COMP-1832

Programming Fundamentals for Data Science

Coursework

Author: Sweejalben Nehal Surti

Student ID: 001127898

Course of Study: MSc Data Science

Course Leader:

Dr. Jia Wang

&

Dr. Konstantin Kapinchev

School of Computing and Mathematical Science

COURSEWORK_PYTHON

December 4, 2021

1 Python Programming

[2]:
```
#imports library

import numpy as np
import pandas as pd
import calendar
import matplotlib.pyplot as plt
from scipy.stats import norm
import networkx as nx
```

2 Portfolio 1

[5]:
```
#1. Download the Dataset.csv file from Moodle, load its content into pandas data␣
↪frame and visualise the entire content of the data frame

data = pd.read_csv("Dataset.csv")
print(data.to_string())
print(data.dtypes.value_counts)
```

```
    longitude  latitude  housing_median_age  total_rooms  population
median_income  median_house_value ocean_proximity
0      -122.23     37.88             41.0000          880          322
8.3252              452600        NEAR BAY
1      -122.23     37.88             41.0000          880          322
8.3252              452600        NEAR BAY
2      -122.22     37.86             21.0000         7099         2401
8.3014              358500        NEAR BAY
3      -122.25     37.84             52.0001         3104         1157
3.1200              241400        NEAR BAY
4      -122.26     37.85             52.0000         3503         1504
3.2705              241800        NEAR BAY
5      -121.65     39.32             40.0000          812          374
2.7891               73500          INLAND
6      -121.69     39.36             29.0000         2220         1170
2.3224               56200          INLAND
7      -121.70     39.37             32.0000         1852          911
```

```
1.7885              57000          INLAND
8     -121.70     39.36            46.0000          1210          523
1.9100              63900          INLAND
9     -121.70     39.36            37.0000          2330         1505
2.0474              56000          INLAND
10    -121.69     39.36            34.0000           842          635
1.8355              63000          INLAND
11    -121.74     39.38            27.0000          2596         1100
2.3243              85500             NaN
12    -121.80     39.33            30.0000          1019          501
2.5259              81300          INLAND
13    -120.46     38.15            16.0000          4221         1516
2.3816             116000          INLAND
14    -120.55     38.12            10.0000          1566          785
2.5000             116100          INLAND
15    -120.56     38.09            34.0000          2745         1150
2.3654              94900          INLAND
16    -124.23     41.75            11.0000          3159         1343
2.4805              73200      NEAR OCEAN
17    -124.21     41.77            17.0000          3461         1947
2.5795              68400          NEAR O
18    -124.19     41.78            15.0000          3140         1645
1.6654              74600          NEAR O
19    -124.16     41.74            15.0000          2715         1532
2.1829              69500      NEAR OCEAN
20    -124.14     41.95            21.0000          2696         1208
NaN             122400      NEAR OCEAN
21    -124.16     41.92            19.0000          1668          841
2.1336              75000      NEAR OCEAN
22    -118.32     33.35            27.0000          1675          744
2.1579             450000          ISLAND
23    -118.33     33.34            52.0000          2359         1100
2.8333             414700          ISLAND
24    -118.32     33.33            52.0000          2127          733
3.3906             300000          ISLAND
25    -118.32     33.34            52.0000           996          341
2.7361             450000          ISLAND
26    -118.48     33.43            29.0000           716          422
2.6042             287500          ISLAND
27    -118.48     33.43            29.0000           716          422
2.6042             287500          ISLAND
<bound method IndexOpsMixin.value_counts of longitude             float64
latitude              float64
housing_median_age    float64
total_rooms             int64
population              int64
median_income         float64
median_house_value      int64
```

```
ocean_proximity        object
dtype: object>
```

The above code download the dataset and show all data in dataframe structure i.e. comma separated values with all columns class types

[6]:
```
#Check the data frame for the following 'data cleaning' issues and resolve them:
#2. Missing values

data.isnull().sum()

#this code check for missing values in dataset.
```

[6]:
```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
population            0
median_income         1
median_house_value    0
ocean_proximity       1
dtype: int64
```

#From above result it can be notice there are misssing values in column 'median_income' and 'ocean_proximity'

[7]:
```
#fill missing values


data.fillna(data.mode().iloc[0], inplace=True)
data
```

[7]:
```
    longitude  latitude  housing_median_age  total_rooms  population  \
0     -122.23     37.88             41.0000          880         322
1     -122.23     37.88             41.0000          880         322
2     -122.22     37.86             21.0000         7099        2401
3     -122.25     37.84             52.0001         3104        1157
4     -122.26     37.85             52.0000         3503        1504
5     -121.65     39.32             40.0000          812         374
6     -121.69     39.36             29.0000         2220        1170
7     -121.70     39.37             32.0000         1852         911
8     -121.70     39.36             46.0000         1210         523
9     -121.70     39.36             37.0000         2330        1505
10    -121.69     39.36             34.0000          842         635
11    -121.74     39.38             27.0000         2596        1100
12    -121.80     39.33             30.0000         1019         501
13    -120.46     38.15             16.0000         4221        1516
14    -120.55     38.12             10.0000         1566         785
15    -120.56     38.09             34.0000         2745        1150
```

16	-124.23	41.75	11.0000	3159	1343
17	-124.21	41.77	17.0000	3461	1947
18	-124.19	41.78	15.0000	3140	1645
19	-124.16	41.74	15.0000	2715	1532
20	-124.14	41.95	21.0000	2696	1208
21	-124.16	41.92	19.0000	1668	841
22	-118.32	33.35	27.0000	1675	744
23	-118.33	33.34	52.0000	2359	1100
24	-118.32	33.33	52.0000	2127	733
25	-118.32	33.34	52.0000	996	341
26	-118.48	33.43	29.0000	716	422
27	-118.48	33.43	29.0000	716	422

	median_income	median_house_value	ocean_proximity
0	8.3252	452600	NEAR BAY
1	8.3252	452600	NEAR BAY
2	8.3014	358500	NEAR BAY
3	3.1200	241400	NEAR BAY
4	3.2705	241800	NEAR BAY
5	2.7891	73500	INLAND
6	2.3224	56200	INLAND
7	1.7885	57000	INLAND
8	1.9100	63900	INLAND
9	2.0474	56000	INLAND
10	1.8355	63000	INLAND
11	2.3243	85500	INLAND
12	2.5259	81300	INLAND
13	2.3816	116000	INLAND
14	2.5000	116100	INLAND
15	2.3654	94900	INLAND
16	2.4805	73200	NEAR OCEAN
17	2.5795	68400	NEAR O
18	1.6654	74600	NEAR O
19	2.1829	69500	NEAR OCEAN
20	2.6042	122400	NEAR OCEAN
21	2.1336	75000	NEAR OCEAN
22	2.1579	450000	ISLAND
23	2.8333	414700	ISLAND
24	3.3906	300000	ISLAND
25	2.7361	450000	ISLAND
26	2.6042	287500	ISLAND
27	2.6042	287500	ISLAND

It can be notice from the dataset that the median_income and ocean_proximity column has one missing value which has been fill by most frequent data of that column.

[8]:
```
#3. Unnecessary duplicates
print(data.duplicated())

data = data.drop_duplicates().reset_index(drop=True)

print(data.to_string())
```

```
0     False
1      True
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12    False
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
21    False
22    False
23    False
24    False
25    False
26    False
27     True
dtype: bool
    longitude  latitude  housing_median_age  total_rooms  population
median_income  median_house_value ocean_proximity
0     -122.23     37.88             41.0000          880         322
8.3252              452600        NEAR BAY
1     -122.22     37.86             21.0000         7099        2401
8.3014              358500        NEAR BAY
2     -122.25     37.84             52.0001         3104        1157
3.1200              241400        NEAR BAY
3     -122.26     37.85             52.0000         3503        1504
3.2705              241800        NEAR BAY
4     -121.65     39.32             40.0000          812         374
2.7891               73500          INLAND
```

```
5      -121.69     39.36               29.0000          2220          1170
2.3224                56200            INLAND
6      -121.70     39.37               32.0000          1852           911
1.7885                57000            INLAND
7      -121.70     39.36               46.0000          1210           523
1.9100                63900            INLAND
8      -121.70     39.36               37.0000          2330          1505
2.0474                56000            INLAND
9      -121.69     39.36               34.0000           842           635
1.8355                63000            INLAND
10     -121.74     39.38               27.0000          2596          1100
2.3243                85500            INLAND
11     -121.80     39.33               30.0000          1019           501
2.5259                81300            INLAND
12     -120.46     38.15               16.0000          4221          1516
2.3816               116000            INLAND
13     -120.55     38.12               10.0000          1566           785
2.5000               116100            INLAND
14     -120.56     38.09               34.0000          2745          1150
2.3654                94900            INLAND
15     -124.23     41.75               11.0000          3159          1343
2.4805                73200        NEAR OCEAN
16     -124.21     41.77               17.0000          3461          1947
2.5795                68400            NEAR O
17     -124.19     41.78               15.0000          3140          1645
1.6654                74600            NEAR O
18     -124.16     41.74               15.0000          2715          1532
2.1829                69500        NEAR OCEAN
19     -124.14     41.95               21.0000          2696          1208
2.6042               122400        NEAR OCEAN
20     -124.16     41.92               19.0000          1668           841
2.1336                75000        NEAR OCEAN
21     -118.32     33.35               27.0000          1675           744
2.1579               450000            ISLAND
22     -118.33     33.34               52.0000          2359          1100
2.8333               414700            ISLAND
23     -118.32     33.33               52.0000          2127           733
3.3906               300000            ISLAND
24     -118.32     33.34               52.0000           996           341
2.7361               450000            ISLAND
25     -118.48     33.43               29.0000           716           422
2.6042               287500            ISLAND
```

It can be notice from the dataset that row 1 and 27 has duplicates value which returns TRUE for that row that confirmed this value as duplicates.That 2 row has been dropped by the function duplicates()

```
[9]: #4. Wrong data types

#housing_median_age has wrong data type which should be integer instead of float

data= data.astype({'housing_median_age': np.int64})
print(data.to_string())
```

```
    longitude  latitude  housing_median_age  total_rooms  population
median_income  median_house_value ocean_proximity
0      -122.23     37.88                  41          880          322
8.3252              452600        NEAR BAY
1      -122.22     37.86                  21         7099         2401
8.3014              358500        NEAR BAY
2      -122.25     37.84                  52         3104         1157
3.1200              241400        NEAR BAY
3      -122.26     37.85                  52         3503         1504
3.2705              241800        NEAR BAY
4      -121.65     39.32                  40          812          374
2.7891               73500          INLAND
5      -121.69     39.36                  29         2220         1170
2.3224               56200          INLAND
6      -121.70     39.37                  32         1852          911
1.7885               57000          INLAND
7      -121.70     39.36                  46         1210          523
1.9100               63900          INLAND
8      -121.70     39.36                  37         2330         1505
2.0474               56000          INLAND
9      -121.69     39.36                  34          842          635
1.8355               63000          INLAND
10     -121.74     39.38                  27         2596         1100
2.3243               85500          INLAND
11     -121.80     39.33                  30         1019          501
2.5259               81300          INLAND
12     -120.46     38.15                  16         4221         1516
2.3816              116000          INLAND
13     -120.55     38.12                  10         1566          785
2.5000              116100          INLAND
14     -120.56     38.09                  34         2745         1150
2.3654               94900          INLAND
15     -124.23     41.75                  11         3159         1343
2.4805               73200      NEAR OCEAN
16     -124.21     41.77                  17         3461         1947
2.5795               68400          NEAR O
17     -124.19     41.78                  15         3140         1645
1.6654               74600          NEAR O
18     -124.16     41.74                  15         2715         1532
2.1829               69500      NEAR OCEAN
19     -124.14     41.95                  21         2696         1208
```

```
2.6042              122400      NEAR OCEAN
20     -124.16      41.92                 19          1668          841
2.1336               75000      NEAR OCEAN
21     -118.32      33.35                 27          1675          744
2.1579              450000          ISLAND
22     -118.33      33.34                 52          2359         1100
2.8333              414700          ISLAND
23     -118.32      33.33                 52          2127          733
3.3906              300000          ISLAND
24     -118.32      33.34                 52           996          341
2.7361              450000          ISLAND
25     -118.48      33.43                 29           716          422
2.6042              287500          ISLAND
```

We can notice from the dataset that the housing_median_age column type is Float which should be an integer value which has been converted by the function np.int64()

[10]:
```
#5. Wrong values
```

From above dataset we can notice that column median_income has wrong values. The income should be in large number. Also the ocean_proximity has wrong values in row no. 16,17, which should be as 'NEAR OCEAN' instead of 'NEAR O'.

[11]:
```
#6. Save the updated data frame into a new CSV file

data.to_csv("New_Dataset.csv")

# The above code is to save the existing data into new csv file.
```

[12]:
```
#The house prices are at the focus of this data frame. By using the updated data␣
↪frame,provide the following values, which describe the column␣
↪'median_house_value':
#7. Mean

print(data['median_house_value'].mean())
```

```
174730.76923076922
```

The above code is to calculate the average price of the house..which is 174730.76923076922 as an average price of the house.

[13]:
```
#8. Median

print(data['median_house_value'].median())
```

```
90200.0
```

The baove code show the median of the house value that is 90200.0

[14]:
```
#9. Range

print("The range of the median_house_value is:", data['median_house_value'].
 ↪max() - data['median_house_value'].min())
```

```
The range of the median_house_value is: 396600
```

The above code find the range of that data which can be find by its Max value - Min value of that column.

[15]:
```
#10. The column 'median_income' contains currency in tens of thousands USD.
#Convert it intro USD and visualise the entire updated data frame

data['median_income']=data['median_income'] * 10000

data
```

[15]:
```
    longitude  latitude  housing_median_age  total_rooms  population  \
0     -122.23     37.88                  41          880         322
1     -122.22     37.86                  21         7099        2401
2     -122.25     37.84                  52         3104        1157
3     -122.26     37.85                  52         3503        1504
4     -121.65     39.32                  40          812         374
5     -121.69     39.36                  29         2220        1170
6     -121.70     39.37                  32         1852         911
7     -121.70     39.36                  46         1210         523
8     -121.70     39.36                  37         2330        1505
9     -121.69     39.36                  34          842         635
10    -121.74     39.38                  27         2596        1100
11    -121.80     39.33                  30         1019         501
12    -120.46     38.15                  16         4221        1516
13    -120.55     38.12                  10         1566         785
14    -120.56     38.09                  34         2745        1150
15    -124.23     41.75                  11         3159        1343
16    -124.21     41.77                  17         3461        1947
17    -124.19     41.78                  15         3140        1645
18    -124.16     41.74                  15         2715        1532
19    -124.14     41.95                  21         2696        1208
20    -124.16     41.92                  19         1668         841
21    -118.32     33.35                  27         1675         744
22    -118.33     33.34                  52         2359        1100
23    -118.32     33.33                  52         2127         733
24    -118.32     33.34                  52          996         341
25    -118.48     33.43                  29          716         422

    median_income  median_house_value ocean_proximity
0         83252.0              452600        NEAR BAY
1         83014.0              358500        NEAR BAY
```

```
2          31200.0                 241400        NEAR BAY
3          32705.0                 241800        NEAR BAY
4          27891.0                  73500          INLAND
5          23224.0                  56200          INLAND
6          17885.0                  57000          INLAND
7          19100.0                  63900          INLAND
8          20474.0                  56000          INLAND
9          18355.0                  63000          INLAND
10         23243.0                  85500          INLAND
11         25259.0                  81300          INLAND
12         23816.0                 116000          INLAND
13         25000.0                 116100          INLAND
14         23654.0                  94900          INLAND
15         24805.0                  73200      NEAR OCEAN
16         25795.0                  68400          NEAR O
17         16654.0                  74600          NEAR O
18         21829.0                  69500      NEAR OCEAN
19         26042.0                 122400      NEAR OCEAN
20         21336.0                  75000      NEAR OCEAN
21         21579.0                 450000          ISLAND
22         28333.0                 414700          ISLAND
23         33906.0                 300000          ISLAND
24         27361.0                 450000          ISLAND
25         26042.0                 287500          ISLAND
```

3 Portfolio 2

3.0.1 Part 1

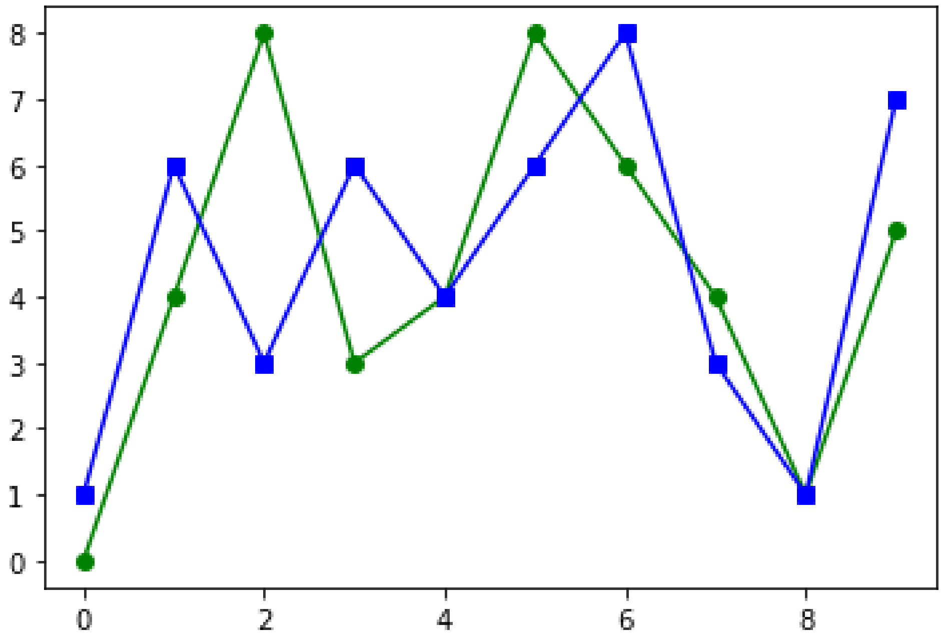
3.0.2 Initialise a two-dimensional array consisting of 5 rows and 10 columns of uniformly distributed data points of integer values from the interval [0..9]. Consider each row of the two-dimensional array as an independent dataset. Display the values on the screen as a table. Plot the first two rows on a single diagram with different colours.

[16]:

```
My_arr = np.empty(shape=(5,10), dtype='int')

for i in range(0,5):
    for j in range(0,10):
        My_arr[i,j] = int(np.random.randint(0,9))

print(My_arr)

plt.plot(My_arr[0,] , marker = "o", color ="green")
plt.plot(My_arr[1,] , marker = "s", color ="blue")
plt.show()
```

```
[[0 4 8 3 4 8 6 4 1 5]
 [1 6 3 6 4 6 8 3 1 7]
```

```
[3 6 7 0 4 2 4 4 1 3]
[3 1 1 8 8 4 8 4 3 6]
[3 6 8 2 7 1 7 3 6 1]]
```



The above code generate the two-dimensional array consisting of 5 rows and 10 columns of uniformly distributed data points of integer values using np.random.randint() function. where plt.plot() function visualize the data as graph.

3.0.3 Part 2

3.0.4 Provide the following information about each individual row:

3.0.5 Mean

3.0.6 Median

3.0.7 Standard deviation

[17]:
```
row_mean = np.mean(My_arr, axis =1)
print("Mean By its row:", row_mean)

row_median = np.median(My_arr, axis =1)
print("Median By its row:", row_median)

row_std = np.std(My_arr, axis =1)
print("Standard Deviation By its row:", row_std)
```

```
# This code find the mean, meadian and Standard Deviation of each individual␣
↪rows.
```

```
Mean By its row: [4.3 4.5 3.4 4.6 4.4]
Median By its row: [4.  5.  3.5 4.  4.5]
Standard Deviation By its row: [2.49198716 2.33452351 2.00997512 2.61533937
2.53771551]
```

3.0.8 Part 3

3.0.9 Initialise a one-dimensional array representing a normal distribution of 1000 data points with mean value 17 and standard deviation 0.2.

[18]:
```
mean = float(17)
std = float(0.2)
size = int(1000)
norm_array = np.random.normal(mean,std, size)
norm_array

# This code represents 1000 normally distributed data with its mean value 17 and␣
↪0.2 sd value
#using np.random.normal() function.
```

[18]:
```
array([17.0871021 , 17.11168963, 16.94467448, 16.4896596 , 16.95476746,
       17.06567924, 17.17335808, 16.7951687 , 16.91907479, 17.1643116 ,
       17.18000543, 17.01106869, 17.0392014 , 16.59103135, 17.13666795,
       16.93715652, 16.87004571, 17.02680475, 17.32366372, 17.10914536,
       16.98544317, 16.69948372, 17.29080071, 17.19402319, 17.24797148,
       17.03986578, 17.29477632, 16.66960589, 16.89329687, 17.10539742,
       17.02425186, 17.23456934, 17.01014785, 17.06396743, 17.00411319,
       16.88756041, 17.44352724, 17.11427959, 16.8673481 , 17.06808373,
       16.8831875 , 17.20674813, 17.10504338, 16.92375556, 16.91714368,
       16.80447175, 16.95268854, 17.15319853, 17.02395346, 16.8188284 ,
       16.92513867, 17.18699766, 17.02696755, 16.77651585, 16.70020168,
       17.38794735, 17.09354312, 17.09739948, 16.97573832, 17.22058579,
       16.74905093, 16.56631596, 16.83290987, 17.26633769, 16.98950945,
       17.05860926, 16.8204387 , 17.14699334, 16.85682185, 17.09290906,
       16.81883683, 17.07050164, 16.86736209, 17.24620704, 17.01713886,
       17.0005637 , 16.7924739 , 16.83954382, 17.07289772, 16.86532834,
       17.31447812, 17.0727732 , 17.08231494, 17.02463888, 16.99143152,
       17.08719624, 17.14764329, 17.05833099, 16.89422403, 16.95774643,
       17.26072463, 17.02790852, 17.0969117 , 16.97463294, 17.21697948,
       17.11829534, 17.17171107, 16.8231039 , 16.90700394, 17.22403978,
       17.08849423, 16.96379442, 17.19300335, 17.53209023, 17.26675625,
       16.63121711, 17.05093245, 17.10449294, 17.01479268, 17.1887665 ,
       17.03844839, 17.06427179, 17.23253528, 16.90747633, 16.85032941,
       16.58846995, 17.03453196, 16.85258579, 17.25193051, 17.1509691 ,
       16.93583732, 17.18558992, 17.05179778, 17.24642586, 16.87496581,
```

17.01876184, 16.76016962, 16.84787505, 16.62248695, 17.08899695,
16.88138056, 17.27993895, 16.79161463, 16.86304199, 16.68506745,
17.06940183, 16.54082787, 17.36471743, 17.1130859 , 16.73405412,
17.04836522, 17.09558439, 17.37753412, 16.3047535 , 17.25442236,
17.00833729, 17.2813321 , 17.02916072, 16.95655001, 16.96556807,
17.12001957, 16.98900228, 17.14103727, 17.19493209, 17.04339813,
16.9682588 , 16.83761681, 16.80996335, 16.77112869, 16.85091437,
16.57369657, 16.66997454, 16.76528644, 17.0831637 , 16.77015311,
17.06081636, 16.84383629, 17.07054635, 16.89896602, 17.15796042,
17.32121117, 17.07293838, 16.99209081, 17.33826773, 17.16697236,
16.97303699, 16.95737729, 16.89811781, 17.1400396 , 17.03198805,
17.4326771 , 17.02159795, 17.26056899, 17.36377224, 16.81623061,
16.63141107, 16.84180498, 17.08166051, 16.85297174, 16.97385609,
17.17690146, 17.39392037, 17.13495342, 17.17263922, 17.095757 ,
16.92279581, 16.57265445, 17.35701406, 16.74115636, 17.28132534,
16.78684841, 16.98123705, 17.18401673, 16.92815912, 16.98921192,
16.88413035, 17.31147289, 16.85391539, 16.99710927, 16.87927108,
16.86867272, 17.00506548, 16.97239156, 16.65560534, 16.85708268,
17.08315371, 16.75007741, 16.92904762, 17.17874049, 16.77178002,
16.73898853, 16.74500984, 17.17072804, 16.91581764, 16.94722422,
16.85015378, 16.84360266, 16.89822629, 17.17538083, 16.91870833,
16.61393145, 16.77649407, 16.91305063, 17.35308936, 17.07171962,
17.0338043 , 17.09277638, 17.19062638, 16.70581859, 16.89875703,
16.65977444, 16.98176876, 16.97479906, 17.09043747, 16.98991103,
17.20978232, 17.02881081, 17.13177579, 16.87954644, 17.03450292,
17.3314627 , 17.32057606, 16.97832425, 17.00240832, 17.48022004,
16.85289057, 17.15678176, 16.96308043, 16.97497726, 17.48521196,
17.07623069, 16.91869127, 16.91310712, 17.06349289, 17.05560457,
17.24810957, 16.74589922, 16.96637097, 17.06863117, 17.01676594,
16.97646644, 17.64330057, 16.8809299 , 17.21902863, 16.8117574 ,
16.71010913, 17.31199186, 16.95538194, 17.20559871, 17.12506202,
17.16564741, 17.11614259, 17.06195758, 17.21349204, 16.88671261,
17.126247 , 16.74844092, 16.80362499, 16.7281345 , 17.04168842,
17.39903817, 16.70725687, 17.00519494, 17.23758694, 16.88363217,
16.79862461, 17.10257875, 17.14033806, 17.27011355, 17.08960661,
16.67101228, 17.24553406, 16.96793299, 17.29078326, 17.02023922,
17.07282967, 17.27432187, 17.09803154, 17.30743341, 16.83019166,
17.05633429, 17.06453818, 17.05004635, 17.0286393 , 17.14408445,
16.80685992, 16.91366161, 16.7160819 , 16.86050876, 16.94289966,
17.27629367, 17.19726841, 16.93227997, 16.84297958, 16.86871134,
16.76381369, 16.47346569, 16.93242973, 16.92818805, 17.11522845,
16.77001698, 16.93497979, 17.27382479, 16.71036307, 16.84721538,
16.48114629, 17.05895385, 16.9285357 , 17.03869522, 16.9797574 ,
16.83449946, 17.06632381, 16.88288106, 16.85442774, 17.27822359,
16.85101851, 16.99124586, 16.89159576, 17.42607425, 16.91302839,
16.99106196, 16.79389608, 17.16649863, 16.97557152, 16.89854355,
17.05486989, 17.03523142, 17.19468506, 17.07287451, 17.1905243 ,

16.99944373, 16.80449277, 16.83278781, 16.93020085, 16.92600206,
17.03384304, 17.34032038, 16.73463849, 17.04692571, 17.21833742,
16.85867064, 17.14092235, 16.87981131, 17.01491885, 17.32250173,
17.34566005, 17.10978567, 17.07633421, 16.73286478, 16.96685802,
16.97307978, 16.72769574, 16.7699041 , 16.79484741, 17.17190562,
17.20742087, 17.03335607, 16.99661785, 17.24671431, 17.11688337,
16.99680907, 16.97466717, 17.07169269, 17.21757271, 17.04241237,
16.98532726, 16.96726361, 16.99587737, 16.97597255, 17.027097 ,
17.11240661, 16.93110885, 17.06017294, 17.16639805, 17.00322924,
17.06244733, 17.2397332 , 16.81379447, 16.95222193, 17.10775715,
17.2316075 , 16.84793781, 17.10059813, 17.21002005, 17.17674973,
17.27232375, 16.68282281, 16.66786976, 17.03270957, 16.7823333 ,
17.17095595, 17.18269907, 16.81570051, 17.25589285, 17.13080517,
16.76238666, 16.94256662, 16.9847868 , 16.90372136, 16.8343269 ,
17.17133821, 17.04579951, 16.83716646, 17.34330914, 16.8920698 ,
17.24365414, 17.1956349 , 16.95577734, 16.95289665, 16.93207337,
16.92225421, 17.12066396, 17.02009611, 16.79014315, 16.90678595,
16.96719557, 16.86034322, 16.95157467, 17.00797235, 17.33032586,
16.95267297, 16.82748735, 17.09759048, 17.05315271, 17.38495279,
16.95664969, 17.13447194, 17.38192329, 16.76829538, 17.1733438 ,
17.40512831, 17.1046592 , 17.07685834, 17.01327833, 16.76704772,
16.9016975 , 16.85377866, 16.71149575, 17.4796177 , 17.12941085,
16.88313308, 17.05669699, 17.02441329, 16.84429367, 16.84250705,
17.24616495, 16.97781493, 16.54931986, 16.77393326, 17.00415797,
16.98130101, 16.91344251, 17.34416445, 16.93714236, 16.70495567,
17.46928548, 16.88869828, 16.86619906, 17.22901323, 16.89848603,
16.59140242, 16.64258538, 16.74538034, 17.18673442, 16.92062214,
17.2232917 , 16.87954024, 17.09111962, 17.01727635, 16.86461898,
16.98959983, 16.73895912, 17.07663837, 17.081587 , 17.13248722,
17.30011702, 17.74613317, 17.08659209, 16.54060377, 17.00931284,
16.72685796, 17.02672841, 16.96274095, 17.23402272, 16.90336752,
16.80846906, 17.23860891, 17.50852385, 16.96046898, 16.84934625,
16.90266804, 16.65091487, 16.79825849, 17.08337478, 16.960757 ,
16.96315973, 17.20761195, 16.92449528, 16.9726497 , 16.99255282,
17.1938683 , 17.05114446, 16.82861775, 16.84630819, 16.8871493 ,
17.19523347, 17.02448867, 16.98378976, 17.00406945, 16.8253051 ,
16.9483232 , 17.3261083 , 17.42901108, 17.06923059, 17.19164582,
17.37536905, 17.09434624, 17.09362463, 16.79120658, 17.14645801,
16.90114575, 17.12943606, 16.83337851, 16.89426052, 16.95572727,
17.36447407, 16.8737966 , 17.10516326, 16.88881866, 16.96769239,
16.63866217, 16.98759322, 17.16753462, 16.73438146, 17.20365204,
17.23844137, 17.11376551, 16.99169414, 17.1135091 , 17.12377164,
17.19181531, 17.34571622, 17.11025025, 16.87899296, 17.1071928 ,
16.80048074, 17.17027833, 17.20421415, 17.0553907 , 17.08924954,
17.10384884, 17.21956301, 17.21478426, 17.27663676, 16.83782446,
16.9170498 , 16.88708824, 16.96642985, 17.01484175, 16.88156577,
17.0030851 , 16.95879819, 16.78575395, 17.18875194, 17.02672629,

```
16.64194505, 16.97390048, 16.82755559, 16.96058157, 17.12452433,
17.25757005, 16.85967945, 17.10377886, 16.74661992, 17.18839546,
16.98744341, 17.39456792, 16.76598346, 17.02971299, 16.98128241,
16.82868227, 16.782239  , 16.60436532, 16.86966767, 16.93331871,
16.73085275, 16.79516687, 16.85663769, 16.97580048, 17.37473405,
16.81098373, 17.29890242, 16.85797936, 16.90469345, 16.86173778,
16.81550164, 16.91101932, 16.98090993, 16.76381872, 16.86897861,
16.75500041, 16.69082812, 17.22408965, 17.12636543, 16.99472073,
17.34737156, 16.94824706, 17.20846778, 16.82514039, 17.1696851 ,
17.11981604, 17.09041056, 17.00007794, 17.09997292, 16.91704093,
16.78689802, 17.05875302, 17.04973822, 17.40754813, 16.94701318,
16.8848001 , 17.28777971, 17.06491415, 17.0132937 , 17.13262051,
17.49059567, 16.85806405, 17.21330979, 16.74457336, 16.96426438,
16.91921888, 16.69834201, 16.78230596, 16.59861587, 17.0603541 ,
17.24532225, 17.13546265, 16.95791634, 16.84573467, 16.57669411,
17.23505742, 16.93780554, 16.81883459, 16.93600869, 16.8627294 ,
17.16100369, 16.99353594, 17.32733492, 16.76103006, 17.00904003,
17.29337853, 16.92267024, 16.86567405, 17.04089918, 17.05737348,
16.94032053, 16.88003865, 17.17771863, 17.10731256, 17.00395725,
17.00082153, 17.14393913, 16.89308447, 16.95850986, 16.83129705,
16.6841111 , 16.87650694, 17.19754278, 17.16567435, 17.0044626 ,
17.27958134, 16.94434594, 16.93575396, 16.93549038, 16.84469841,
17.01802707, 16.6425294 , 16.81430267, 17.13915078, 16.93349289,
16.75516253, 17.05414939, 17.06148566, 17.28072456, 17.05030592,
16.94289829, 16.96605361, 16.96171429, 16.93183783, 17.06043563,
16.88714   , 16.71415556, 17.34401315, 17.02916367, 16.69972944,
16.93445907, 16.8454594 , 16.54754045, 17.02490163, 17.0631384 ,
17.08256759, 16.98198949, 17.00991817, 17.23012272, 17.18708674,
16.86298247, 17.17425369, 17.04430136, 16.79487456, 16.74000569,
16.95717913, 16.6888197 , 16.96525916, 17.3445201 , 16.82798005,
17.17582521, 17.03053868, 17.21548101, 16.97501587, 16.76272883,
17.07502127, 16.69603914, 17.11770649, 16.54362446, 16.90569892,
17.1964836 , 17.01421042, 17.15853756, 17.10497158, 16.80374771,
17.21172221, 16.55604671, 16.98927839, 17.08279814, 17.11821798,
16.83200999, 17.2289492 , 16.99199146, 17.32612999, 16.51099364,
16.93114227, 16.88046342, 16.83721403, 16.8473788 , 17.20399105,
16.89971402, 16.72703244, 17.07191142, 17.19474086, 17.20692533,
17.19957971, 16.85371874, 16.63790937, 16.85944777, 17.27811408,
16.92783534, 17.04694248, 16.93293716, 17.06517992, 17.00382864,
17.09771436, 16.65670799, 17.10962741, 16.95737722, 17.06685537,
16.9784586 , 17.00109718, 16.82550141, 17.27336228, 16.85342134,
17.26332537, 17.12944886, 16.88087533, 16.96850505, 17.01160292,
16.92065022, 17.15109711, 16.90710918, 16.80267196, 17.07506539,
17.02446751, 16.76985395, 16.78416286, 17.12967547, 16.89559053,
17.19361502, 17.27108156, 17.0043217 , 16.796008  , 16.7974345 ,
17.17234208, 16.81718383, 17.29691849, 16.79296394, 17.31713979,
16.94227121, 16.97889025, 17.38714219, 17.04971647, 17.17534798,
```

```
17.18242677, 16.87779983, 17.1646998 , 16.98367641, 17.0847319 ,
16.72898733, 16.83396185, 16.97797797, 16.94345655, 16.87616145,
17.04642455, 16.94049581, 16.83857511, 16.43868869, 17.14004766,
16.93586751, 16.82711972, 16.95025948, 17.00493603, 16.92561354,
16.81484952, 16.73457267, 16.92937634, 17.1973348 , 16.77731167,
17.16773373, 17.20536366, 17.12719632, 16.83867001, 16.72455727,
17.47762048, 17.20979336, 17.2102932 , 17.4095791 , 17.13224638,
17.05135516, 16.62855893, 17.17711174, 16.71069633, 16.93285961,
16.65298256, 16.65576471, 16.8400396 , 16.56191472, 16.80629858,
17.12135027, 16.74329593, 17.36098369, 17.20022874, 16.6376544 ,
16.90673524, 17.21433879, 17.21450323, 17.13379987, 17.02501604,
16.8680352 , 16.95781816, 16.94253805, 17.06658364, 16.67124587,
17.04358793, 17.02347247, 17.0909941 , 17.03970977, 16.9710721 ,
17.10969294, 16.90568685, 17.17004159, 17.06147354, 17.25295724,
16.83005984, 17.00540568, 16.92761069, 16.81518194, 16.86661849,
17.13114053, 17.05908618, 16.73089603, 16.69676875, 16.7536834 ,
17.24660716, 16.92573589, 17.02117265, 17.00712351, 17.09411461,
17.31279555, 17.25387602, 16.84554709, 16.92005015, 17.15717627,
17.29406874, 17.17870898, 17.32335584, 17.19176775, 16.78917576,
16.5959618 , 17.37509984, 16.9505088 , 17.18634627, 16.75749451,
16.91265389, 16.88205949, 16.81673387, 17.01146871, 16.72569015,
17.00173771, 16.92465372, 16.79806707, 17.10172157, 17.15011894,
16.99575949, 16.88103069, 17.08960643, 17.04211538, 17.02180548,
16.99650496, 16.91095905, 16.99626111, 16.91375668, 16.5431922 ,
17.17730138, 16.93034734, 17.06799828, 17.18508633, 17.3766111 ,
17.43597271, 16.88795565, 16.85000319, 16.75325955, 17.08245439,
17.04770094, 16.79470935, 16.93201491, 16.8917455 , 17.05949328,
17.24072714, 16.72237777, 16.85163948, 17.05170608, 16.95802344,
17.03577655, 17.10602765, 17.11049152, 16.67338261, 17.18592262,
16.96741021, 16.93910632, 16.85223359, 16.76595325, 17.16496138,
17.12245535, 16.99069928, 17.05271393, 17.2190864 , 16.96177382,
17.16255832, 16.59707838, 16.78837975, 16.96228402, 17.06367125,
16.97194593, 16.7468086 , 16.89275522, 16.66783309, 17.08288435,
16.85791867, 17.0622175 , 17.0148367 , 17.23322922, 17.07695526])
```

3.0.10 Part 4

3.0.11 Find the maximum and the minimum values of the dataset and calculate the range.

[19]:
```python
print("Max:", np.max(norm_array))

print("Min:", np.min(norm_array))

print("Range:", round(np.max(norm_array) - np.min(norm_array),2))

# The above code finds the Max, min and range of the data.
```

```
Max: 17.74613317359515
```

```
Min: 16.30475350250239
Range: 1.44
```

3.0.12 Part 5 (2 points):

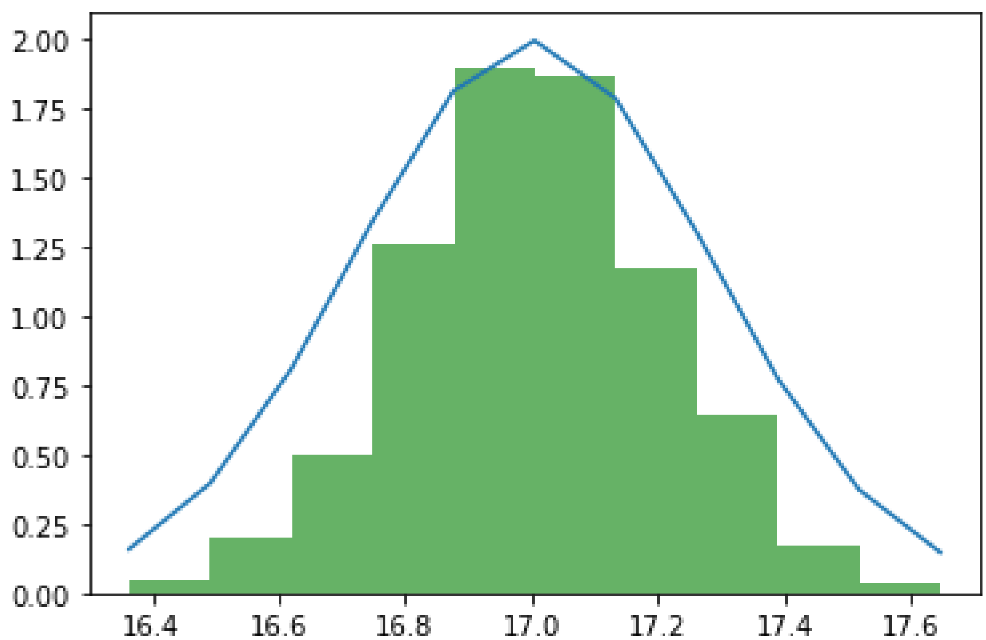
3.0.13 Visualise the dataset by using a histogram with 10 bins. Visualise the probability density function.

3.0.14 Probability density function f:

3.0.15 f=1/(2) e$^{(((x-)}$2)/2^2)

3.0.16 Where X represents the data points, is the mean value and is the standard deviation.

[18]:
```
A,X,C = plt.hist(norm_array, bins=10, density=True, alpha=0.6, color='g')
# plt.show()

f = (1/(std*np.sqrt(2*np.pi)))*np.exp(-((X-mean)/(2*std))**2)
#f = (1/(std*np.sqrt(2*np.pi)))*np.exp(-(X-mean)**2/(2*std)**2)

plt.plot(X,f)
plt.show()
```



The above code plots the data on histogram and shows the normally distibuted data with its density. The density of that data has been found using Probability density function of math function.

4 Portfolio 3

4.0.1 Task 1:

4.0.2 Initialize the following matrix

4.0.3 A: {[2,5,1],[4,3,7],[1,3,2]}

4.0.4 Find the determinant, the trace and the inverse of matrix A

[21]:
```python
# Initialise a matrix M with 3 by 3 elements
M = np.array([[2,5,1],[4,3,7],[1,3,2]])

# Display the matrix
print(M)

# Calculate the determinant of M
D = np.linalg.det(M)

# Display the determinant
print('Determinant:', D)

# Calculate the trace of M
T = np.trace(M)

#Display the trace
print('Trace:', T)

# Calculate the inverse of M
I = np.linalg.inv(M)

print('Matrix:',M)
print('Inverse of Matrix:', I)
```

```
[[2 5 1]
 [4 3 7]
 [1 3 2]]
Determinant: -26.000000000000014
Trace: 7
Matrix: [[2 5 1]
 [4 3 7]
 [1 3 2]]
Inverse of Matrix: [[ 0.57692308  0.26923077 -1.23076923]
 [ 0.03846154 -0.11538462  0.38461538]
 [-0.34615385  0.03846154  0.53846154]]
```

The above code find the determinant of the matrix using algebric function of python det(). Display the Trace of the matrix and the inverse of the matrix using fuction trace() and inv() respectively.

4.0.5 Task 2:

4.0.6 Initialise the following square matrices B and C:

4.0.7 B ={[4,7,2],[3,2,5],[6,4,3]}

4.0.8 C = {[3,1,9],[7,5,8],[2,1,1]}

4.0.9 Find the product P of the matrices B and C by using the Python function for matrix

4.0.10 multiplication. Display the result on the screen.

[23]:
```
# Initialise a matrix B with 3x3 elements
B = np.array([[4,7,2],[3,2,5],[6,4,3]], dtype='int')

# Initialise a matrix C with 3x3 elements
C = np.array([[3,1,9],[7,5,8],[2,1,1]], dtype='int')

# Calculate p = B*C
p = np.matmul(B,C)

#print Product p
print('p:', p)

# The above code multiply the two matrix and generate the new matrix using␣
↪function np.matmul().
```

```
p: [[65 41 94]
 [33 18 48]
 [52 29 89]]
```

4.0.11 Task 3:

4.0.12 Consider the following system of linear equations:

4.0.13 3x + 2y -z =25

4.0.14 2x -y + 4z =19

4.0.15 4x - 2y + 3z = 18

4.0.16 Represent the system of linear equations by using matrices.

4.0.17 Solution:

Representation the system of linear equations by matrices:

```
( 3,2,-1 ) (x)     (25)
( 2,-1,4 ) (y)   = (19)
( 4,-2,3 ) (z)     (18)
```

And by following notation:

MX = C, where (3,2,-1) (x) (25) M = (2,-1,4) , X = (y) , C = (19) (4,-2,3) (z) (18)

4.0.18 Task 4:

4.0.19 Provide the algebraic steps for solving the system of linear equations from Task 3 by

4.0.20 using matrix notation.

4.0.21 Solution:

Step 1: MX = c find the Matrix M , X and C as below for Task 3

```
The matrix has 3 rows and 1 column

    ( 3,2,-1 )      (x)      (25)
M = ( 2,-1,4 ) , X = (y) , C = (19)
    ( 4,-2,3 )      (z)      (18)
```

Step 2: M^-1 MX = M^-1 C

Step 3: Find the Inverse of matrix M

```
IX = M^-1 C
```

Step 4: Calculate X = M^-1 * C which solve the linear equation system and gives values for X,Y,Z.

4.0.22 Task 5:

4.0.23 Solve the system of linear equations from Task 3 by using Python script utilising matrix

4.0.24 multiplication and inverse matrix

[21]:
```python
# Initialise a matrix M with 3x3 elements
M = np.array([[3,2,-1],[2,-1,4],[4,-2,3]])

#Initialise a matrix c
C = np.array([25,19,18])

# Calculate the inverse of Matrix M
I  = np.linalg.inv(M)

#Calculate X = I*C
X = np.matmul(I,C)

#Display Matrix X
print('X:', X)
```

```
X: [5. 7. 4.]
```

The above code solved the linear equation from the task 3 and give the values of X,Y,Z which we put in the eqution to solved the linear equation.

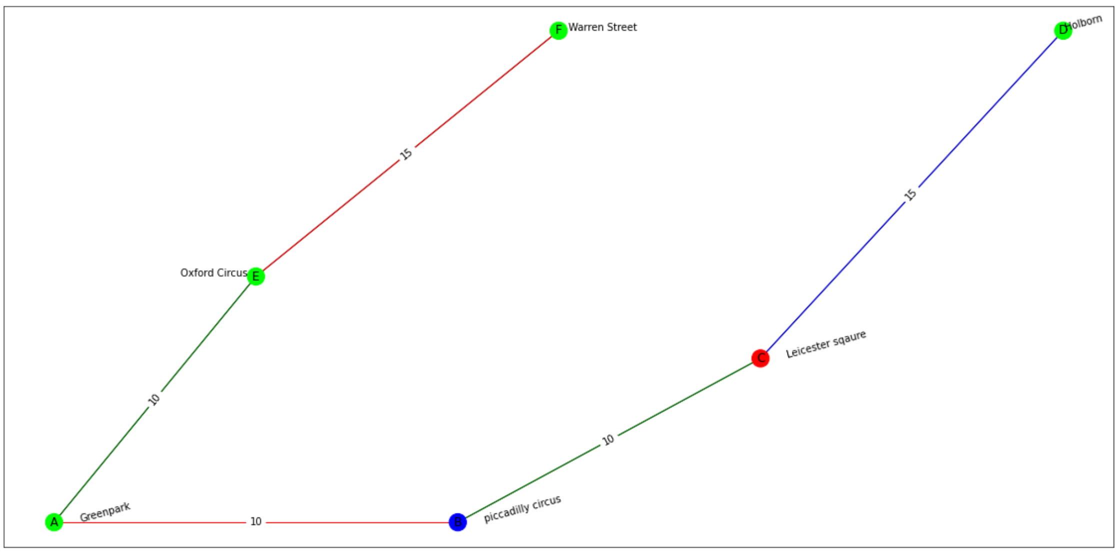
5 Portfolio 4

5.0.1 Task 1 (5 points)

Develop a graph which represents the public transport network of a city of your choice. Some cities have extensive public transport networks. In such case, represent minimum 3 lines with minimum 4 stations on each line. When visualising the network, use different colours for the different lines and their corresponding stations. Provide attributes to the edges which correspond to the distances between stations and visualise them. If the actual distances between the stations are not available, approximate them by using online map services. Visualise the names of the stations.

[66]:
```
# Create a graph object

G= nx.Graph()

# Add $ station to the graph

G.add_node('A',npos=(10,20),color_node='#00FF00')
G.add_node('B',npos=(18,20),color_node='#0000FF')
G.add_node('C',npos=(24,21),color_node='#FF0000')
G.add_node('D',npos=(30,23),color_node='#00FF00')
G.add_node('E',npos=(14,21.5),color_node='#00FF00')
G.add_node('F',npos=(20,23),color_node='#00FF00')

# Connect nodes

G.add_edge('A', 'B', color='r',distance=10)
G.add_edge('B', 'C', color='g',distance=10)
G.add_edge('C', 'D', color='b',distance=15)
G.add_edge('A', 'E', color='g',distance=10)
G.add_edge('E', 'F', color='r',distance=15)

# Extract attributes from the graph to dictionaries
pos = nx.get_node_attributes(G, 'npos')
node_colour = nx.get_node_attributes(G, 'color_node')
edge_colour = nx.get_edge_attributes(G, 'color')
distance = nx.get_edge_attributes(G, 'distance')

# Place the dictionary values in lists
NodeList = list(node_colour.values())
EdgeList = list(edge_colour.values())



# Set the size of the figure
plt.figure(figsize=(20, 10))

# Display the names of the stations
plt.text(10.5,20,s='Greenpark', rotation=15)
```

```
plt.text(18.5,20, s='piccadilly circus', rotation=15)
plt.text(24.5,21, s='Leicester sqaure', rotation=15)
plt.text(30,23, s='Holborn', rotation=15)
plt.text(12.5,21.5, s='Oxford Circus', rotation=0)
plt.text(20.2,23, s='Warren Street', rotation=0)

# Draw the nodes and the edges
nx.draw_networkx(G, pos, node_color=NodeList)
nx.draw_networkx_edges(G, pos, edge_color=EdgeList)
nx.draw_networkx_edge_labels(G, pos, edge_labels=distance)

# Visualise the graph
plt.show()
```



The below graph shows the public transport network of london city which describes some DLR station of the city using NetworkX library The round shapes describes the node as station and the lines as edges which connects the stations and each edges has its attributes and the number between the edges describes the distance between two stations.

5.0.2 Task 2 (5 points)

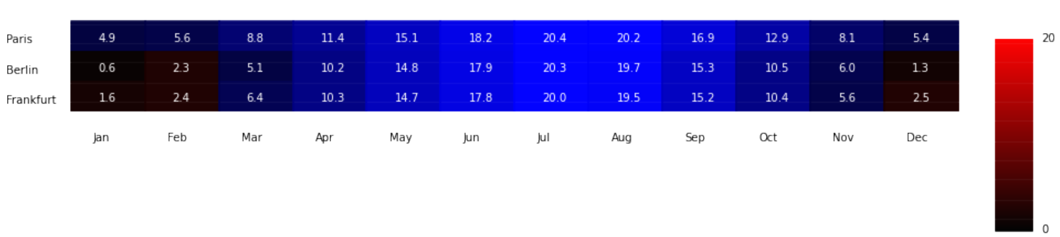
Find the average monthly temperatures of three cities of your choice. Represent the data by using a heat map. Provide a colour scale for guidance. Allow the user to specify a threshold for the heat map. Based on this threshold value, use different base colours when representing the data points.

[24]:
```
def DrawBox(x, y, size, r, g, b):
    if r < 0:
        r = int(0)
    if g < 0:
```

```
        g = int(0)
    if b < 0:
        b = int(0)
    if r > 255:
        r = int(255)
    if g > 255:
        g = int(255)
    if b > 255:
        b = int(255)
    for i in range(0, int(size)):
        plt.plot([x, x + size], [y + i, y + i], '#{:02x}{:02x}{:02x}'.format(r, g, b))


# Store the dataset into a data frame
df = pd.read_csv('HeatMap.csv')
# Print the content on the screen
print(df.head(7))

# Set the plot
plt.figure(figsize=(18, 5))
plt.axis([0, 600, 0, 400])
plt.xticks([])
plt.yticks([])
plt.axis('off')

Min = int(min(df.min(numeric_only=True)))
Max = int(max(df.max(numeric_only=True)))

BoxSize = int(40)
OffsetX = int(15)
OffsetY = int(12)

thresold = int(input("Specify Thresold Value:"))


# Generate the heat map
for i in range(0, df.shape[0]):
    for j in range(1, df.shape[1]):
        ColourCode = int(((df.values[i, j]-Min)/(Max-Min))*255)
        if df.values[i, j] > thresold:
            DrawBox(20+BoxSize*j, 300-BoxSize*i, BoxSize, 0, 0, ColourCode)
        if df.values[i, j] <= thresold:
            DrawBox(20+BoxSize*j, 300-BoxSize*i, BoxSize, ColourCode, 0, 0)
        plt.text(OffsetX+20+BoxSize*j, OffsetY+300-BoxSize*i, str(df.values[i, j]), color='white')
```

```python
# Generate the scale
for i in range(0, 256):
    plt.plot([560, 580], [i + 60, i + 60], '#{:02x}{:02x}{:02x}'.format(int(i), 0, 0))
plt.text(585, 58, Min)
plt.text(585, 312, Max)

plt.text(72, 180, 'Jan')
plt.text(112, 180, 'Feb')
plt.text(152, 180, 'Mar')
plt.text(192, 180, 'Apr')
plt.text(232, 180, 'May')
plt.text(272, 180, 'Jun')
plt.text(312, 180, 'Jul')
plt.text(352, 180, 'Aug')
plt.text(392, 180, 'Sep')
plt.text(432, 180, 'Oct')
plt.text(472, 180, 'Nov')
plt.text(512, 180, 'Dec')

plt.text(25, 310, str(df.values[0, 0]))
plt.text(25, 270, str(df.values[1, 0]))
plt.text(25, 230, str(df.values[2, 0]))

plt.show()
```

```
        City    1    2    3     4     5     6     7     8     9    10   11  \
0      Paris  4.9  5.6  8.8  11.4  15.1  18.2  20.4  20.2  16.9  12.9  8.1
1     Berlin  0.6  2.3  5.1  10.2  14.8  17.9  20.3  19.7  15.3  10.5  6.0
2  Frankfurt  1.6  2.4  6.4  10.3  14.7  17.8  20.0  19.5  15.2  10.4  5.6

    12
0  5.4
1  1.3
2  2.5
Specify Thresold Value:3
```



The above code creat the heat map for the data from csv file HatMap.csv. Also user can specify the thresold value. if thresold value is greater than data points then it shows the balck and orange color either blue and black. The below graph shows the heat map for the 3 city's monthly average temperature for the year. The Dark black color shows the lower temperature of the month and the dark orange color shows the highest temperature of the month July and August and it also represents the strong correlation between the 3 city's temperature for the same months.

5.0.3 Task 3 (5 points)

Represent the data from Task 2 by using parallel coordinates. Use different colours for each city's average monthly temperatures.

[25]:
```
# Read dataset from CSV file
Data = pd.read_csv('HeatMap.csv',index_col=False)

Tempdata = np.array(Data.drop(['City'], axis = 1))
Tempdata
```

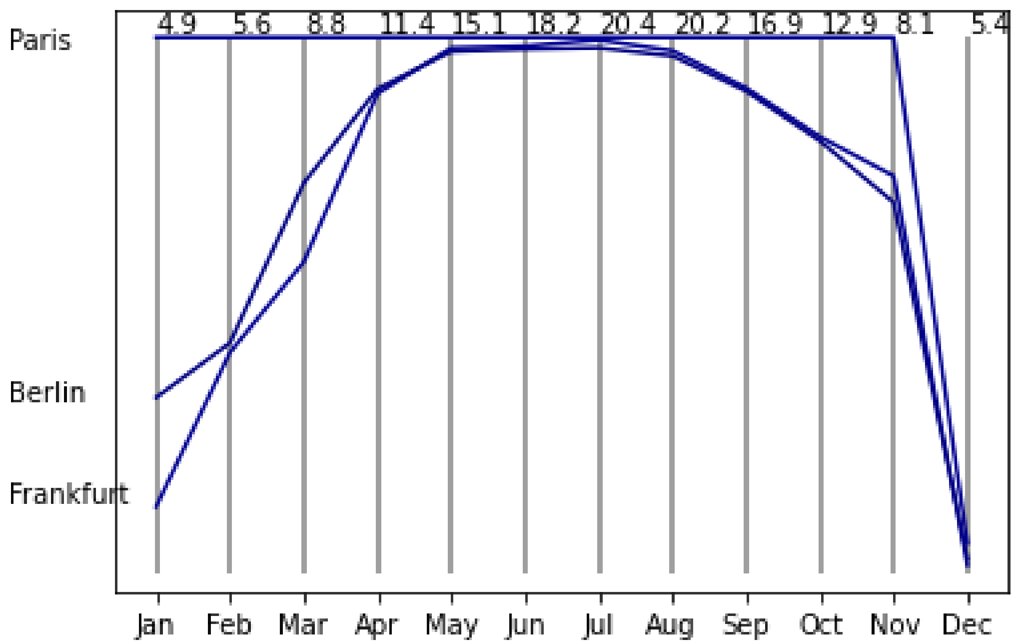
[25]:
```
array([[ 4.9,  5.6,  8.8, 11.4, 15.1, 18.2, 20.4, 20.2, 16.9, 12.9,  8.1,
         5.4],
       [ 0.6,  2.3,  5.1, 10.2, 14.8, 17.9, 20.3, 19.7, 15.3, 10.5,  6. ,
         1.3],
       [ 1.6,  2.4,  6.4, 10.3, 14.7, 17.8, 20. , 19.5, 15.2, 10.4,  5.6,
         2.5]])
```

[26]:
```
# Print dimension of dataset
print('Number of Rows: ', Tempdata.shape[0])
print('Number of Columns: ', Tempdata.shape[1])

# Find the maximum values per column
Maximum = np.amax(Tempdata, axis=0)

# Normalise to interval 0 .. 100
for i in range(0, Tempdata.shape[0]):
    for j in range(0, Tempdata.shape[1]-1):
        Tempdata[i, j] = float(Tempdata[i, j]*(100.0/Maximum[j]))

# Spine names
Name = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']

# Test for random RGB in hex
k = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E',␣
 ↪'F']
MyC = '#'
for i in range(0, 6):
    s = np.random.choice(k)
    MyC = MyC + s
```

```python
plt.yticks([])

# Spines
for i in range(0, Tempdata.shape[1]):
    plt.vlines(i, 0, 100, '#808080')

# Generate the parallel coordinates
for i in range(0, Tempdata.shape[0]):
    if Tempdata[i, 4] < float(40.0):
        MyC = '#'
        for j in range(0, 6):
            s1 = np.random.choice(k)
            s2 = np.random.choice(k)
            MyC = '#' + s1 + s2 + '0000'
        plt.plot(Name, Tempdata[i], MyC)

for i in range(0, Tempdata.shape[0]):
    if float(40.0) < Tempdata[i, 4] < float(60.0):
        MyC = '#'
        for j in range(0, 6):
            s1 = np.random.choice(k)
            s2 = np.random.choice(k)
            MyC = '#' + '00' + s1 + s2 + '00'
        plt.plot(Name, Tempdata[i], MyC)

for i in range(0, Tempdata.shape[0]):
    if float(60.0) < Tempdata[i, 4]:
        MyC = '#'
        for j in range(0, 6):
            s1 = np.random.choice(k)
            s2 = np.random.choice(k)
            MyC = '#' + '0000' + s1 + s2
        plt.plot(Name, Tempdata[i], MyC)

plt.text(0, 101, round(Maximum[0], 2))
plt.text(1, 101, round(Maximum[1], 2))
plt.text(2, 101, round(Maximum[2], 2))
plt.text(3, 101, round(Maximum[3], 2))
plt.text(4, 101, round(Maximum[4], 2))
plt.text(5, 101, round(Maximum[5], 2))
plt.text(6, 101, round(Maximum[6], 2))
plt.text(7, 101, round(Maximum[7], 2))
plt.text(8, 101, round(Maximum[8], 2))
plt.text(9, 101, round(Maximum[9], 2))
plt.text(10, 101, round(Maximum[10], 2))
plt.text(11, 101, round(Maximum[11], 2))
```

```
plt.text(-2, 13, 'Frankfurt')
plt.text(-2, 32, 'Berlin')
plt.text(-2, 98, 'Paris')

plt.show()
```

```
Number of Rows:  3
Number of Columns:  12
```



Parallel Coordinates Plots are ideal for comparing many variables together and seeing the relationships between them. The above parallel coordinates graph shows the data of the three different cities temperature data, Paris, Berlin, Frankfurt. We can notice fro the above graph that the Paris city has highest temperature compare to two other cities.We can also notice from the graph that berlin and frankfurt has almost the same temperature over the year.for the month April-october both berlin and franfurt have almost the same temperature.

COURSEWORK_R-Final

December 4, 2021

1 R Programming

2 Portfolio 1

[21]:
```
install.packages("tidyverse")
install.packages("data.table")
install.packages("modeest")
library(tidyr)
library(ggplot2)
library(dplyr)
library(stringr)
library(data.table)
library(modeest)
```

```
Installing package into 'C:/Users/khala/OneDrive/Documents/R/win-library/4.1'
(as 'lib' is unspecified)


package 'tidyverse' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\khala\AppData\Local\Temp\Rtmp4gu1tR\downloaded_packages

Warning message:
"package 'data.table' is in use and will not be installed"
Warning message:
"package 'modeest' is in use and will not be installed"
```

2.0.1 The tidyr::who dataset contains tuberculosis (TB) cases broken down by year, country, age, gender, and diagnosis method. The data comes from the 2014 World Health Organization Global Tuberculosis Report

[22]:
```
tidyr::who #Shows the who dataset
```

A tibble: 7240 × 60

country <chr>	iso2 <chr>	iso3 <chr>	year <int>	new_sp_m014 <int>	new_sp_m1524 <int>	new_sp_m2534 <int>
Afghanistan	AF	AFG	1980	NA	NA	NA
Afghanistan	AF	AFG	1981	NA	NA	NA
Afghanistan	AF	AFG	1982	NA	NA	NA
Afghanistan	AF	AFG	1983	NA	NA	NA
Afghanistan	AF	AFG	1984	NA	NA	NA
Afghanistan	AF	AFG	1985	NA	NA	NA
Afghanistan	AF	AFG	1986	NA	NA	NA
Afghanistan	AF	AFG	1987	NA	NA	NA
Afghanistan	AF	AFG	1988	NA	NA	NA
Afghanistan	AF	AFG	1989	NA	NA	NA
Afghanistan	AF	AFG	1990	NA	NA	NA
Afghanistan	AF	AFG	1991	NA	NA	NA
Afghanistan	AF	AFG	1992	NA	NA	NA
Afghanistan	AF	AFG	1993	NA	NA	NA
Afghanistan	AF	AFG	1994	NA	NA	NA
Afghanistan	AF	AFG	1995	NA	NA	NA
Afghanistan	AF	AFG	1996	NA	NA	NA
Afghanistan	AF	AFG	1997	0	10	6
Afghanistan	AF	AFG	1998	30	129	128
Afghanistan	AF	AFG	1999	8	55	55
Afghanistan	AF	AFG	2000	52	228	183
Afghanistan	AF	AFG	2001	129	379	349
Afghanistan	AF	AFG	2002	90	476	481
Afghanistan	AF	AFG	2003	127	511	436
Afghanistan	AF	AFG	2004	139	537	568
Afghanistan	AF	AFG	2005	151	606	560
Afghanistan	AF	AFG	2006	193	837	791
Afghanistan	AF	AFG	2007	186	856	840
Afghanistan	AF	AFG	2008	187	941	773
Afghanistan	AF	AFG	2009	200	906	705
...	...	...	...	...	...	...
Zimbabwe	ZW	ZWE	1984	NA	NA	NA
Zimbabwe	ZW	ZWE	1985	NA	NA	NA
Zimbabwe	ZW	ZWE	1986	NA	NA	NA
Zimbabwe	ZW	ZWE	1987	NA	NA	NA
Zimbabwe	ZW	ZWE	1988	NA	NA	NA
Zimbabwe	ZW	ZWE	1989	NA	NA	NA
Zimbabwe	ZW	ZWE	1990	NA	NA	NA
Zimbabwe	ZW	ZWE	1991	NA	NA	NA
Zimbabwe	ZW	ZWE	1992	NA	NA	NA
Zimbabwe	ZW	ZWE	1993	NA	NA	NA
Zimbabwe	ZW	ZWE	1994	NA	NA	NA
Zimbabwe	ZW	ZWE	1995	NA	NA	NA
Zimbabwe	ZW	ZWE	1996	NA	NA	NA
Zimbabwe	ZW	ZWE	1997	NA	NA	NA
Zimbabwe	ZW	ZWE	1998	NA	NA	NA
Zimbabwe	ZW	ZWE	1999	NA	NA	NA
Zimbabwe	ZW	ZWE	2000	NA	NA	NA
Zimbabwe	ZW	ZWE	2001	NA	NA	NA
Zimbabwe	ZW	ZWE	2002	191	600	2548
Zimbabwe	ZW	ZWE	2003	133	874	3048

***There are 7240 rows and 60 columns in the above dataset of who.

2.0.2 1.Gather together all the columns from _new_spm014 to _newrelf65

[23]:
```
who1 <- who %>%
  pivot_longer(
    cols = starts_with("new"), #Columns starts with new
    names_to = "Key",  #new column name for all that columns gathers
    values_to = "Cases", # columns name for all values
    values_drop_na = TRUE  # Drops rows that correspond to missing values
  )
```

The above code gathers all columns start with "new" prefix into new column as "Key" and its value as column "Cases" and drops rows that correspond to missing values

[24]:
```
who1 #The new Dataset agthers all the columns as new key column "Key" with its␣
↪values as"Cases"
```

A tibble: 76046 × 6

country	iso2	iso3	year	Key	Cases
<chr>	<chr>	<chr>	<int>	<chr>	<int>
Afghanistan	AF	AFG	1997	new_sp_m014	0
Afghanistan	AF	AFG	1997	new_sp_m1524	10
Afghanistan	AF	AFG	1997	new_sp_m2534	6
Afghanistan	AF	AFG	1997	new_sp_m3544	3
Afghanistan	AF	AFG	1997	new_sp_m4554	5
Afghanistan	AF	AFG	1997	new_sp_m5564	2
Afghanistan	AF	AFG	1997	new_sp_m65	0
Afghanistan	AF	AFG	1997	new_sp_f014	5
Afghanistan	AF	AFG	1997	new_sp_f1524	38
Afghanistan	AF	AFG	1997	new_sp_f2534	36
Afghanistan	AF	AFG	1997	new_sp_f3544	14
Afghanistan	AF	AFG	1997	new_sp_f4554	8
Afghanistan	AF	AFG	1997	new_sp_f5564	0
Afghanistan	AF	AFG	1997	new_sp_f65	1
Afghanistan	AF	AFG	1998	new_sp_m014	30
Afghanistan	AF	AFG	1998	new_sp_m1524	129
Afghanistan	AF	AFG	1998	new_sp_m2534	128
Afghanistan	AF	AFG	1998	new_sp_m3544	90
Afghanistan	AF	AFG	1998	new_sp_m4554	89
Afghanistan	AF	AFG	1998	new_sp_m5564	64
Afghanistan	AF	AFG	1998	new_sp_m65	41
Afghanistan	AF	AFG	1998	new_sp_f014	45
Afghanistan	AF	AFG	1998	new_sp_f1524	350
Afghanistan	AF	AFG	1998	new_sp_f2534	419
Afghanistan	AF	AFG	1998	new_sp_f3544	194
Afghanistan	AF	AFG	1998	new_sp_f4554	118
Afghanistan	AF	AFG	1998	new_sp_f5564	61
Afghanistan	AF	AFG	1998	new_sp_f65	20
Afghanistan	AF	AFG	1999	new_sp_m014	8
Afghanistan	AF	AFG	1999	new_sp_m1524	55
...	...	...	...	...	...
Zimbabwe	ZW	ZWE	2012	new_sn_f5564	516
Zimbabwe	ZW	ZWE	2012	new_sn_f65	432
Zimbabwe	ZW	ZWE	2012	new_ep_m014	233
Zimbabwe	ZW	ZWE	2012	new_ep_m1524	214
Zimbabwe	ZW	ZWE	2012	new_ep_m2534	658
Zimbabwe	ZW	ZWE	2012	new_ep_m3544	789
Zimbabwe	ZW	ZWE	2012	new_ep_m4554	331
Zimbabwe	ZW	ZWE	2012	new_ep_m5564	178
Zimbabwe	ZW	ZWE	2012	new_ep_m65	182
Zimbabwe	ZW	ZWE	2012	new_ep_f014	208
Zimbabwe	ZW	ZWE	2012	new_ep_f1524	319
Zimbabwe	ZW	ZWE	2012	new_ep_f2534	710
Zimbabwe	ZW	ZWE	2012	new_ep_f3544	579
Zimbabwe	ZW	ZWE	2012	new_ep_f4554	228
Zimbabwe	ZW	ZWE	2012	new_ep_f5564	140
Zimbabwe	ZW	ZWE	2012	new_ep_f65	143
Zimbabwe	ZW	ZWE	2013	newrel_m014	1315
Zimbabwe	ZW	ZWE	2013	newrel_m1524	1642
Zimbabwe	ZW	ZWE	2013	newrel_m2534	5331
Zimbabwe	ZW	ZWE	2013	newrel_m3544	5363

After gathering columns into one column key the dimension of data is 76046(rows) × 6(columns).

2.0.3 2.Make variable names consistent Instead of _newrel we have newrel. It is hard to spot this here but if you do not fix it,we will get errors in subsequent steps.

[25]:
```
who1$Key <- str_replace(who1$Key,'newrel','new_rel') # Replace String "newrel"␣
↪with "new_rel"
```

[26]:
```
who2 <- who1
who2
```

A tibble: 76046 × 6

country	iso2	iso3	year	Key	Cases
<chr>	<chr>	<chr>	<int>	<chr>	<int>
Afghanistan	AF	AFG	1997	new_sp_m014	0
Afghanistan	AF	AFG	1997	new_sp_m1524	10
Afghanistan	AF	AFG	1997	new_sp_m2534	6
Afghanistan	AF	AFG	1997	new_sp_m3544	3
Afghanistan	AF	AFG	1997	new_sp_m4554	5
Afghanistan	AF	AFG	1997	new_sp_m5564	2
Afghanistan	AF	AFG	1997	new_sp_m65	0
Afghanistan	AF	AFG	1997	new_sp_f014	5
Afghanistan	AF	AFG	1997	new_sp_f1524	38
Afghanistan	AF	AFG	1997	new_sp_f2534	36
Afghanistan	AF	AFG	1997	new_sp_f3544	14
Afghanistan	AF	AFG	1997	new_sp_f4554	8
Afghanistan	AF	AFG	1997	new_sp_f5564	0
Afghanistan	AF	AFG	1997	new_sp_f65	1
Afghanistan	AF	AFG	1998	new_sp_m014	30
Afghanistan	AF	AFG	1998	new_sp_m1524	129
Afghanistan	AF	AFG	1998	new_sp_m2534	128
Afghanistan	AF	AFG	1998	new_sp_m3544	90
Afghanistan	AF	AFG	1998	new_sp_m4554	89
Afghanistan	AF	AFG	1998	new_sp_m5564	64
Afghanistan	AF	AFG	1998	new_sp_m65	41
Afghanistan	AF	AFG	1998	new_sp_f014	45
Afghanistan	AF	AFG	1998	new_sp_f1524	350
Afghanistan	AF	AFG	1998	new_sp_f2534	419
Afghanistan	AF	AFG	1998	new_sp_f3544	194
Afghanistan	AF	AFG	1998	new_sp_f4554	118
Afghanistan	AF	AFG	1998	new_sp_f5564	61
Afghanistan	AF	AFG	1998	new_sp_f65	20
Afghanistan	AF	AFG	1999	new_sp_m014	8
Afghanistan	AF	AFG	1999	new_sp_m1524	55
...	...	...	...	...	...
Zimbabwe	ZW	ZWE	2012	new_sn_f5564	516
Zimbabwe	ZW	ZWE	2012	new_sn_f65	432
Zimbabwe	ZW	ZWE	2012	new_ep_m014	233
Zimbabwe	ZW	ZWE	2012	new_ep_m1524	214
Zimbabwe	ZW	ZWE	2012	new_ep_m2534	658
Zimbabwe	ZW	ZWE	2012	new_ep_m3544	789
Zimbabwe	ZW	ZWE	2012	new_ep_m4554	331
Zimbabwe	ZW	ZWE	2012	new_ep_m5564	178
Zimbabwe	ZW	ZWE	2012	new_ep_m65	182
Zimbabwe	ZW	ZWE	2012	new_ep_f014	208
Zimbabwe	ZW	ZWE	2012	new_ep_f1524	319
Zimbabwe	ZW	ZWE	2012	new_ep_f2534	710
Zimbabwe	ZW	ZWE	2012	new_ep_f3544	579
Zimbabwe	ZW	ZWE	2012	new_ep_f4554	228
Zimbabwe	ZW	ZWE	2012	new_ep_f5564	140
Zimbabwe	ZW	ZWE	2012	new_ep_f65	143
Zimbabwe	ZW	ZWE	2013	new_rel_m014	1315
Zimbabwe	ZW	ZWE	2013	new_rel_m1524	1642
Zimbabwe	ZW	ZWE	2013	new_rel_m2534	5331
Zimbabwe	ZW	ZWE	2013	new_rel_m3544	5363

The who2 is new dataset afeter replacing the sring String “newrel” with “new_rel” of column Key

2.0.4 3. Run the following code

2.0.5 wh3 <- who2 %>%

2.0.6 separate(key, c(“new”, “type”, “sexage”), sep = "_")

2.0.7 Now you get a dataset who3. Comment these two lines of code. What is the purpose of using %>%.

[27]:
```
who3  <- who2 %>% separate(Key, c("new", "type", "sexage"), sep = "_")
```

The above code separate the one column into three column by "_" using separate function. The purpose of the %>% is that its work as a pipe for the data. it passes the left hand side of the operator to the first argument of the right hand side of the operator. In the above code, the data frame who2 gets passed to separate().

