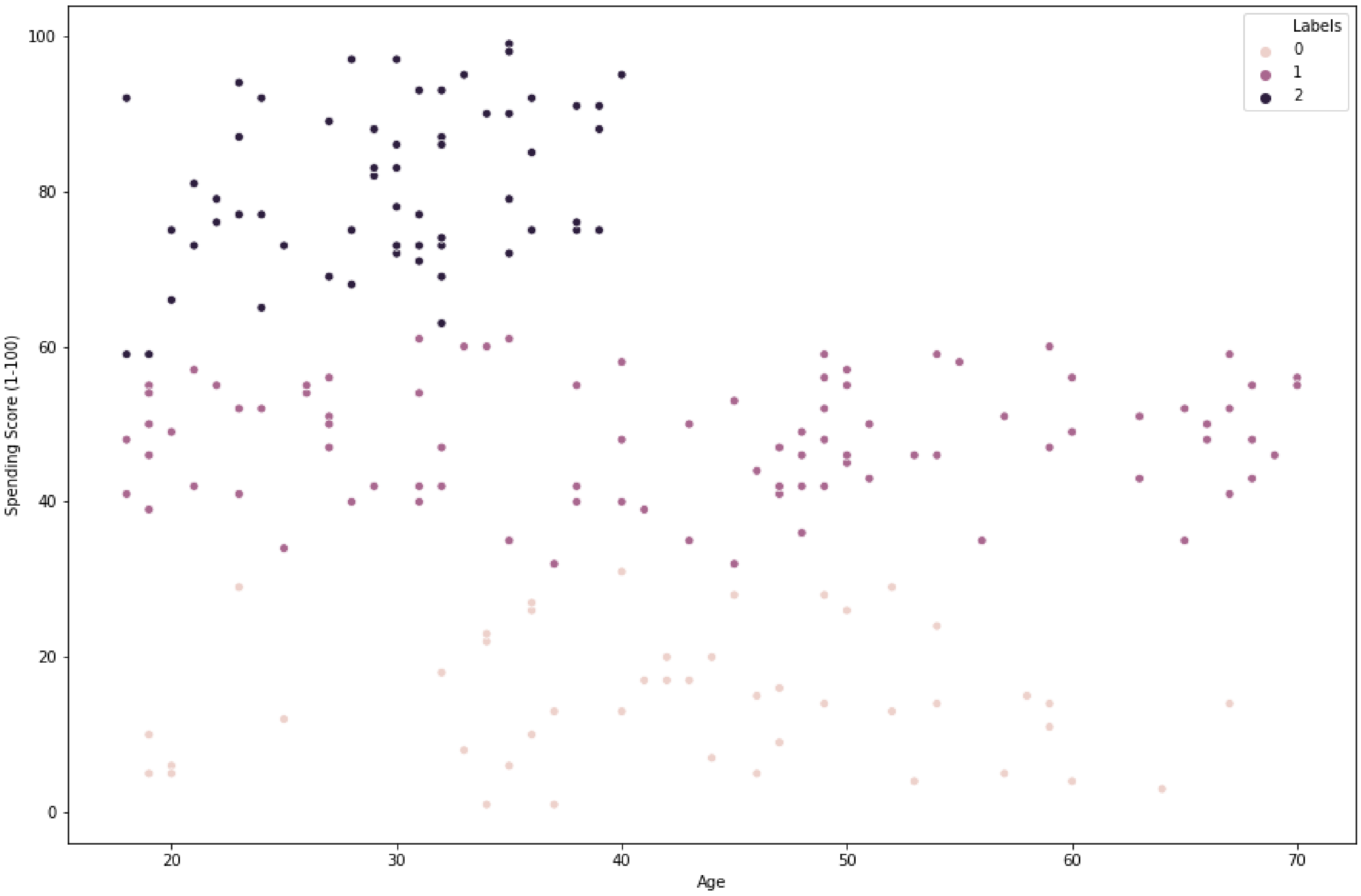
| | Genre | Age | Spending Score (1-100) | Labels |
|---|---|---|---|---|
| **0** | 1 | 19 | 39 | 1 |
| **1** | 1 | 21 | 81 | 2 |
| **2** | 0 | 20 | 6 | 0 |
| **3** | 0 | 23 | 77 | 2 |
| **4** | 0 | 31 | 40 | 1 |
| **...** | ... | ... | ... | ... |
| **195** | 0 | 35 | 79 | 2 |

| | Genre | Age | Spending Score (1-100) | Labels |
|---|---|---|---|---|
| **196** | 0 | 45 | 28 | 0 |
| **197** | 1 | 32 | 74 | 2 |
| **198** | 1 | 32 | 18 | 0 |
| **199** | 1 | 30 | 83 | 2 |

**200 rows × 4 columns**

In [73]:

```
plt.figure(figsize=(15,10))
sns.scatterplot(ds2['Age'],temp['Spending Score (1-100)'],hue=ds2['Labels'])
plt.show()
```




In [74]:

```
km.cluster_centers_
```

Out[74]:

```
array([[ 0.5106383 , 42.95744681, 14.59574468],
       [ 0.3956044 , 43.05494505, 47.78021978],
       [ 0.4516129 , 29.56451613, 80.74193548]])
```

In [75]:

```
ds2.Labels.value_counts()
```

Out[75]:

```
1    91
2    62
0    47
Name: Labels, dtype: int64
```

In [ ]: