Data Processing

Task 1

import numpy, pandas and calendar libraries.

```
import numpy as np
import pandas as pd
import calendar
```

Initialize 12 random integers from the range [65...75]

```
#set seed so that results can be replicated
np.random.seed(12)

#Initialize 12 random integers within the range [65..75]
rints = np.random.randint(65, high = 75, size = 12)

#visualize the generated random integers
print(rints)
```

```
[71 66 67 68 68 65 71 66 69 70 74 67]
```

Create a pandas series

```
#create a pandas series and visualize the series
mean_temps = pd.Series(rints, dtype=int)
print(mean_temps)
```

```
0     71
1     66
2     67
3     68
4     68
5     65
6     71
7     66
8     69
9     70
10    74
11    67
dtype: int32
```

Set the indices of the series as full names of months

```
#set the indices of the pandas series as the full names of the months and visualize the series object
months = ["January","February","March","April","May","June","July","August","September","October","November","
mean_temps = pd.Series(rints, index = months, dtype=int)
print(mean_temps)
```

```
January      71
February     66
March        67
April        68
May          68
June         65
July         71
August       66
September    69
October      70
November     74
December     67
dtype: int32
```

Convert temperatures from Fahrenheit into Celsius

```
#Convert the temperatures from Fahrenheit into Celsius and visualize them Note: formula 5/9 x (F-32)
mean_temps_celcius = (mean_temps.iloc[:] - 32)*5/9

#convert the series to integers
mean_temps_celcius_ints = mean_temps_celcius.astype(int)

print(mean_temps_celcius_ints)
```

```
0     21
1     18
2     19
3     20
4     20
5     18
6     21
7     18
8     20
9     21
10    23
11    19
dtype: int32
```

Find mode

```
#determine the mode
mean_temps_celcius_ints.mode()
```

```
0    18
1    20
2    21
dtype: int32
```

Number of months average was recorded

This can be accomplished by using the *len* function.

```
#number of months this average temperature was recorded
num_months = len(mean_temps_celcius_ints.mode())
print(num_months)
```

```
3
```

Task 2

1. Download dataset from (https://www.kaggle.com/camnugent/california-housing-prices)

Employ *pd.read_csv* function to load the dataset into memory. Calling the shape function on the data frame returns a tuple of the number of rows and the number of columns.

```
path = "C:\\Users\\joshua\\Desktop\\ASSIGNMENTS\\5582744_1500739164_ProgrammingFundamentals\\Programming Fundam
housing = pd.read_csv(path)
housing.shape
```

```
(20640, 10)
```

2. Check the data frame for Missing values and resolve

Chain *isnull* and *any* function on the data frame. This will show whether any columns contain any missing values ie null values.

```
#this operation reveals that there is some missing values in the column total_bedrooms
print(housing.isnull().any())
```

```
longitude             False
latitude              False
housing_median_age    False
total_rooms           False
total_bedrooms         True
population            False
households            False
median_income         False
median_house_value    False
ocean_proximity       False
dtype: bool
```

```
# creating bool series True for NaN values
bool_total_bedrooms = pd.isnull(housing["total_bedrooms"])

#filter to see missing values of column total_bedrooms
housing[bool_total_bedrooms]
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
290	-122.16	37.77	47.0	1256.0	NaN	570.0	218.0	4.3750	161900.0
341	-122.17	37.75	38.0	992.0	NaN	732.0	259.0	1.6196	85100.0
538	-122.28	37.78	29.0	5154.0	NaN	3741.0	1273.0	2.5762	173400.0
563	-122.24	37.75	45.0	891.0	NaN	384.0	146.0	4.9489	247100.0
696	-122.10	37.69	41.0	746.0	NaN	387.0	161.0	3.9063	178400.0
...	...	...	...	...	...	...	...	...	...
20267	-119.19	34.20	18.0	3620.0	NaN	3171.0	779.0	3.3409	220500.0
20268	-119.18	34.19	19.0	2393.0	NaN	1938.0	762.0	1.6953	167400.0
20372	-118.88	34.17	15.0	4260.0	NaN	1701.0	669.0	5.1033	410700.0
20460	-118.75	34.29	17.0	5512.0	NaN	2734.0	814.0	6.6073	258100.0
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376	218600.0

207 rows × 10 columns

```
#Given that the dataset has over 20,000 rows, we can afford to fix the data by dropping 207 the missing values
housing1 = housing.dropna(axis = 0, how ='any')
housing1.shape
```

```
(20433, 10)
```

3. Check the data frame for Unnecessary duplicates and resolve

Based on which column are we checking for duplicates?

```
#The new dataset is held in variable housing1
housing1.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0

```
#select the first column of the data frame and call method duplicated and then call method any on it
housing1.iloc[:, 0].duplicated().shape
```

```
(20433,)
```

4. Check the data frame for Wrong data types and resolve

```
(20433,)
False
```

5. Check the data frame for Wrong values and resolve

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

7. Describe the Mean of column 'median_house_value'

Call function *mean* on the Series object.

```
housing1["median_house_value"].mean()
```

```
206864.41315519012
```

8. Describe the Median of column 'median_house_value'

Call function *median* on the Series object.

```
housing1["median_house_value"].median()
```

```
179700.0
```

9. Describe the Range of column 'median_house_value'

```
housing1["median_house_value"].describe()
```

```
count     20433.000000
mean     206864.413155
std      115435.667099
min       14999.000000
25%      119500.000000
50%      179700.000000
75%      264700.000000
max      500001.000000
Name: median_house_value, dtype: float64
```

10. The column 'median_income' contains currency in tens of thousands USD.

Convert it into USD and visualise the entire updated data frame

```
housing1["median_house_value"] = housing1["median_house_value"] * 10000
housing1.head()
```

```
<ipython-input-23-99b7521b875c>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  housing1["median_house_value"] = housing1["median_house_value"] * 10000
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0

Statistics with Python

Task 1

Part 1

import numpy, pandas and the matplotlib libraries.

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

Initialise an array representing a uniformly distributed dataset containing 20 data points of integer type with minimum value 0 and maximum value 100. Visualise the values of the datapoints and plot the dataset.

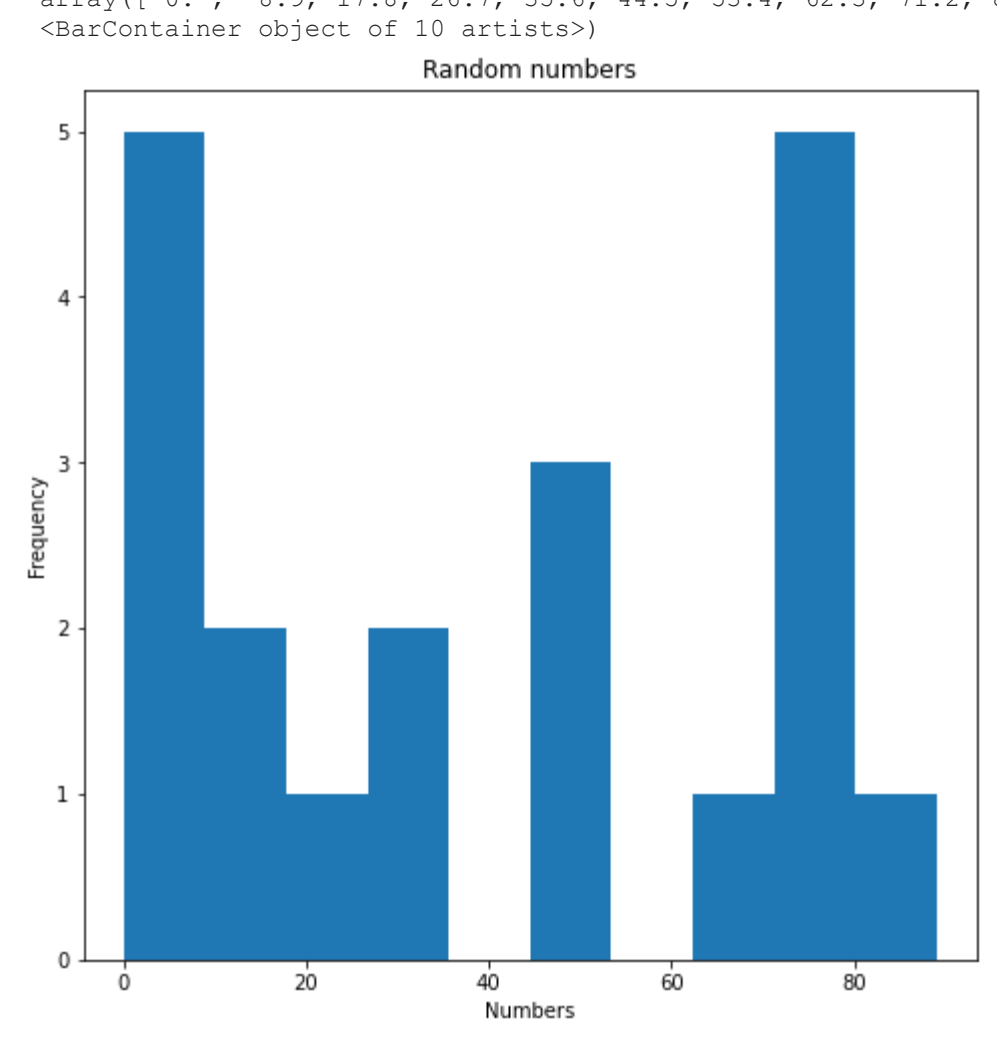
```
#ensure our results are replicated
np.random.seed(12)

#Initialise an array representing a uniformly distributed
vals = np.random.randint(low=0, high=100, size=20, dtype=int)

print(vals)
```

```
[75 27  6  2  3 67 76 48 22 49 52  5 13 89 34 75 74  0 76 13]
```

A histogram of random numbers

```
#plot the dataset
plt.figure(figsize=(8, 8))
plt.title("Random numbers")
plt.xlabel("Numbers")
plt.ylabel("Frequency")
plt.hist(vals)
```

```
(array([5., 2., 1., 2., 0., 3., 0., 1., 5., 1.]),
 array([ 0. ,  8.9, 17.8, 26.7, 35.6, 44.5, 53.4, 62.3, 71.2, 80.1, 89. ]),
 <BarContainer object of 10 artists>)
```



Part 2

Provide the following information for the generated dataset.

Mean

Call function *mean* on the Series object.

```
#Mean

#create a pandas series object
pd_series = pd.Series(vals)

#calculate mean
pd_series.mean()
```

```
40.3
```

Mode

Call function mode on the Series object.

```
#Mode

#calculate mode
pd_series.mode()
```

```
0    13
1    75
2    76
dtype: int32
```

Lower quartile

Call function *quantile* on the Series object and pass in 0.25 as the fraction the coincides with the lower quartile.

```
#lower quartile

#calculate the lower quartile
pd_series.quantile(0.25)
```

```
11.25
```

Median

Call function *median* on the Series object.

```
#Median
pd_series.median()
```

```
41.0
```

Upper quartile

Call function *quantile* on the Series object and pass in 0.75 as the fraction that coincides with the upper quartile.

```
#Upper quartile
pd_series.quantile(0.75)
```

```
74.25
```

Interquartile range

The difference between the upper and lower quartile will constitute the interquartile range.

```
#interquartile range

#calculate the interquartile range ie upper minus lower quartile
pd_series.quantile(0.75) - pd_series.quantile(0.25)
```

```
63.0
```

Two different solutions for calculating the standard deviation

1.Using a library function

```
pd_series.std()
```

```
31.116250617937825
```

2.Applying a formula without using a library function

```
#Find the mean
series_mean = pd_series.mean()
print(series_mean)
```

```
#calculate the deviations from the mean
deviations = pd_series - series_mean
print(deviations)
```

```
#square the deviations
square_of_deviations = deviations ** 2
print(square_of_deviations)
```

```
#Find the summation of the deviations
sum_of_deviations = square_of_deviations.sum()
print(sum_of_deviations)
```

```
#Divide the sum of deviations by the number of observations
average_of_deviations = sum_of_deviations / len(pd_series)
```

```
#Find the square root of the result
average_of_deviations
```

```
standard_deviation = np.sqrt(average_of_deviations)
print(standard_deviation)
```

Task 2

Part 1

Initialise a two-dimensional array consisting of 5 rows and 10 columns...

