Data Wrangling I

import the requried libraries

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

load the dataset

In [2]:
```
data = pd.read_csv('weatherAUS.csv')
```

In [3]:
```
#data.head displays top 5 records by default
data.head(10)
```

Out[3]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Hum
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	
5	2008-12-06	Albury	14.6	29.7	0.2	NaN	NaN	WNW	56.0	W	...	
6	2008-12-07	Albury	14.3	25.0	0.0	NaN	NaN	W	50.0	SW	...	
7	2008-12-08	Albury	7.7	26.7	0.0	NaN	NaN	W	35.0	SSE	...	
8	2008-12-09	Albury	9.7	31.9	0.0	NaN	NaN	NNW	80.0	SE	...	
9	2008-12-10	Albury	13.1	30.1	1.4	NaN	NaN	W	28.0	S	...	

10 rows × 24 columns

In [4]:
```
data.tail()                  #It displays bottom 5 records
```

Out[4]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...
145455	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	E	31.0	SE	...
145456	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NNW	22.0	SE	...
145457	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	N	37.0	SE	...
145458	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	SE	28.0	SSE	...
145459	2017-06-25	Uluru	14.9	NaN	0.0	NaN	NaN	NaN	NaN	ESE	...

5 rows × 24 columns

data preprocessing

In [5]:
```
data.info()                                  #dispaly information about dataet
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 24 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           145460 non-null  object
 1   Location       145460 non-null  object
 2   MinTemp        143975 non-null  float64
 3   MaxTemp        144199 non-null  float64
 4   Rainfall       142199 non-null  float64
 5   Evaporation    82670 non-null   float64
 6   Sunshine       75625 non-null   float64
 7   WindGustDir    135134 non-null  object
 8   WindGustSpeed  135197 non-null  float64
 9   WindDir9am     134894 non-null  object
 10  WindDir3pm     141232 non-null  object
 11  WindSpeed9am   143693 non-null  float64
 12  WindSpeed3pm   142398 non-null  float64
 13  Humidity9am    142806 non-null  float64
 14  Humidity3pm    140953 non-null  float64
 15  Pressure9am    130395 non-null  float64
 16  Pressure3pm    130432 non-null  float64
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
 20  Temp3pm        141851 non-null  float64
 21  RainToday      142199 non-null  object
 22  RISK_MM        142193 non-null  float64
 23  RainTomorrow   142193 non-null  object
dtypes: float64(17), object(7)
memory usage: 26.6+ MB
```

In [6]:
```
data.columns                       # gives the column names
```

Out[6]:
```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation',
       'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RISK_MM', 'RainTomorrow'],
      dtype='object')
```

Descriptive Statistics: It is used to summarize and describe the features of data in a meaningful way to extract insights.

In [7]:
```
print(data.describe(exclude=[object]))          # It gives descriptive statistics of numerical variable
```

```
             MinTemp        MaxTemp       Rainfall   Evaporation  \
count  143975.000000  144199.000000  142199.000000  82670.000000
mean       12.194034      23.221348       2.360918      5.468232
std         6.398495       7.119049       8.478060      4.193704
min        -8.500000      -4.800000       0.000000      0.000000
25%         7.600000      17.900000       0.000000      2.600000
50%        12.000000      22.600000       0.000000      4.800000
75%        16.900000      28.200000       0.800000      7.400000
max        33.900000      48.100000     371.000000    145.000000

           Sunshine  WindGustSpeed   WindSpeed9am   WindSpeed3pm  \
count  75625.000000  135197.000000  143693.000000  142398.000000
mean       7.611178      40.035230      14.043426      18.662657
std        3.785483      13.607062       8.915375       8.809800
min        0.000000       6.000000       0.000000       0.000000
25%        4.800000      31.000000       7.000000      13.000000
50%        8.400000      39.000000      13.000000      19.000000
75%       10.600000      48.000000      19.000000      24.000000
max       14.500000     135.000000     130.000000      87.000000

         Humidity9am    Humidity3pm   Pressure9am    Pressure3pm  \
count  142806.000000  140953.000000  130395.00000  130432.000000
mean       68.880831      51.539116    1017.64994    1015.255889
std        19.029164      20.795902       7.10653       7.037414
min         0.000000       0.000000     980.50000     977.100000
25%        57.000000      37.000000    1012.90000    1010.400000
50%        70.000000      52.000000    1017.60000    1015.200000
75%        83.000000      66.000000    1022.40000    1020.000000
max       100.000000     100.000000    1041.00000    1039.600000
```

```
          Cloud9am      Cloud3pm        Temp9am        Temp3pm         RISK_MM
Count  89572.000000  86102.000000  143693.000000  141851.00000  142193.000000
mean       4.447461      4.509930      16.990631      21.68339        2.360682
std        2.887159      2.720357       6.488753       6.93665        8.477969
min        0.000000      0.000000      -7.200000      -5.40000        0.000000
25%        1.000000      2.000000      12.300000      16.60000        0.000000
50%        5.000000      5.000000      16.700000      21.10000        0.000000
75%        7.000000      7.000000      21.600000      26.40000        0.800000
max        9.000000      9.000000      40.200000      46.70000      371.000000
```

In [8]:
```python
print(data.describe(include=[object]))
```

```
              Date  Location WindGustDir WindDir9am WindDir3pm RainToday  \
count       145460    145460      135134     134894     141232    142199
unique        3436        49          16         16         16         2
top     2013-11-12  Canberra           W          N         SE        No
freq            49      3436        9915      11758      10838    110319

       RainTomorrow
count        142193
unique            2
top              No
freq         110316
```

Finding Categorical and Numerical Features in a Data set:

Categorical features in Dataset:

In [9]:
```python
categorical_features = [column_name for column_name in data.columns if data[column_name].dtype == 'O']
print("Number of Categorical Features: {}".format(len(categorical_features)))
print("Categorical Features: ",categorical_features)
```

```
Number of Categorical Features: 7
Categorical Features: ['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday', 'RainTo
morrow']
```

In [10]:
```python
numerical_features = [column_name for column_name in data.columns if data[column_name].dtype != 'O']
print("Number of Numerical Features: {}".format(len(numerical_features)))
print("Numerical Features: ",numerical_features)
```

```
Number of Numerical Features: 17
Numerical Features: ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSp
eed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3p
m', 'Temp9am', 'Temp3pm', 'RISK_MM']
```

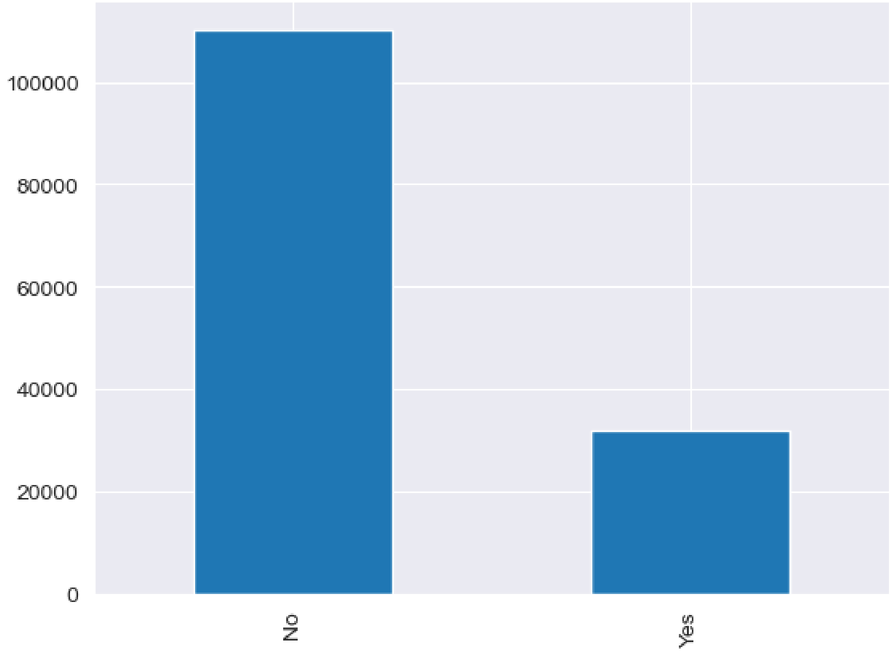
In [11]:
```python
data.isnull().sum()
```

Out[11]:
```
Date                 0
Location             0
MinTemp           1485
MaxTemp           1261
Rainfall          3261
Evaporation      62790
Sunshine         69835
WindGustDir      10326
WindGustSpeed    10263
WindDir9am       10566
WindDir3pm        4228
WindSpeed9am      1767
WindSpeed3pm      3062
Humidity9am       2654
Humidity3pm       4507
Pressure9am      15065
Pressure3pm      15028
Cloud9am         55888
Cloud3pm         59358
Temp9am           1767
Temp3pm           3609
RainToday         3261
RISK_MM           3267
RainTomorrow      3267
dtype: int64
```

In [12]:
```python
data.shape
```

Out[12]:
```
(145460, 24)
```

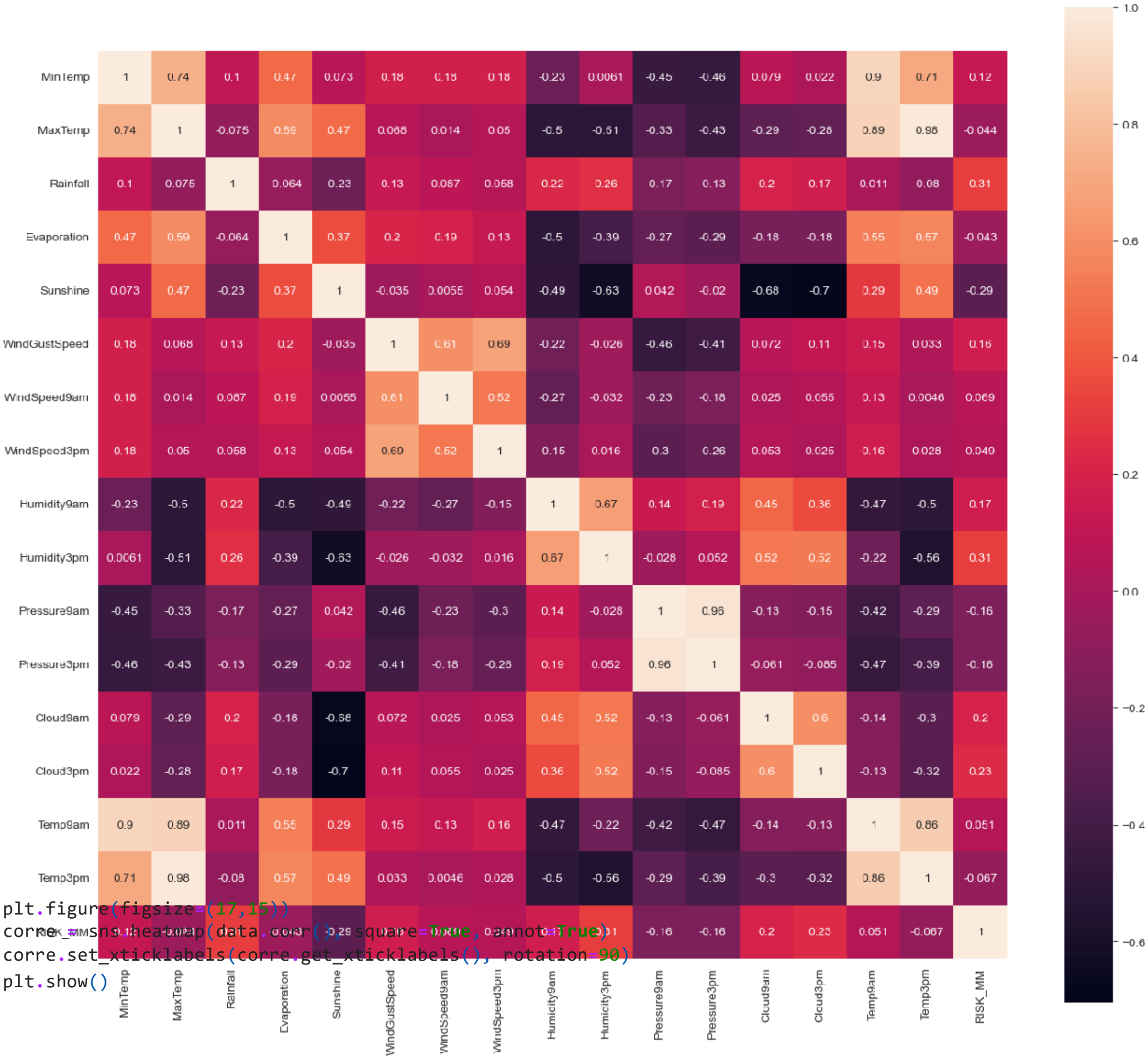
In [13]:
```python
data['RainTomorrow'].value_counts().plot.bar()
```
Out[13]: <Axes: >



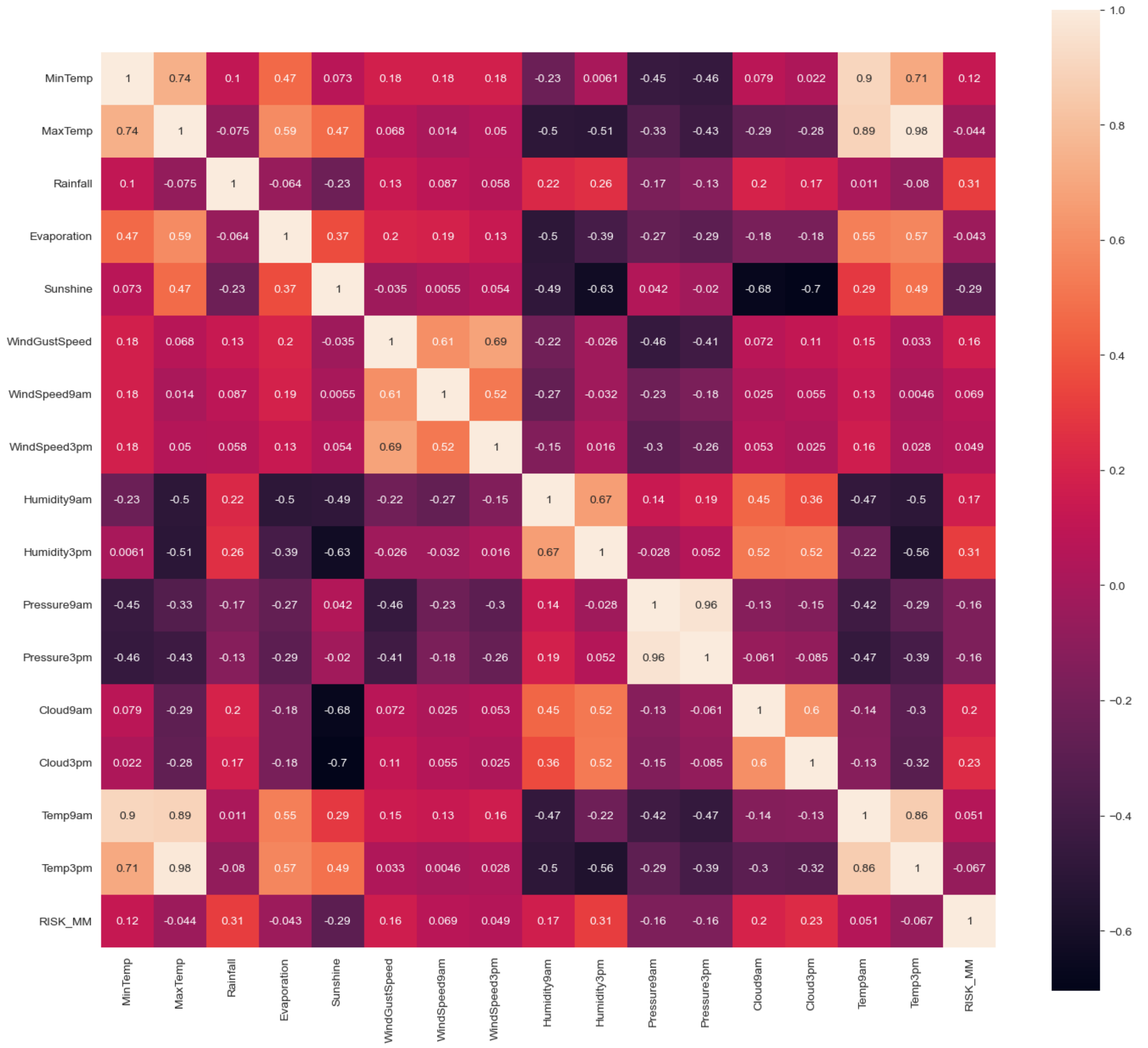
In [14]:
```python
sns.catplot(x = 'RainTomorrow', kind = 'count', data = data)
```
Out[14]: <seaborn.axisgrid.FacetGrid at 0x1cdb15d6b30>

In [15]:

Out[15]:



In [16]:
```python
plt.figure(figsize=(17,15))
corre = sns.heatmap(data.corr(), square=True, annot=True)
corre.set_xticklabels(corre.get_xticklabels(), rotation=90)
plt.show()
```



Handling Missing values in Categorical Features:

In [17]:
```
categorical_features = [column_name for column_name in data.columns if data[column_name].dtype == 'O']
data[categorical_features].isnull().sum()
```

Out[17]:
```
Date                0
Location            0
WindGustDir     10326
WindDir9am      10566
WindDir3pm       4228
RainToday        3261
RainTomorrow     3267
dtype: int64
```

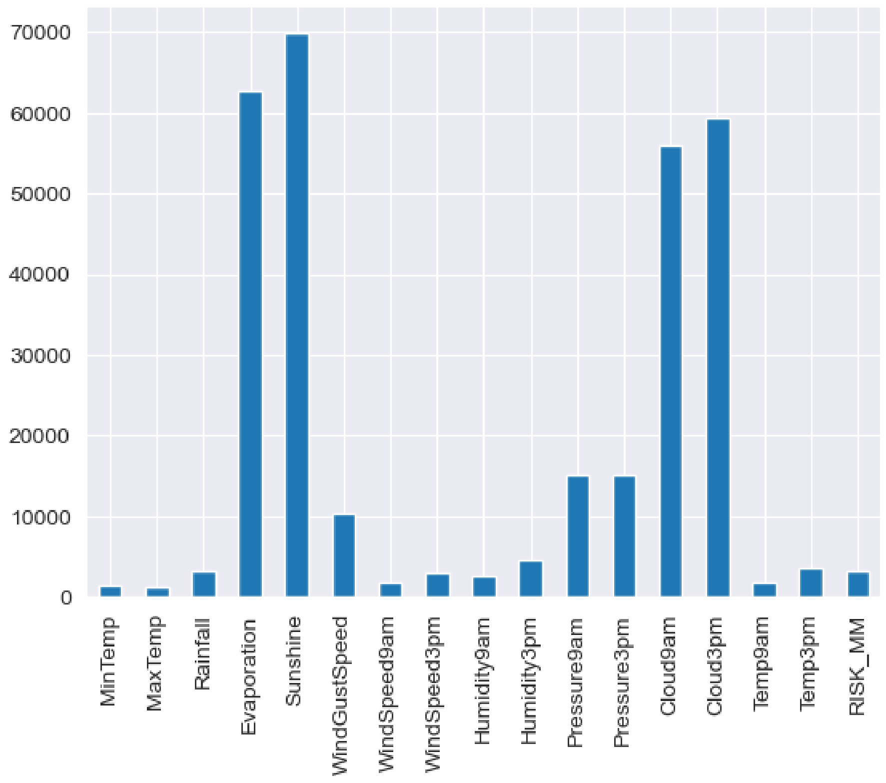
In [18]:
```
# Imputing the missing values in categorical features using the most frequent value which is mode:
categorical_features_with_null = [feature for feature in categorical_features if data[feature].isnull().sum
for each_feature in categorical_features_with_null:
    mode_val = data[each_feature].mode()[0]
    data[each_feature].fillna(mode_val,inplace=True)
```

In [19]:
```
# Handling Missing values in Numerical features:

numerical_features = [column_name for column_name in data.columns if data[column_name].dtype != 'O']
data[numerical_features].isnull().sum()
```

Out[19]:
```
MinTemp           1485
MaxTemp           1261
Rainfall          3261
Evaporation      62790
Sunshine         69835
WindGustSpeed    10263
WindSpeed9am      1767
WindSpeed3pm      3062
Humidity9am       2654
Humidity3pm       4507
Pressure9am      15065
Pressure3pm      15028
Cloud9am         55888
Cloud3pm         59358
Temp9am           1767
Temp3pm           3609
RISK_MM           3267
dtype: int64
```

In [20]:
```
data[numerical_features].isnull().sum().plot.bar()
```

Out[20]: <Axes: >



In [21]:
```
# Outlier Treatment to remove outliers from Numerical Features:
features_with_outliers = ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'WindGustSpeed','WindSpeed9am',
for feature in features_with_outliers:
    q1 = data[feature].quantile(0.25)
    q3 = data[feature].quantile(0.75)
    IQR = q3-q1
    lower_limit = q1 - (IQR*1.5)
    upper_limit = q3 + (IQR*1.5)
    data.loc[data[feature]<lower_limit,feature] = lower_limit
    data.loc[data[feature]>upper_limit,feature] = upper_limit
```

Now, numerical features are free from outliers. Let’s Impute missing values in numerical features using mean

In [22]:
```
numerical_features_with_null = [feature for feature in numerical_features if data[feature].isnull().sum()]
for feature in numerical_features_with_null:
    mean_value = data[feature].mean()
    data[feature].fillna(mean_value,inplace=True)
```

In [23]:
```
data.isnull().sum()
```

Out[23]:
```
Date             0
Location         0
MinTemp          0
MaxTemp          0
Rainfall         0
Evaporation      0
Sunshine         0
WindGustDir      0
WindGustSpeed    0
WindDir9am       0
WindDir3pm       0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Cloud9am         0
Cloud3pm         0
Temp9am          0
Temp3pm          0
RainToday        0
RISK_MM          0
RainTomorrow     0
dtype: int64
```

Encoding

In [24]:
```
def encode_data(feature_name):

    '''

    This function takes feature name as a parameter and returns mapping dictionary to replace(or map) categ

    '''

    mapping_dict = {}

    unique_values = list(data[feature_name].unique())

    for idx in range(len(unique_values)):

        mapping_dict[unique_values[idx]] = idx

    return mapping_dict
```

In [25]:
```
data['RainToday'].replace({'No':0, 'Yes': 1}, inplace = True)

data['RainTomorrow'].replace({'No':0, 'Yes': 1}, inplace = True)

data['WindGustDir'].replace(encode_data('WindGustDir'),inplace = True)

data['WindDir9am'].replace(encode_data('WindDir9am'),inplace = True)

data['WindDir3pm'].replace(encode_data('WindDir3pm'),inplace = True)

data['Location'].replace(encode_data('Location'), inplace = True)
```

In [26]:
```
encode_data('RainToday')
data['RainTomorrow']
data['WindGustDir']
data['WindDir9am']
data['WindDir3pm']
data['Location']
```

Out[26]:
```
0          0
1          0
2          0
3          0
4          0
          ..
145455    48
145456    48
145457    48
145458    48
145459    48
Name: Location, Length: 145460, dtype: int64
```

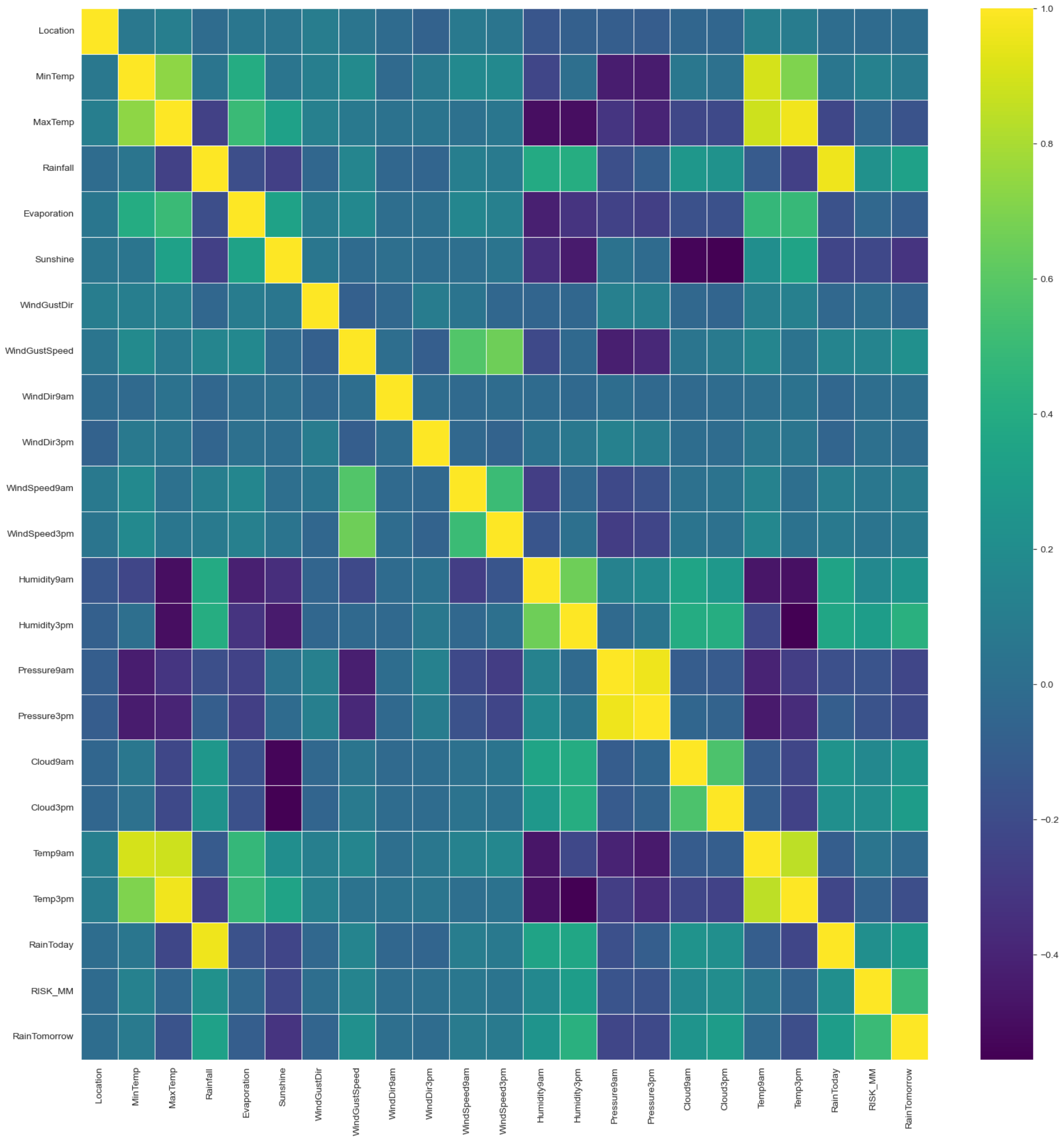
In [27]:

```python
data['RainToday'].replace({'No':0, 'Yes': 1}, inplace = True)

data['RainTomorrow'].replace({'No':0, 'Yes': 1}, inplace = True)

data['WindGustDir'].replace(encode_data('WindGustDir'),inplace = True)

data['WindDir9am'].replace(encode_data('WindDir9am'),inplace = True)

data['WindDir3pm'].replace(encode_data('WindDir3pm'),inplace = True)

data['Location'].replace(encode_data('Location'), inplace = True)
```

In [28]:

```python
##Correlation Analysis
plt.figure(figsize=(20,20))
sns.heatmap(data.corr(), linewidths=0.5, annot=False, fmt=".2f", cmap = 'viridis')
```

Out[28]: <Axes: >



Data Wrangling II

import the requried libraries

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

In [ ]:
```
data = pd.read_csv('/content/xAPI-Edu-Data.csv')
data.head()
```

Out[ ]:

	gender	NationalITy	PlaceofBirth	StageID	GradeID	SectionID	Topic	Semester	Relation	raisedhands	VisITedResources
0	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	Father	15	16
1	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	Father	20	20
2	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	Father	10	7
3	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	Father	30	25
4	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	Father	40	50

In [ ]:
```
# Let us check the columns in the dataset
data.columns
```

Out[ ]:
```
Index(['gender', 'NationalITy', 'PlaceofBirth', 'StageID', 'GradeID',
       'SectionID', 'Topic', 'Semester', 'Relation', 'raisedhands',
       'VisITedResources', 'AnnouncementsView', 'Discussion',
       'ParentAnsweringSurvey', 'ParentschoolSatisfaction',
       'StudentAbsenceDays', 'Class'],
      dtype='object')
```

In [ ]:
```
# For outliers we can identify the continuous / numerical variables
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 480 entries, 0 to 479
Data columns (total 17 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   gender                    480 non-null    object
 1   NationalITy               480 non-null    object
 2   PlaceofBirth              480 non-null    object
 3   StageID                   480 non-null    object
 4   GradeID                   480 non-null    object
 5   SectionID                 480 non-null    object
 6   Topic                     480 non-null    object
 7   Semester                  480 non-null    object
 8   Relation                  480 non-null    object
 9   raisedhands               480 non-null    int64
 10  VisITedResources          480 non-null    int64
 11  AnnouncementsView         480 non-null    int64
 12  Discussion                480 non-null    int64
 13  ParentAnsweringSurvey     480 non-null    object
 14  ParentschoolSatisfaction  480 non-null    object
 15  StudentAbsenceDays        480 non-null    object
 16  Class                     480 non-null    object
dtypes: int64(4), object(13)
memory usage: 63.9+ KB
```

In [ ]:
```
# Int type columns are - raisedhands, VisITedResources, AnnouncementsView
data[['raisedhands', 'VisITedResources', 'AnnouncementsView']]
```

Out[ ]:

	raisedhands	VisITedResources	AnnouncementsView
0	15	16	2
1	20	20	3
2	10	7	0
3	30	25	5
4	40	50	12
...	...	...	...
475	5	4	5
476	50	77	14
477	55	74	25
478	30	17	14
479	35	14	23

480 rows × 3 columns

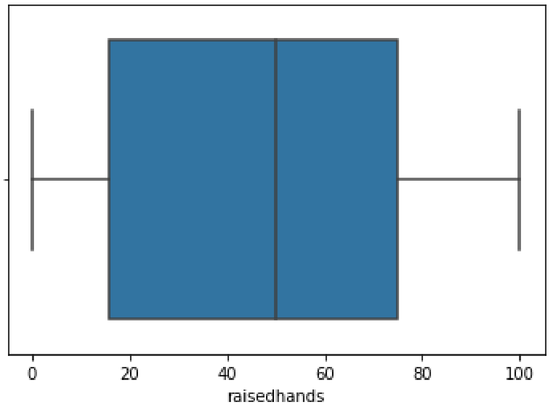
In [ ]:

```
# Identify (Detect the outliers) with the help of BoxPlot
import seaborn as sns
```

In [ ]:

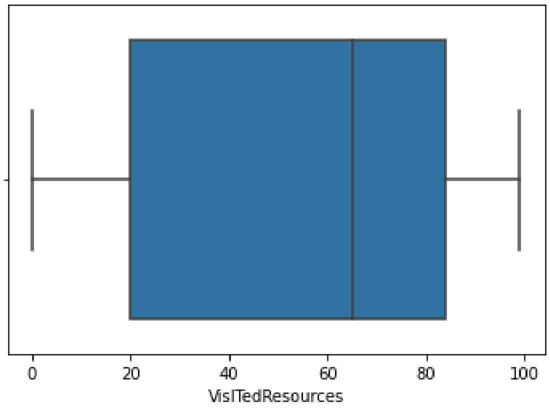
```
sns.boxplot(data['raisedhands'])
```

Out[ ]: <AxesSubplot:xlabel='raisedhands'>



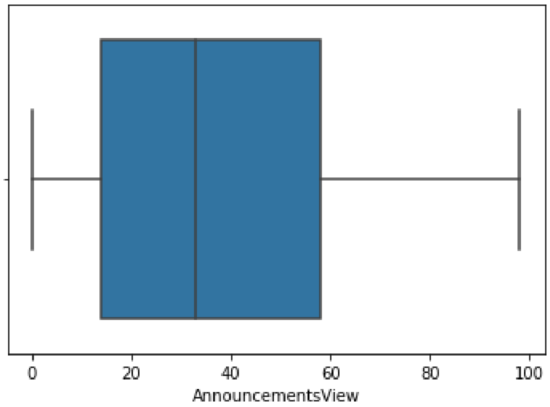

```
sns.boxplot(data['VisITedResources'])
```

<AxesSubplot:xlabel='VisITedResources'>

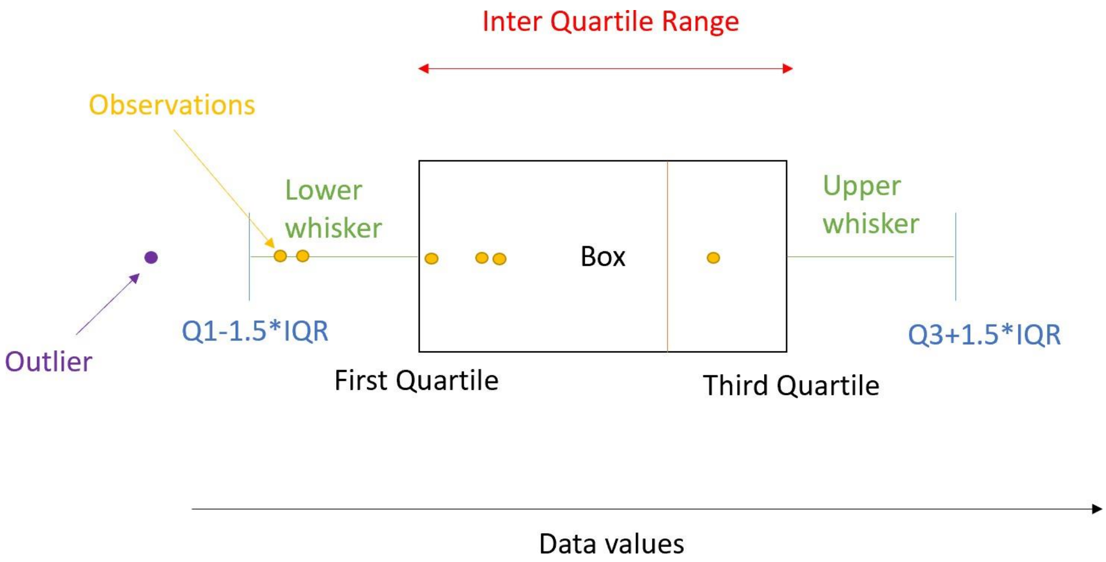



```
sns.boxplot(data['AnnouncementsView'])
```

Out[ ]: <AxesSubplot:xlabel='AnnouncementsView'>




```
# in all the three diagrams above there is not outlier
```



Descriptive Statistics - Measures of Central Tendancy and variability

1) Provide summary statistics(mean, median, minimum, maximum, standard deviation) for a dataset with numeric variables grouped by one of the qualitative variable.

Data Description:

Link of the dataset:

https://www.kaggle.com/datasets/aungpyaeap/supermarket-sales

About dataset:

The growth of supermarkets in most populated cities are increasing and market competitions are also high. The dataset is one of the historical sales of supermarket company which has recorded in 3 different branches for 3 months data.

Attribute information:

- Invoice id: Computer generated sales slip invoice identification number
- Branch: Branch of supercenter (3 branches are available identified by A, B and C).
- City: Location of supercenters
- Customer type: Type of customers, recorded by Members for customers using member card and Normal for without member card.
- Gender: Gender type of customer
- Product line: General item categorization groups - Electronic accessories, Fashion accessories, Food and beverages, Health and beauty, Home and lifestyle, Sports and travel
- Unit price: Price of each product in $
- Quantity: Number of products purchased by customer
- Tax: 5% tax fee for customer buying
- Total: Total price including tax
- Date: Date of purchase (Record available from January 2019 to March 2019)
- Time: Purchase time (10am to 9pm)
- Payment: Payment used by customer for purchase (3 methods are available – Cash, Credit card and Ewallet)
- COGS: Cost of goods sold
- Gross margin percentage: Gross margin percentage
- Gross income: Gross income
- Rating: Customer stratification rating on their overall shopping experience (On a scale of 1 to 10)

In [5]:
```
# import the required librabries

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

In [6]:
```
# Read the datasets

sales = pd.read_csv('supermarket_sales.csv')
```

In [7]:
```
sales.head()
```

Out[7]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Dat
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	1/5/201
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	3/8/201
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/201
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1/27/201
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2/8/201

In [8]:
```
sales.tail()
```

Out[8]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	2.0175	42.3675	1/29/
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	48.6900	1022.4900	3/2/
997	727-02-1313	A	Yangon	Member	Male	Food and beverages	31.84	1	1.5920	33.4320	2/9/
998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	65.82	1	3.2910	69.1110	2/22/
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7	30.9190	649.2990	2/18/

In [9]:
```
sales.shape
```

Out[9]:
```
(1000, 17)
```

In [10]:
```
# Check the missing value

sales.isnull().sum()
```

Out[10]:
```
Invoice ID                 0
Branch                     0
City                       0
Customer type              0
Gender                     0
Product line               0
Unit price                 0
Quantity                   0
Tax 5%                     0
Total                      0
Date                       0
Time                       0
Payment                    0
cogs                       0
gross margin percentage    0
gross income               0
Rating                     0
dtype: int64
```

In [11]:
```python
# Total number of non-missing calues -- using count()

sales.count()
```

Out[11]:
```
Invoice ID                 1000
Branch                     1000
City                       1000
Customer type              1000
Gender                     1000
Product line               1000
Unit price                 1000
Quantity                   1000
Tax 5%                     1000
Total                      1000
Date                       1000
Time                       1000
Payment                    1000
cogs                       1000
gross margin percentage    1000
gross income               1000
Rating                     1000
dtype: int64
```

In [12]:
```python
# What is the total sales amount -- using sum()

sales.sum()
```

Out[12]:
```
Invoice ID                 750-67-8428226-31-3081631-41-3108123-19-
Branch                     117637...
City                       ACAAACACABBBAAABAAABCBBAAABABABBBACCAACBBCBCCB..
Customer type              .
Gender                     YangonNaypyitawYangonYangonYangonNaypyitawYang..
Product line               .
Unit price                 MemberNormalNormalMemberNormalNormalMemberNorm..
Quantity                   .
Tax 5%                     FemaleFemaleMaleMaleMaleMaleFemaleFemaleFemale..
Total                      .
Date                       Health and beautyElectronic accessoriesHome an...
Time                                                               55672.13
Payment                                                                5510
cogs                                                              15379.369
gross margin percentage                                          322966.749
gross income               1/5/20193/8/20193/3/20191/27/20192/8/20193/25/...
Rating                     13:0810:2913:2320:3310:3718:3014:3611:3817:151...
dtype: object              EwalletCashCredit cardEwalletEwalletEwalletEwa...
                                                                  307587.38
```

In [13]:
```python
sales['Total'].sum()
```

Out[13]:
```
322966.749                                                           6972.7
```

In [14]:
```
sales[['Total','gross income']].sum()
```

Out[14]:
```
Total           322966.749
gross income     15379.369
dtype: float64
```

In [15]:
```
# What is the average satisfaction level and what is the spread of it -- using mean and
st

sales.mean()
```

Out[15]:
```
Unit price                  55.672130
Quantity                     5.510000
Tax 5%                      15.379369
Total                      322.966749
cogs                       307.587380
gross margin percentage      4.761905
gross income                15.379369
Rating                       6.972700
dtype: float64
```

```
sales['Rating'].mean()
```

```
6.972700000000003
```

In [16]:
```
sales.std()
```

```
Unit price                 2.649463e+01
Quantity                   2.923431e+00
Tax 5%                     1.170883e+01
Total                      2.458853e+02
cogs                       2.341765e+02
gross margin percentage    6.131498e-14
gross income               1.170883e+01
Rating                     1.718580e+00
dtype: float64
```

```
sales['Rating'].std()
```

```
1.718580294379123
```

```
# Most frequently ussed payment method -- using mode()

print('The Mode of : ',sales['Payment'].mode)
```

In [18]:
```
The Mode of :  <bound method Series.mode of 0          Ewallet
1             Cash
2      Credit card
3          Ewallet
4          Ewallet
          ...
995        Ewallet
996        Ewallet
997           Cash
998           Cash
999           Cash
Name: Payment, Length: 1000, dtype: object>
```

In [20]:
```python
sales['Payment'].mode()  #Ewallet is the payment method which has been used most frequentl
```
Out[20]:
```
0    Ewallet
Name: Payment, dtype: object
```

In [21]:
```python
# Minimum and Maximum Quantity sold  -- using min() and max()

sales['Quantity'].min()           # 1 is the minimum quantity which has been sold
```
Out[21]:
```
1
```

In [22]:
```python
sales['Quantity'].max()
```
Out[22]:
```
10
```

In [23]:
```python
# Daily increasing sales(how the sales increasing in daily bases)  -- using cumsum()

sales['Total'].cumsum().head()      # total amount sold like 548.9715,...
```
Out[23]:
```
0      548.9715
1      629.1915
2      969.7170
3     1458.7650
4     2093.1435
Name: Total, dtype: float64
```

In [24]:
```python
sales['Total'].head()
```
Out[24]:
```
0    548.9715
1     80.2200
2    340.5255
3    489.0480
4    634.3785
Name: Total, dtype: float64
```

In [25]:
```python
# Median

median =
sales['Quantity'].median() median
```
Out[25]:
```
5.0
```

In [26]:
```python
print('median quantity: ' + str(median))
```
```
median quantity: 5.0
```

In [27]:
```python
# Var

var = sales['Quantity'].var()
var
```
Out[27]:
```
8.546446446446451
```

In [28]:
```python
print('var of quantity: ' + str(var))
```
```
var of quantity: 8.546446446446451
```

In [29]:
```python
# group by

groupby_sum =
sales.groupby(['City']).sum() groupby_sum
```

Out[29]:

City	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rating
Mandalay	18478.88	1820	5057.0320	106197.6720	101140.64	1580.952381	5057.0320	2263.6
Naypyitaw	18567.76	1831	5265.1765	110568.7065	105303.53	1561.904762	5265.1765	2319.9
Yangon	18625.49	1859	5057.1605	106200.3705	101143.21	1619.047619	5057.1605	2389.2

In [30]:

```
groupby_count =
sales.groupby(['City']).count() groupby_count
```

Out[30]:

City	Invoice ID	Branch	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Paymen
Mandalay	332	332	332	332	332	332	332	332	332	332	332	33
Naypyitaw	328	328	328	328	328	328	328	328	328	328	328	32
Yangon	340	340	340	340	340	340	340	340	340	340	340	34

In [31]:

```
groupby_sum1 =
sales.groupby(['Payment']).sum() groupby_sum1
```

Out[31]:

Payment	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rating
Cash	19525.09	1896	5343.170	112206.570	106863.40	1638.095238	5343.170	2397.7
Credit card	16916.68	1722	4798.432	100767.072	95968.64	1480.952381	4798.432	2178.0
Ewallet	19230.36	1892	5237.767	109993.107	104755.34	1642.857143	5237.767	2397.0

In [32]:

```
groupby_count1 =
sales.groupby(['Payment']).count() groupby_count1
```

Out[32]:

Payment	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	cog
Cash	344	344	344	344	344	344	344	344	344	344	344	344	34
Credit card	311	311	311	311	311	311	311	311	311	311	311	311	31
Ewallet	345	345	345	345	345	345	345	345	345	345	345	345	34

In [33]:

```
print('sum of values, grouped by the payment: ' + str(groupby_sum1))
```

```
sum of values, grouped by the payment:              Unit price  Quantity    Tax 5%       T
otal       cogs  \
Payment
Cash            19525.09      1896  5343.170  112206.570  106863.40
Credit card     16916.68      1722  4798.432  100767.072   95968.64
Ewallet         19230.36      1892  5237.767  109993.107  104755.34

             gross margin percentage  gross income  Rating
Payment
Cash                     1638.095238      5343.170  2397.7
Credit card              1480.952381      4798.432  2178.0
Ewallet                  1642.857143      5237.767  2397.0
```

In [34]:

```
print('count of values, grouped by the payment: ' + str(groupby_count1))
```

```
count of values, grouped by the payment:              Invoice ID  Branch  City  Customer t
ype  Gender  Product line  \
Payment
Cash                344     344   344            344     344            344
Credit card         311     311   311            311     311            311
Ewallet             345     345   345            345     345            345

             Unit price  Quantity  Tax 5%  Total  Date  Time  cogs  \
Payment
Cash                344       344     344    344   344   344   344
Credit card         311       311     311    311   311   311   311
Ewallet             345       345     345    345   345   345   345

             gross margin percentage  gross income  Rating
Payment
Cash                             344           344     344
Credit card                      311           311     311
Ewallet                          345           345     345
```

In [35]:

```
# describe

sales.describe()  # it does not provide categorical attribute summay, bydefault it
provide
```

Out[35]:

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rat
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1.000000e+03	1000.000000	1000.00
mean	55.672130	5.510000	15.379369	322.966749	307.58738	4.761905e+00	15.379369	6.97
std	26.494628	2.923431	11.708825	245.885335	234.17651	6.131498e-14	11.708825	1.71
min	10.080000	1.000000	0.508500	10.678500	10.17000	4.761905e+00	0.508500	4.00
25%	32.875000	3.000000	5.924875	124.422375	118.49750	4.761905e+00	5.924875	5.50
50%	55.230000	5.000000	12.088000	253.848000	241.76000	4.761905e+00	12.088000	7.00
75%	77.935000	8.000000	22.445250	471.350250	448.90500	4.761905e+00	22.445250	8.50
max	99.960000	10.000000	49.650000	1042.650000	993.00000	4.761905e+00	49.650000	10.00

In [36]:

```
sales.describe(include='object')
```

Out[36]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Date	Time	Payment
count	1000	1000	1000	1000	1000	1000	1000	1000	1000
unique	1000	3	3	2	2	6	89	506	3
top	750-67-8428	A	Yangon	Member	Female	Fashion accessories	2/7/2019	19:48	Ewallet
freq	1	340	340	501	501	178	20	7	345

In [37]:
```
sales.describe(include='all')
```

Out[37]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	
count	1000	1000	1000	1000	1000	1000 1	1000.000000	1000.000000	1000.000000	
unique	1000	3	3	2	2	6	NaN	NaN	NaN	
top	750-67-8428	A	Yangon	Member	Female	Fashion accessories	NaN	NaN	NaN	
freq	1	340	340	501	501	178	NaN	NaN	NaN	
mean	NaN	NaN	NaN	NaN	NaN	NaN	55.672130	5.510000	15.379369	
std	NaN	NaN	NaN	NaN	NaN	NaN	26.494628	2.923431	11.708825	
min	NaN	NaN	NaN	NaN	NaN	NaN	10.080000	1.000000	0.508500	
25%	NaN	NaN	NaN	NaN	NaN	NaN	32.875000	3.000000	5.924875	
50%	NaN	NaN	NaN	NaN	NaN	NaN	55.230000	5.000000	12.088000	
75%	NaN	NaN	NaN	NaN	NaN	NaN	77.935000	8.000000	22.445250	
max	NaN	NaN	NaN	NaN	NaN	NaN	99.960000	10.000000	49.650000	1

2) Write a Python program to display some basic statistical details like percentile, mean, standard deviation etc. of the species of 'iris-setosa','iris-versicolor' of iris.csv dataset.

In [38]:
```
# Load the libraries

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

In [39]:
```
# Read the data
iris = pd.read_csv('Iris.csv')
```

In [40]:
```
iris.head()
```

Out[40]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [41]:

```
iris.tail()
```

Out[41]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

In [42]:

```
# How many data-points and features?
iris.shape
```

Out[42]:

```
(150, 6)
```

In [43]:

```
# What are the column names in our dataset?
print(iris.columns)
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm',
       'PetalWidthCm', 'Species'],
      dtype='object')
```

In [44]:

```
iris.isnull().sum()
```

Out[44]:

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [45]:

```
iris.describe()
```

Out[45]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [46]:

```
# How many data points for each class are present or flowers for each species are
present?
```

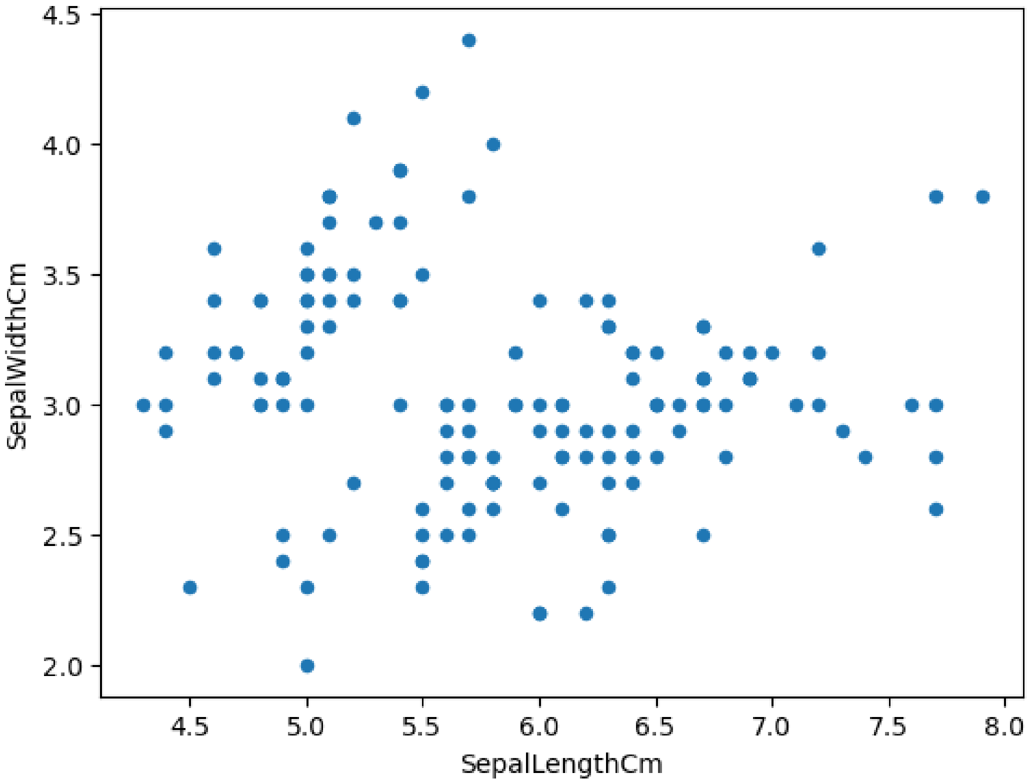
```
iris['Species'].value_counts()
```

Out[46]:

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

In [47]:

```
# 2D scatter plot

iris.plot(kind='scatter', x='SepalLengthCm',
y='SepalWidthCm') plt.show()
```



In [48]:

```
iris.mean()
```

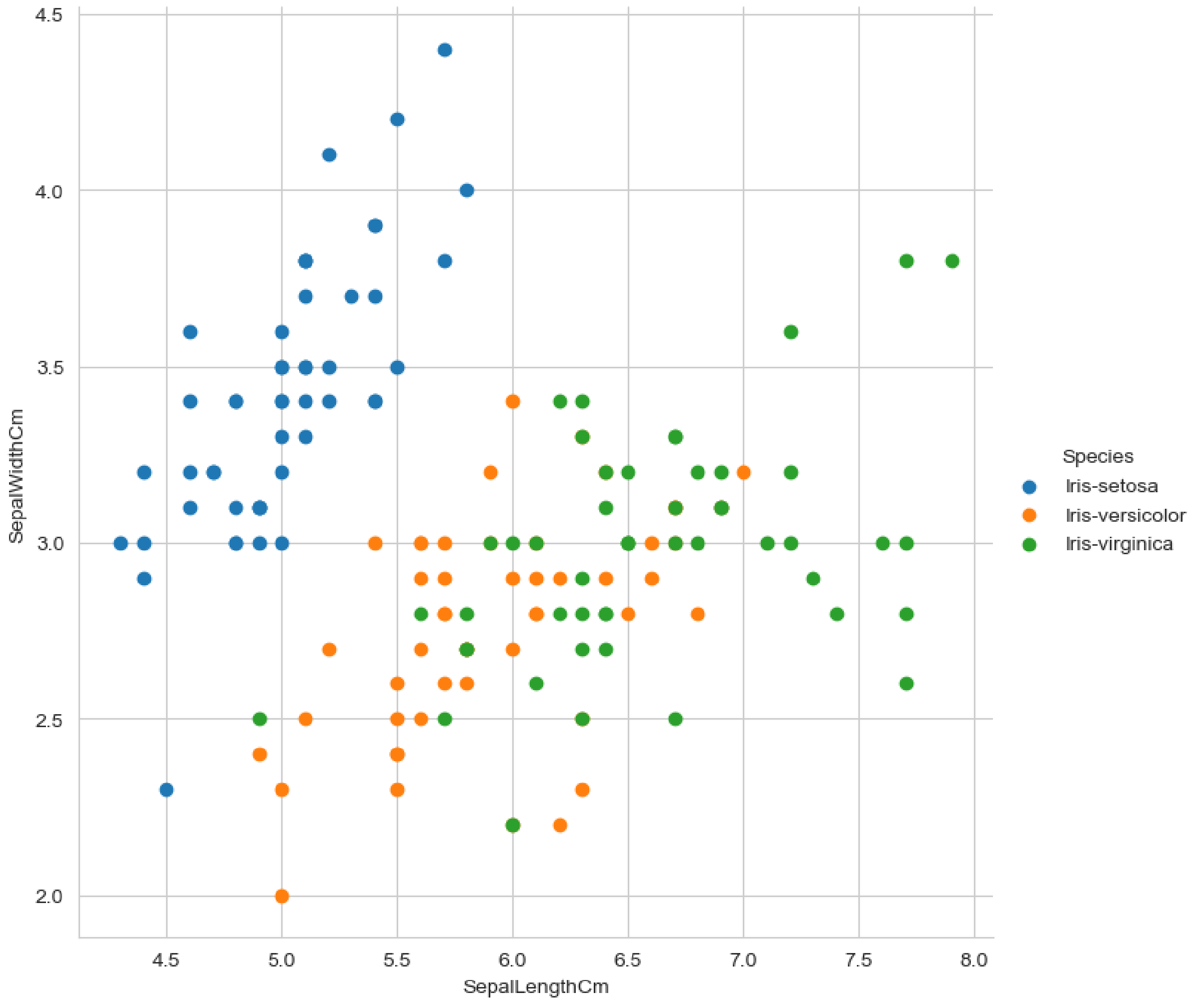
Out[48]:

```
Id               75.500000
SepalLengthCm     5.843333
SepalWidthCm      3.054000
PetalLengthCm     3.758667
PetalWidthCm      1.198667
dtype: float64
```

```
iris.Species.mode()
```

```
0        Iris-setosa
1    Iris-versicolor
2     Iris-virginica
Name: Species, dtype: object
```

```
import seaborn as sns
sns.set_style("whitegrid")
sns.FacetGrid(iris, hue="Species", height=7).map(plt.scatter, "SepalLengthCm",
"SepalWidth
plt.show()
```



In [51]:
```python
# Filter the data for the species 'iris-setosa' and 'iris-versicolor'

setosa_data = iris[iris['Species'] == 'Iris-setosa']
versicolor_data = iris[iris['Species'] == 'Iris-versicolor']
```

In [52]:
```python
setosa_data
```

Out[52]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa
10	11	5.4	3.7	1.5	0.2	Iris-setosa
11	12	4.8	3.4	1.6	0.2	Iris-setosa
12	13	4.8	3.0	1.4	0.1	Iris-setosa
13	14	4.3	3.0	1.1	0.1	Iris-setosa
14	15	5.8	4.0	1.2	0.2	Iris-setosa

Out[52]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
15	16	5.7	4.4	1.5	0.4	Iris-setosa
16	17	5.4	3.9	1.3	0.4	Iris-setosa
17	18	5.1	3.5	1.4	0.3	Iris-setosa
18	19	5.7	3.8	1.7	0.3	Iris-setosa
19	20	5.1	3.8	1.5	0.3	Iris-setosa
20	21	5.4	3.4	1.7	0.2	Iris-setosa
21	22	5.1	3.7	1.5	0.4	Iris-setosa
22	23	4.6	3.6	1.0	0.2	Iris-setosa
23	24	5.1	3.3	1.7	0.5	Iris-setosa
24	25	4.8	3.4	1.9	0.2	Iris-setosa
25	26	5.0	3.0	1.6	0.2	Iris-setosa
26	27	5.0	3.4	1.6	0.4	Iris-setosa
27	28	5.2	3.5	1.5	0.2	Iris-setosa
28	29	5.2	3.4	1.4	0.2	Iris-setosa
29	30	4.7	3.2	1.6	0.2	Iris-setosa
30	31	4.8	3.1	1.6	0.2	Iris-setosa
31	32	5.4	3.4	1.5	0.4	Iris-setosa
32	33	5.2	4.1	1.5	0.1	Iris-setosa
33	34	5.5	4.2	1.4	0.2	Iris-setosa
34	35	4.9	3.1	1.5	0.1	Iris-setosa
35	36	5.0	3.2	1.2	0.2	Iris-setosa
36	37	5.5	3.5	1.3	0.2	Iris-setosa
37	38	4.9	3.1	1.5	0.1	Iris-setosa
39	40	5.1	3.4	1.5	0.2	Iris-setosa
40	41	5.0	3.5	1.3	0.3	Iris-setosa
41	42	4.5	2.3	1.3	0.3	Iris-setosa
42	43	4.4	3.2	1.3	0.2	Iris-setosa
43	44	5.0	3.5	1.6	0.6	Iris-setosa
44	45	5.1	3.8	1.9	0.4	Iris-setosa
45	46	4.8	3.0	1.4	0.3	Iris-setosa
46	47	5.1	3.8	1.6	0.2	Iris-setosa
47	48	4.6	3.2	1.4	0.2	Iris-setosa
48	49	5.3	3.7	1.5	0.2	Iris-setosa
49	50	5.0	3.3	1.4	0.2	Iris-setosa

```
In [53]: versicolor_data
```

Out[53]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
50	51	7.0	3.2	4.7	1.4	Iris-versicolor
51	52	6.4	3.2	4.5	1.5	Iris-versicolor
52	53	6.9	3.1	4.9	1.5	Iris-versicolor
53	54	5.5	2.3	4.0	1.3	Iris-versicolor
54	55	6.5	2.8	4.6	1.5	Iris-versicolor
55	56	5.7	2.8	4.5	1.3	Iris-versicolor
56	57	6.3	3.3	4.7	1.6	Iris-versicolor
57	58	4.9	2.4	3.3	1.0	Iris-versicolor
58	59	6.6	2.9	4.6	1.3	Iris-versicolor
59	60	5.2	2.7	3.9	1.4	Iris-versicolor
60	61	5.0	2.0	3.5	1.0	Iris-versicolor
61	62	5.9	3.0	4.2	1.5	Iris-versicolor
62	63	6.0	2.2	4.0	1.0	Iris-versicolor
63	64	6.1	2.9	4.7	1.4	Iris-versicolor
64	65	5.6	2.9	3.6	1.3	Iris-versicolor
65	66	6.7	3.1	4.4	1.4	Iris-versicolor
66	67	5.6	3.0	4.5	1.5	Iris-versicolor
67	68	5.8	2.7	4.1	1.0	Iris-versicolor
68	69	6.2	2.2	4.5	1.5	Iris-versicolor
69	70	5.6	2.5	3.9	1.1	Iris-versicolor
70	71	5.9	3.2	4.8	1.8	Iris-versicolor
71	72	6.1	2.8	4.0	1.3	Iris-versicolor
72	73	6.3	2.5	4.9	1.5	Iris-versicolor
73	74	6.1	2.8	4.7	1.2	Iris-versicolor
74	75	6.4	2.9	4.3	1.3	Iris-versicolor
75	76	6.6	3.0	4.4	1.4	Iris-versicolor
76	77	6.8	2.8	4.8	1.4	Iris-versicolor
77	78	6.7	3.0	5.0	1.7	Iris-versicolor
78	79	6.0	2.9	4.5	1.5	Iris-versicolor
79	80	5.7	2.6	3.5	1.0	Iris-versicolor
80	81	5.5	2.4	3.8	1.1	Iris-versicolor
81	82	5.5	2.4	3.7	1.0	Iris-versicolor
82	83	5.8	2.7	3.9	1.2	Iris-versicolor
83	84	6.0	2.7	5.1	1.6	Iris-versicolor
84	85	5.4	3.0	4.5	1.5	Iris-versicolor
85	86	6.0	3.4	4.5	1.6	Iris-versicolor
86	87	6.7	3.1	4.7	1.5	Iris-versicolor
87	88	6.3	2.3	4.4	1.3	Iris-versicolor
88	89	5.6	3.0	4.1	1.3	Iris-versicolor

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
89	90	5.5	2.5	4.0	1.3	Iris-versicolor
90	91	5.5	2.6	4.4	1.2	Iris-versicolor
91	92	6.1	3.0	4.6	1.4	Iris-versicolor
92	93	5.8	2.6	4.0	1.2	Iris-versicolor
93	94	5.0	2.3	3.3	1.0	Iris-versicolor
94	95	5.6	2.7	4.2	1.3	Iris-versicolor
95	96	5.7	3.0	4.2	1.2	Iris-versicolor
96	97	5.7	2.9	4.2	1.3	Iris-versicolor
97	98	6.2	2.9	4.3	1.3	Iris-versicolor
98	99	5.1	2.5	3.0	1.1	Iris-versicolor
99	100	5.7	2.8	4.1	1.3	Iris-versicolor

In [54]:

```
# Calculate some basic statistical details for each species
setosa_stats = {
    'percentile': np.percentile(setosa_data['SepalLengthCm'],
    50), 'mean': np.mean(setosa_data['SepalLengthCm']),
    'std_dev': np.std(setosa_data['SepalLengthCm'])
}
```

In [55]:

```
versicolor_stats = {
    'percentile': np.percentile(versicolor_data['SepalLengthCm'],
    50), 'mean': np.mean(versicolor_data['SepalLengthCm']),
    'std_dev': np.std(versicolor_data['SepalLengthCm'])
}
```

In [56]:

```
# Print the results
print('Statistics for Iris Setosa:')
print(setosa_stats)
print('')

print('Statistics for Iris
Versicolor:') print(versicolor_stats)
```

```
Statistics for Iris Setosa:
{'percentile': 5.0, 'mean': 5.005999999999999, 'std_dev':
0.348946987377739}

Statistics for Iris Versicolor:
{'percentile': 5.9, 'mean': 5.936, 'std_dev': 0.5109833656783752}
```

=

Data Analytics I

Create a Linear Regression Model using Python to predict home prices using Boston Housing Dataset. The Bostom Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset.

Link of the Dataset: https://www.kaggle.com/datasets/altavish/boston-housing-dataset

The objective is to predict the value of prices of the house using the given features.

Problem Statement:

- The dataset used in this project comes from the kaggle websites.
- This data was collected in 1978 and each of the 506 entries represents aggreagate information about 14 features of homes located in Boston.

Attribute information:

1) CRIM: This is the per capita crime rate by town
2) ZN: This is the proportion of residential land zoned for lots larger than 25,000 sq.ft.
3)INDUS: This is the proportion of non-retail business acres per town. 4) CHAS: this is the Charles River dummy variable (this is equal to 1 if tract bounds river; 0 otherwise)
5) NOX: This is the nitric oxides concentration (parts per 10 million) 6) RM: This is the average number of rooms per dwelling
7) AGE: This is the proportion of owner-occupied units built prior to 1940
8) DIS: This is the weighted distances to five Boston employment centers
9) RAD: This is the index of accessibility to radial highways
10) TAX: This is the full-value property-tax rate per $10,000

11) PTRATIO: This is the pupil-teacher ratio by town
12) B: This is calculated as 1000(Bk -- 0.63)^2 , where Bk is the proportion of people of African American descent by town
13) LSTAT: This is the percentage lower status of the population
14) MEDV: This is the median value of owner-occupied homes in $1000s

Linear Regression

Linear Regression is one of the most fundamental and widely known Machine Learning Algorithm.
A Linear Regression model predicts the dependent variable using a regression line based on the independent variables.
The equation of the Linear Regression is:

Y = a*X + C + e
Where, C is the intercept,
m is the slope of the line
e is the error term

The equation above is used to predict the value of the targer variable based on the given predictor variable(s).

In [1]:
```
# import required libraries

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

In [2]:
```
df = pd.read_csv('HousingData.csv')
```

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

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2

In [4]:
```
df.tail()
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	NaN	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	NaN	2.5050	1	273	21.0	396.90	7.88	11.9

In [5]:
```
df.describe()
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PT
count	486.000000	486.000000	486.000000	486.000000	506.000000	506.000000	486.000000	506.000000	506.000000	506.000000	506.
mean	3.611874	11.211934	11.083992	0.069959	0.554695	6.284634	68.518519	3.795043	9.549407	408.237154	18.
std	8.720192	23.388876	6.835896	0.255340	0.115878	0.702617	27.999513	2.105710	8.707259	168.537116	2.
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.
25%	0.081900	0.000000	5.190000	0.000000	0.449000	5.885500	45.175000	2.100175	4.000000	279.000000	17.
50%	0.253715	0.000000	9.690000	0.000000	0.538000	6.208500	76.800000	3.207450	5.000000	330.000000	19.
75%	3.560263	12.500000	18.100000	0.000000	0.624000	6.623500	93.975000	5.188425	24.000000	666.000000	20.
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.

In [6]:
```
df.shape
```

Out[6]:
```
(506, 14)
```

In [7]:
```
df.dtypes
```

Out[7]:
```
CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD          int64
TAX          int64
PTRATIO    float64
B          float64
LSTAT      float64
MEDV       float64
dtype: object
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     486 non-null    float64
 1   ZN       486 non-null    float64
 2   INDUS    486 non-null    float64
 3   CHAS     486 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      486 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64  
 9   TAX      506 non-null    int64  
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    486 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

In [9]:
```
df.isna().sum()
```

Out[9]:
```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

dealing missing value:

1. Mean Imputation- If the number of missing values is small and you want to retain all the rows in your dataset, you could impute the missing values using the mean value of the column.

In [10]:
```
mean_value = df['CRIM'].mean()
```

In [11]:
```
# Calculate the mean of each numeric column
means = df.mean()

# Impute missing values with the mean values
df.fillna(value=means, inplace=True)

# Check for any remaining missing values
print(df.isnull().sum())
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

The mean() method is used to calculate the mean of each numeric column in the dataset.
The fillna() method is then used to replace all missing values in the dataframe with the mean values for their respective columns.
The isnull() method is used to check if there are any remaining missing values in the dataframe.

In [12]:
```
target_feature = 'MEDV'
```

In [13]:
```
# Splitting the dataset

x = df.drop(target_feature, axis=1)
y = df[target_feature]
```

In [14]:
```
x.head()
```

Out[14]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.980000
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.140000
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.030000
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.940000
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	12.715432

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

Out[15]:
```
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: MEDV, dtype: float64
```

Use model_selection.train_test_split from sklearn to split the data into training and testing sets. Set test_size=0.2 and random_state=0

In [16]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=0)
```

Training the Model

Import LinearRegression from sklearn.linear_model

In [17]:
```
from sklearn.linear_model import LinearRegression

# Create an instance of a LinearRegression() model named regression.
regression = LinearRegression()
```

Fit regression on the training data

In [18]:
```
regression.fit(x_train, y_train)
```

Out[18]:
```
LinearRegression()
```

In [19]:
```
# train score
train_score=round(regression.score(x_train,y_train)*100,2)
print('Train score of Linear Regression:',train_score)
```

```
Train score of Linear Regression: 76.49
```

In [20]:
```
# Print out the coefficients of the model
print('Coefficients" ', regression.coef_)
```

```
Coefficients"  [-1.26194005e-01  3.76363553e-02 -6.26295345e-02  2.70382928e+00
 -1.45015824e+01  4.08006958e+00 -2.11509464e-02 -1.41798662e+00
  1.96343241e-01 -8.70651696e-03 -1.01396225e+00  8.29504244e-03
 -4.19861039e-01]
```

Predicting Test Data

Use regression.predict() to predict of the x_test of the data

(Lets check how well model fits the test data)

In [21]:
```python
predictions = regression.predict(x_test)
```

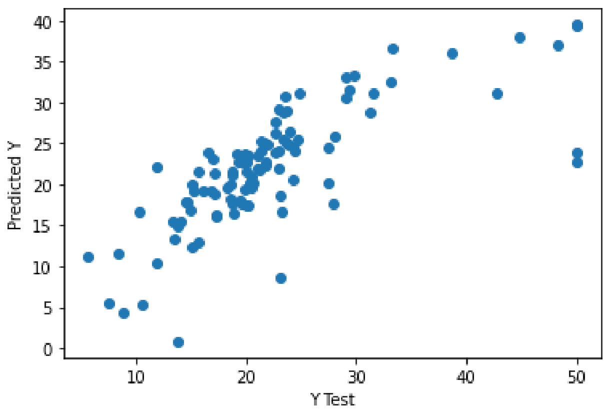
In [22]:
```python
predictions
```

Out[22]:
```
array([26.175296  , 22.64747588, 29.1456294 , 11.52971235, 21.65312134,
       19.42320699, 20.18413017, 21.46914355, 19.1985363 , 19.98228162,
        4.32483046, 16.16891668, 16.87682404,  5.31232373, 39.36827861,
       33.09358732, 21.9152876 , 36.61918436, 31.52676377, 23.52713482,
       24.96022461, 23.69866912, 20.88033802, 30.55074901, 22.74081741,
        8.66805959, 17.65119072, 17.93088633, 36.01223185, 21.16299556,
       17.83464361, 17.43306603, 19.5240167 , 23.50605522, 28.97262793,
       19.21808862, 11.23997435, 23.94256597, 17.86786717, 15.40849806,
       26.3630836 , 21.5193299 , 23.78733694, 14.84041522, 23.9445175 ,
       24.97067627, 20.11366175, 23.08636158, 10.42208266, 24.52832122,
       21.60847326, 18.66228165, 24.53362832, 31.03502944, 12.97457826,
       22.38536236, 21.34822822, 16.10928673, 12.37477824, 22.78596712,
       18.28714824, 21.91802045, 32.49771603, 31.21256855, 17.47867791,
       33.18861907, 19.17896285, 19.94662594, 20.17142015, 23.90228857,
       22.81288844, 24.17911208, 30.83402844, 28.87481037, 25.14581721,
        5.55072029, 37.0183454 , 24.15428003, 27.67587636, 19.63884644,
       28.74874123, 18.83204358, 17.63305678, 37.97947167, 39.49507972,
       24.17228966, 25.33605088, 16.75044819, 25.43224687, 16.65089426,
       16.49186628, 13.37283452, 24.81689254, 31.21188699, 22.0891919 ,
       20.49360168,  0.8229737 , 25.5004737 , 15.5481509 , 17.72901193,
       25.77663998, 22.43131323])
```

Create a scatterplot of the real test values versus the predicted values

In [23]:
```python
plt.scatter(y_test, predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

Out[23]:
```
Text(0, 0.5, 'Predicted Y')
```



In [24]:
```python
from sklearn.metrics import r2_score
score = round(r2_score(y_test,predictions)*100,2)
print("r_2 score:", score)
```

```
r_2 score: 57.03
```

In [25]:
```python
round(regression.score(x_test, y_test)*100,2)
```

Out[25]:
```
57.03
```

it means , it looks like our model r2 score is less on the test data

In [26]:
```
from sklearn import metrics
print('Mean Absolute Error on test data of Linear Regression: ',metrics.mean_absolute_error(y_test, predictions
print('Mean Squared Error on test data of Linear Regression: ',metrics.mean_squared_error(y_test, predictions))
print('Root Mean Squared Error on test data of Linear Regression: ',np.sqrt(metrics.mean_squared_error(y_test,
```

```
Mean Absolute Error on test data of Linear Regression: 3.961621123959114
Mean Squared Error on test data of Linear Regression:  34.9873895442387
Root Mean Squared Error on test data of Linear Regression:  5.915013909048626
```

In [27]:
```
df1 = pd.DataFrame({'Actual':y_test, 'Predicted':predictions, 'Variance':y_test-predictions})
df1.head()
```

Out[27]:

	Actual	Predicted	Variance
329	22.6	26.175296	-3.575296
371	50.0	22.647476	27.352524
219	23.0	29.145629	-6.145629
403	8.3	11.529712	-3.229712
78	21.2	21.653121	-0.453121

In [28]:
```
df.head(15)
```

Out[28]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.000000	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.980000	24.0
1	0.02731	0.0	7.07	0.000000	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.140000	21.6
2	0.02729	0.0	7.07	0.000000	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.030000	34.7
3	0.03237	0.0	2.18	0.000000	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.940000	33.4
4	0.06905	0.0	2.18	0.000000	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	12.715432	36.2
5	0.02985	0.0	2.18	0.000000	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.210000	28.7
6	0.08829	12.5	7.87	0.069959	0.524	6.012	66.6	5.5605	5	311	15.2	395.60	12.430000	22.9
7	0.14455	12.5	7.87	0.000000	0.524	6.172	96.1	5.9505	5	311	15.2	396.90	19.150000	27.1
8	0.21124	12.5	7.87	0.000000	0.524	5.631	100.0	6.0821	5	311	15.2	386.63	29.930000	16.5
9	0.17004	12.5	7.87	0.069959	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.100000	18.9
10	0.22489	12.5	7.87	0.000000	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.450000	15.0
11	0.11747	12.5	7.87	0.000000	0.524	6.009	82.9	6.2267	5	311	15.2	396.90	13.270000	18.9
12	0.09378	12.5	7.87	0.000000	0.524	5.889	39.0	5.4509	5	311	15.2	390.50	15.710000	21.7
13	0.62976	0.0	8.14	0.000000	0.538	5.949	61.8	4.7075	4	307	21.0	396.90	8.260000	20.4
14	0.63796	0.0	8.14	0.069959	0.538	6.096	84.5	4.4619	4	307	21.0	380.02	10.260000	18.2

In [29]:
```
regression.predict([[0.62976,0.0,8.14,0.0,0.538,5.949,61.8,4.7075,4,307,21.0,396.60,8.26]])
```

Out[29]:
```
array([19.58009845])
```

In [30]:
```
regression.intercept_
```

Out[30]:
```
35.0401660294875
```

In [31]:
```
regression.coef_
```

Out[31]:
```
array([-1.26194005e-01,  3.76363553e-02, -6.26295345e-02,  2.70382928e+00,
       -1.45015824e+01,  4.08006958e+00, -2.11509464e-02, -1.41798662e+00,
        1.96343241e-01, -8.70651696e-03, -1.01396225e+00,  8.29504244e-03,
       -4.19861039e-01])
```

In [32]:
```
lr_coefficient = pd.DataFrame()
lr_coefficient["columns"] = x_train.columns
```

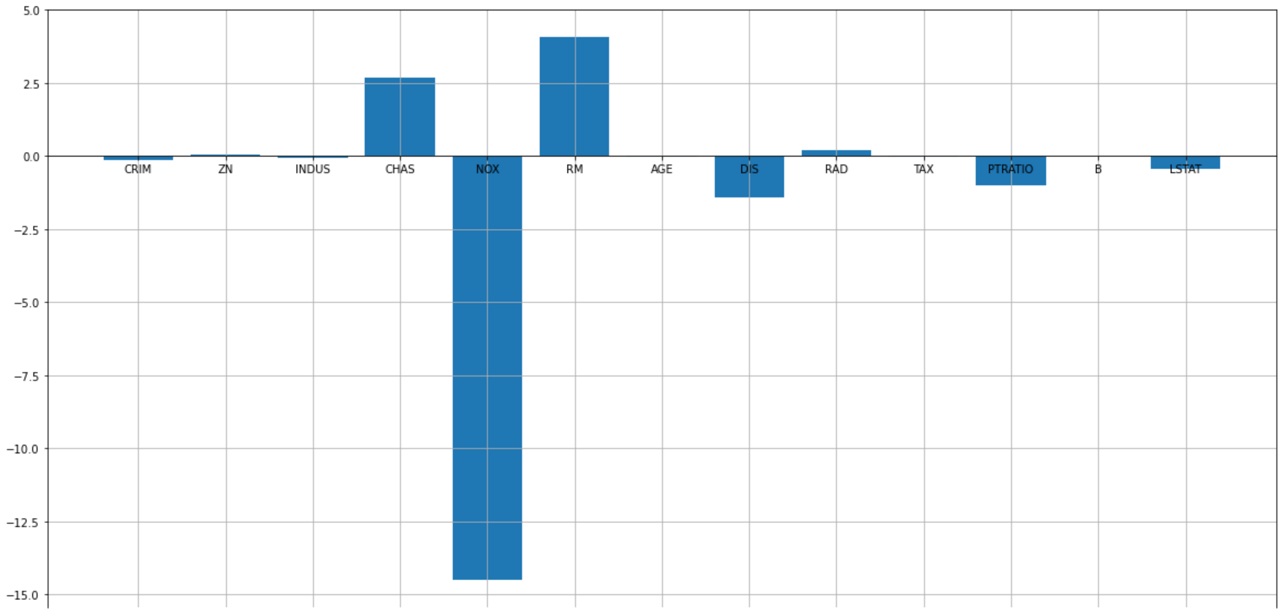
```
lr_coefficient['Coefficient Estimate'] = pd.Series(regression.coef_)
print(lr_coefficient)
```

```
    columns  Coefficient Estimate
0      CRIM             -0.126194
1        ZN              0.037636
2     INDUS             -0.062630
3      CHAS              2.703829
4       NOX            -14.501582
5        RM              4.080070
6       AGE             -0.021151
7       DIS             -1.417987
8       RAD              0.196343
9       TAX             -0.008707
10  PTRATIO             -1.013962
11        B              0.008295
12    LSTAT             -0.419861
```

let's plot a bar chart of above coefficients using matplotlb plotting library

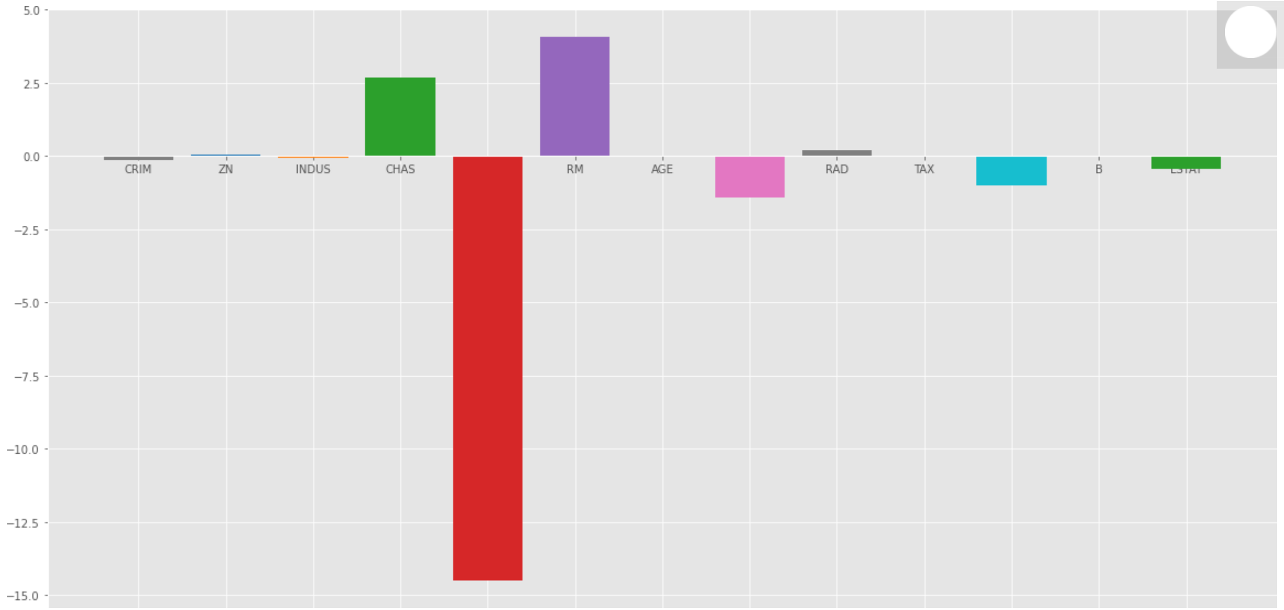
In [38]:

```
# plotting the coefficient score
fig, ax = plt.subplots(figsize =(20, 10))

ax.bar(lr_coefficient["columns"],
lr_coefficient['Coefficient Estimate'])

ax.spines['bottom'].set_position('zero')

plt.style.use('ggplot')
plt.grid()
plt.show()
```



In [40]:

```
fig, ax = plt.subplots(figsize =(20, 10))

color = ['tab:gray', 'tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown', 'tab:pink', '

ax.bar(lr_coefficient["columns"],
lr_coefficient['Coefficient Estimate'],color = color)

ax.spines['bottom'].set_position('zero')

plt.style.use('ggplot')
plt.show()
```


5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
CRIM
ZN
INDUS
CHAS
RM
AGE
RAD
TAX
B

Data Analytics II

1. Implement logistic regression using Python to perform classification on Social_Network_Ads.csv dataset.
2. Compute Confusion matrix to find TP,FP,Tn,FN,Accuracy,Error rate,Precision,Recall on the given dataset.