[28]:
```
who3
```

A tibble: 76046 × 8

country	iso2	iso3	year	new	type	sexage	Cases
<chr>	<chr>	<chr>	<int>	<chr>	<chr>	<chr>	<int>
Afghanistan	AF	AFG	1997	new	sp	m014	0
Afghanistan	AF	AFG	1997	new	sp	m1524	10
Afghanistan	AF	AFG	1997	new	sp	m2534	6
Afghanistan	AF	AFG	1997	new	sp	m3544	3
Afghanistan	AF	AFG	1997	new	sp	m4554	5
Afghanistan	AF	AFG	1997	new	sp	m5564	2
Afghanistan	AF	AFG	1997	new	sp	m65	0
Afghanistan	AF	AFG	1997	new	sp	f014	5
Afghanistan	AF	AFG	1997	new	sp	f1524	38
Afghanistan	AF	AFG	1997	new	sp	f2534	36
Afghanistan	AF	AFG	1997	new	sp	f3544	14
Afghanistan	AF	AFG	1997	new	sp	f4554	8
Afghanistan	AF	AFG	1997	new	sp	f5564	0
Afghanistan	AF	AFG	1997	new	sp	f65	1
Afghanistan	AF	AFG	1998	new	sp	m014	30
Afghanistan	AF	AFG	1998	new	sp	m1524	129
Afghanistan	AF	AFG	1998	new	sp	m2534	128
Afghanistan	AF	AFG	1998	new	sp	m3544	90
Afghanistan	AF	AFG	1998	new	sp	m4554	89
Afghanistan	AF	AFG	1998	new	sp	m5564	64
Afghanistan	AF	AFG	1998	new	sp	m65	41
Afghanistan	AF	AFG	1998	new	sp	f014	45
Afghanistan	AF	AFG	1998	new	sp	f1524	350
Afghanistan	AF	AFG	1998	new	sp	f2534	419
Afghanistan	AF	AFG	1998	new	sp	f3544	194
Afghanistan	AF	AFG	1998	new	sp	f4554	118
Afghanistan	AF	AFG	1998	new	sp	f5564	61
Afghanistan	AF	AFG	1998	new	sp	f65	20
Afghanistan	AF	AFG	1999	new	sp	m014	8
Afghanistan	AF	AFG	1999	new	sp	m1524	55
...	...	...	...	...	...	...	...
Zimbabwe	ZW	ZWE	2012	new	sn	f5564	516
Zimbabwe	ZW	ZWE	2012	new	sn	f65	432
Zimbabwe	ZW	ZWE	2012	new	ep	m014	233
Zimbabwe	ZW	ZWE	2012	new	ep	m1524	214
Zimbabwe	ZW	ZWE	2012	new	ep	m2534	658
Zimbabwe	ZW	ZWE	2012	new	ep	m3544	789
Zimbabwe	ZW	ZWE	2012	new	ep	m4554	331
Zimbabwe	ZW	ZWE	2012	new	ep	m5564	178
Zimbabwe	ZW	ZWE	2012	new	ep	m65	182
Zimbabwe	ZW	ZWE	2012	new	ep	f014	208
Zimbabwe	ZW	ZWE	2012	new	ep	f1524	319
Zimbabwe	ZW	ZWE	2012	new	ep	f2534	710
Zimbabwe	ZW	ZWE	2012	new	ep	f3544	579
Zimbabwe	ZW	ZWE	2012	new	ep	f4554	228
Zimbabwe	ZW	ZWE	2012	new	ep	f5564	140
Zimbabwe	ZW	ZWE	2012	new	ep	f65	143
Zimbabwe	ZW	ZWE	2013	new	rel	m014	1315
Zimbabwe	ZW	ZWE	2013	new	rel	m1524	1642
Zimbabwe	ZW	ZWE	2013	new	rel	m2534	5331
Zimbabwe	ZW	ZWE	2013	new	rel	m3544	5363

2.0.8 4.Separate sexage into sex and age: Use the function separate(). Name the dataset who4

[29]:
```
who4 <- (separate(who3, col=sexage, into=c("sex","age"), sep='(?<=[a-zA-Z])(?
↪=[0-9])'))
```

The above code separate the sexage column into two column sex and age by numeric and alpha regular expression using separate function.

2.0.9 5.Print the first 5 rows and the last 5 rows of the dataset who4 to the screen.

[30]:
```
head(who4) #shows the first 6 data from the datafarme who4
tail(who4) #shows the last 6 data from the datafarme who4
```

A tibble: 6 × 9

country <chr>	iso2 <chr>	iso3 <chr>	year <int>	new <chr>	type <chr>	sex <chr>	age <chr>	Cases <int>
Afghanistan	AF	AFG	1997	new	sp	m	014	0
Afghanistan	AF	AFG	1997	new	sp	m	1524	10
Afghanistan	AF	AFG	1997	new	sp	m	2534	6
Afghanistan	AF	AFG	1997	new	sp	m	3544	3
Afghanistan	AF	AFG	1997	new	sp	m	4554	5
Afghanistan	AF	AFG	1997	new	sp	m	5564	2

A tibble: 6 × 9

country <chr>	iso2 <chr>	iso3 <chr>	year <int>	new <chr>	type <chr>	sex <chr>	age <chr>	Cases <int>
Zimbabwe	ZW	ZWE	2013	new	rel	f	1524	2069
Zimbabwe	ZW	ZWE	2013	new	rel	f	2534	4649
Zimbabwe	ZW	ZWE	2013	new	rel	f	3544	3526
Zimbabwe	ZW	ZWE	2013	new	rel	f	4554	1453
Zimbabwe	ZW	ZWE	2013	new	rel	f	5564	811
Zimbabwe	ZW	ZWE	2013	new	rel	f	65	725

2.0.10 6.Export who4 as an csv file and save it in your local directory.

[31]:
```
fwrite(who4, "C:/Users/khala/OneDrive/Desktop/who.csv") #saved file in my local␣
↪directory
```

The above fwrite function id much better and faster then read_csv and other function

3 Portfolio 2

[49]:
```
# Print the first 6 rows

head(iris)
```

A data.frame: 6 × 5

	Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	Species <fct>
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

[50]:
```
df_iris=iris
df_iris # Iris  Dataset
```

A data.frame: 150 × 5

Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	Species <fct>
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa
5.4	3.4	1.7	0.2	setosa
5.1	3.7	1.5	0.4	setosa
4.6	3.6	1.0	0.2	setosa
5.1	3.3	1.7	0.5	setosa
4.8	3.4	1.9	0.2	setosa
5.0	3.0	1.6	0.2	setosa
5.0	3.4	1.6	0.4	setosa
5.2	3.5	1.5	0.2	setosa
5.2	3.4	1.4	0.2	setosa
4.7	3.2	1.6	0.2	setosa
...	...	...	...	...
6.9	3.2	5.7	2.3	virginica
5.6	2.8	4.9	2.0	virginica
7.7	2.8	6.7	2.0	virginica
6.3	2.7	4.9	1.8	virginica
6.7	3.3	5.7	2.1	virginica
7.2	3.2	6.0	1.8	virginica
6.2	2.8	4.8	1.8	virginica
6.1	3.0	4.9	1.8	virginica
6.4	2.8	5.6	2.1	virginica
7.2	3.0	5.8	1.6	virginica
7.4	2.8	6.1	1.9	virginica
7.9	3.8	6.4	2.0	virginica
6.4	2.8	5.6	2.2	virginica
6.3	2.8	5.1	1.5	virginica
6.1	2.6	5.6	1.4	virginica
7.7	3.0	6.1	2.3	virginica
6.3	3.4	5.6	2.4	virginica
6.4	3.1	5.5	1.8	virginica
6.0	3.0	4.8	1.8	virginica
6.9	3.1	5.4	2.1	virginica

The above dataset is for Iris which show the different species by its different attributes such as Sepal.Length,Sepal.Width,Petal.Length and Petal.Width.

[51]:
```
new_df_iris <- na.omit(df_iris) #remove NA values
new_df_iris
```

A data.frame: 150 × 5

	Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	Species <fct>
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa
21	5.4	3.4	1.7	0.2	setosa
22	5.1	3.7	1.5	0.4	setosa
23	4.6	3.6	1.0	0.2	setosa
24	5.1	3.3	1.7	0.5	setosa
25	4.8	3.4	1.9	0.2	setosa
26	5.0	3.0	1.6	0.2	setosa
27	5.0	3.4	1.6	0.4	setosa
28	5.2	3.5	1.5	0.2	setosa
29	5.2	3.4	1.4	0.2	setosa
30	4.7	3.2	1.6	0.2	setosa
...	...	...	...	...	...
121	6.9	3.2	5.7	2.3	virginica
122	5.6	2.8	4.9	2.0	virginica
123	7.7	2.8	6.7	2.0	virginica
124	6.3	2.7	4.9	1.8	virginica
125	6.7	3.3	5.7	2.1	virginica
126	7.2	3.2	6.0	1.8	virginica
127	6.2	2.8	4.8	1.8	virginica
128	6.1	3.0	4.9	1.8	virginica
129	6.4	2.8	5.6	2.1	virginica
130	7.2	3.0	5.8	1.6	virginica
131	7.4	2.8	6.1	1.9	virginica
132	7.9	3.8	6.4	2.0	virginica
133	6.4	2.8	5.6	2.2	virginica
134	6.3	2.8	5.1	1.5	virginica
135	6.1	2.6	5.6	1.4	virginica
136	7.7	3.0	6.1	2.3	virginica
137	6.3	3.4	5.6	2.4	virginica
138	6.4	3.1	5.5	1.8	virginica
139	6.0	3.0	4.8	1.8	virginica
140	6.9	3.1	5.4	2.1	virginica

From above dataset we can notice that this dataset doesnot contain NA values

3.0.1 Q1. Compute the mean, median and mode of sepal length

[52]:
```
paste("mean of speal.Length is:", mean(new_df_iris$Sepal.Length)) #the function␣
↪mean()find mean of the column
paste("median of speal.Length is:",median(new_df_iris$Sepal.Length)) #the␣
↪function median()find median value of the column
paste("mode of speal.Length is:",mfv(new_df_iris$Sepal.Length)) #the function␣
↪mfv()find mode  of the column
```

'mean of speal.Length is: 5.84333333333333'

'median of speal.Length is: 5.8'

'mode of speal.Length is: 5'

3.0.2 Q2. Compute how "spread out" the data are. Here you need to calculate the minimum, maximum and range of sepal length

[53]:
```
paste("Minimum value of speal.Length is:",min(new_df_iris$Sepal.Length))␣
↪#Minimum value of the column
paste("Maximum value of speal.Length is:",max(new_df_iris$Sepal.Length))␣
↪#Maximum value of the column
paste("The range of speal.Length is:",range(new_df_iris$Sepal.Length)) #range()␣
↪function fine the maximum and minimum value of the column
```

'Minimum value of speal.Length is: 4.3'

'Maximum value of speal.Length is: 7.9'

1. 'The range of speal.Length is: 4.3' 2. 'The range of speal.Length is: 7.9'

3.0.3 Q3. Calculate the interquartile (IQR) range of sepal length. Use the function quantile() to measure quantiles for the same variable, sepal length, and comment what is the difference and relation of these two functions regarding the results shown on your screen?

[54]:
```
summary(new_df_iris$Sepal.Length)
paste("The interquartile(IQR) of speal.Length is:",IQR(new_df_iris$Sepal.Length))
paste("The quantile of speal.Length is:",quantile(new_df_iris$Sepal.Length))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.300   5.100   5.800   5.843   6.400   7.900
```

'The interquartile(IQR) of speal.Length is: 1.3'

1. 'The quantile of speal.Length is: 4.3' 2. 'The quantile of speal.Length is: 5.1' 3. 'The quantile of speal.Length is: 5.8' 4. 'The quantile of speal.Length is: 6.4' 5. 'The quantile of speal.Length is: 7.9'

Quantiles are points in a distribution that relate to the rank order of values in that distribution.The middle value(middle quantile, 50th quartile) is known as the median. The first and last values is limit, the minimum and maximum values. the second and 4th values are lower quartile(25th quartile) and upper quartile((75th quartile) of the data.

However the interquartile(IQR) is the upper quartile((75th quartile) - lower quartile(25th quartile) of the data.

3.0.4 Q4. Compute the variance and standard deviation of sepal length

[55]:
```
paste("The variance of speal.Length is:",var(new_df_iris$Sepal.Length)) # the␣
 ↪function var() finds the variance of the data
paste("The standard variance of speal.Length is:",sd(new_df_iris$Sepal.Length))␣
 ↪# the function sd() finds the standard deviation of the data
```

'The variance of speal.Length is: 0.685693512304251'

'The standard variance of speal.Length is: 0.828066127977863'

3.0.5 Q5. Choose the right function to show min, max, mean, median, 1st and 3rd quantiles all at once of the variable sepal length

[56]:
```
summary(new_df_iris$Sepal.Length) #This function summary() gives descriptiv␣
 ↪statistics of the column sepal.length.
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  4.300   5.100   5.800   5.843   6.400   7.900 
```

3.0.6 Q6. Use sapply() to compute the mean and quantiles of each column in the dataset iris.

[57]:
```
colMeans(new_df_iris[sapply(new_df_iris, is.numeric)]) #the function clsMeans()␣
 ↪Use to find mean of all columns
sapply(new_df_iris[,1:4],quantile) #The below table shows the all columns␣
 ↪quantile
```

Sepal.Length 5.84333333333333 **Sepal.Width** 3.05733333333333 **Petal.Length** 3.758 **Petal.Width** 1.19933333333333

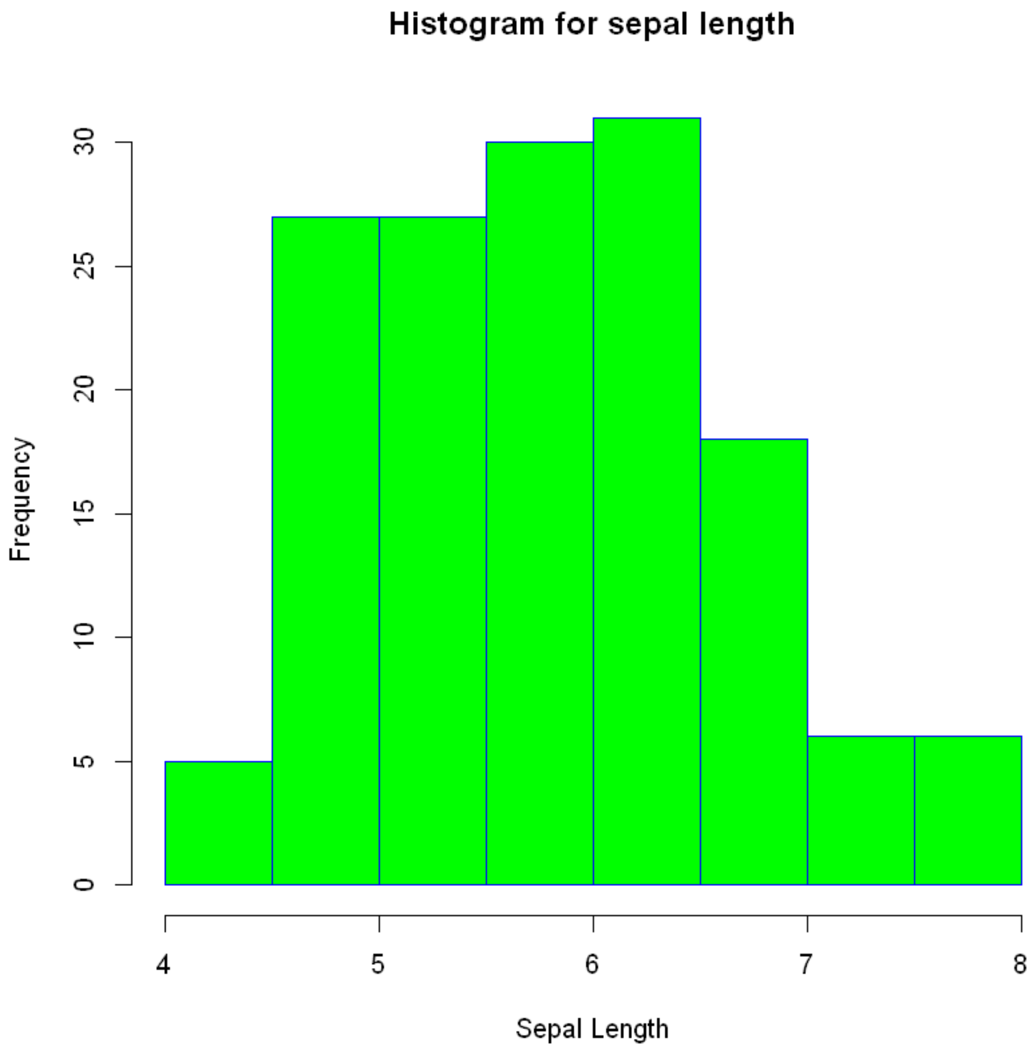
A matrix: 5 × 4 of type dbl

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
0%	4.3	2.0	1.00	0.1
25%	5.1	2.8	1.60	0.3
50%	5.8	3.0	4.35	1.3
75%	6.4	3.3	5.10	1.8
100%	7.9	4.4	6.90	2.5