```
#set seed
np.random.seed(1234)

#generate the data points, since it is a 5 by 10, the size will be 50
data_points = np.random.randint(low=0, high=9, size=50, dtype=int)

#make the data points into a 5 by 10 matrix
data_points_matrix = np.reshape(data_points, (5, 10))
print(data_points_matrix)
```

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

Displaying a matrix as a table.

```
#display as data points as a table
df = pd.DataFrame(data_points_matrix)
print(df)
```

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

Plot the first two rows on a single diagram with different colors

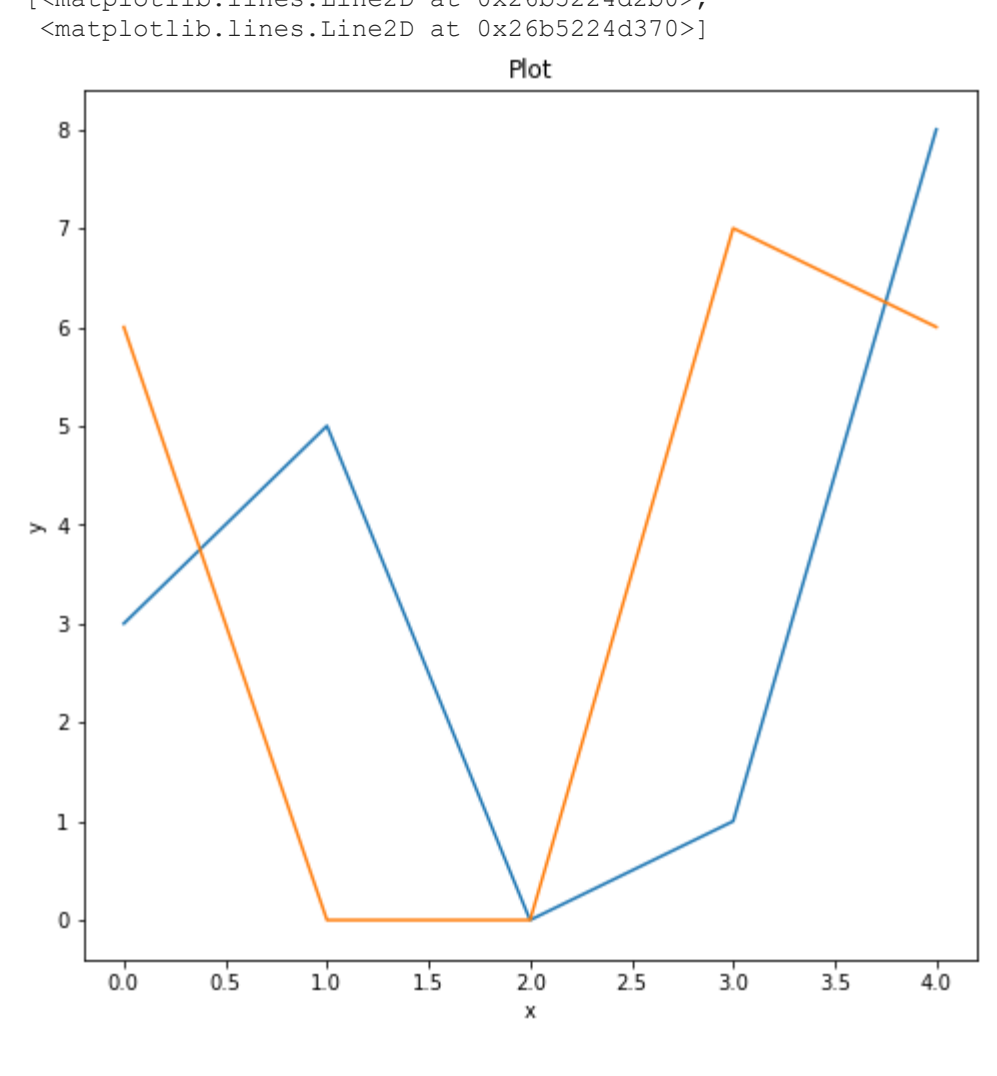
```
#selecting the first two rows
first_two_rows = df.iloc[:, 0:2]
print(first_two_rows)
```

```
   0  1
0  3  6
1  5  0
2  0  0
3  1  7
4  8  6
```

Plot the first the rows above

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 8))
plt.title('Plot')
plt.xlabel('x')
plt.ylabel('y')
plt.plot(first_two_rows)
```

```
[<matplotlib.lines.Line2D at 0x26b5224d2b0>,
 <matplotlib.lines.Line2D at 0x26b5224d370>]
```