Data Link: https://www.kaggle.com/datasets/rakeshrau/social-network-ads

Our dataset contains some information about all of our users in the social network, including their User ID, Gender, Age, and Estimated Salary. The last column of the dataset is a vector of booleans describing whether or not each individual ended up clicking on the advertisement (0 = False, 1 = True).

In [87]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [88]:
```
data = pd.read_csv('Social_Network_Ads.csv')
```

In [89]:
```
data
```

Out[89]:

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

400 rows × 5 columns

In [90]:
```
data.head(5)
```

Out[90]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [91]:
```
data.tail()
```

Out[91]:

	User ID	Gender	Age	EstimatedSalary	Purchased
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

In [92]:
```
data.shape
```
Out[92]:
```
(400, 5)
```
In [93]:
```
data.columns
```
Out[93]:
```
Index(['User ID', 'Gender', 'Age', 'EstimatedSalary', 'Purchased'], dtype='object')
```
In [94]:
```
data.describe()
```
Out[94]:

	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.655000	69742.500000	0.357500
std	7.165832e+04	10.482877	34096.960282	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	29.750000	43000.000000	0.000000
50%	1.569434e+07	37.000000	70000.000000	0.000000
75%	1.575036e+07	46.000000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

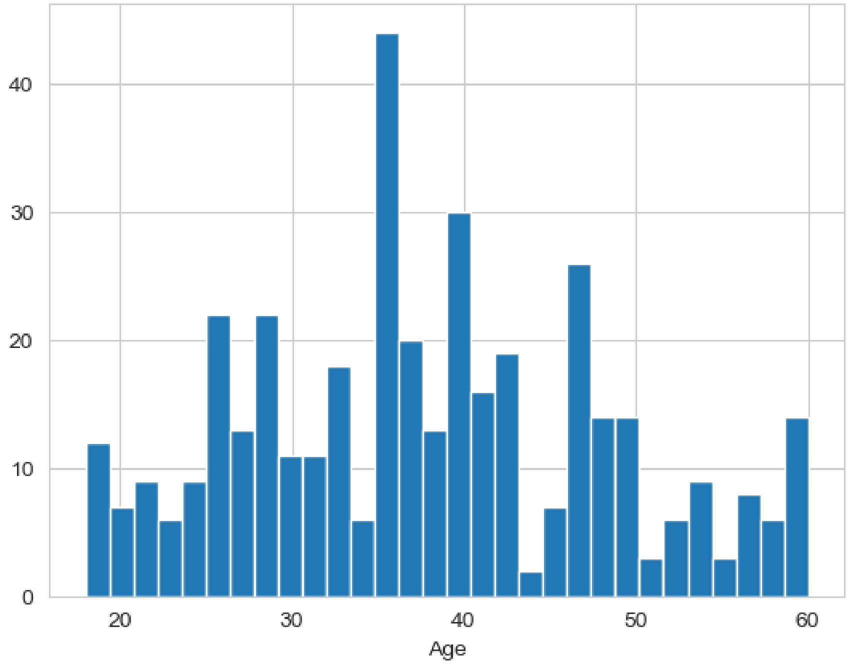
In [95]:
```
data.isnull().sum()
```
Out[95]:
```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```
In [96]:
```
# requier 2 columns age and salary from given data frame
data.iloc[:,2:4]
```
Out[96]:

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
...	...	...
395	46	41000
396	51	23000
397	50	20000
398	36	33000
399	49	36000

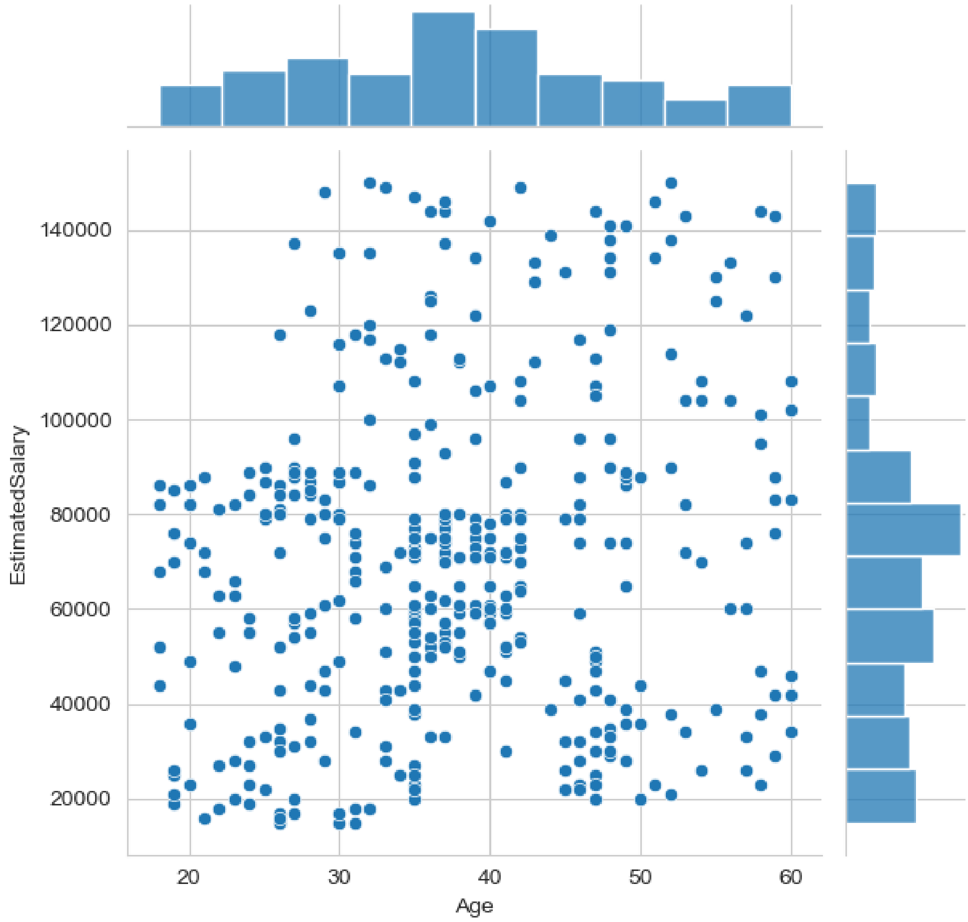
400 rows × 2 columns

In [97]:
```
# in the form of numpy array
ndata = data.iloc[:,2:4].values
```
In [98]:
```
import seaborn as sns
```

In [99]:
```
sns.set_style('whitegrid')
data['Age'].hist(bins=30)
plt.xlabel('Age')
```
Out[99]: Text(0.5, 0, 'Age')

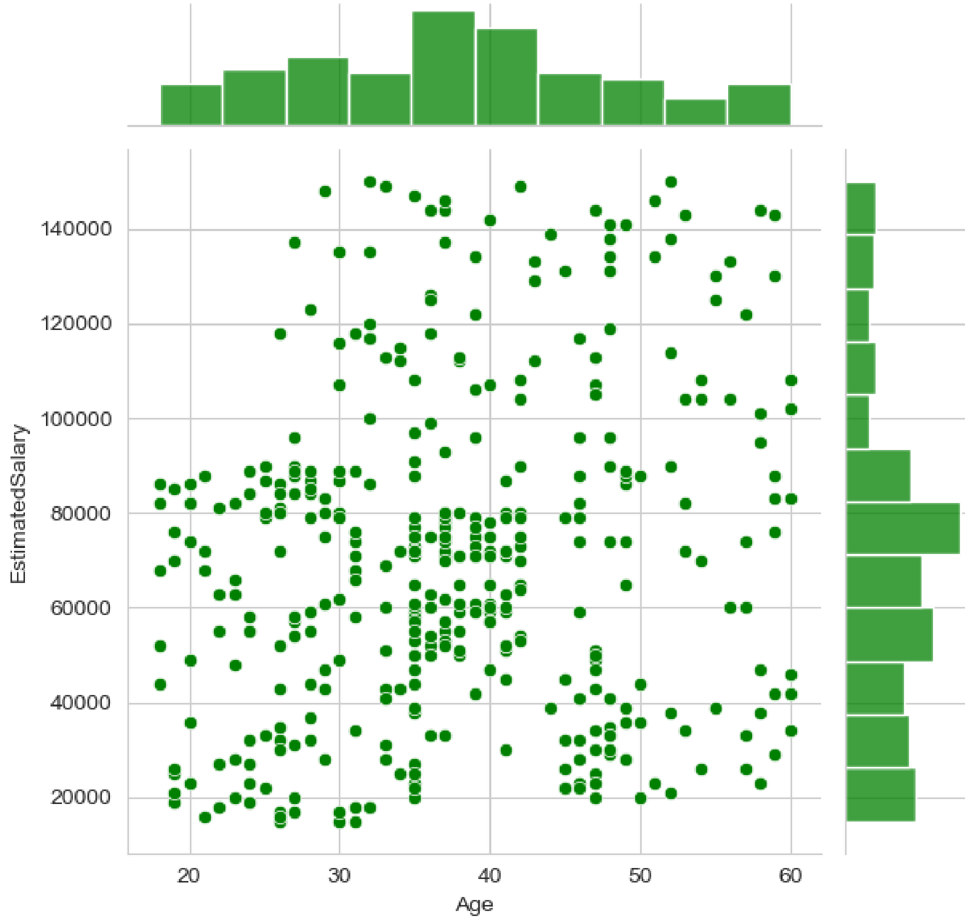


In [100…
```
sns.jointplot(x='Age', y='EstimatedSalary', data = data)
```
Out[100]: <seaborn.axisgrid.JointGrid at 0x1cf865638b0>



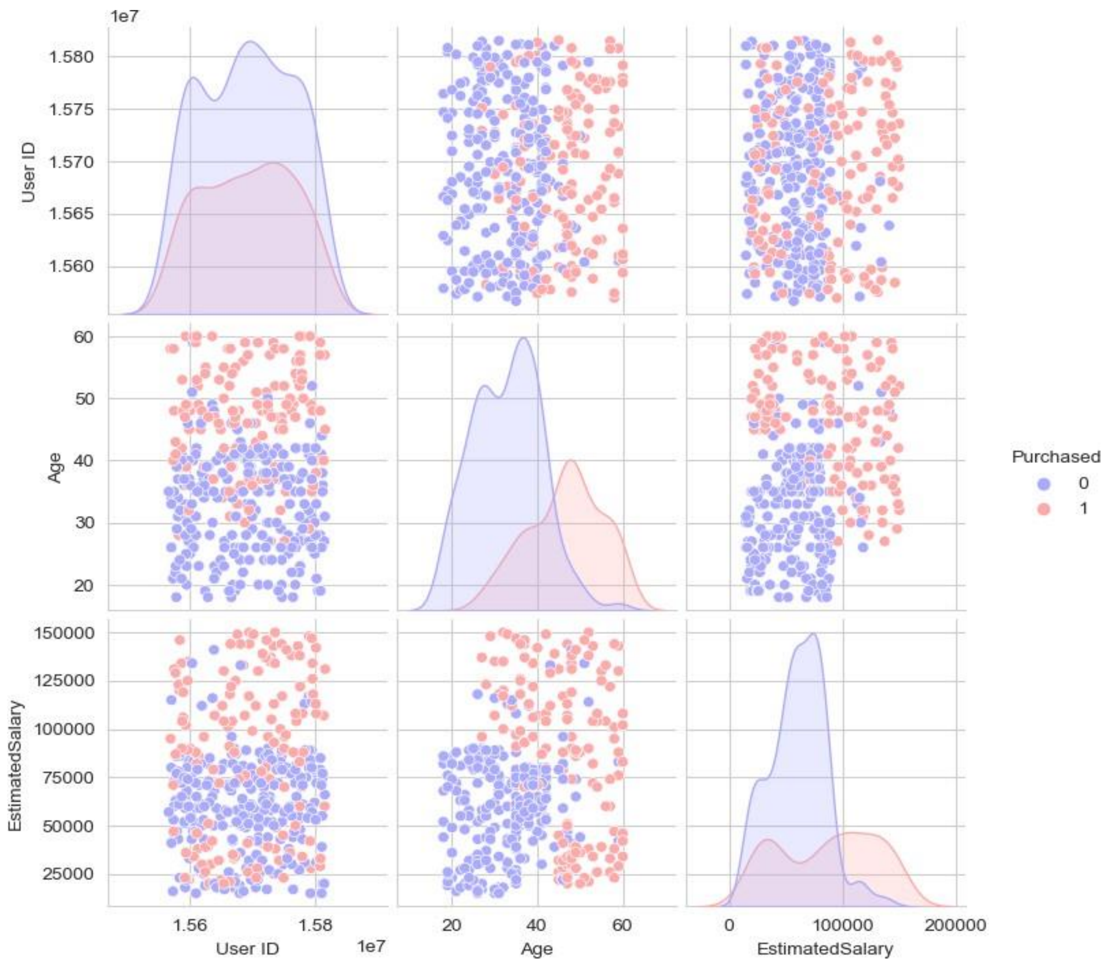
In [101…
```
sns.jointplot(x='Age',y='EstimatedSalary',data= data,color='green')
```

Out[101]: <seaborn.axisgrid.JointGrid at 0x1cf8eabe350>



In [102…
```
sns.pairplot(data,hue='Purchased',palette='bwr')
```

Out[102]: <seaborn.axisgrid.PairGrid at 0x1cf8eabd6f0>



Logistic Regression

In [103…
```
# ** Split the data into training set and testing set using train_test_split**

from sklearn.model_selection import train_test_split
```

In [104…
```
X = data[['Age','EstimatedSalary']]
y = data['Purchased']
```

In [105…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

In [106…
```
X_train.shape
```

Out[106]:
```
(300, 2)
```

In [107…
```
X_test.shape
```

Out[107]:
```
(100, 2)
```

In [108…
```
y_train.shape
```

Out[108]:
```
(300,)
```

In [109…
```
y_test.shape
```

Out[109]:
```
(100,)
```

Train and fit a logistic regression model on the training set.

In [110…
```
from sklearn.linear_model import LogisticRegression
```

In [111…
```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

Out[111]:
```
▾ LogisticRegression
LogisticRegression()
```

Predictions and Evaluations

In [112…
```
predictions = logmodel.predict(X_test)
```

Create a classification report for the model.

In [113…
```
from sklearn.metrics import classification_report
```

In [114…
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.68      1.00      0.81        68
           1       0.00      0.00      0.00        32

    accuracy                           0.68       100
   macro avg       0.34      0.50      0.40       100
weighted avg       0.46      0.68      0.55       100
```

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, predictions)
print(cm)
```

```
[[68  0]
 [32  0]]
```

This Confusion Matrix tells us that there were 68 correct predictions and 32 incorrect ones, meaning the model overall accomplished an 68% accuracy rating.

Data Analytics III

Que: Implement simple naive bayes classification algorithm using python on iris.csv dataset and compare confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset

Probability: Probability is a number that reflects the chance. **Conditional Probability:** Conditional probability is the probability of an event happening, given that another event has already happened. For example, the probability of it raining tomorrow given that it is cloudy today. Think of it as "if-then" probability. If a certain condition is met (it is cloudy), then the likelihood of another event occurring (it raining) can change. It allows us to see how one event influences the probability of another event. **Naive Bayes Classifier:** Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as: **Naïve:** It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. **Bayes:** It is called Bayes because it depends on the principle of Bayes' Theorem. Bayes' Theorem: Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.