3.0.7 Q7. Use the in-built R basic functions (no need to import any library) to create a histogram for sepal length. Make sure you add the following arguments:

[62]:
```
hist(new_df_iris$Sepal.Length,
      main='Histogram for sepal length',
        xlab='Sepal Length',
        col='green',
```

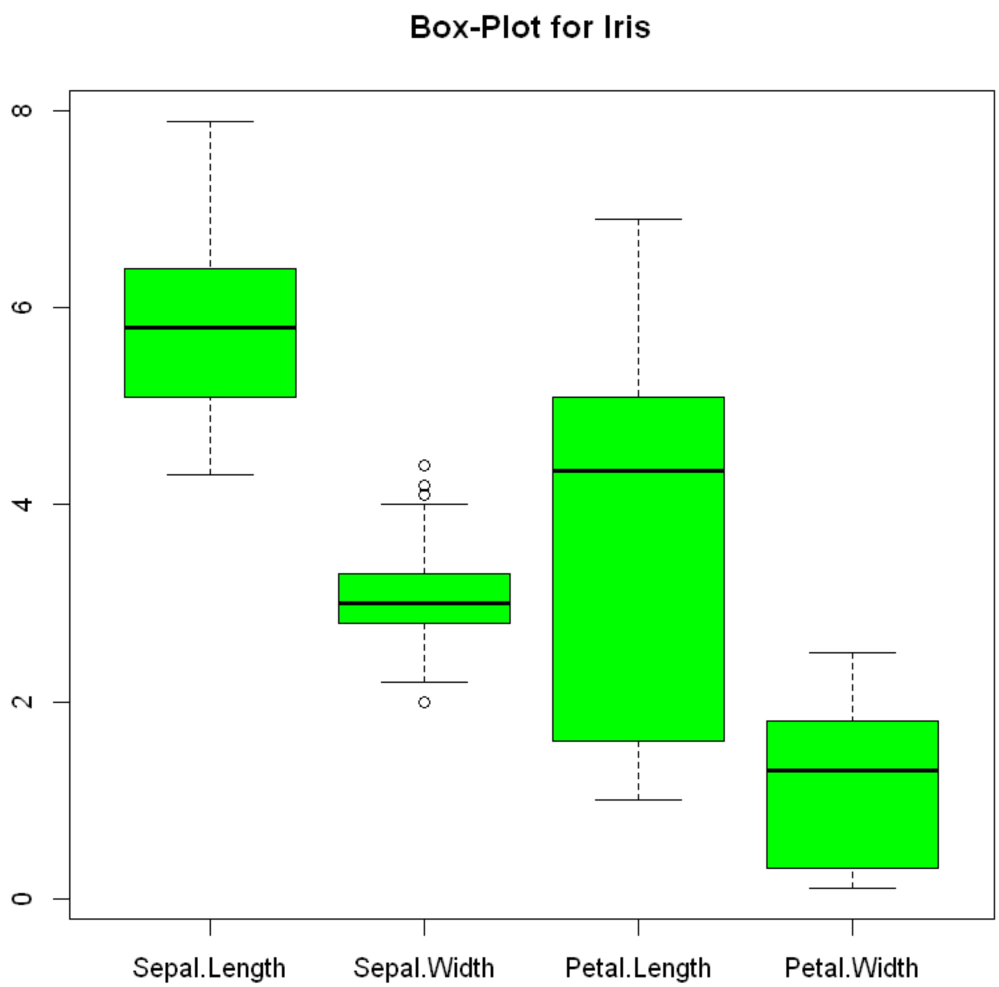
```
      border='blue') # Plot hsitogram using hist() function
```



The histogram is used to evaluate the distribution of the data. The above histogram gram graph we can notice that the graph is for sepal Length of the species. The green vertical line indicates the bin, y-axis shows the frequency of the sepal legth data and x-axis shows the values of the sepal Length.We can notice from the above histogram that the maximum sepal.length is between 5.5 to 6.5.

3.0.8 Q8. Use the in-built R basic function to create one boxplot for sepal length, sepal width, petal length and petal width

[42]:
```
boxplot(new_df_iris[,1:4],
        main="Box-Plot for Iris",
        col='green',
        border='black')
```



Box plots can be used to compare variables, particularly for illustrating median/spread between different groups of data. The above graph shows the Box-plot graph for the sepal length, sepal width, petal length and petal width of the Iris data. The box shows the Interquartile of the data. The middle line between box represents the median of the data, the portion of the box beside the median it shows the upper quartile and lower quartile of the data,the two vertical line shows the range of the data, the bubble outside the range shows the outliers. It can be seen from the graph that the Sepal.Width has the outliers in its data.

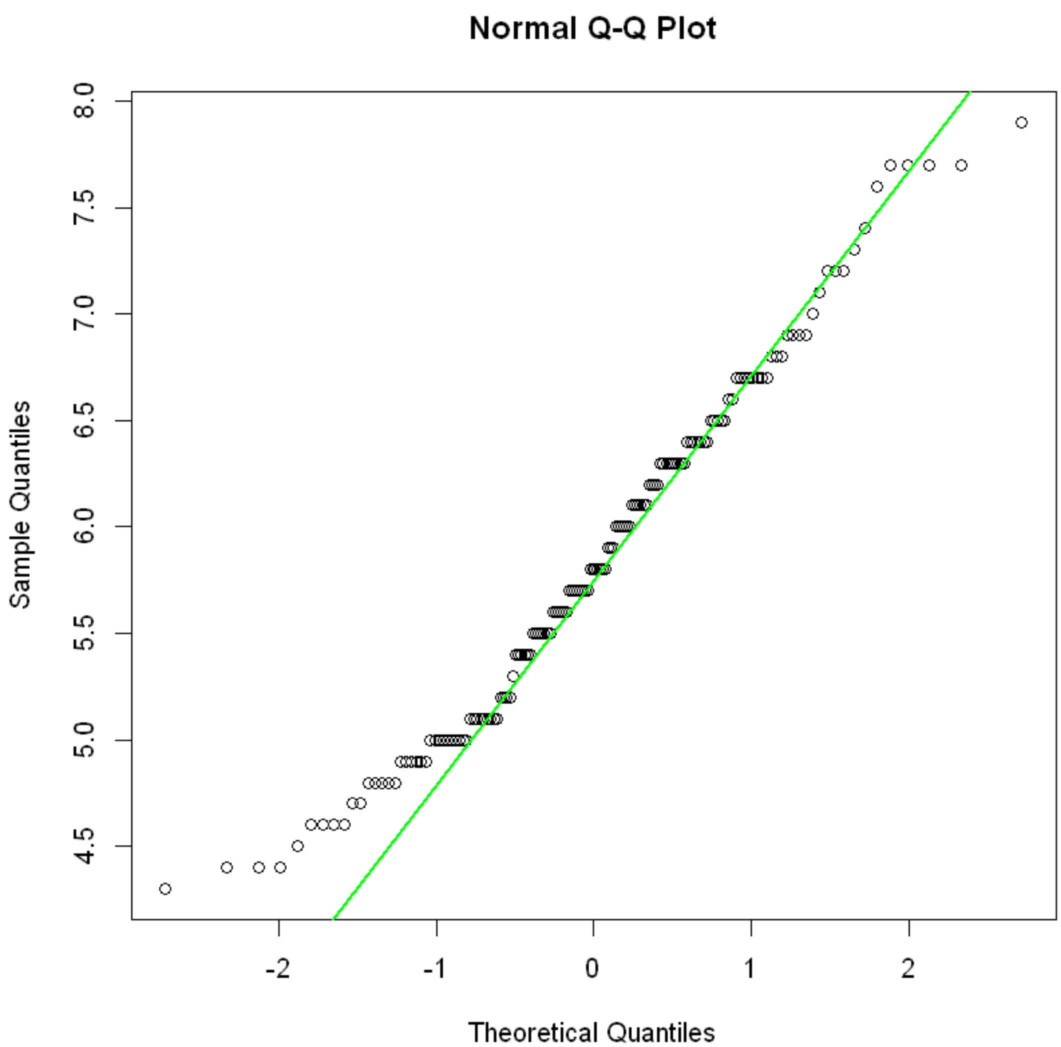
3.0.9 Q9. The R base functions qqnorm() and qqplot() are used to produce quantile-quantile plots:

3.0.10 • qqnorm(): produces a normal QQ plot of the variable

3.0.11 • qqline(): adds a reference lineUse these two functions to create a QQ plot for sepal length.

3.0.12 You will need to set the reference line colour as green, and its width as 2)

[43]:
```
qqnorm(new_df_iris$Sepal.Length)
qqline(new_df_iris$Sepal.Length,col='green', lwd=2)
```



From the above plot It can be notice that the QQplot is not following the straight QQline, indicating that the sepal Length is not distribted normally.

4 R Programming Portfolio 3

ggplot2::ggplot() // use a function defined in ggplot2 ggplot2::mpg //load a dataset from package ggplot2

[63]:
```
mpg=ggplot2::mpg
mpg
```

A tibble: 234 × 11

manufacturer	model	displ	year	cyl	trans	drv	cty
<chr>	<chr>	<dbl>	<int>	<int>	<chr>	<chr>	<int>
audi	a4	1.8	1999	4	auto(l5)	f	18
audi	a4	1.8	1999	4	manual(m5)	f	21
audi	a4	2.0	2008	4	manual(m6)	f	20
audi	a4	2.0	2008	4	auto(av)	f	21
audi	a4	2.8	1999	6	auto(l5)	f	16
audi	a4	2.8	1999	6	manual(m5)	f	18
audi	a4	3.1	2008	6	auto(av)	f	18
audi	a4 quattro	1.8	1999	4	manual(m5)	4	18
audi	a4 quattro	1.8	1999	4	auto(l5)	4	16
audi	a4 quattro	2.0	2008	4	manual(m6)	4	20
audi	a4 quattro	2.0	2008	4	auto(s6)	4	19
audi	a4 quattro	2.8	1999	6	auto(l5)	4	15
audi	a4 quattro	2.8	1999	6	manual(m5)	4	17
audi	a4 quattro	3.1	2008	6	auto(s6)	4	17
audi	a4 quattro	3.1	2008	6	manual(m6)	4	15
audi	a6 quattro	2.8	1999	6	auto(l5)	4	15
audi	a6 quattro	3.1	2008	6	auto(s6)	4	17
audi	a6 quattro	4.2	2008	8	auto(s6)	4	16
chevrolet	c1500 suburban 2wd	5.3	2008	8	auto(l4)	r	14
chevrolet	c1500 suburban 2wd	5.3	2008	8	auto(l4)	r	11
chevrolet	c1500 suburban 2wd	5.3	2008	8	auto(l4)	r	14
chevrolet	c1500 suburban 2wd	5.7	1999	8	auto(l4)	r	13
chevrolet	c1500 suburban 2wd	6.0	2008	8	auto(l4)	r	12
chevrolet	corvette	5.7	1999	8	manual(m6)	r	16
chevrolet	corvette	5.7	1999	8	auto(l4)	r	15
chevrolet	corvette	6.2	2008	8	manual(m6)	r	16
chevrolet	corvette	6.2	2008	8	auto(s6)	r	15
chevrolet	corvette	7.0	2008	8	manual(m6)	r	15
chevrolet	k1500 tahoe 4wd	5.3	2008	8	auto(l4)	4	14
chevrolet	k1500 tahoe 4wd	5.3	2008	8	auto(l4)	4	11
...	...	...	...	...	...	...	...
toyota	toyota tacoma 4wd	3.4	1999	6	auto(l4)	4	15
toyota	toyota tacoma 4wd	4.0	2008	6	manual(m6)	4	15
toyota	toyota tacoma 4wd	4.0	2008	6	auto(l5)	4	16
volkswagen	gti	2.0	1999	4	manual(m5)	f	21
volkswagen	gti	2.0	1999	4	auto(l4)	f	19
volkswagen	gti	2.0	2008	4	manual(m6)	f	21
volkswagen	gti	2.0	2008	4	auto(s6)	f	22
volkswagen	gti	2.8	1999	6	manual(m5)	f	17
volkswagen	jetta	1.9	1999	4	manual(m5)	f	33
volkswagen	jetta	2.0	1999	4	manual(m5)	f	21
volkswagen	jetta	2.0	1999	4	auto(l4)	f	19
volkswagen	jetta	2.0	2008	4	auto(s6)	f	22
volkswagen	jetta	2.0	2008	4	manual(m6)	f	21
volkswagen	jetta	2.5	2008	5	auto(s6)	f	21
volkswagen	jetta	2.5	2008	5	manual(m5)	f	21
volkswagen	jetta	2.8	1999	6	auto(l4)	f	16
volkswagen	jetta	2.8	1999	6	manual(m5)	f	17
volkswagen	new beetle	1.9	1999	4	manual(m5)	f	35
volkswagen	new beetle	1.9	1999	4	auto(l4)	f	29
volkswagen	new beetle	2.0	1999	4	manual(m5)	f	21

