Programming Fundamentals for Data Science

(COMP-1832)

Coursework submitted in partial fulfillment

Of the requirements for the degree of

Master of Science

In

Data Science

By

Rabin Chhetri

(ID: 001185145)

Submitted to: Dr Jia Wang

Module Leader (COMP-1832)

School of Computing and Mathematical Sciences

Part 1: Python Portfolio Tasks

1. Data Processing

Develop the Notebook cells in the following order:

1. Download the Dataset.csv file from Moodle, load its content into pandas data frame and visualise the entire content of the data frame Check the data frame for the following 'data cleaning' issues and resolve them:

2. Missing values

3. Unnecessary duplicates

4. Wrong data types

5. Wrong values

6. Save the updated data frame into a new CSV file 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

8. Median

9. Range

10. The column 'median_income' contains currency in tens of thousands USD. Convert it into USD and visualise the entire updated data frame.

Python code along with code comments and generated result is as follows:

```
# importing pandas library as pd
import pandas as pd
#reading the Dataset.csv file downloaded from moodle and loading its content to pandas data frame as DataFrame
DataFrame=pd.read_csv('C:/Users/Rabin Thapa Kshetry/Downloads/Dataset.csv')
# converting the dataframe into string and printing the content
print(DataFrame.to_string())
```

```
    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
```

```
# Checking for the duplicate data in the datset
#rows containing duplicate data is represented with True
print(DataFrame.duplicated())
```

```
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
```

```
#duplicates are removed from DataFrame and the updated content is loaded into NewDataFrame
NewDataFrame=DataFrame.drop_duplicates()
#Updated Dataframe NewDataFrame is converted to string and is displayed
print(NewDataFrame.to_string())
```

```
    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
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
```

```
#All the rows having NaN are dropped and the updated content is loaded into NewDataFrame1
NewDataFrame1=NewDataFrame.dropna()
# Updated Dataset from NewDataFrame1 is converted into string and is displayed
print(NewDataFrame1.to_string())
```

```
    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
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
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
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
```

```
# after dropping the duplicate rows and rows with missing values the index is reset and the dataframe is displayed
NewDataFrame1.reset_index(drop=True, inplace=True)
print(NewDataFrame1.to_string())
```

```
    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     -121.80     39.33             30.0000         1019         501         2.5259               81300          INLAND
11     -120.46     38.15             16.0000         4221        1516         2.3816              116000          INLAND
12     -120.55     38.12             10.0000         1566         785         2.5000              116100          INLAND
13     -120.56     38.09             34.0000         2745        1150         2.3654               94900          INLAND
14     -124.23     41.75             11.0000         3159        1343         2.4805               73200      NEAR OCEAN
15     -124.21     41.77             17.0000         3461        1947         2.5795               68400          NEAR O
16     -124.19     41.78             15.0000         3140        1645         1.6654               74600          NEAR O
17     -124.16     41.74             15.0000         2715        1532         2.1829               69500      NEAR OCEAN
18     -124.16     41.92             19.0000         1668         841         2.1336               75000      NEAR OCEAN
19     -118.32     33.35             27.0000         1675         744         2.1579              450000          ISLAND
20     -118.33     33.34             52.0000         2359        1100         2.8333              414700          ISLAND
21     -118.32     33.33             52.0000         2127         733         3.3906              300000          ISLAND
22     -118.32     33.34             52.0000          996         341         2.7361              450000          ISLAND
23     -118.48     33.43             29.0000          716         422         2.6042              287500          ISLAND
```

```
#Updated data frame is loaded into a new csv file UpdatedDataset.csv
NewDataFrame1.to_csv('C:/Users/Rabin Thapa Kshetry/Downloads/UpdatedDataset.csv')
```

```
# from the updated data frame NewDataFrame1 , Mean is calculated for column median_house_value and is displayed
print('Mean :',NewDataFrame1.median_house_value.mean())
```

```
Mean : 180629.16666666666
```

```
# from the updated data frame NewDataFrame1 , Median is calculated for column median_house_value and is displayed
print('Median :',NewDataFrame1.median_house_value.median())
# from the updated data frame NewDataFrame1 , Range is calculated for column median_house_value and is displayed
print('Range :',NewDataFrame1.median_house_value.max()-NewDataFrame1.median_house_value.min())
```

```
Median : 88100.0
Range : 396600
```

```
# column median_income has currencies in tens of thousands USD so it is converted into USD by multiplying with 1000
# Updated median income is displayed
print(NewDataFrame1.median_income*1000)
```

```
0     8325.2
1     8301.4
2     3120.0
3     3270.5
4     2789.1
5     2322.4
6     1788.5
7     1910.0
8     2047.4
9     1835.5
10    2525.9
11    2381.6
12    2500.0
13    2365.4
14    2480.5
15    2579.5
16    1665.4
17    2182.9
18    2133.6
19    2157.9
20    2833.3
21    3390.6
22    2736.1
23    2604.2
Name: median_income, dtype: float64
```

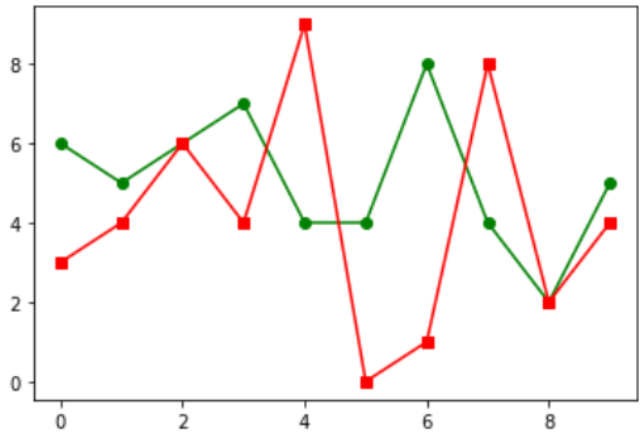
2. Statistics with Python

```
Part 1
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.
```

```
# importing numpy library as np
import numpy as np
# importing matplotlib.pyplot library as plt
import matplotlib.pyplot as plt
# initialising 2D Array My2DArray with 5 rows and 10 columns of integer data type
My2DArray= np.empty(shape=(5,10), dtype='int')
# assigning values to My2DArray with random integers from 0 to 9
for i in range(0, 5):
     for j in range(0, 10):
        My2DArray[i, j] = int(np.random.randint(0, 10))
# displaying the My2DArray
print(My2DArray)
```

```
[[6 5 6 7 4 4 8 4 2 5]
 [3 4 6 4 9 0 1 8 2 4]
 [3 3 0 3 9 1 2 6 3 3]
 [1 8 8 7 2 3 3 8 4 6]
 [4 3 9 7 9 6 9 5 5 5]]
```

```
# Displaying the first two rows of My2DArray with two different colours red and green
plt.plot(My2DArray[0, ], marker='o', color='g')
plt.plot(My2DArray[1, ], marker='s', color='r')
plt.show()
```



```
Part 2
Provide the following information about each individual row:
Mean
Median
Standard deviation
```

```
# Calculating mean, median and standard Deviation of the above My2DArray for each row
Mean = np.mean(My2DArray, axis=1)
Median = np.median(My2DArray, axis=1)
StandardDeviation = np.std(My2DArray, axis=1)
# Displaying Mean
print('Mean :', Mean)
# displaying Median
print('Median :', Median)
# Displaying Standard Deviation
print('Standard Deviation :', StandardDeviation)
```

```
Mean : [5.1 4.1 3.3 5.  6.2]
Median : [5.  4.  3.  5.  5.5]
Standard Deviation : [1.64012195 2.73678644 2.41039416 2.56904652 2.0880613 ]
```

```
Part 3
Initialise a one-dimensional array representing a normal distribution of 1000 data points
with mean value 17 and standard deviation 0.2.
```

```
# Generating normal distribution using mean as 17 and SD as 0.2
Mean = float(17)
SD = float(0.2)
OneDimensionalArray = np.random.normal(Mean, SD, 1000)
# Displaying the content of OneDimensionalArray
print(OneDimensionalArray)
```

```
[16.9708252  17.10329435 16.84983347 17.01848034 16.96420276 17.0171752
 16.59083733 17.12589279 18.02825117 16.79525457 16.74376453 16.99636315
 17.09519516 16.8039567  17.2077012  16.66233552 17.22623557 17.1951874
 17.13987884 16.86167088 16.96646253 17.30526126 16.73890145 16.52559016
 17.15647851 17.15549084 17.04646858 17.04776805 17.07698074 17.32993888
 16.58160344 17.24229328 16.99098327 17.20958781 16.97836472 16.97838384
 17.09433217 16.65913908 16.91613976 17.17517041 17.12157469 16.91476999
 17.05737674 16.98156872 16.94378173 17.03782602 16.70511947 16.8637206
 16.92696551 17.04808991 17.11111543 16.88980847 17.12391238 17.41385949
 17.28619762 16.8320034  17.07228908 17.03917693 16.86253573 16.92993151
 17.10843071 16.88003132 17.17474308 16.95249584 16.99793833 17.10866331
 17.16600361 16.93851343 16.96526854 17.16525333 17.22138025 17.10322002
 16.89923813 17.02060357 17.43561166 17.19609009 17.03165588 17.23372126
 17.09208216 17.04304781 17.18190584 17.01798489 16.4859301  17.11190404
 17.12905599 16.85116378 17.08104139 17.1320785  16.64603055 16.9679506
 16.99531948 17.11564541 17.08940075 17.04533101 17.10575415 16.78868333
 17.05161589 16.8438647  16.8815094  17.04774894 16.96486607 17.01450132
 16.90185934 17.10776238 16.80736349 17.3529954  16.9363488  17.07121053
 16.78921623 16.60240536 16.85825382 16.87532801 16.76023022 17.12238751
```

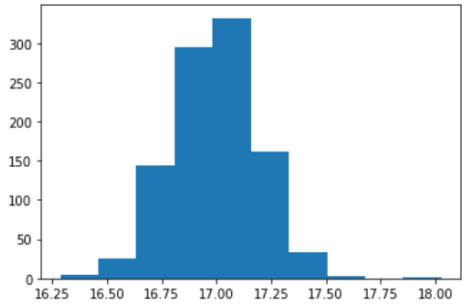
```
Part 4
Find the maximum and the minimum values of the dataset and calculate the range.
```

```
# finding and displaying the maximum and minimum values from OneDimensionalArray
print('Maximum Value :', np.max(OneDimensionalArray))
print('Minimum Value :', np.min(OneDimensionalArray))
```

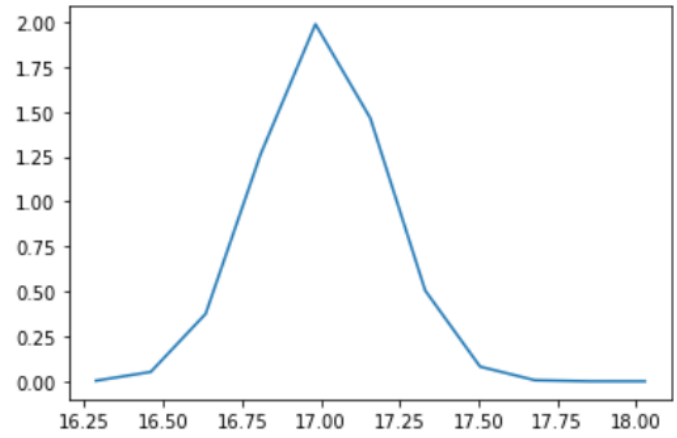
```
Maximum Value : 18.028251174540898
Minimum Value : 16.28619476441503
```

```
Part 5
Visualise the dataset by using a histogram with 10 bins. Visualise the probability density
function.
Probability density function f:
{f}
Where X represents the data points, μ is the mean value and σ is the standard deviation
```

```
# creating a histogram for OneDimensionalArray with 10 bins
A, X, C = plt.hist(OneDimensionalArray, bins=10)
# displaying the histogram
plt.show()
```




```
# calculating the probability density function
f = (1/(SD*np.sqrt(2*np.pi)))*np.exp(-((X-Mean)**2/(2*SD**2)))
# Plotting the probability density function
plt.plot(X, f)
# displaying  the plot
plt.show()
```



3. Linear Algebra

```
Task 1
Initialize the Matrix A with given integer values. Find the determinant, Trace and Inverse of Matrix A
```

```
# initiaizing matrix A with given integer values
A= np.array([[2,5,1],[4,3,7],[1,3,2]], dtype='int')
# finding determinant of matrix A
D=np.linalg.det(A)
# finding trace of matrix A
T=np.trace(A)
# finding inverse of matrix A
I=np.linalg.inv(A)
# displaying Matrix A
print('Matrix A= \n',A)
# displaying determinant of Matrix A
print('Determinant of Matrix A= ',D)
# displaying trace of Matrix A
print('Trace of Matrix A= ',T)
# displaying Inverse of Matrix a
print('Inverse of Matrix A= \n',I)
```

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

```
Task 2
Initialize the following square matrices B and C.
Find the product P of the matrices B and C by using the Python function for matrix
multiplication. Display the result on the screen.
```

```
# Initializing Matrices B and C with given integer values
B=np.array([[4,7,2],[3,2,5],[6,4,3]], dtype='int')
C= np.array([[3,1,9],[7,5,8],[2,1,1]], dtype='int')
# Multiplying Matrices B and C and storing in Matrix P
P= np.matmul(B,C)
# displaying Matrices B and C
print('Matrix B= \n',B)
print('Matrix C= \n',C)
# displaying product of B and C as P
print('Product of Matrices B and C is \n',P)
```

```
Matrix B=
 [[4 7 2]
 [3 2 5]
 [6 4 3]]
Matrix C=
 [[3 1 9]
 [7 5 8]
 [2 1 1]]
Product of Matrices B and C is
 [[65 41 94]
 [33 18 48]
 [52 29 89]]
```

```
Task 3
Consider the following system of linear equations:
3x+2y-z=25
2x-y+4z=19
4x-2y+3z=18
Represent the system of linear equations using Matrix
```

$$\begin{matrix} 3 & 2 & -1 \\ 2 & -1 & 4 \\ 4 & -2 & 3 \end{matrix} \begin{matrix} x \\ y \\ z \end{matrix} = \begin{matrix} 25 \\ 19 \\ 18 \end{matrix}$$

```
Task 4
Provide the algebraic steps for solving the system of linear equations from Task 3 by
using matrix notation.
```

The steps to solve the above linear equations is given below:

MX=C

M^{-1}MX = M^{-1}C

IX = M^{-1}C

X = M^{-1}C

```
Task 5
Solve the system of linear equations from Task 3 by using Python script utilising matrix
multiplication and inverse matrix.
```

```
# Initializing Matrix M with given integer values
M=np.array([[3,2,-1],[2,-1,4],[4,-2,3]], dtype='int')
# Initializing the Constant in C
C=np.array([[25],[19],[18]], dtype='int')
# finding the inverse of Matrix M
I= np.linalg.inv(M)
# finding the product of inverse and Constant
X=np.matmul(I,C)
# displaying the product as X
print(X)
```

```
[[5.]
 [7.]
 [4.]]
```

4. Data Visualization

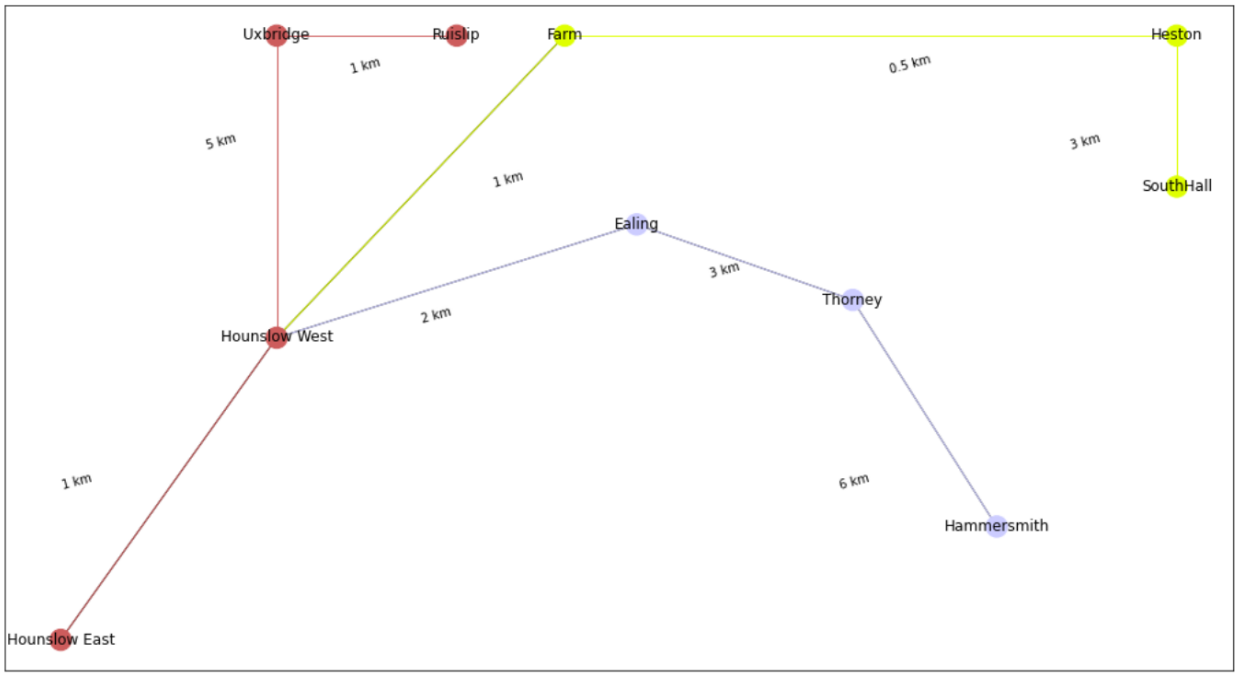
```
Task 1
Develop a graph which represents the public transport network of a city of your choice.
Some cities have extensive public transport networks. In such case, represent minimum
3 lines with minimum 4 stations on each line. When visualising the network, use different
colours for the different 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.
```

```
# importing networkx library as nx
import networkx as nx

# Creating a graph object
MyNetworkGraph = nx.Graph()

# Adding the  nodes
# IndianRed Line
MyNetworkGraph.add_node('Hounslow East', npos=(20, 10), ccn='#CD5C5C')
MyNetworkGraph.add_node('Hounslow West', npos=(50, 50), ccn='#CD5C5C')
MyNetworkGraph.add_node('Uxbridge', npos=(50, 90), ccn='#CD5C5C')
MyNetworkGraph.add_node('Ruislip', npos=(75, 90), ccn='#CD5C5C')
# Yellow Line
MyNetworkGraph.add_node('Farm', npos=(90, 90), ccn='#DFFF00')
MyNetworkGraph.add_node('Heston', npos=(175, 90), ccn='#DFFF00')
MyNetworkGraph.add_node('SouthHall', npos=(175, 70), ccn='#DFFF00')
# Grey Line
MyNetworkGraph.add_node('Hammersmith', npos=(150, 25), ccn='#CCCCFF')
MyNetworkGraph.add_node('Thorney', npos=(130, 55), ccn='#CCCCFF')
MyNetworkGraph.add_node('Ealing', npos=(100, 65), ccn='#CCCCFF')
```

```
# Connecting the nodes
# IndianRed Line
MyNetworkGraph.add_edge('Hounslow East', 'Hounslow West', cce='#CD5C5C')
MyNetworkGraph.add_edge('Hounslow West', 'Uxbridge', cce='#CD5C5C')
MyNetworkGraph.add_edge('Uxbridge', 'Ruislip', cce='#CD5C5C')
# Yellow Line
MyNetworkGraph.add_edge('Hounslow West', 'Farm', cce='#DFFF00')
MyNetworkGraph.add_edge('Farm', 'Heston', cce='#DFFF00')
MyNetworkGraph.add_edge('Heston', 'SouthHall', cce='#DFFF00')
# Grey Line
MyNetworkGraph.add_edge('Hammersmith', 'Thorney', cce='#CCCCFF')
MyNetworkGraph.add_edge('Thorney', 'Ealing', cce='#CCCCFF')
MyNetworkGraph.add_edge('Ealing', 'Hounslow West', cce='#CCCCFF')

# Extract attributes from the graph to dictionaries
pos = nx.get_node_attributes(MyNetworkGraph, 'npos')
nodecolour = nx.get_node_attributes(MyNetworkGraph, 'ccn')
edgecolour = nx.get_edge_attributes(MyNetworkGraph, 'cce')

# Place the dictionary values in lists
NodeList = list(nodecolour.values())
EdgeList = list(edgecolour.values())

# Set the size of the figure
plt.figure(figsize=(18, 10))
```

```
# Display the names of the stations
plt.text(20, 30, s='1 km', rotation=15)
plt.text(32, 45, s='', rotation=15)
plt.text(40, 75, s='5 km ', rotation=15)
plt.text(60, 85, s='1 km', rotation=15)

plt.text(80, 70, s='1 km', rotation=15)
plt.text(135, 85, s='0.5 km', rotation=15)
plt.text(160, 75, s='3 km', rotation=15)

plt.text(128, 30, s='6 km', rotation=15)
plt.text(110, 58, s='3 km', rotation=15)
plt.text(70, 52, s='2 km', rotation=15)

# Draw the nodes and the edges
nx.draw_networkx(MyNetworkGraph, pos, node_color=NodeList)
nx.draw_networkx_edges(MyNetworkGraph, pos, edge_color=EdgeList)

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

Output:



Task 2
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.

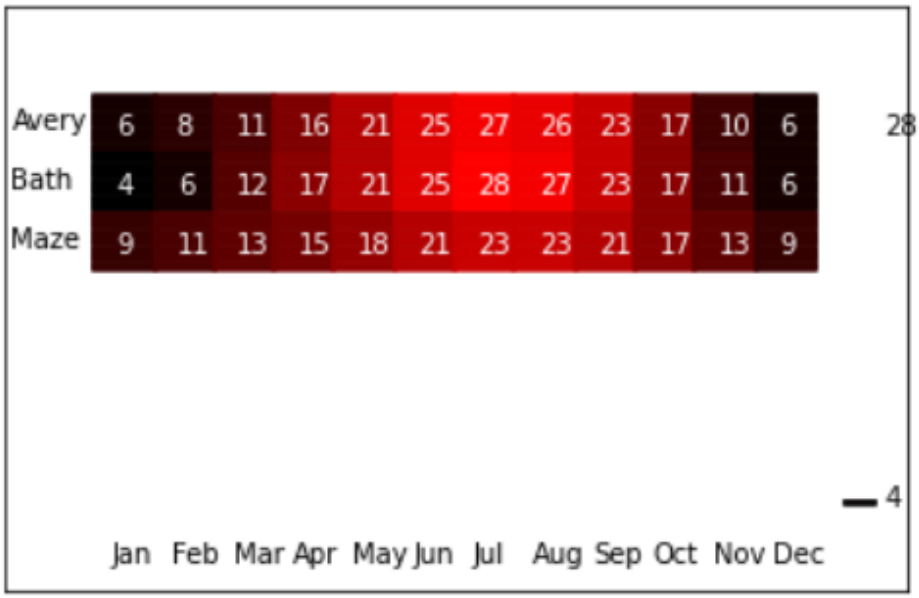
```
# importing pandas library as pd
import pandas as pd
# This subroutine encapsulates the 'plot' method
def DrawBox(x, y, size, r, g, b):
    if r < 0:
        r = int(0)
    if g < 0:
        g = int(0)
    if b < 0:
        b = int(0)
    if r > 255:
        r = int(255)
    if g > 255:
        g = 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))
```

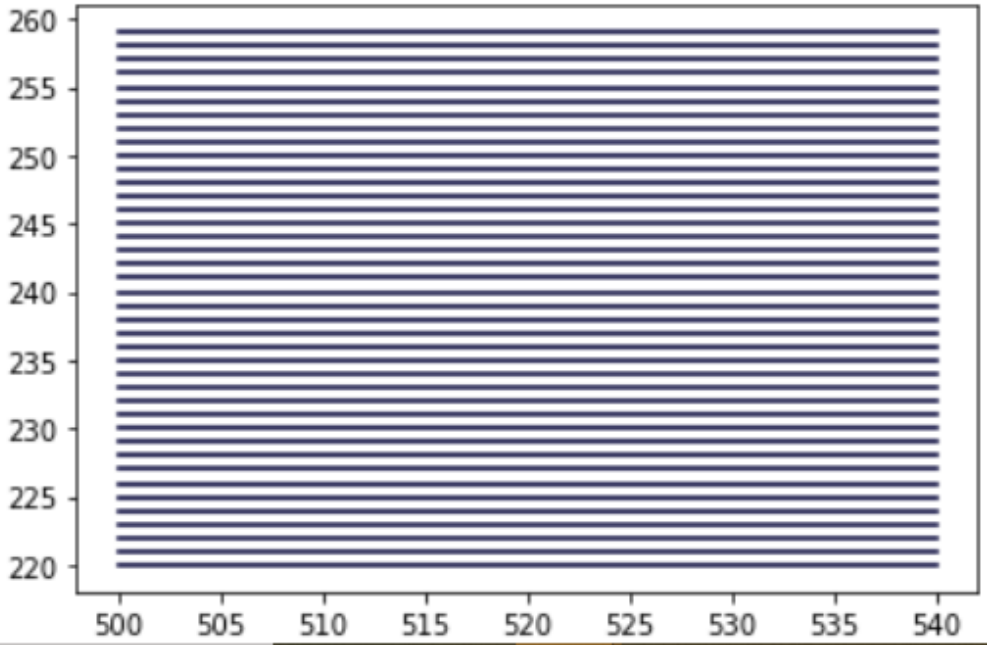
```
# Store the dataset into a data frame
df = pd.read_csv('C:/Users/Rabin Thapa Kshetry/Downloads/HeatMap.csv')
# Print the content on the screen
print(df.head())

# Set the plot
plt.axis([0, 600, 0, 400])
plt.xticks([])
plt.yticks([])

# minimum and maximum
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)

# Generating 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)
        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')
```

```
# Generating the scale
for i in range(0, 3):
    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, 20, 'Jan')
plt.text(112, 20, 'Feb')
plt.text(152, 20, 'Mar')
plt.text(192, 20, 'Apr')
plt.text(232, 20, 'May')
plt.text(272, 20, 'Jun')
plt.text(312, 20, 'Jul')
plt.text(352, 20, 'Aug')
plt.text(392, 20, 'Sep')
plt.text(432, 20, 'Oct')
plt.text(472, 20, 'Nov')
plt.text(512, 20, 'Dec')

plt.text(5, 315, str(df.values[0, 0]))
plt.text(5, 275, str(df.values[1, 0]))
plt.text(5, 235, str(df.values[2, 0]))

#displaying the heat map
plt.show()
# maintaining the threshold
Threshold=int(3)
# comparing with the thrreshold
if df.values[i, j] > Threshold:
    DrawBox(20+BoxSize*j, 300-BoxSize*i, BoxSize, 0, 0, ColourCode)
if df.values[i, j] <= Threshold:
    DrawBox(20+BoxSize*j, 300-BoxSize*i, BoxSize, ColourCode, 0, 0)
```

Output:

	City	1	2	3	4	5	6	7	8	9	10	11	12
0	Avery	6	8	11	16	21	25	27	26	23	17	10	6
1	Bath	4	6	12	17	21	25	28	27	23	17	11	6
2	Maze	9	11	13	15	18	21	23	23	21	17	13	9





Part 2: R Portfolio Tasks

1. Data Transformation

Q1.

(2 marks) Gather together all the columns from _new_sp*m014* to _newrel*f65*

- Use pivot_longer()
- We do not know what those values represent yet, so we give them the generic name as "key"
- We know the cells represent the count of cases, so we use the variable cases.
- There are a lot of missing values in the current representation, so for now we use values_drop_na just so we can focus on the values that are present.
- Name the new dataset **who1**

Code:

```
library(tidyr)
library(dplyr)
library(readr)
library(tidyverse)
# accessing the who data using tidyr::who
who<-tidyr::who

# viewing the data
View(who)
# gathering together the data from multiple columns
# renaming new data as who1
who1<-who %>% pivot_longer(
  cols = new_sp_m014:newrel_f65,
  names_to = "key",
  values_to = "cases",
  values_drop_na = TRUE
)
# viewing the who1 dataset
who1<-who
```

Output:

	country	iso2	iso3	year	key	cases
1	Afghanistan	AF	AFG	1997	new_sp_m014	0
2	Afghanistan	AF	AFG	1997	new_sp_m1524	10
3	Afghanistan	AF	AFG	1997	new_sp_m2534	6
4	Afghanistan	AF	AFG	1997	new_sp_m3544	3
5	Afghanistan	AF	AFG	1997	new_sp_m4554	5
6	Afghanistan	AF	AFG	1997	new_sp_m5564	2
7	Afghanistan	AF	AFG	1997	new_sp_m65	0
8	Afghanistan	AF	AFG	1997	new_sp_f014	5
9	Afghanistan	AF	AFG	1997	new_sp_f1524	38
10	Afghanistan	AF	AFG	1997	new_sp_f2534	36
11	Afghanistan	AF	AFG	1997	new_sp_f3544	14

Q2.

(2 marks) Make variable names consistent

Instead of _new*rel* we have *newrel*. It is hard to spot this here but if you do not fix it, we will get errors in subsequent steps.

- Use stringr::str_replace() in strings: replace the characters "*newrel*" with "*new_rel*".
- Name the dataset **who2**

```
# replacing newrel with new_rel in key column
# renaming updated dataset as who2
who2<- who1 %>% stringr::str_replace (key, "newrel", "new_rel")
# viewing who2
View(who2)
```

Q3.

(2 mark) Run the following code

```
who3 <- who2 %>%

separate(key, c("new", "type", "sexage"), sep = "_")
```

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

Ans: %>% is used to update a value by initially piping it to one or more expressions and then assigning the final result to some other variable.

Here, initially it is updated using who2 and the final result is placed in who3.

Q4. (1 mark) Separate sexage into sex and age: Use the function separate(). Name the dataset who4

Ans: who4 <- who3 %>% separate(key, c("new", "type", "sexage"), sep = "_")

Q5. (2 mark) Print the first 5 rows and the last 5 rows of the dataset who4 to the screen.

Ans: head(who4,n= 5)

Tail(who4, n=5)

```
  country     iso2  iso3   year key          cases
  <chr>       <chr> <chr> <int> <chr>        <int>
1 Afghanistan AF    AFG    1997 new_sp_m014      0
2 Afghanistan AF    AFG    1997 new_sp_m1524    10
3 Afghanistan AF    AFG    1997 new_sp_m2534     6
4 Afghanistan AF    AFG    1997 new_sp_m3544     3
5 Afghanistan AF    AFG    1997 new_sp_m4554     5
> |
```

Q6. (1 mark) Export who4 as an csv file and save it in your local directory.

Ans: write.csv(who4, “C:/Users/Desktop/who4.csv”)

2. Basic Statistics

Q1. Compute the mean, median and mode of sepal length (2 marks).

R code as follows:

```
# viewing the heads of iris dataset
head(iris)
# calculating the mean of sepal length
mean.(iris$Sepal.Length, na.rm=TRUE)
# calculating the median of sepal length
median(iris$Sepal.Length, na.rm=TRUE)
# calculating the mode of sepal length
mode(iris$Sepal.Length)
```

Output:

```
> # viewing the heads of iris dataset
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
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
> # calculating the mean of sepal length
> mean(iris$Sepal.Length)
[1] 5.843333
> # calculating the median of sepal length
> median(iris$Sepal.Length)
[1] 5.8
> # calculating the mode of sepal length
> mode(iris$Sepal.Length)
[1] "numeric"
```

Q2. Compute how "spread out" the data are. Here you need to calculate the minimum, maximum and range of sepal length (2 marks).

R code:

```
# calculating the minimum of sepal length
min(iris$Sepal.Length, na.rm=TRUE)
# calculating the maximum of sepal length
max(iris$Sepal.Length, na.rm=TRUE)
# calculating the range of sepal length
range(iris$Sepal.Length, na.rm=TRUE)
```

Output:

```
> # calculating the minimum of sepal length
> min(iris$Sepal.Length, na.rm=TRUE)
[1] 4.3
> # calculating the maximum of sepal length
> max(iris$Sepal.Length, na.rm=TRUE)
[1] 7.9
> # calculating the range of sepal length
> range(iris$Sepal.Length, na.rm=TRUE)
[1] 4.3 7.9
>
```

Q3. Calculate the interquartile (IQR) range of sepal length (1 mark). 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? (2 marks)

Code:

```
# calculating the inter quartile range of sepal length
IQR(iris$Sepal.Length, na.rm=TRUE)
# calculating the quantiles of sepal length
quantile(iris$Sepal.Length, na.rm=TRUE)
```

Output:

```
> # calculating the inter quartile range of sepal length
> IQR(iris$Sepal.Length, na.rm=TRUE)
[1] 1.3
> # calculating the quantiles of sepal length
> quantile(iris$Sepal.Length, na.rm=TRUE)
  0%  25%  50%  75% 100%
 4.3  5.1  5.8  6.4  7.9
```

IQR() directly gives inter quartile range value where quantile() gives values of each quarter so the relation is IQR= Q3-Q2.

Q4. Compute the variance (1 mark) and standard deviation of sepal length (1 mark).

```
# calculating the variance of sepal length
var(iris$Sepal.Length, na.rm=TRUE)
# calculating the standard deviation of sepal length
sd(iris$Sepal.Length, na.rm=TRUE)
```

Output:

```
> var(iris$Sepal.Length, na.rm=TRUE)
[1] 0.6856935
> # calculating the standard deviation of sepal length
> sd(iris$Sepal.Length, na.rm=TRUE)
[1] 0.8280661
```

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

```
# finding min, max, mean, median, 1st and 3rd quartiles of sepal length at once
summary(iris$Sepal.Length)
```

Output:

```
# finding min, max, mean, median, 1st and 3rd quartiles of sepal length at once
summary(iris$Sepal.Length)
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
4.300   5.100   5.800   5.843   6.400   7.900
```

Q6. Use sapply() to compute the mean (1 marks) and quantiles (1 marks) of each column in the dataset iris.

```
#Using sapply to compute mean and quantile for each column
sapply(iris, mean, na.rm=TRUE)
sapply(iris$Sepal.Length, quantile)
```

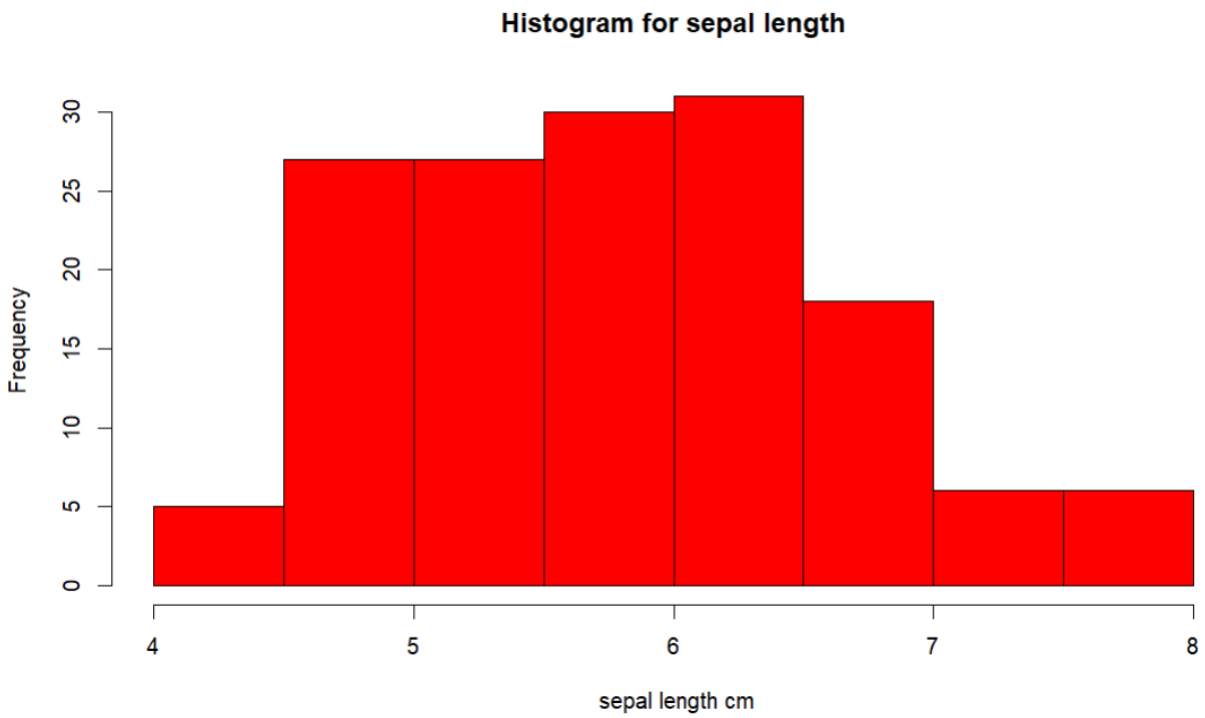
Output:

```
> sapply(iris, mean, na.rm=TRUE)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
    5.843333     3.057333     3.758000     1.199333           NA