Part 2

Mean, Median, Standard deviation for the rows

To be able to provide the above information for rows of the above data frame, you first have to find the transpose of the data frame.

```
#find the transpose of the data frame to ensure that you are working with rows
df.transpose().describe(percentiles=[0.5])
```

	0	1	2	3	4
count	10.000000	10.00000	10.000000	10.000000	10.000000
mean	4.800000	3.60000	2.300000	3.500000	4.600000
std	2.780887	2.54733	2.057507	2.877113	2.503331
min	0.000000	0.00000	0.000000	0.000000	1.000000
50%	5.500000	4.00000	2.500000	4.000000	5.000000
max	8.000000	7.00000	7.000000	8.000000	8.000000

Part 3 (2 points):

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

```
data = np.random.normal(17, 0.2, size=1000)
print(data)
```

```
[16.84976106 17.0357555  16.83265219 17.10187925 16.99177522 16.97761808
 16.84819852 16.9908128  16.81845443 16.94417726 17.04882405 17.07321128
 16.94579975 16.98575857 17.01696737 17.02408286 16.74627063 17.10496747
 17.19676754 16.7470184  17.06651319 17.15008752 16.78415756 16.87437692
 17.05986941 17.02545545 17.18523799 17.49104798 16.93582197 16.93767467
 17.02619238 17.0525571  17.2047529  16.98748099 16.76353994 16.93104985
 16.95090187 16.79480095 17.01088496 16.91863028 16.85747389 17.17246912
 16.88448994 16.89763639 17.08511034 16.98724835 17.03937896 16.97737236
 16.94519519 17.05404379 16.79982214 17.28702067 17.03190393 17.12311667
 17.00566797 17.38350738 16.85752839 17.08226517 17.10657388 17.1443531
 16.83736441 16.98339772 17.12217256 16.96423219 16.70687904 16.91036682
 17.3554183  17.35726139 17.3654275  16.75874764 17.11245849 16.74235736
 17.090738   16.99572875 16.92141364 16.9422558  16.94572605 16.75014807
 17.15970103 17.38234353 17.38028709 17.17025804 17.09373638 16.97276449
 17.08378002 16.92740098 16.97761375 17.08670747 16.76110337 16.83106343
 16.79545627 17.10637862 16.79358244 16.66339816 17.4227614  16.90987438
 17.00099429 17.02830963 16.87578204 16.97288396 17.02573405 16.9079012
 16.99669003 16.98017188 16.99797204 16.98679153 17.2635519  16.69555863
 16.95311059 16.99218981 17.72084987 16.91365665 17.30167105 17.03229173
 17.17228169 17.68348896 16.9716351  16.86558328 17.08885263 17.04254368
 16.60721796 17.11747428 17.08782633 17.02781984 17.03138022 16.97786263
 17.32230935 17.03578395 17.49829239 17.11789427 16.86009748 16.95674017
 17.32049774 17.03700159 16.9741107  17.12171323 17.29272396 16.94604237
 17.20196141 17.03980619 16.80137958 16.61859605 17.06170164 17.07810079
 16.50113189 16.96012649 17.03357251 17.16680143 16.97030865 17.18218823
 16.99998081 17.38537909 17.33654872 17.00584765 16.93993665 17.08122102
 17.19285458 16.60481436 16.91518486 16.98289671 16.94861069 16.75485107
 16.92573101 17.28598209 16.89940506 17.25612145 16.82531783 16.71430784
 17.0006265  17.01425704 17.06547083 17.06053811 16.55298592 17.10529378
 16.82128579 17.08022481 16.96973071 17.22336654 17.00675905 17.07711338
 17.08436854 17.03540341 16.75213619 17.03457649 17.38102437 16.97868038
 16.75175031 16.93844124 17.04538375 17.29632759 17.09110731 17.27422999
 16.99417775 16.94101418 16.81560234 16.9022322  17.06050213 16.95585604
 17.01082745 17.15508658 16.93158192 17.17083479 17.06664008 17.07838885
 17.20621401 16.90893231 17.26697732 16.72250012 16.8564125  16.81397645