[45]:

```
summary(mpg)
```

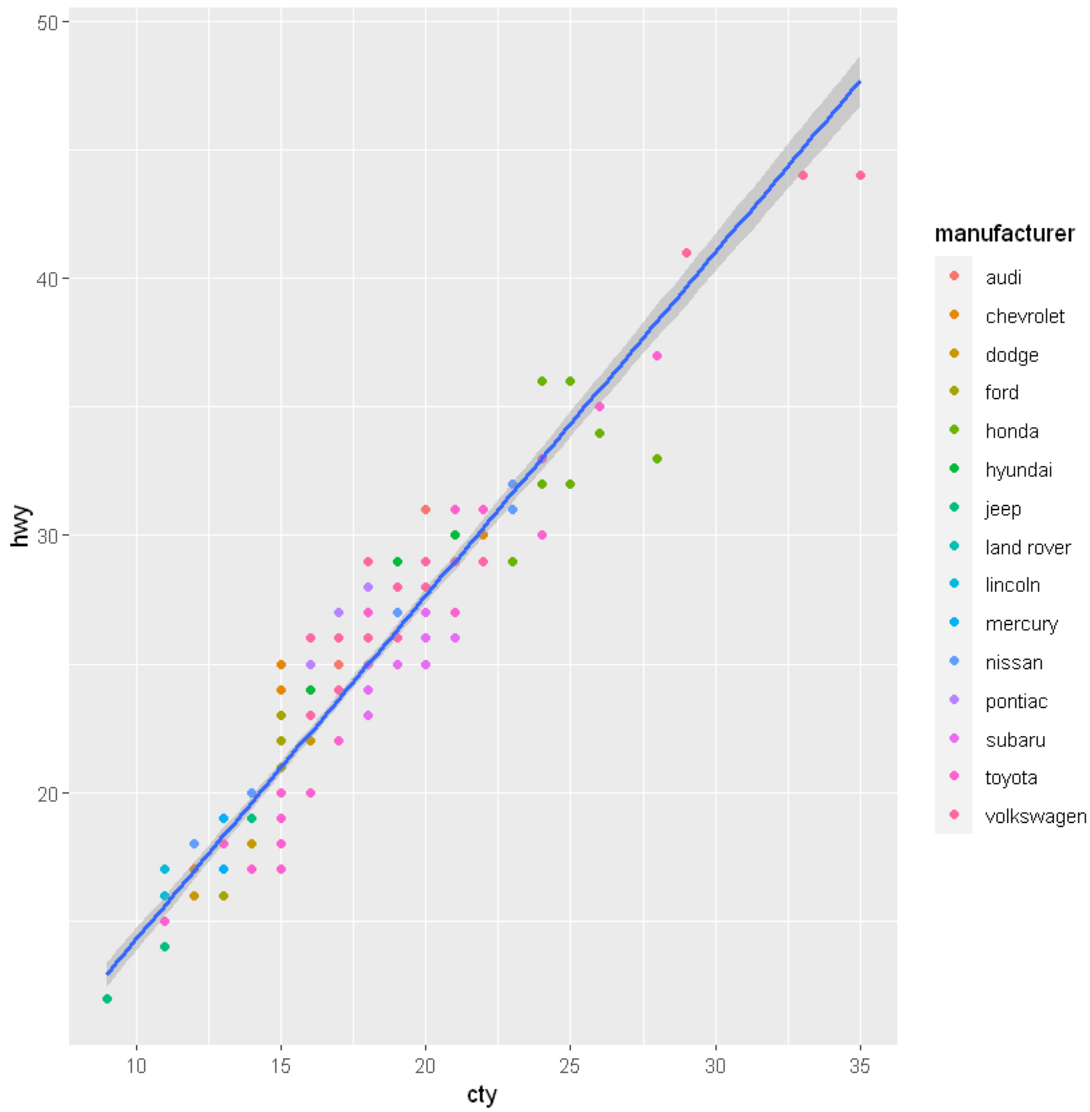
```
 manufacturer          model               displ            year     
 Length:234         Length:234         Min.   :1.600   Min.   :1999  
 Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999  
 Mode  :character   Mode  :character   Median :3.300   Median :2004  
                                       Mean   :3.472   Mean   :2004  
                                       3rd Qu.:4.600   3rd Qu.:2008  
                                       Max.   :7.000   Max.   :2008  
      cyl           trans               drv                 cty       
 Min.   :4.000   Length:234         Length:234         Min.   : 9.00  
 1st Qu.:4.000   Class :character   Class :character   1st Qu.:14.00  
 Median :6.000   Mode  :character   Mode  :character   Median :17.00  
 Mean   :5.889                                         Mean   :16.86  
 3rd Qu.:8.000                                         3rd Qu.:19.00  
 Max.   :8.000                                         Max.   :35.00  
      hwy             fl               class          
 Min.   :12.00   Length:234         Length:234        
 1st Qu.:18.00   Class :character   Class :character  
 Median :24.00   Mode  :character   Mode  :character  
 Mean   :23.44                                        
 3rd Qu.:27.00                                        
 Max.   :44.00                                        
```

4.0.1 Q1. Plot and explain: Which vehicle brand (or manufacturer), offers the best mpg in both city and in the highway?

[46]:

```
mpg_cityHwy <- ggplot(mpg, aes(cty, hwy))
mpg_cityHwy + geom_point(aes(colour = manufacturer))+
  stat_smooth(method = "lm")
```

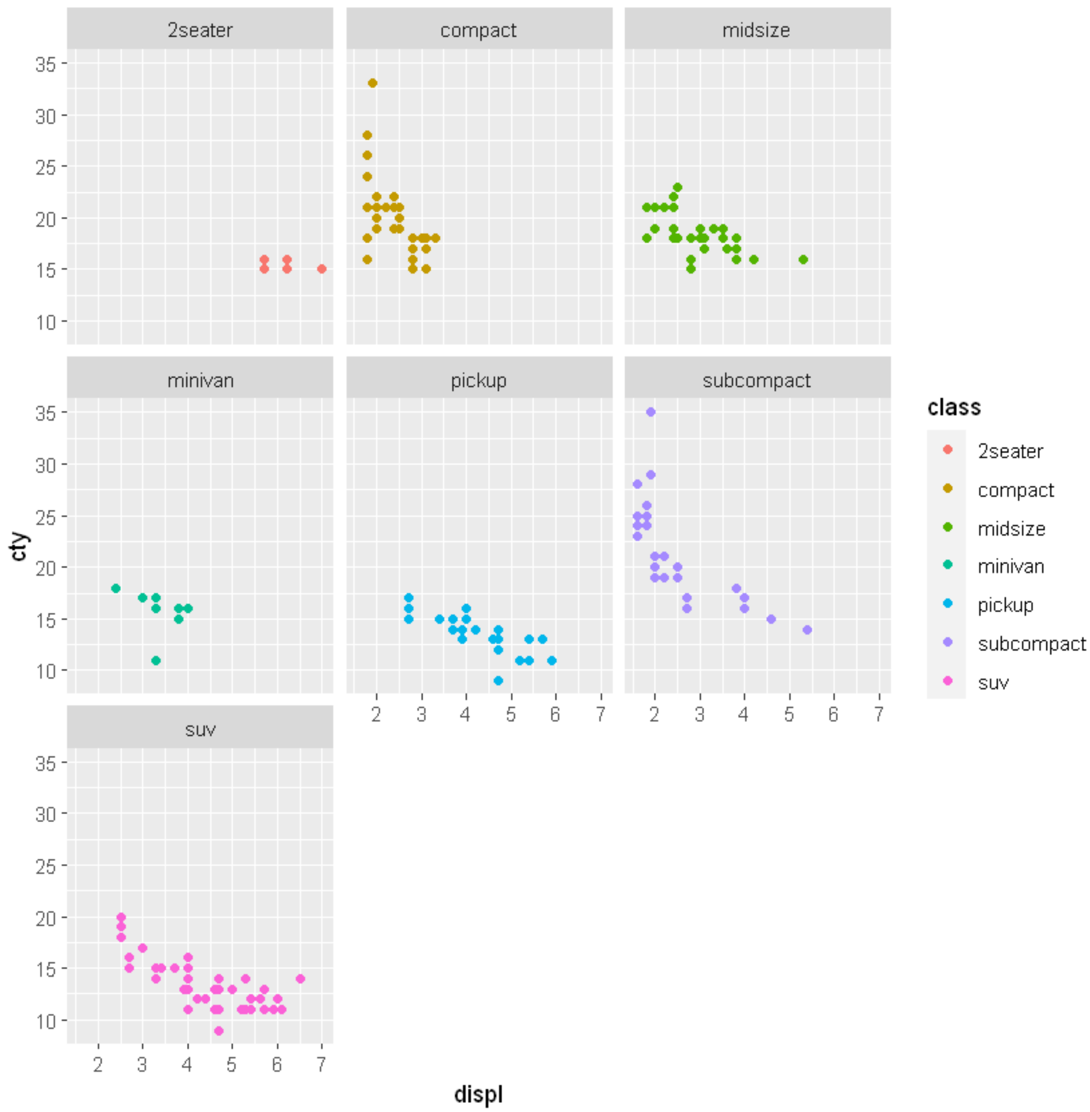
```
`geom_smooth()` using formula 'y ~ x'
```



The above scatter plots give a great view of best mpg offers by both city and highway.The x-axis shows the city mpg(Miles per gallon) and y-axis shows the highway mpg where mapping vehicle manufacturing company as a color aesthetic with regression line . It can be notice from the above scatter plots that the Vehicle company ***Volkswagen*** has the highest mpg performance in both city is 35 and highway is 44 compares to other vehicle brands.

4.0.2 Q2. Plot and explain: Which type of car, regarding their displ range (size of engine) has the lowest mpg in the city categorised by the vehicle type (e.g., compact, suv or 2seaters defined in the variable class)?

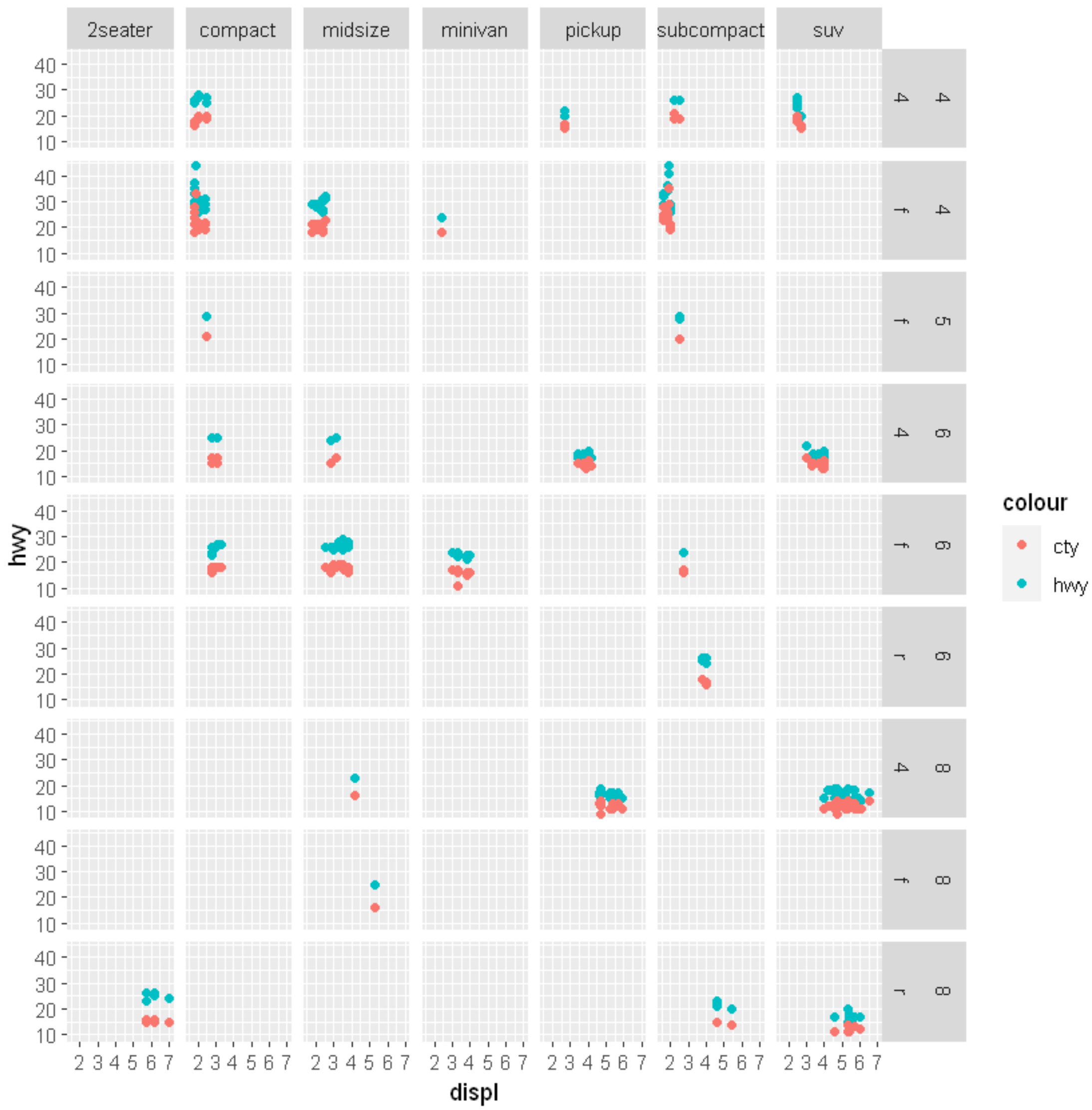
[47]:
```
mpg_displ <- ggplot(mpg, aes(displ,cty,color=class))
mpg_displ + facet_wrap(~ class)+ geom_point()
```



The above scatter plots give a great view of how mpg decrease with high dispacement range of vehicles(size of engine)in city. The x-axis shows the displacement range and y-axis shows the city mpg however the data is categorised by vehicle types using facet_wrap() function.Facets split our plot into facets(subplots) that each displays one subset of the data.here we have used single variable so we have used facet_wrap(). from above chart, we can clearly notice that the ***SUV and pickup*** has the lowest mpg which is 9 in city with its size of the engine which is 4.7.

4.0.3 Q3.Plot and explain: Which type of car, regarding their displ range (size of engine) has the best mpg performance in both city and highway? Display the resulting plot categorised by the number of cylinders and the drive type (the type of drive train, where f = front-wheel drive, r = rear wheel drive, 4 = 4wd). You are a buyer who wants a high litre engine vehicle and drives mostly in the highway, which type of car would you choose?

[48]:
```
ggplot(data=mpg)+

geom_point(aes(x = displ , y = hwy,color = "hwy"))+

geom_point( aes(x = displ, y = cty,color = "cty"))+

facet_grid(cyl ~ drv~ class)

#scatterplot_class <-ggplot(mpg, aes(x = cty , y = hwy))

#scatterplot_class + geom_point( aes(color = class))+
#facet_grid(cyl ~ drv ~  displ )
```



Above scatter plot describes the mpg performance of both city and highway and with thier displacement range and it is categorised by cylinder size of the vehicle and vehicle types. The x-axis show the displacement range and y -axis shows the highway mpg and city mpg performance blue color, orange color, respectively.

From above plot It can be notice that Compact, subcompact and midsize vehicle has the highest mpg in both city and highway but with 4 wheel-drive and front-wheel drive and 4 cylinder which is vehicle types of Volkswagen brand but the low dispalcement range. From above plot It can be notice that in respect to the higher litter engine and good performance on highway, it is recommended to choose the ***2seater*** which has 7 diplacement range and 8 cylinder and rear-wheel drive with highway performace around 25-30 mpg.