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17]
0%    5.1  4.9  4.7  4.6    5  5.4  4.6    5  4.4   4.9   5.4   4.8   4.8   4.3   5.8   5.7   5.4
25%   5.1  4.9  4.7  4.6    5  5.4  4.6    5  4.4   4.9   5.4   4.8   4.8   4.3   5.8   5.7   5.4
50%   5.1  4.9  4.7  4.6    5  5.4  4.6    5  4.4   4.9   5.4   4.8   4.8   4.3   5.8   5.7   5.4
75%   5.1  4.9  4.7  4.6    5  5.4  4.6    5  4.4   4.9   5.4   4.8   4.8   4.3   5.8   5.7   5.4
100%  5.1  4.9  4.7  4.6    5  5.4  4.6    5  4.4   4.9   5.4   4.8   4.8   4.3   5.8   5.7   5.4
     [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33]
0%     5.1   5.7   5.1   5.4   5.1   4.6   5.1   4.8     5     5   5.2   5.2   4.7   4.8   5.4   5.2
25%    5.1   5.7   5.1   5.4   5.1   4.6   5.1   4.8     5     5   5.2   5.2   4.7   4.8   5.4   5.2
50%    5.1   5.7   5.1   5.4   5.1   4.6   5.1   4.8     5     5   5.2   5.2   4.7   4.8   5.4   5.2
75%    5.1   5.7   5.1   5.4   5.1   4.6   5.1   4.8     5     5   5.2   5.2   4.7   4.8   5.4   5.2
100%   5.1   5.7   5.1   5.4   5.1   4.6   5.1   4.8     5     5   5.2   5.2   4.7   4.8   5.4   5.2
     [,34] [,35] [,36] [,37] [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49]
0%     5.5   4.9     5   5.5   4.9   4.4   5.1     5   4.5   4.4     5   5.1   4.8   5.1   4.6   5.3
25%    5.5   4.9     5   5.5   4.9   4.4   5.1     5   4.5   4.4     5   5.1   4.8   5.1   4.6   5.3
50%    5.5   4.9     5   5.5   4.9   4.4   5.1     5   4.5   4.4     5   5.1   4.8   5.1   4.6   5.3
75%    5.5   4.9     5   5.5   4.9   4.4   5.1     5   4.5   4.4     5   5.1   4.8   5.1   4.6   5.3
100%   5.5   4.9     5   5.5   4.9   4.4   5.1     5   4.5   4.4     5   5.1   4.8   5.1   4.6   5.3
     [,50] [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62] [,63] [,64] [,65]
0%      5     7   6.4   6.9   5.5   6.5   5.7   6.3   4.9   6.6   5.2     5   5.9     6   6.1   5.6
25%     5     7   6.4   6.9   5.5   6.5   5.7   6.3   4.9   6.6   5.2     5   5.9     6   6.1   5.6
50%     5     7   6.4   6.9   5.5   6.5   5.7   6.3   4.9   6.6   5.2     5   5.9     6   6.1   5.6
75%     5     7   6.4   6.9   5.5   6.5   5.7   6.3   4.9   6.6   5.2     5   5.9     6   6.1   5.6
100%    5     7   6.4   6.9   5.5   6.5   5.7   6.3   4.9   6.6   5.2     5   5.9     6   6.1   5.6
     [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74] [,75] [,76] [,77] [,78] [,79] [,80] [,81]
0%     6.7   5.6   5.8   6.2   5.6   5.9   6.1   6.3   6.1   6.4   6.6   6.8   6.7     6   5.7   5.5
25%    6.7   5.6   5.8   6.2   5.6   5.9   6.1   6.3   6.1   6.4   6.6   6.8   6.7     6   5.7   5.5
50%    6.7   5.6   5.8   6.2   5.6   5.9   6.1   6.3   6.1   6.4   6.6   6.8   6.7     6   5.7   5.5
75%    6.7   5.6   5.8   6.2   5.6   5.9   6.1   6.3   6.1   6.4   6.6   6.8   6.7     6   5.7   5.5
100%   6.7   5.6   5.8   6.2   5.6   5.9   6.1   6.3   6.1   6.4   6.6   6.8   6.7     6   5.7   5.5
     [,82] [,83] [,84] [,85] [,86] [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97]
0%     5.5   5.8     6   5.4     6   6.7   6.3   5.6   5.5   5.5   6.1   5.8     5   5.6   5.7   5.7
25%    5.5   5.8     6   5.4     6   6.7   6.3   5.6   5.5   5.5   6.1   5.8     5   5.6   5.7   5.7
50%    5.5   5.8     6   5.4     6   6.7   6.3   5.6   5.5   5.5   6.1   5.8     5   5.6   5.7   5.7
75%    5.5   5.8     6   5.4     6   6.7   6.3   5.6   5.5   5.5   6.1   5.8     5   5.6   5.7   5.7
100%   5.5   5.8     6   5.4     6   6.7   6.3   5.6   5.5   5.5   6.1   5.8     5   5.6   5.7   5.7
     [,98] [,99] [,100] [,101] [,102] [,103] [,104] [,105] [,106] [,107] [,108] [,109] [,110] [,111]
0%     6.2   5.1    5.7    6.3    5.8    7.1    6.3    6.5    7.6    4.9    7.3    6.7    7.2    6.5
25%    6.2   5.1    5.7    6.3    5.8    7.1    6.3    6.5    7.6    4.9    7.3    6.7    7.2    6.5
50%    6.2   5.1    5.7    6.3    5.8    7.1    6.3    6.5    7.6    4.9    7.3    6.7    7.2    6.5
75%    6.2   5.1    5.7    6.3    5.8    7.1    6.3    6.5    7.6    4.9    7.3    6.7    7.2    6.5
100%   6.2   5.1    5.7    6.3    5.8    7.1    6.3    6.5    7.6    4.9    7.3    6.7    7.2    6.5
     [,112] [,113] [,114] [,115] [,116] [,117] [,118] [,119] [,120] [,121] [,122] [,123] [,124]
0%      6.4    6.8    5.7    5.8    6.4    6.5    7.7    7.7      6    6.9    5.6    7.7    6.3
25%     6.4    6.8    5.7    5.8    6.4    6.5    7.7    7.7      6    6.9    5.6    7.7    6.3
50%     6.4    6.8    5.7    5.8    6.4    6.5    7.7    7.7      6    6.9    5.6    7.7    6.3
75%     6.4    6.8    5.7    5.8    6.4    6.5    7.7    7.7      6    6.9    5.6    7.7    6.3
100%    6.4    6.8    5.7    5.8    6.4    6.5    7.7    7.7      6    6.9    5.6    7.7    6.3
     [,125] [,126] [,127] [,128] [,129] [,130] [,131] [,132] [,133] [,134] [,135] [,136] [,137]
```

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: (4 marks).

```
# creating histogram
hist(iris$Sepal.Length, main = "Histogram for sepal length", xlab=" sepal length cm", xlim = c(4,8), col = "red")
```

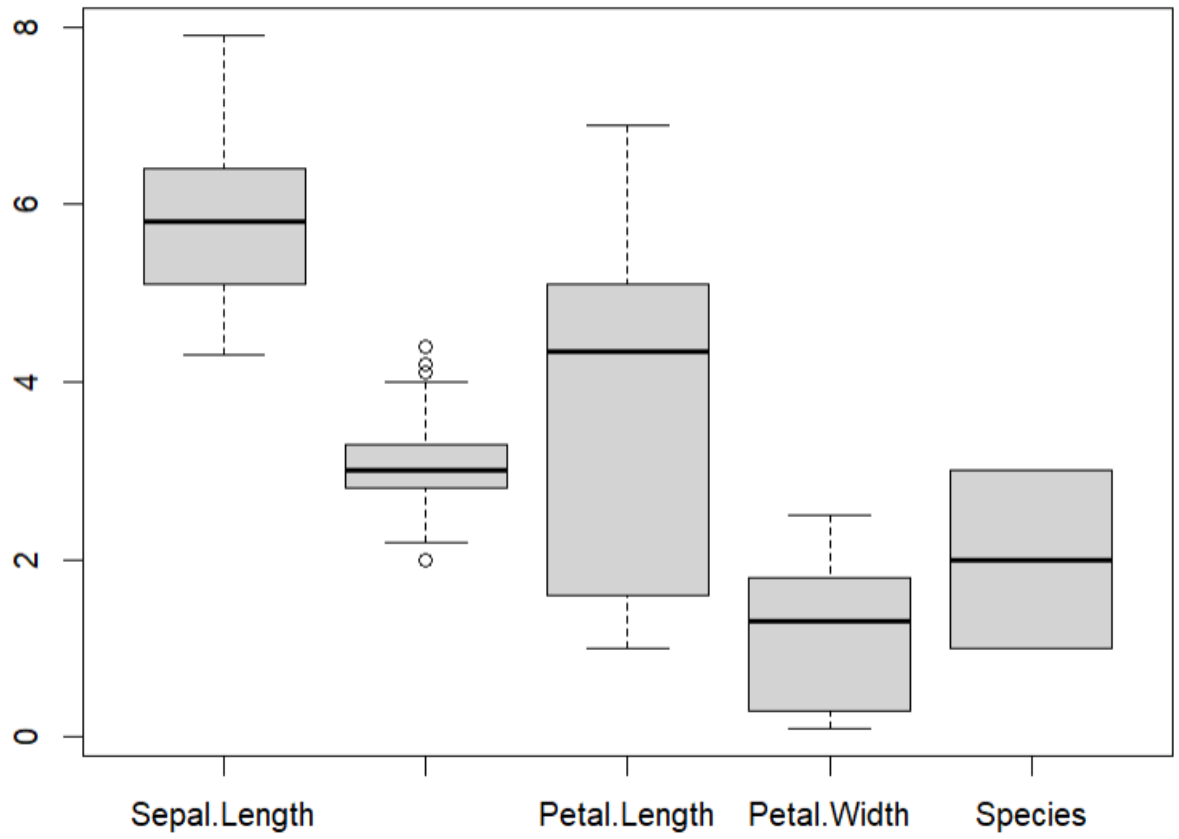
Output:



Q8. Use the in-built R basic function to create one boxplot for sepal length, sepal width, petal length and petal width (2 marks).