In [1]:
```
# Import required libraries
import pandas as pd
import numpy as np
```

In [2]:
```
df = pd.read_csv('iris.csv')
```

In [3]:
```
df
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

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

Out[4]:
```
(150, 6)
```

In [5]:
```python
df.describe()
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [6]:
```python
df.isnull()
```

Out[6]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False
148	False	False	False	False	False	False
149	False	False	False	False	False	False

150 rows × 6 columns

In [7]:
```python
df.isnull().sum()
```

Out[7]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [8]:
```python
x = df.drop(["Species"],axis=1)
y = df["Species"]
```

In [10]:
```python
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(120, 5)
(120,)
(30, 5)
(30,)
```

Train Naive Bayes Classfier Model

In [11]:
```
from sklearn.naive_bayes import MultinomialNB
```

In [12]:
```
classifier = MultinomialNB()
classifier.fit(x_train,y_train)
```

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

In [13]:
```
classifier.score(x_test, y_test)
```

Out[13]:
```
0.8333333333333334
```

In [14]:
```
y_pred = classifier.predict(x_test)
```

In [15]:
```
y_pred
```

Out[15]:
```
array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-setosa', 'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-setosa', 'Iris-virginica', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-virginica', 'Iris-setosa'], dtype='<U15')
```

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

Out[16]:
```
114     Iris-virginica
62     Iris-versicolor
33         Iris-setosa
107     Iris-virginica
7          Iris-setosa
100     Iris-virginica
40         Iris-setosa
86     Iris-versicolor
76     Iris-versicolor
71     Iris-versicolor
134     Iris-virginica
51     Iris-versicolor
73     Iris-versicolor
54     Iris-versicolor
63     Iris-versicolor
37         Iris-setosa
78     Iris-versicolor
90     Iris-versicolor
45         Iris-setosa
16         Iris-setosa
121     Iris-virginica
66     Iris-versicolor
24         Iris-setosa
8          Iris-setosa
126     Iris-virginica
22         Iris-setosa
44         Iris-setosa
97     Iris-versicolor
93     Iris-versicolor
26         Iris-setosa
Name: Species, dtype: object
```

Confusion Matrix

In [17]:
```
import sklearn.metrics

lbs = ['Iris-versicolor','Iris-setosa','Iris-virginica']
print(sklearn.metrics.confusion_matrix(y_test, y_pred, labels = lbs))
```

```
[[10  0  3]
 [ 1 10  0]
 [ 1  0  5]]
```

In [18]:
```python
from sklearn.metrics import classification_report
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      0.91      0.95        11
Iris-versicolor       0.83      0.77      0.80        13
 Iris-virginica       0.62      0.83      0.71         6

       accuracy                           0.83        30
      macro avg       0.82      0.84      0.82        30
   weighted avg       0.85      0.83      0.84        30
```

In [21]:
```python
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)
```

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

In [22]:
```python
classifier.score(x_test, y_test)
```

Out[22]:
```
0.8333333333333334
```

In [23]:
```python
y_pred = gnb.predict(x_test)
```

In [24]:
```python
y_pred
```

Out[24]:
```
array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa'], dtype='<U15')
```

In [25]:
```python
y_test
```

Out[25]:
```
114     Iris-virginica
62     Iris-versicolor
33         Iris-setosa
107     Iris-virginica
7          Iris-setosa
100     Iris-virginica
40         Iris-setosa
86     Iris-versicolor
76     Iris-versicolor
71     Iris-versicolor
134     Iris-virginica
51     Iris-versicolor
73     Iris-versicolor
54     Iris-versicolor
63     Iris-versicolor
37         Iris-setosa
78     Iris-versicolor
90     Iris-versicolor
45         Iris-setosa
16         Iris-setosa
121     Iris-virginica
66     Iris-versicolor
24         Iris-setosa
8          Iris-setosa
126     Iris-virginica
22         Iris-setosa
44         Iris-setosa
97     Iris-versicolor
93     Iris-versicolor
26         Iris-setosa
```

Natural Langauage Processing (NLP)

NLTK installation

pip3 install nltk

#import nltk

#nltk.download()

Tokenization of words:

We use the method word_tokenize() to split a sentence into words.
word tokenization becomes a crucial part of the text (string) to numeric data conversion

In [1]:
```
import nltk
#nltk.download()
```

In [2]:
```
import nltk
nltk.download('punkt')
```

Out[2]:
```
True
```

Word Tokenizer

In [3]:
```
import nltk
from nltk.tokenize import word_tokenize
text = "Welcome to the Python Programming at Indeed Insprining Infotech"
print(word_tokenize(text))
```

```
['Welcome', 'to', 'the', 'Python', 'Programming', 'at', 'Indeed', 'Insprining', 'Infotech']
```

Sentence Tokenizer

In [4]:
```
from nltk.tokenize import sent_tokenize
text = "Hello Everyone. Welcome to the Python Programming"
print(sent_tokenize(text))
```

```
['Hello Everyone.', 'Welcome to the Python Programming']
```

Stemming

When we have many variations of the same word for example...the word is dance and the variations are "dancing", "dances","danced".
Stemming algorithm works by cutting the suffix from the word.

In [5]:
```
from nltk.stem import PorterStemmer
# words = ['Wait','Waiting','Waited','Waits']
words = ['clean','cleaning','cleans','cleaned']
ps = PorterStemmer()
for w in words:
    words=ps.stem(w)
    print(words)
```

```
clean
clean
clean
clean
```

lemmatization

Why is Lemmatization better than Stemming?

Stemming algorithm woks by cutting the suffix from the word and Lemmatization is a more powerful operation because it perform morphological analysis of the words.

Stemming Code:

In [6]:
```
import nltk
from nltk.stem.porter import PorterStemmer
porter_stemmer = PorterStemmer()
text  = "studies studying floors cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
    print('Stemming for ', w,'is',porter_stemmer.stem(w))
```

```
Stemming for  studies is studi
Stemming for  studying is studi
Stemming for  floors is floor
Stemming for  cry is cri
```

lemmatization Code:

In [7]:
```
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
```

Out[7]:
```
True
```

In [8]:
```
import nltk
from nltk.stem import WordNetLemmatizer
Wordnet_lemmatizer = WordNetLemmatizer()
text  = "studies study floors cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
    print('Lemma for ', w,'is',Wordnet_lemmatizer.lemmatize(w))
```

```
Lemma for  studies is study
Lemma for  study is study
Lemma for  floors is floor
Lemma for  cry is cry
```

NLTK stop words

Text may contain stop words like 'the','is','are','a'. Stop words can be filterd from the text to be processed.

In [9]:
```
nltk.download('stopwords')
```

Out[9]:
```
True
```

In [10]:
```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

data = 'AI was introduced in the year 1956 but it gained popularity recently.'
stopwords = set(stopwords.words('english'))
words = word_tokenize(data)
wordsFiltered = []

for w in words:
    if w not in stopwords:
        wordsFiltered.append(w)

print(wordsFiltered)
```

```
['AI', 'introduced', 'year', '1956', 'gained', 'popularity', 'recently', '.']
```

In [11]:
```
print(len(stopwords))
print(stopwords)
```

```
179
{'but', 'only', 'couldn', "don't", 'who', 'each', 'yours', 'with', 'had', 'they', 'don', 'does', 'herself', 'i
f', 'doesn', 'my', 'are', 'an', 'shouldn', 'she', 'doing', 'hasn', 'ourselves', 's', 'i', 'itself', "won't",
'o', "couldn't", 'shan', 'no', 'just', 'more', 'above', 'ain', 'ma', 'through', 'him', 'once', 'didn', 'wasn',
'few', "you're", "wasn't", 'both', 'not', 'having', 'over', 'to', 'hers', 'or', 'on', 'because', 'off', "is
n't", "shan't", 'is', 'below', 'same', "you'd", 'you', 'the', 'y', 'me', "haven't", 'those', 'that', 'wouldn',
'of', 'own', 'we', 'then', 'will', 'them', 'too', 'mustn', 'until', 'some', 'nor', "you've", "doesn't", "it's",
't', "mustn't", 'aren', 've', 'did', 'a', "should've", 'ours', 'd', 'himself', 'all', 'before', 'haven', 'out',
'between', 'now', 'been', 'into', 'needn', 'what', 'was', 'be', 'am', 'as', "aren't", 'down', 'after', 'their',
'during', 'so', 'it', 'which', 'here', 'other', 'than', 'm', "mightn't", 'won', 'for', "that'll", 'from', 'bein
g', 'against', "you'll", 'themselves', "needn't", 'its', 'very', 'he', 'in', 'her', 'most', 'can', "weren't",
"shouldn't", 'again', 'while', "she's", 'and', "didn't", 'further', 'such', "hasn't", 'at', 'these', 'where',
'weren', 'up', 'do', 'll', 'isn', 'should', "wouldn't", 'myself', 'under', 'any', 'how', 'your', 'mightn', 'hav
e', 'theirs', 'whom', 'when', 'yourselves', 'this', 'has', "hadn't", 'were', 'there', 'about', 'why', 'his', 'o
ur', 're', 'yourself', 'by', 'hadn'}
```

Text Analytics

1. Extract Sample document and apply following document preprocessing methods: Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization.
2. Create representation of document by calculating Term Frequency and Inverse Document Frequency.

In [12]:
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag
```

In [13]:
```
# Example document
document = "This is an example document that we will use to demonstrate document preprocessing."
```

In [14]:
```
# Tokenization
tokens = word_tokenize(document)
```

In [15]:
```
tokens
```

Out[15]:
```
['This',
 'is',
 'an',
 'example',
 'document',
 'that',
 'we',
 'will',
 'use',
 'to',
 'demonstrate',
 'document',
 'preprocessing',
 '.']
```

In [16]:
```
import nltk
nltk.download('averaged_perceptron_tagger')
```

Out[16]:
```
True
```

In [17]:
```
# POS tagging
#These tags can indicate whether a word is a noun, verb, adjective, adverb, preposition, conjunction, or other
pos_tags = pos_tag(tokens)
```

In [18]:
```
pos_tags
# DT for determiner , NN for noun , VBD for past tense word , IN for preprosition
#VBZ: verb, 3rd person singular present tense (e.g. "runs")
# VB: verb, base form (e.g. "run")
# PRP: personal pronoun (e.g. "he", "she", "it", "they")
# MD: modal verb (e.g. "can", "should", "will")
# TO: to (e.g. "to run", "to go")
# Here are some additional tags that you may find useful:

# NN: noun, singular or mass (e.g. "cat", "dog", "water")
# NNS: noun, plural (e.g. "cats", "dogs")
# JJ: adjective (e.g. "happy", "blue")
# RB: adverb (e.g. "quickly", "very")
# IN: preposition or subordinating conjunction (e.g. "in", "on", "because")
```

Out[18]:
```
[('This', 'DT'),
 ('is', 'VBZ'),
 ('an', 'DT'),
 ('example', 'NN'),
 ('document', 'NN'),
 ('that', 'IN'),
 ('we', 'PRP'),
 ('will', 'MD'),
 ('use', 'VB'),
 ('to', 'TO'),
 ('demonstrate', 'VB'),
 ('document', 'NN'),
 ('preprocessing', 'NN'),
 ('.', '.')]
```

In [19]:
```
# Stopwords removal
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if not word.lower() in stop_words]
```

In [20]:
```
# Stemming
ps = PorterStemmer()
stemmed_tokens = [ps.stem(word) for word in filtered_tokens]
```

In [21]:
```
# Lemmatization
wnl = WordNetLemmatizer()
lemmatized_tokens = [wnl.lemmatize(word) for word in filtered_tokens]
```

In [22]:
```
# Print the results
print("Tokens: ", tokens)
# print("POS tags: ", pos_tags)
print("Filtered tokens: ", filtered_tokens)
print("Stemmed tokens: ", stemmed_tokens)
print("Lemmatized tokens: ", lemmatized_tokens)
#NLTK is capable of performing all the document preprocessing methods that you have mentioned.
```

```
Tokens: ['This', 'is', 'an', 'example', 'document', 'that', 'we', 'will', 'use', 'to', 'demonstrate', 'documen
t', 'preprocessing', '.']
Filtered tokens:  ['example', 'document', 'use', 'demonstrate', 'document', 'preprocessing', '.']
Stemmed tokens:  ['exampl', 'document', 'use', 'demonstr', 'document', 'preprocess', '.']
Lemmatized tokens:  ['example', 'document', 'use', 'demonstrate', 'document', 'preprocessing', '.']
```

Text Analytics

Create representation of document by calculating Term Frequency and Inverse Document Frequency.

In [23]:
```
import math
from collections import Counter
```

In [24]:
```
# Sample corpus of documents
corpus = [
    'The quick brown fox jumps over the lazy dog',
    'The brown fox is quick',
    'The lazy dog is sleeping'
]
```

In [25]:
```
# Tokenize the documents
tokenized_docs = [doc.lower().split() for doc in corpus]
```

In [26]:
```
# Count the term frequency for each document
tf_docs = [Counter(tokens) for tokens in tokenized_docs]
```

In [27]:
```
# Calculate the inverse document frequency for each term
n_docs = len(corpus)
idf = {}
for tokens in tokenized_docs:
    for token in set(tokens):
        idf[token] = idf.get(token, 0) + 1
for token in idf:
    idf[token] = math.log(n_docs / idf[token])
```

In [28]:
```
# Calculate the TF-IDF weights for each document
tfidf_docs = []
for tf_doc in tf_docs:
    tfidf_doc = {}a
    for token, freq in tf_doc.items():
        tfidf_doc[token] = freq * idf[token]
    tfidf_docs.append(tfidf_doc)
```

In [29]:
```
# Print the resulting TF-IDF representation for each document
for i, tfidf_doc in enumerate(tfidf_docs):
    print(f"Document {i+1}: {tfidf_doc}")
```

```
Document 1: {'the': 0.0, 'quick': 0.4054651081081644, 'brown': 0.4054651081081644, 'fox': 0.4054651081081644,
'jumps': 1.0986122886681098, 'over': 1.0986122886681098, 'lazy': 0.4054651081081644, 'dog': 0.4054651081081644}
Document 2: {'the': 0.0, 'brown': 0.4054651081081644, 'fox': 0.4054651081081644, 'is': 0.4054651081081644, 'qui
ck': 0.4054651081081644}
Document 3: {'the': 0.0, 'lazy': 0.4054651081081644, 'dog': 0.4054651081081644, 'is': 0.4054651081081644, 'slee
ping': 1.0986122886681098}
```

This code uses the Counter class from the collections module to count the term frequency for each document. It then calculates the inverse document frequency by iterating over the tokenized documents and keeping track of the number of documents that each term appears in. Finally, it multiplies the term frequency of each term in each document by its corresponding inverse document frequency to get the TF-IDF weight for each term in each document. The resulting TF-IDF representation for each document is printed to the console.

Data Visualization I

i)Use the inbuilt dataset 'titanic'. The dataset contains 891 rows and contains information about the passengers who boarded the unfortunate Titanic ship. Use the Seaborn library to see if we can find any patterns in the data.

ii)Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.

iii)Use the inbuilt dataset 'titanic' as used in the above problem. Plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (column names: 'sex' and 'age')

iv) Write observations on the inference from the above statistics.

Downloading the Seaborn Library

The seaborn library can be downloaded in a couple of ways. If you are using pip installer for Python libraries, you can execute the following command to download the library:
pip install seaborn

Alternatively, if you are using the Anaconda distribution of Python, you can use execute the following command to download the seaborn library:
conda install seaborn

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

In [2]:
```
dataset = sns.load_dataset('titanic')

dataset.head()
```

Out[2]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

The dataset contains 891 rows and 15 columns and contains information about the passengers who boarded the unfortunate Titanic ship. The original task is to predict whether or not the passenger survived depending upon different features such as their age, ticket, cabin they boarded, the class of the ticket, etc. We will use the Seaborn library to see if we can find any patterns in the data.

In [3]:
```
dataset.shape
```

Out[3]:
```
(891, 15)
```

In [5]:
```
dataset.isnull()
```

Out[5]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	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
6	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False

Out[5]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
871	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
872	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
879	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
887	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False

182 rows × 15 columns

In [6]:

```
dataset.isnull().sum()
```

Out[6]:

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

In [7]:

```
# remove all null values from the dataset
dataset = dataset.dropna()
```

Distributional Plots

Distributional plots, as the name suggests are type of plots that show the statistical distribution of data.

The Dist Plot

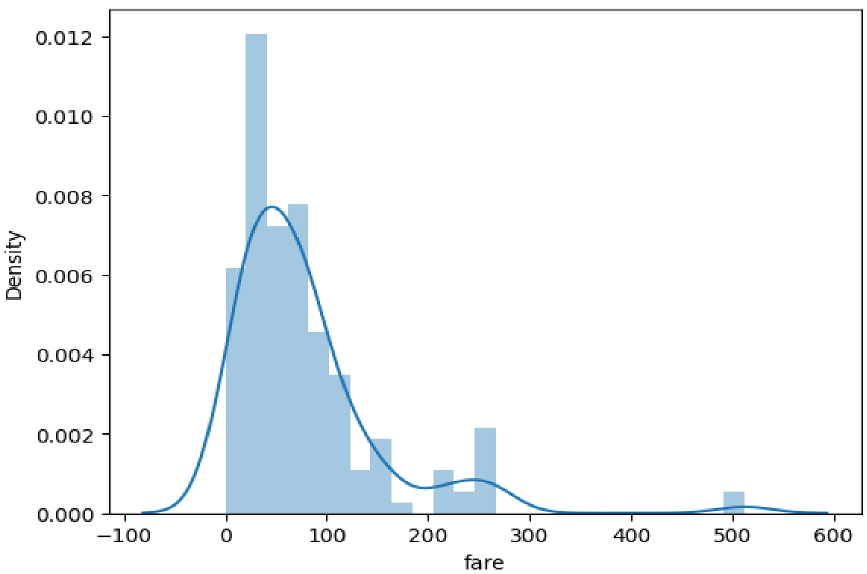
The distplot() shows the histogram distribution of data for a single column. The column name is passed as a parameter to the distplot() function.

In [8]:

```
# Let's see how the price of the ticket for each passenger is distributed.

sns.distplot(dataset['fare'])
```

Out[8]:

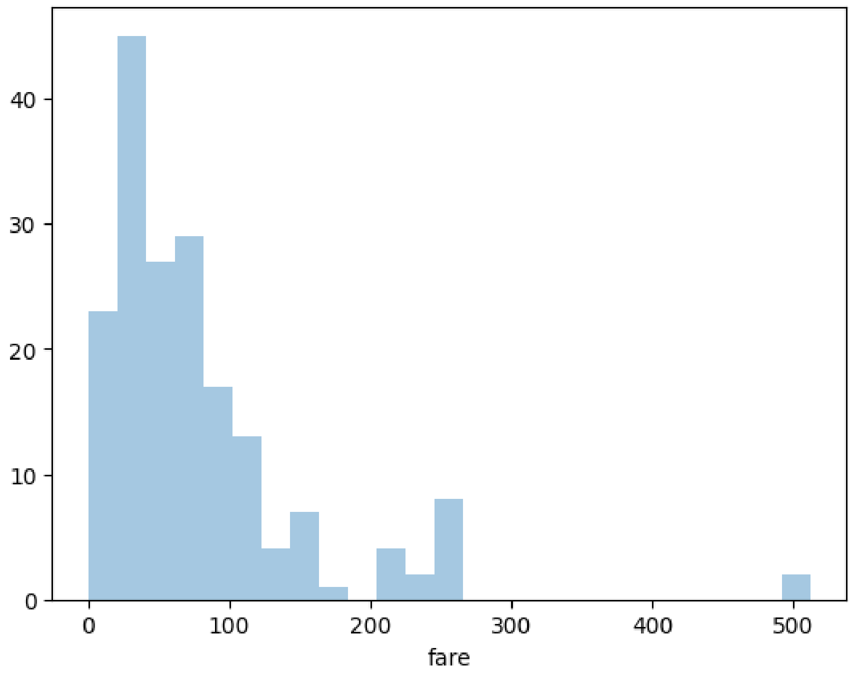
```
<Axes: xlabel='fare', ylabel='Density'>
```



You can see that most of the tickets have been solved between 0-50 dollars. The line that you see represents the kernel density estimation. You can remove this line by passing False as the parameter for the kde attribute as shown below:

In [9]:
```
sns.distplot(dataset['fare'], kde=False)
```

Out[9]: <Axes: xlabel='fare'>

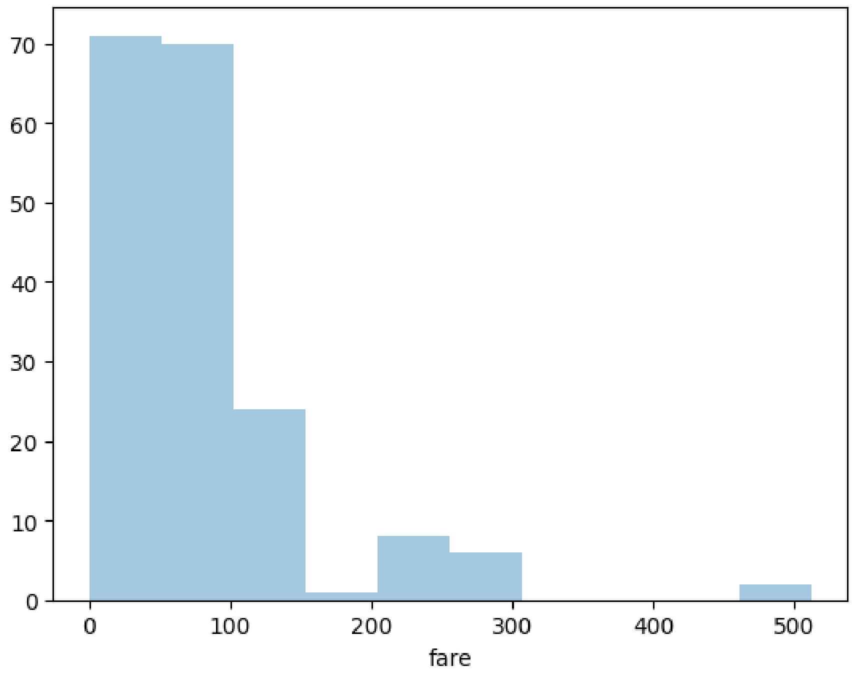


Now you can see there is no line for the kernel density estimation on the plot.

You can also pass the value for the bins parameter in order to see more or less details in the graph