 16.84947807 17.14596277 16.9575177  16.51395559 17.31675433 17.5982187
 16.85795934 17.25271963 16.5773694  17.03820246 17.04591739 17.21967139
 16.6542951  16.82384528 16.87173189 17.47823573 16.75755212 16.70375839
 16.69132326 17.31150848 17.21784047 17.37160524 16.86111839 16.91070805
 16.88369636 16.69041589 17.22491913 16.71661261 16.95054441 16.72735791
 17.12471749 16.63825116 16.92645324 17.35748831 16.71595729 16.88901964
 16.92087964 17.29046724 16.91478916 17.04242377 16.9203232  16.70399652
 17.13258251 16.84703668 17.31361789 16.47743419 17.1958613  16.77172667
 16.70716121 17.27179909 16.89022258 16.86164265 16.90485345 16.8072823
 16.62317735 16.82291914 17.23831201 16.90697433 17.13980479 16.95855678
 16.96143144 16.74684061 16.88878682 16.8141338  17.1577337  17.20978202
 17.55863424 16.8573904  16.84790753 17.21300786 16.8490519  17.17536482
 16.90158315 16.97046473 17.7555028  16.95554737 16.90264628 17.16766512
 16.72450286 16.85203371 16.6151275  16.63963058 17.31061706 17.12269738
 16.96566544 16.892      16.9636609  16.70837763 16.94130358 17.11276443
 16.92314385 17.2593254  16.63915775 17.11200309 17.12806963 16.95451721
 16.88347625 16.8194251  17.00798214 16.94597234 17.22300887 17.19388085
 16.84376971 16.44303092 16.7596429  16.95362478 16.97150664 17.01203551
 16.83542834 17.37519991 16.81346836 16.87289332 17.07582431 16.61810157
 16.71375784 17.26593069 16.88756691 17.11714572 16.89117921 17.16517014
 16.9875055  17.40641788 17.12789583 16.68985755 17.1806989  17.09530018
 16.83991307 16.68063278 17.04854021 17.06045959 17.249943   16.69501911
 16.85464438 17.05591575 17.21191242 16.64321188 16.72458616 17.03001548
 16.7398107  16.93148325 16.60557926 17.19229195 17.24446405 17.08411935
 16.87362975 16.78903136 17.11762682 17.29070854 17.13379831 16.99519443
 17.25389469 17.20783647 17.19125093 17.2897837  17.04769407 17.03480629
 16.84134168 17.01030899 17.29056843 17.02305101 16.91158676 16.88268977
 16.80997376 17.17812208 16.96580921 17.07110176 16.94310832 17.21887636
 17.01094396 17.00600934 17.39565317 16.91435723 16.97668585 17.00265935
 16.87343196 16.81879403 17.01285773 17.20939481 16.85589352 17.22019397
 17.08352171 17.19728711 16.74442274 17.30680221 17.17919146 17.38884042
 16.89059926 16.90737728 16.75340489 17.1763089  16.63950461 16.99852378
 16.75604119 17.02911552 16.95013576 16.79070418 17.26287458 17.14335789
 17.0771589  16.92693697 17.07419107 17.28570048 16.94140666 16.74988096
 17.06663007 17.12329421 16.81691662 16.99503661 16.84097503 16.91832312
 16.63015958 17.15634448 17.02666624 16.94030143 17.72647121 17.07844892
 16.85220569 17.07156342 17.25822939 16.84250991 17.20477003 17.09516875
 17.03184261 17.20052938 17.02741256 17.05759161 16.86960631 16.91545237
 16.93915913 17.24696878 17.13852502 16.58129172 17.13764601 17.21218862
 17.23055363 17.25295333 17.02813943 17.0115831  17.08110836 17.016944
 17.36662227 17.4206798  17.0146127  16.86250292 16.99684109 16.95150156
 17.13945233 17.23024735 17.1254936  17.07955715 16.83774697 16.96032266
 17.28065669 17.00481947 16.84534096 17.09272001 17.39394413 17.18971809
 16.90186702 17.06261837 16.88230183 17.04063313 17.3265992  16.88849029
 17.02524075 17.328723   16.98645681 17.02541274 17.07922876 17.20865787
 16.95407463 17.03163722 16.94360694 17.25102956 17.61269281 17.06095414