```
# creating boxplot for sepal length, sepal width, petal length, petal width
boxplot(iris)
```

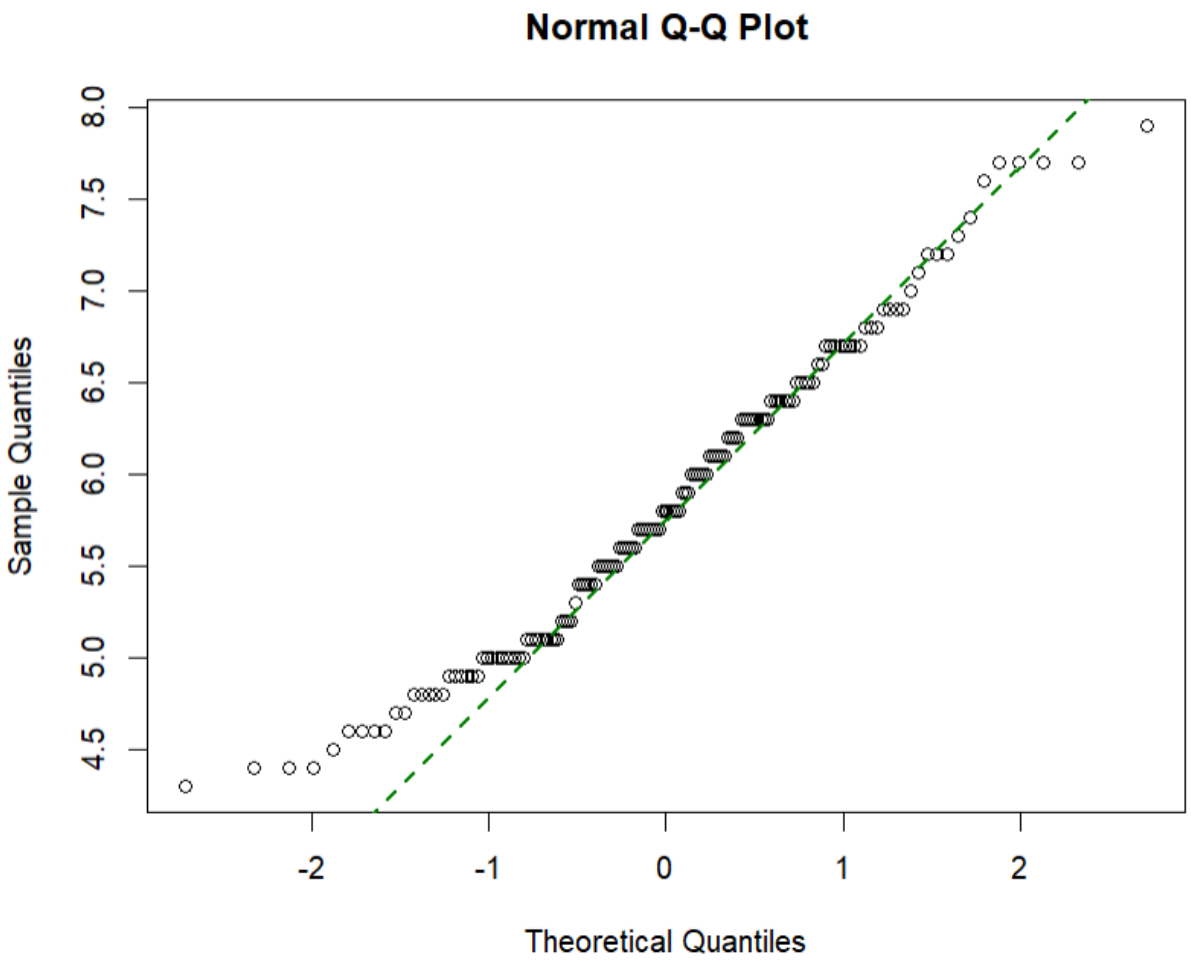
Output:



Q9. Use these two functions (qqnorm() and qqline()) to create a QQ plot for sepal length. You will need to set the reference line colour as green, and its width as 2) (2 marks).

```
# QQ plot for sepal length
qqnorm(iris$Sepal.Length)
# QQ plot for sepal length with reference line
qqline(iris$Sepal.Length, col='#008000', lwd=2, lty=2)
```

Output:



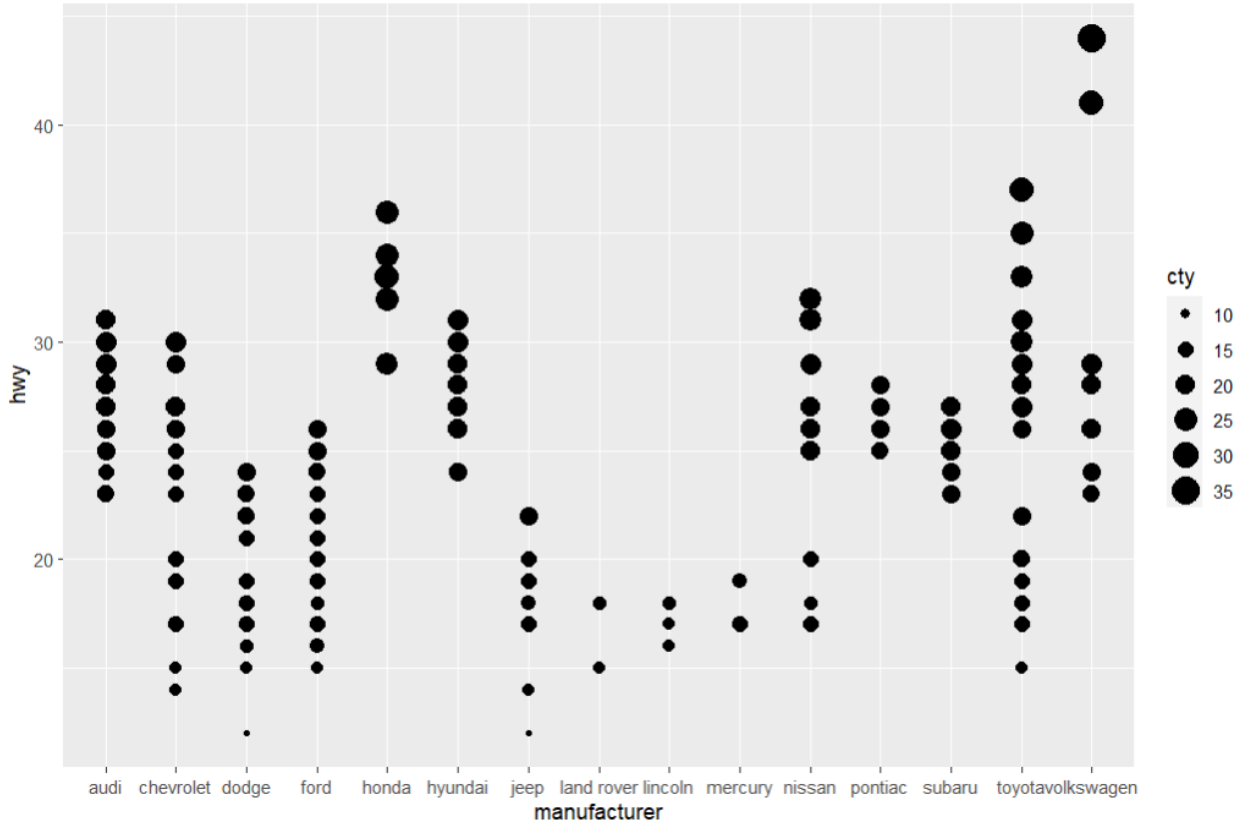
3. Data Visualization

Q1. Plot and explain: Which vehicle brand (or manufacturer), offers the best mpg in both city and in the highway? (6 marks).

Code:

```
# using ggplot to visualize between manufacturer and city and highway
ggplot(mpg, aes(x = manufacturer, y = hwy, size = cty)) +
  geom_point()
```

Output:

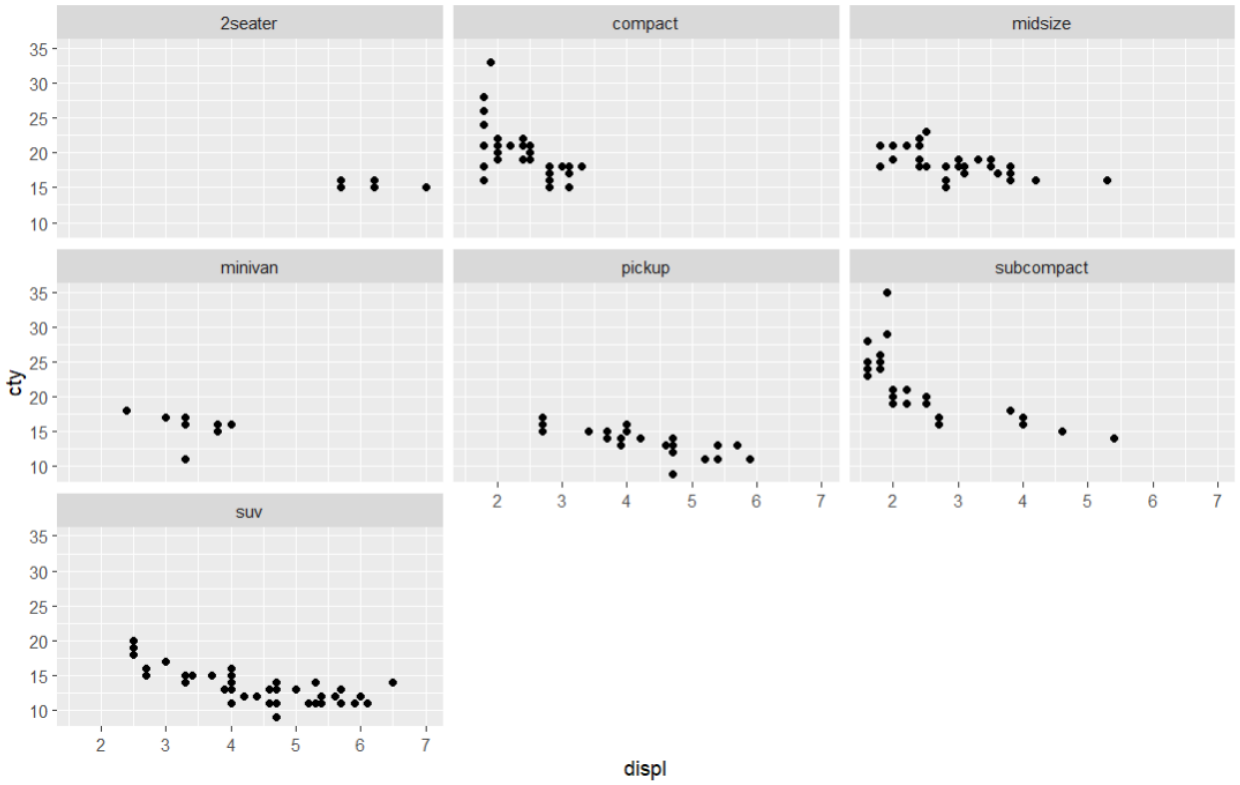


As per the above graph, I noticed that the manufacturer Volkswagen offers best mpg both in highway and city.

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)? Display the resulting plot categorised by the vehicle type. (6 marks)

```
# using facet_wrap to categorise the variables
ggplot(mpg, aes(x = displ, y = cty)) +
  geom_point() +
  facet_wrap( ~ class)
```

Output:



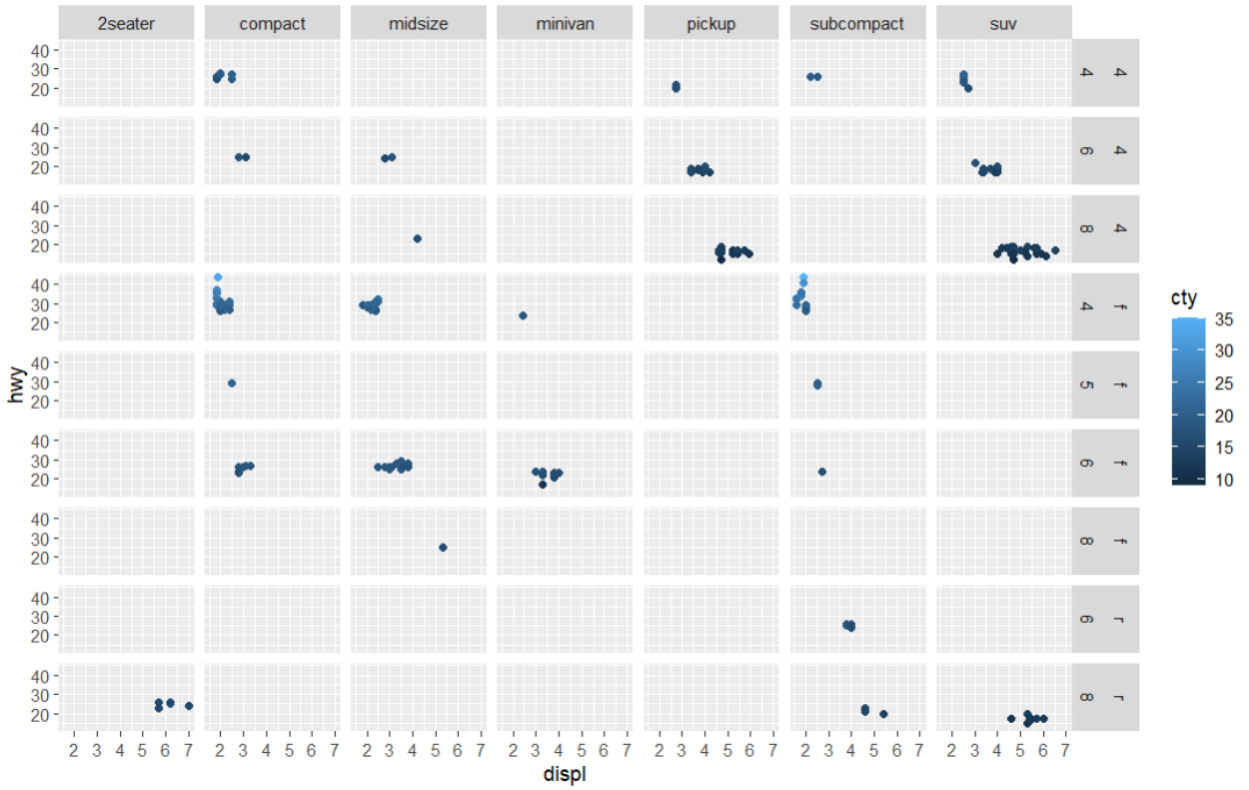
As per the above graph, I found that pickup vans have comparatively lower mpg value than other vehicle types in the cities according to their engine size.

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? (8 marks)

Code:

```
#using facet_grid to categorise with multiple variables
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, colour = cty)) +
  facet_grid(drv ~ cyl~class)
```

Output:



The above graph shows that the compact cars have best performance value in both city and highways regarding the size of their engine.so if I had to buy a car then I would prefer to buy compact cars as they high mpg performance value above 40.