```
sns.distplot(dataset['fare'], kde=False, bins=10)  #Here we set the number of bins to 10.
```

<Axes: xlabel='fare'>



In the output, you will see data distributed in 10 bins You can clearly see that for more than 700 passengers, the ticket price is between 0 and 50.

Data Visualization III

Download the Iris flower dataset or any other dataset into a DataFrame. Scan the dataset and give the inference as:

1. List down the features and theri types (ex, numeric, nominal) available in the dataset.
2. Create a histogram for each feature in the dataset to illustrate the feature distributions.
3. Create a boxplot for each feature in the dataset.
4. Compare distribution and identity outliers.

In [1]:
```
# Loading libraries

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

In [2]:
```
Iris = pd.read_csv('/content/Iris.csv')
Iris
```

Out[2]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [3]:
```
Iris.shape
```

Out[3]:
```
(150, 6)
```

In [4]:
```
Iris.describe()
```

Out[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [5]:
```
Iris.dtypes
```

Out[5]:
```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

In [ ]:
```
Iris.isnull().sum()
```

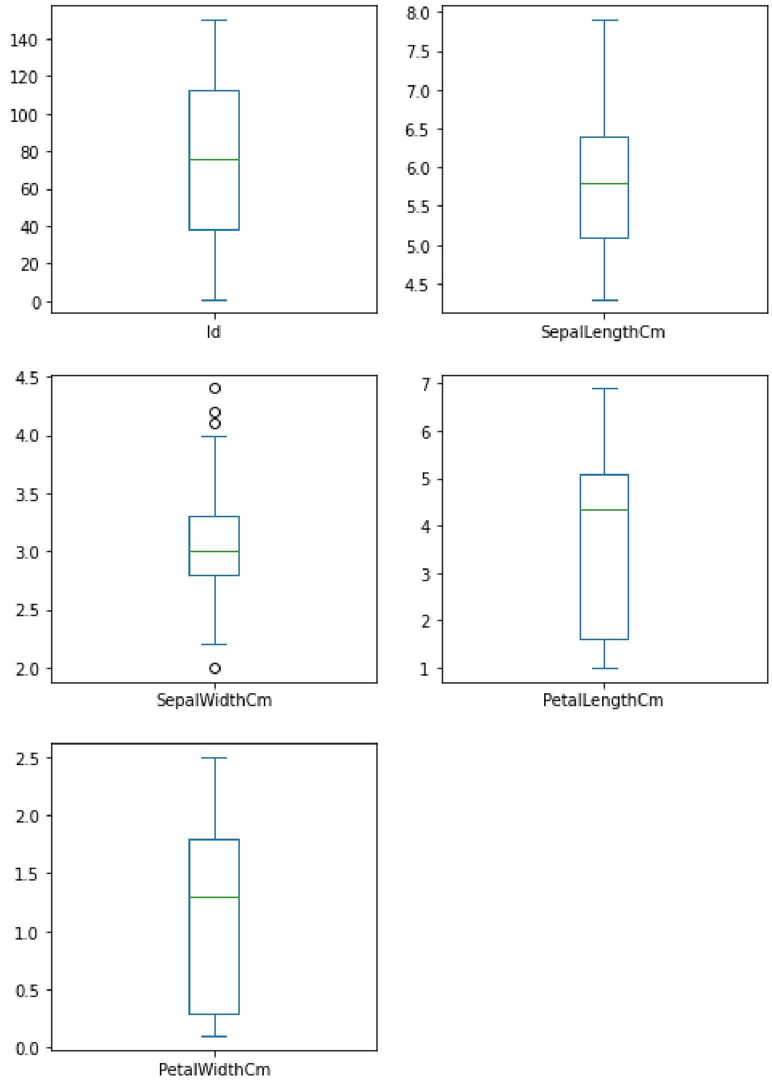
Out[ ]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [ ]:
```
print(Iris.groupby('Species').size())
```

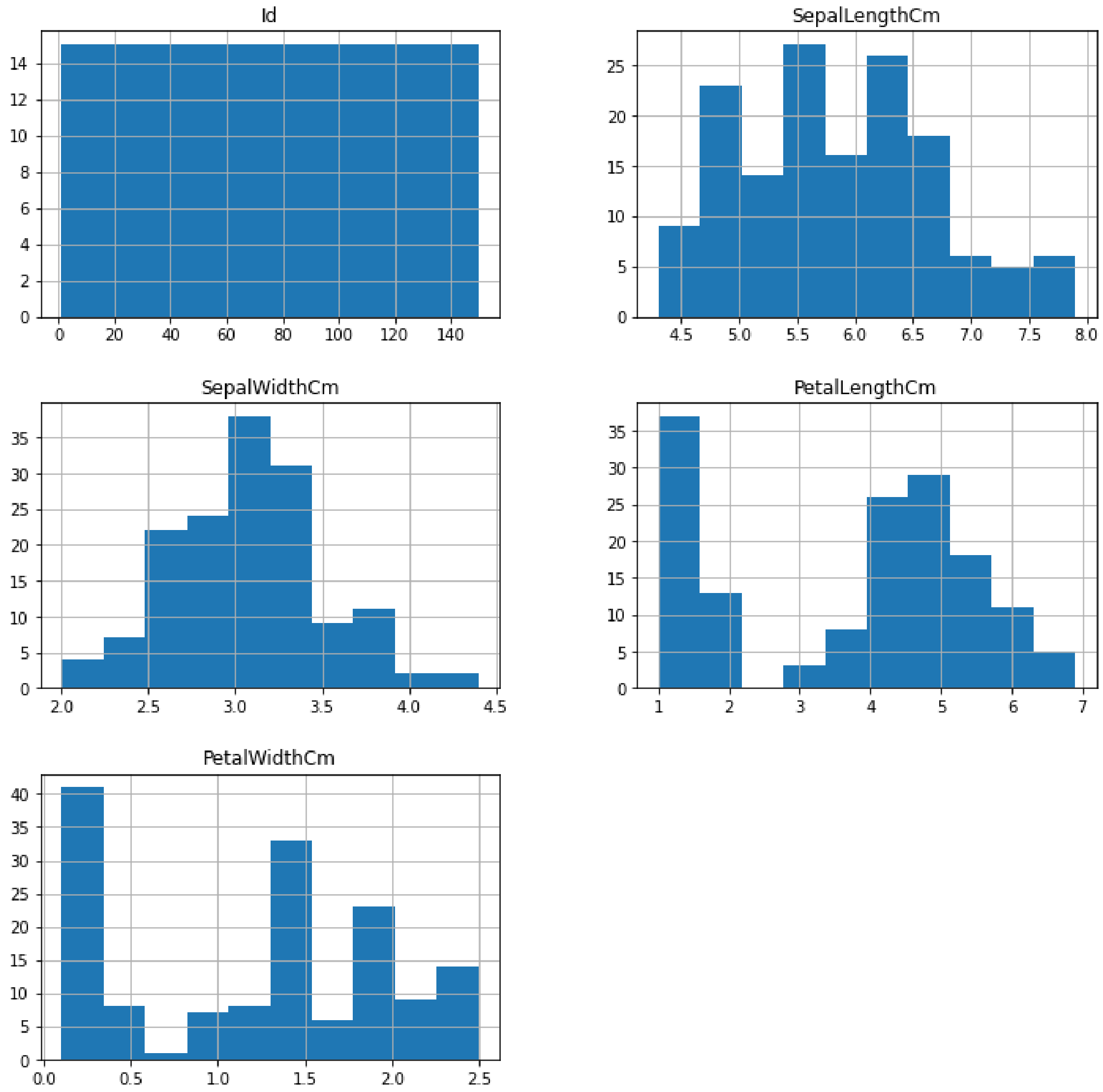
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

Data Visualization

In [ ]:
```
# Box and whisker plots
Iris.plot(kind='box', subplots=True, layout=(3,2), figsize=(8,12));
```



```
In [ ]: Iris.hist(figsize=(12,12))
        plt.show()
```



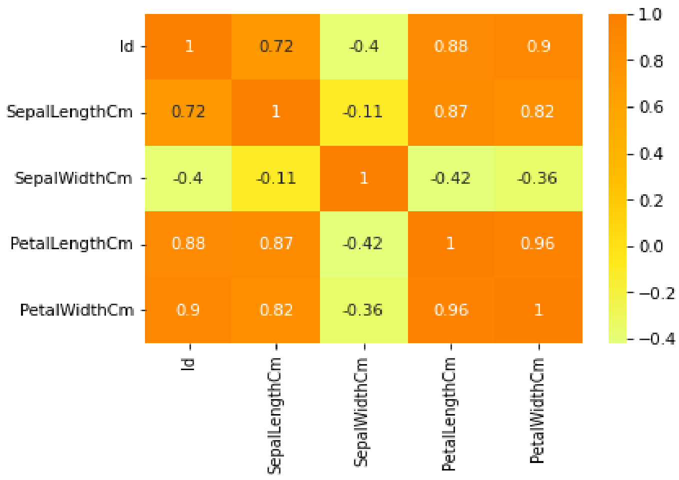

```
In [ ]: Iris.corr()
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Id	1.000000	0.716676	-0.397729	0.882747	0.899759
SepalLengthCm	0.716676	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.397729	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.882747	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.899759	0.817954	-0.356544	0.962757	1.000000

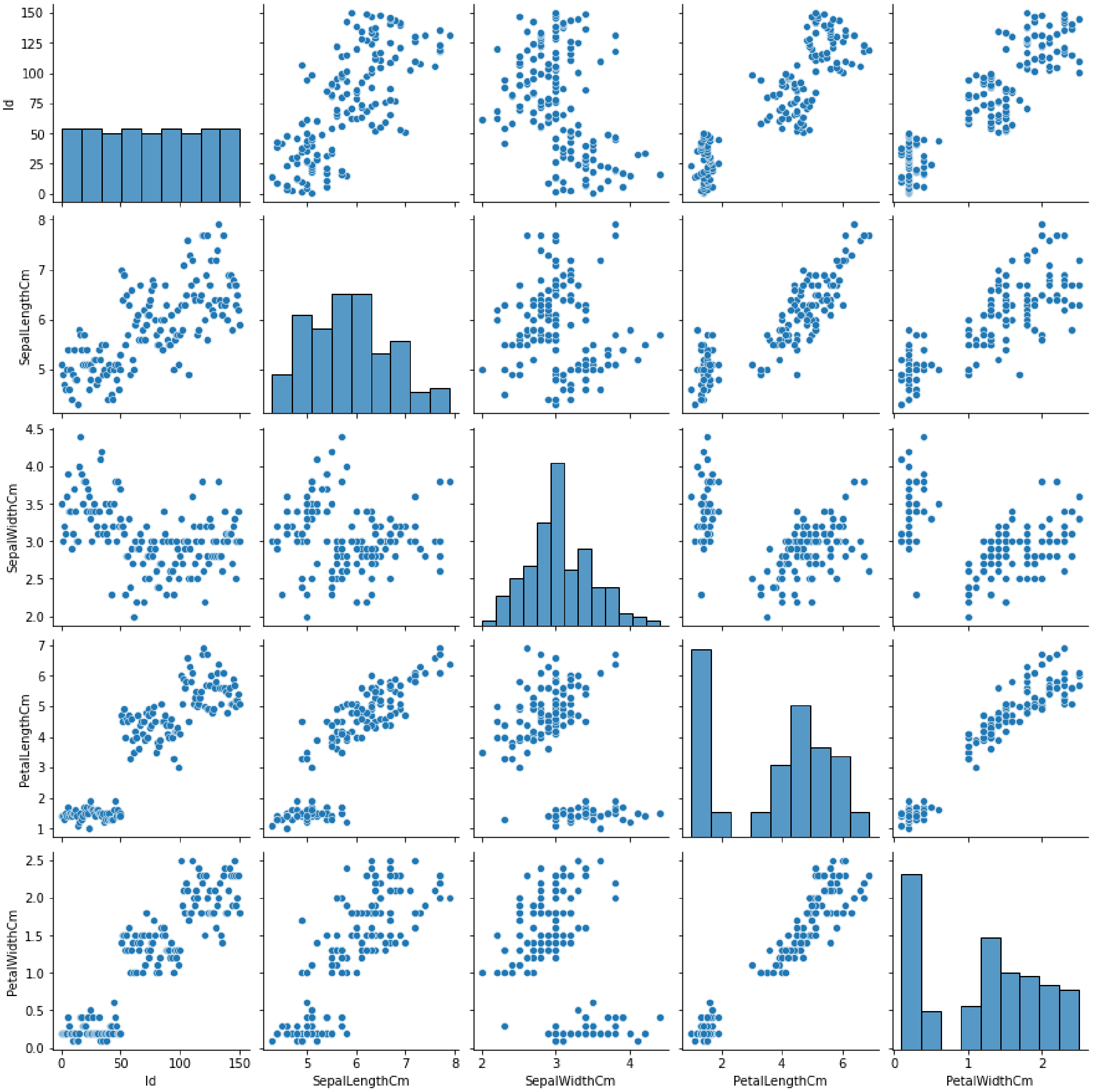
```
In [ ]: sns.heatmap(Iris.corr(), annot=True, cmap='Wistia')
```

```
Out[ ]: <AxesSubplot:>
```



```
In [ ]: sns.pairplot(Iris)
```

```
Out[ ]: <seaborn.axisgrid.PairGrid at 0x7f6327f7c1c0>
```



Step 1: Load the dataset into a DataFrame

```
In [ ]: from sklearn.datasets import load_iris
        import pandas as pd

        iris = load_iris()
        iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
        iris_df['target'] = iris.target
```

Step 2: List down the features and their types

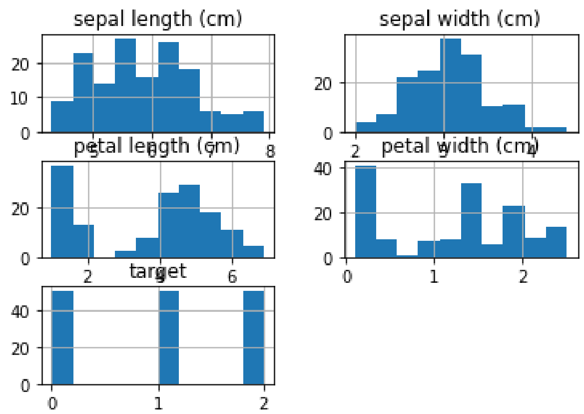
The Iris flower dataset contains the following features:

sepal length (numeric)
sepal width (numeric)
petal length (numeric)
petal width (numeric)
target (nominal)

Step 3: Create a histogram for each feature

To create a histogram for each feature, we can use the hist method of the DataFrame as follows:

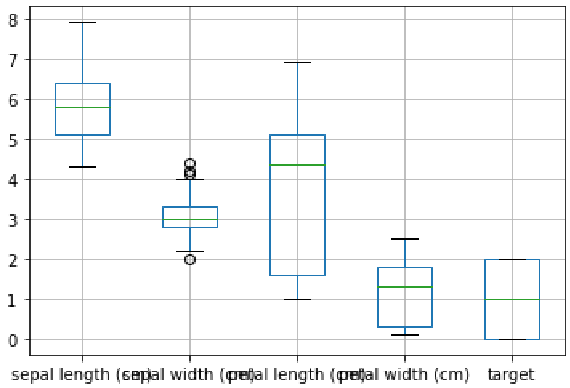
```
In [ ]: import matplotlib.pyplot as plt

iris_df.hist()
plt.show()
```



Step 4: Create a boxplot for each feature

To create a boxplot for each feature, we can use the boxplot method of the DataFrame as follows:

```
In [ ]: iris_df.boxplot()
plt.show()
```



Step 5: Compare distribution and identify outliers

By looking at the histograms and boxplots, we can see the distribution of each feature and identify any outliers.

For example, we can see that the petal length and petal width have a bimodal distribution. The sepal length and sepal width have a more normal distribution. We can also see that there are some outliers in the sepal width and petal length features.

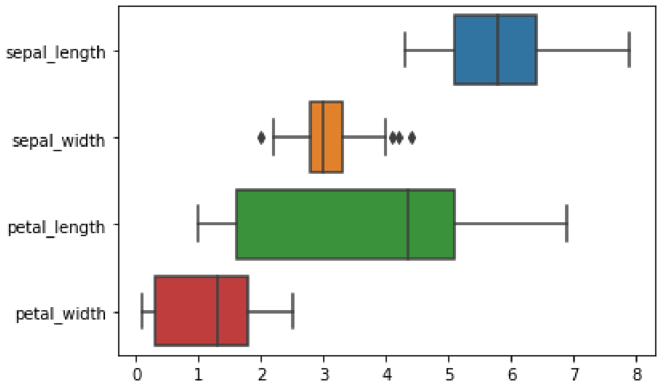
We can further investigate the outliers using the describe method of the DataFrame as follows:

In [ ]:
```
iris_df.describe()
```

Out[ ]:

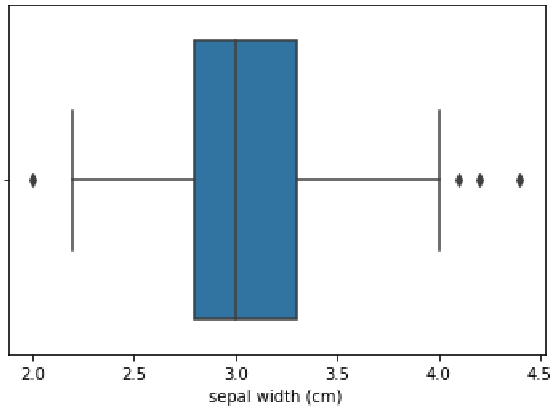
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.boxplot(data=iris, orient="h")
plt.show()
```



In [ ]:
```
sns.boxplot(x = 'sepal width (cm)', data = iris_df)
```

Out[ ]: <AxesSubplot:xlabel='sepal width (cm)'>



In [ ]:
```
Q1 = Iris.SepalWidthCm.quantile(0.25)
Q3 = Iris.SepalWidthCm.quantile(0.75)
IQR = Q3-Q1
print(IQR)
```

```
0.5
```

In [ ]:
```
data = Iris[Iris.SepalWidthCm < (Q1 - 1.5 * IQR) / (Iris.SepalWidthCm > (Q3 + 1.5 * IQR))]
```

In [ ]:
```
data
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

147 rows × 6 columns

Data Visualization II

i)Use the inbuilt dataset 'titanic'. The dataset contains 891 rows and contains information about the passengers who boarded the unfortunate Titanic ship. Use the Seaborn library to see if we can find any patterns in the data.

ii)Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.

iii)Use the inbuilt dataset 'titanic' as used in the above problem. Plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (column names: 'sex' and 'age')

iv) Write observations on the inference from the above statistics.