 16.84663604 16.82422278 17.02112408 17.04961027 17.33826482 16.96966611
 17.3533154  17.12088473 16.93252341 17.01444494 16.73039668 16.9684129
 17.15330763 17.18544448 17.15963676 17.24772749 17.03898429 17.05780944
 17.20678386 17.00822645 17.3042031  17.10866879 16.89751587 16.971618
 16.71943572 16.69992202 16.60469478 17.01487462 17.07601221 17.07080102
 17.07054523 17.06312313 16.93939442 17.12377913 16.82255881 17.28512255
 17.17510707 17.06536103 16.66031562 16.75736061 17.06035004 16.9348154
 17.00216045 16.91663747 17.05951202 17.16301808 17.20374179 17.16824452
 16.88204067 16.93752033 16.65953922 17.153216   17.04549312 16.8314547
 16.69458886 17.20651685 17.01077191 16.9624767  17.17485182 16.6251975
 16.97147502 17.09175784 16.61399617 17.03183366 17.61343536 16.95465398
 16.92134066 16.9822862  17.20528928 16.87894028 16.97877752 16.85646403
 16.91654274 16.78531981 17.17073755 17.07045674 16.89593637 16.79591081
 16.93269904 17.1079563  16.88202129 17.19545382 16.94060674 16.76996577
 17.09351365 16.92774878 16.74297816 16.62276787 17.15778972 16.92669416
 17.22686147 17.02398681 17.20409536 17.22887947 17.03684664 17.01005886
 17.50417401 17.15530131 17.35848244 16.91084441 17.02946423 16.86195626
 17.09228309 17.56470023 17.36942814 16.86997585 16.9466686  16.85852617
 17.12765543 17.03464345 16.88180108 16.93078149 17.05575373 16.76902885
 16.88961283 17.1829066  16.84966367 16.74855033 16.86210367 16.65467097
 16.69560693 16.96873409 17.17447506 16.88549717 16.95410207 16.6935657
 16.9905778  16.94725165 17.02879936 16.97482355 16.93019704 16.70236144
 17.31756321 17.00530443 17.15721261 17.10889403 17.04307004 17.00176809
 16.89898381 17.09376663 16.89853466 16.67599675 16.9370659  16.85927265
 16.72541411 17.36249249 16.91158327 16.85748099 16.93163935 16.92303078
 17.10122366 17.26372489 17.0121536  16.95834477 17.08096297 16.7949391
 16.95010478 16.76369163 16.99633446 17.25150596 16.69894476 17.05007911
 16.60009514 16.94377681 16.8316861  16.88194831 17.15784037 17.12421325
 16.71157228 16.84722345 16.97138472 16.96958543 17.03896212 16.8737725
 16.52049975 16.81760639 16.97495218 17.0778757  17.06721605 17.1144428
 17.06095658 17.15275024 16.78981376 16.71603418 17.01200984 17.01362876
 16.69353557 17.21441563 16.7470007  17.18948666 17.17509991 17.00804541
 17.09760908 16.77324942 16.96381359 17.22652306 17.14353327 17.21666627
 17.10906673 17.1205598  16.56721857 16.91509273 17.06036246 17.18165392
 17.27873324 16.92566349 16.72734879 16.85778419 17.01183882 16.95446555
 16.77267277 17.34936703 17.18538971 17.09490725 17.427043   17.03993586
 17.3312636  17.16859373 17.15600752 16.83127922 17.26553776 17.10312938
 17.06107072 17.07420591 16.68865209 16.81252266 16.96289744 16.60170699
 17.36032054 16.97102914 17.13587568 17.25842405 16.90624611 17.22944342
 16.94628283 16.73726038 17.06275192 17.17248962 17.03316144 17.01657641
 17.13202001 17.14392817 17.07595834 16.91588229 17.47954594 16.98289794
 17.04551961 16.75645421 16.89887735 17.34486668 17.1346251  17.3588741
 16.95394479 17.08602887 17.37141212 17.20376476 17.34039141 17.00217514
 16.99887202 16.96857784 17.13779026 17.02530318 17.0872091  17.07109911
 16.56807053 17.00225195 16.84385883 17.27983562 17.00456507 16.5648766
 17.08769583 17.10026033 17.0346123  16.71444868 17.19739212 17.27761291
 16.92262324 17.48279496 16.97943177 16.96166239 16.89559802 17.28205752