Downloading the Seaborn Library

The seaborn library can be downloaded in a couple of ways. If you are using pip installer for Python libraries, you can execute the following command to download the library:

pip install seaborn

Alternatively, if you are using the Anaconda distribution of Python, you can use execute the following command to download the seaborn library:

conda install seaborn

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

In [ ]:
```
dataset = sns.load_dataset('titanic')

dataset.head()
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

The dataset contains 891 rows and 15 columns and contains information about the passengers who boarded the unfortunate Titanic ship. The original task is to predict whether or not the passenger survived depending upon different features such as their age, ticket, cabin they boarded, the class of the ticket, etc. We will use the Seaborn library to see if we can find any patterns in the data.

In [ ]:
```
dataset.shape
```

Out[ ]:
```
(891, 15)
```

In [ ]:
```
dataset.isnull()
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	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
6	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
871	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
872	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
879	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
887	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
889	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False

182 rows × 15 columns

In [ ]:
```
dataset.isnull().sum()
```

Out[ ]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

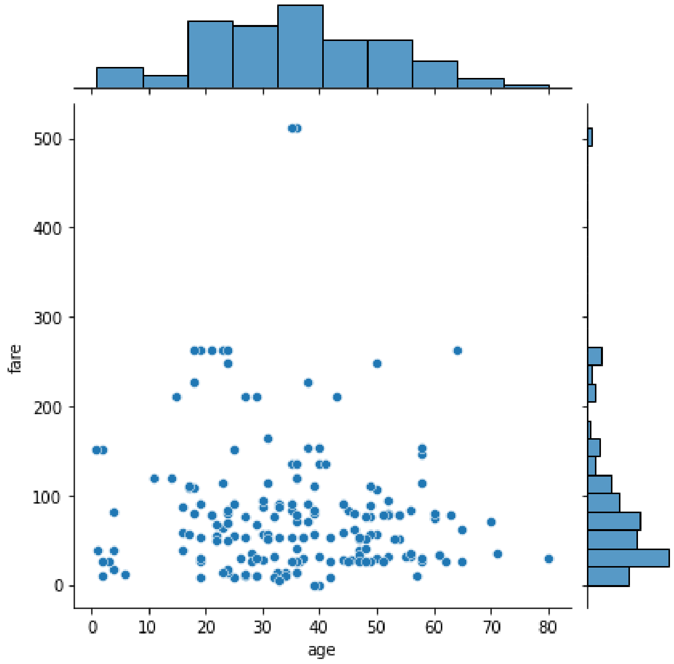
In [ ]:
```
# remove all null values from the dataset
dataset = dataset.dropna()
```

The Joint Plot

The jointplot()is used to display the mutual distribution of each column. You need to pass three parameters to jointplot. The first parameter is the column name for which you want to display the distribution of data on x-axis. The second parameter is the column name for which you want to display the distribution of data on y-axis. Finally, the third parameter is the name of the data frame.

In [ ]:
```
# Let's plot a joint plot of age and fare columns to see if we can find any relationship between the two.
sns.jointplot(x='age', y='fare', data=dataset)
```

Out[ ]:
```
<seaborn.axisgrid.JointGrid at 0x1dfe96c8e20>
```

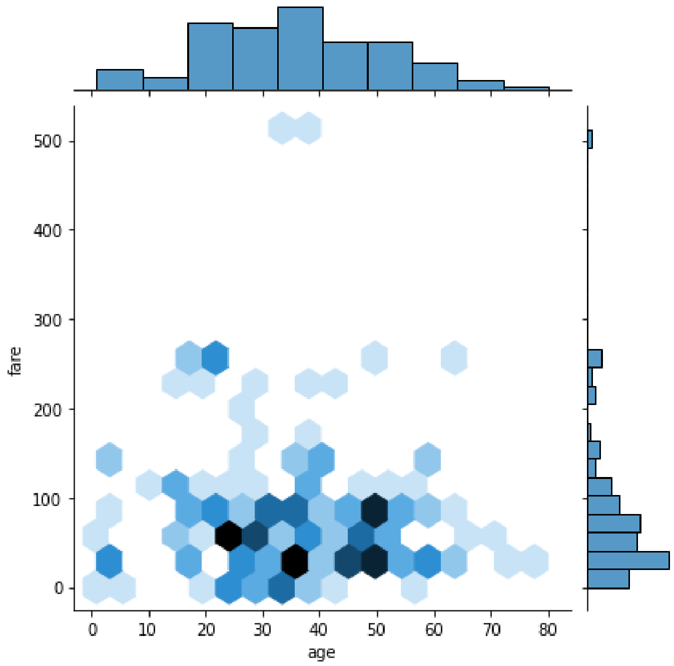


From the output, you can see that a joint plot has three parts. A distribution plot at the top for the column on the x-axis, a distribution plot on the right for the column on the y-axis and a scatter plot in between that shows the mutual distribution of data for both the columns. You can see that there is no correlation observed between prices and the fares.

You can change the type of the joint plot by passing a value for the kind parameter. For instance, if instead of scatter plot, you want to display the distribution of data in the form of a hexagonal plot, you can pass the value hex for the kind parameter.

In [ ]:
```
sns.jointplot(x='age', y='fare', data=dataset, kind='hex')
```

Out[ ]:
```
<seaborn.axisgrid.JointGrid at 0x1dfe97b90a0>
```



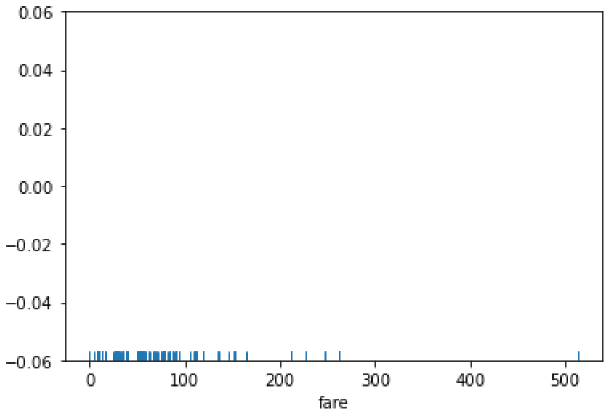
In the hexagonal plot, the hexagon with most number of points gets darker color. So if you look at the above plot, you can see that most of the passengers are between age 20 and 30 and most of them paid between 10-50 for the tickets.

The Rug Plot

The rugplot() is used to draw small bars along x-axis for each point in the dataset.

In [ ]:
```
# To plot a rug plot, you need to pass the name of the column. Let's plot a rug plot for fare.

sns.rugplot(dataset['fare'])
```

Out[ ]:
```
<AxesSubplot:xlabel='fare'>
```



you can see that as was the case with the distplot(), most of the instances for the fares have values between 0 and 100.

These are some of the most commonly used distribution plots offered by the Python's Seaborn Library.

Let's see some of categorical plots in the Seaborn library.

Categorical Plots

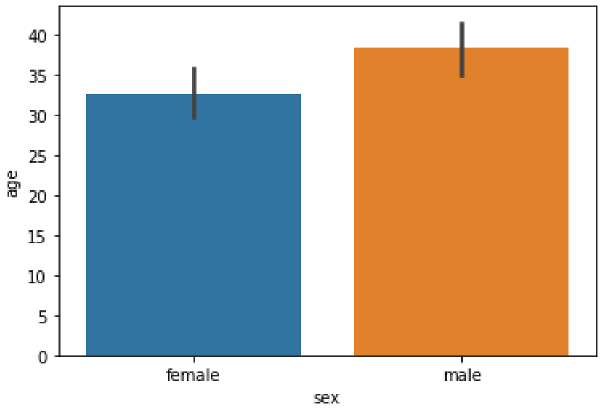
Categorical plots, as the name suggests are normally used to plot categorical data. The categorical plots plot the values in the categorical column against another categorical column or a numeric column. Let's see some of the most commonly used categorical data.

The Bar Plot

The barplot() is used to display the mean value for each value in a categorical column, against a numeric column. The first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset.

In [ ]:
```
#  to know the mean value of the age of the male and female passengers, you can use the bar plot

sns.barplot(x='sex', y='age', data=dataset)
```

Out[ ]:
```
<AxesSubplot:xlabel='sex', ylabel='age'>
```

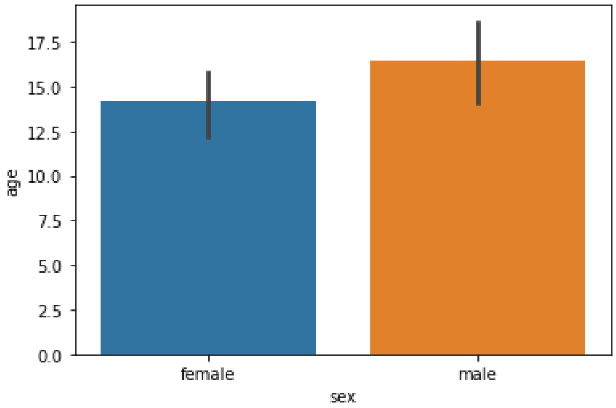


you can clearly see that the average age of male passengers is just less than 40 while the average age of female passengers is around 33.

to finding the average, the bar plot can also be used to calculate other aggregate values for each category. To do so, you need to pass the aggregate function to the estimator.

In [ ]:
```python
# # To calculate the standard deviation for the age of each gender

import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

sns.barplot(x='sex', y='age', data=dataset, estimator=np.std)

# we use the std aggregate function from the numpy library to calculate the standard deviation for the ages of
```

Out[ ]:
```
<AxesSubplot:xlabel='sex', ylabel='age'>
```

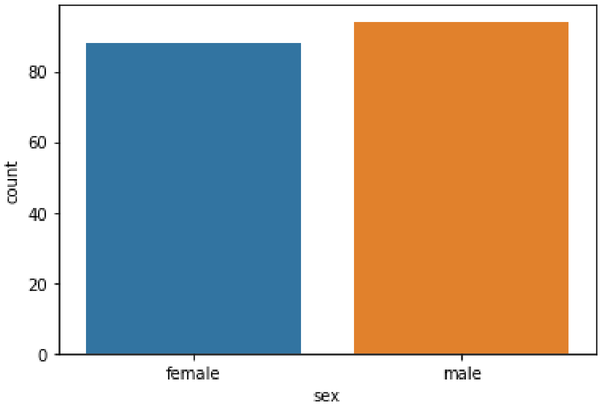


The Count Plot

The count plot is similar to the bar plot, however it displays the count of the categories in a specific column.

In [ ]:
```python
# to count the number of males and women passenger we can do so using count plot

sns.countplot(x='sex', data=dataset)
```

Out[ ]:
```
<AxesSubplot:xlabel='sex', ylabel='count'>
```

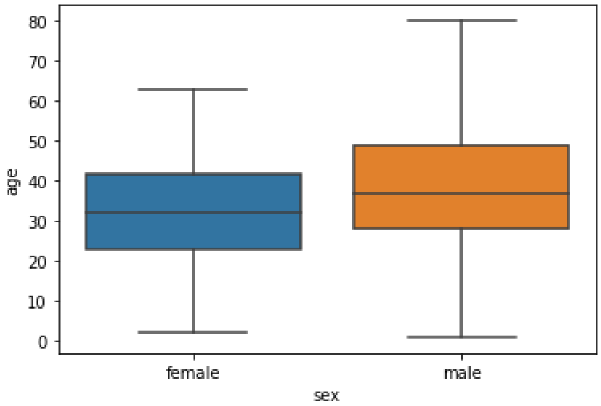


Box Plot

The box plot is used to display the distribution of the categorical data in the form of quartiles. The center of the box shows the median value. The value from the lower whisker to the bottom of the box shows the first quartile. From the bottom of the box to the middle of the box lies the second quartile. From the middle of the box to the top of the box lies the third quartile and finally from the top of the box to the top whisker lies the last quartile.

In [ ]:
```
#  let's plot a box plot that displays the distribution for the age with respect to each gender. You need to pa

sns.boxplot(x='sex', y='age', data=dataset)
```

Out[ ]:
```
<AxesSubplot:xlabel='sex', ylabel='age'>
```

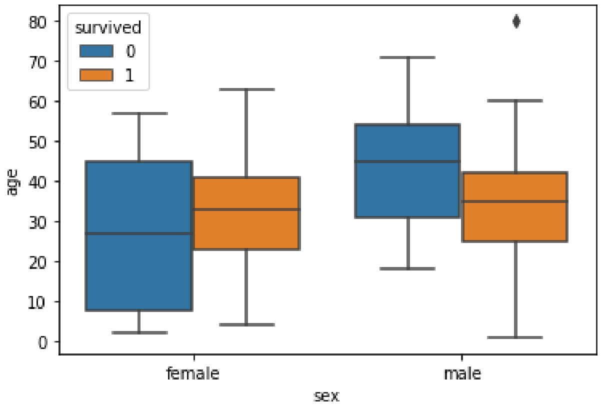


Let's try to understand the box plot for female. The first quartile starts at around 5 and ends at 22 which means that 25% of the passengers are aged between 5 and 25. The second quartile starts at around 23 and ends at around 32 which means that 25% of the passengers are aged between 23 and 32. Similarly, the third quartile starts and ends between 34 and 42, hence 25% passengers are aged within this range and finally the fourth or last quartile starts at 43 and ends around 65.

If there are any outliers or the passengers that do not belong to any of the quartiles, they are called outliers and are represented by dots on the box plot.

In [ ]:
```
#  if you want to see the box plots of forage of passengers of both genders, along with the information about w

sns.boxplot(x='sex', y='age', data=dataset, hue="survived")
```

Out[ ]:
```
<AxesSubplot:xlabel='sex', ylabel='age'>
```



The Violin Plot

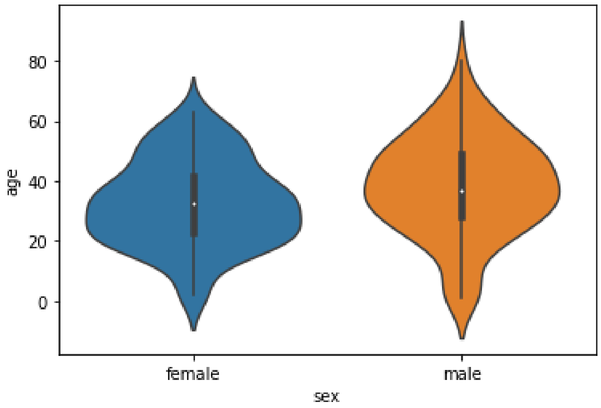
The violin plot is similar to the box plot, however, the violin plot allows us to display all the components that actually correspond to the data point. The violinplot() function is used to plot the violin plot. Like the box plot, the first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset.

In [ ]:

```
# Let's plot a violin plot that displays the distribution for the age with respect to each gender.

sns.violinplot(x='sex', y='age', data=dataset)
```

Out[ ]:

```
<AxesSubplot:xlabel='sex', ylabel='age'>
```



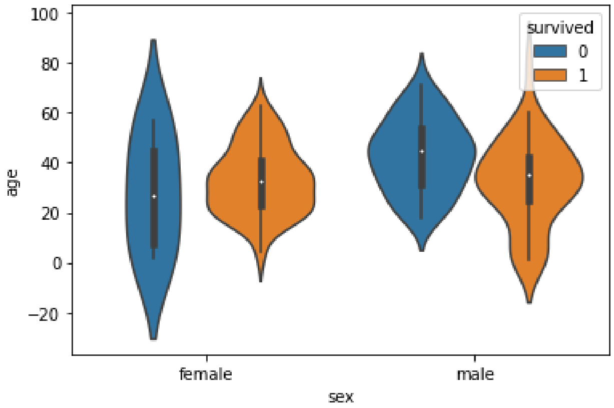
violin plots provide much more information about the data as compared to the box plot. Instead of plotting the quartile, the violin plot allows us to see all the components that actually correspond to the data. The area where the violin plot is thicker has a higher number of instances for the age.

In [ ]:

```
# add another categorical variable to the violin plot using the hue parameter

sns.violinplot(x='sex', y='age', data=dataset, hue='survived')
```

Out[ ]:

```
<AxesSubplot:xlabel='sex', ylabel='age'>
```



Now you can see a lot of information on the violin plot. For instance, if you look at the bottom of the violin plot for the males who survived (left-orange), you can see that it is thicker than the bottom of the violin plot for the males who didn't survive (left-blue). This means that the number of young male passengers who survived is greater than the number of young male passengers who did not survive. The violin plots convey a lot of information, however, on the downside, it takes a bit of time and effort to understand the violin plots.

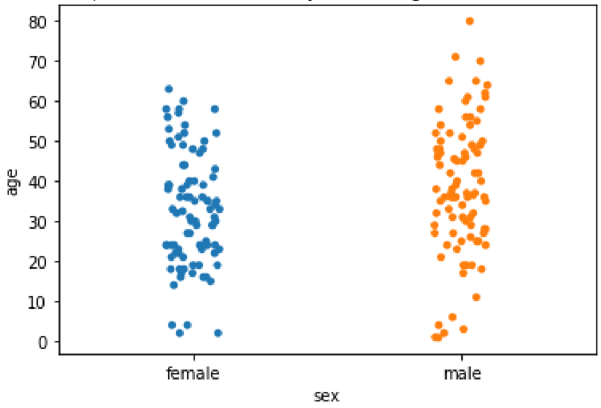
The Strip Plot

The strip plot draws a scatter plot where one of the variables is categorical. We have seen scatter plots in the joint plot and the pair plot sections where we had two numeric variables. The strip plot is different in a way that one of the variables is categorical in this case, and for each category in the categorical variable, you will see scatter plot with respect to the numeric column.

The stripplot() function is used to plot the violin plot.

In [ ]:
```
sns.stripplot(x='sex', y='age', data=dataset)
```

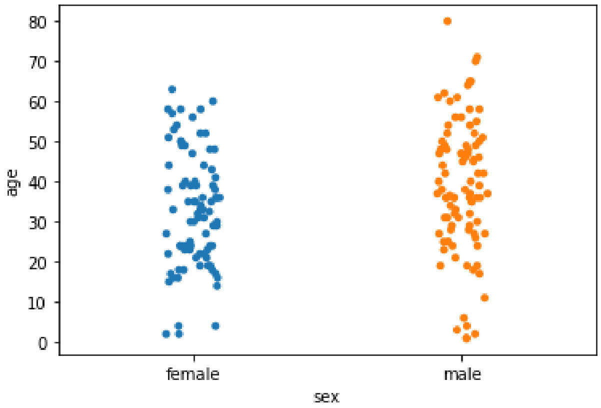
Out[ ]: <AxesSubplot:xlabel='sex', ylabel='age'>



see the scattered plots of age for both males and females. The data points look like strips. It is difficult to comprehend the distribution of data in this form.

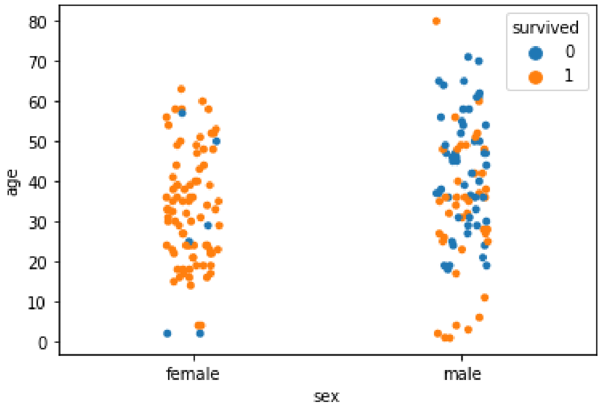
In [ ]:
```
sns.stripplot(x='sex', y='age', data=dataset, jitter=True)
#pass True for the jitter parameter which adds some random noise to the data.
```

Out[ ]: <AxesSubplot:xlabel='sex', ylabel='age'>



In [ ]:
```
# Like violin and box plots, you can add an additional categorical column to strip plot using hue parameter as

sns.stripplot(x='sex', y='age', data=dataset, jitter=True, hue='survived')
```

Out[ ]: <AxesSubplot:xlabel='sex', ylabel='age'>



Seaborn is an advanced data visualization library built on top of Matplotlib library. In this above all code, we looked at how we can draw distributional and categorical plots using Seaborn library.