 17.14410473 17.09723208 17.23572576 17.10656769 16.99378085 17.15682665
 17.22215803 16.92940689 16.89391401 17.2778258  17.01274928 16.98258185
 16.88611026 17.09118112 17.18299728 17.36211242 16.96898194 17.22113234
 17.19747461 16.69578975 16.89306468 16.87930137 16.92144704 17.08582367
 16.69674905 16.95432711 17.02038919 17.02358994 17.26699863 16.768485
 17.18987457 17.07741116 17.00425507 16.93295494 17.00336386 16.81073031
 16.93042664 16.95760356 17.23615644 16.97756245 16.74372452 16.91099407
 16.78748285 16.75031637 16.98156978 17.1219577  17.25421152 16.81206504
 16.8009062  16.46608985 17.02341614 16.87788479 17.00338456 17.07128438
 17.34609237 17.12458486 17.06852002 16.76852002 16.63967319 17.21326158
 16.73358673 16.99332394 17.26344099 17.08304884 17.04612488 16.81439661
 16.87337168 17.26240577 16.98156047 16.93430061 16.75669191 16.67236788
 17.22549865 17.15974191 17.34752185 16.90091694 17.03734341 16.92007652
 16.8473368  17.30295595 17.0325293  17.01532773 17.06998633 17.15262809
 16.78982834 16.8119928  17.01174313 17.08522661 17.17437737 16.99497134
 17.11991184 16.94719392 17.02884672 16.7329318  17.4375564  16.98009716
 17.05831643 17.02056474 17.59154788 16.99806884 16.96133092 17.13176948
 17.24878766 16.82136906 16.79400272 17.13704421 17.39597988 17.07765889
 17.06860971 16.86407045 17.1688247  16.86367556 17.04251149 16.99178163
 16.97910938 17.2191611  17.03070893 17.09062211 16.69245267 17.02864139
 17.20285709 17.48815253 16.71753808 17.32686772 17.03727698 16.89707701
 16.76556295 17.01673738 17.08532613 17.28766175 17.22897796 16.81781862
 17.23963447 17.12816496 16.97881274 16.85863704 17.1534179  17.00016785
 17.05708117 16.73285505 17.36048335 17.10217553 17.16092395 17.17559807
 17.30901912 17.26328855 17.15234893 17.04561372 16.77005416 16.67466894
 17.10762091 17.05978226 17.09955727 17.30084424 17.23488528 16.7456814
 16.91382428 16.79748943 17.03480739 17.20727097 16.78208487 17.23995066
 17.10476927 16.73621192 16.6079323  16.99353479 17.38212698 16.86998938
 17.22205262 17.38953827 17.40763864 16.98755479 16.86164855 16.72981879
 16.98368083 17.34867617 16.86089821 16.71478426 16.72121088 17.05075644
 17.11791968 16.98784537 16.90914096 17.04945274 17.04842585 17.20436256
 16.80775445 16.82254316 16.69441294 17.01380822 17.06386383 16.85379477
 16.90246326 16.86738457 17.14394893 16.76392918 17.10396223 17.06727987
 16.77833359 17.03652208 16.86898426 16.71112232 17.33488605 16.71799926
 17.12321033 16.6516557  16.99206759 16.77655254 17.0665583  16.83785015
 17.15668825 17.03455036 17.18512719 17.39895527 17.10797819 16.78959038
 16.81152311 16.88501648 17.75246664 16.6102609  17.22246542 16.55701815
 17.3400848  16.98151019 16.94597558 17.40128802 16.96918489 16.97428234
 16.77697397 16.82635306 17.01397664 16.98575416]
```

Call function *std* on the numpy array to get standard deviation.

```
data.std()
```

```
0.20033168500868037
```

Part 4

Maximum

Call function *max* on the numpy object to get the maximum value.

```
data.max()
```

```
17.752466639839604
```

Minimum

Can function *min* on the numpy array object to get the minimum value.

```
data.min()
```

```
16.37160524113363
```

Range

Find the difference between the output of calling function *max* and function *min*

```
data.max() - data.min()
```

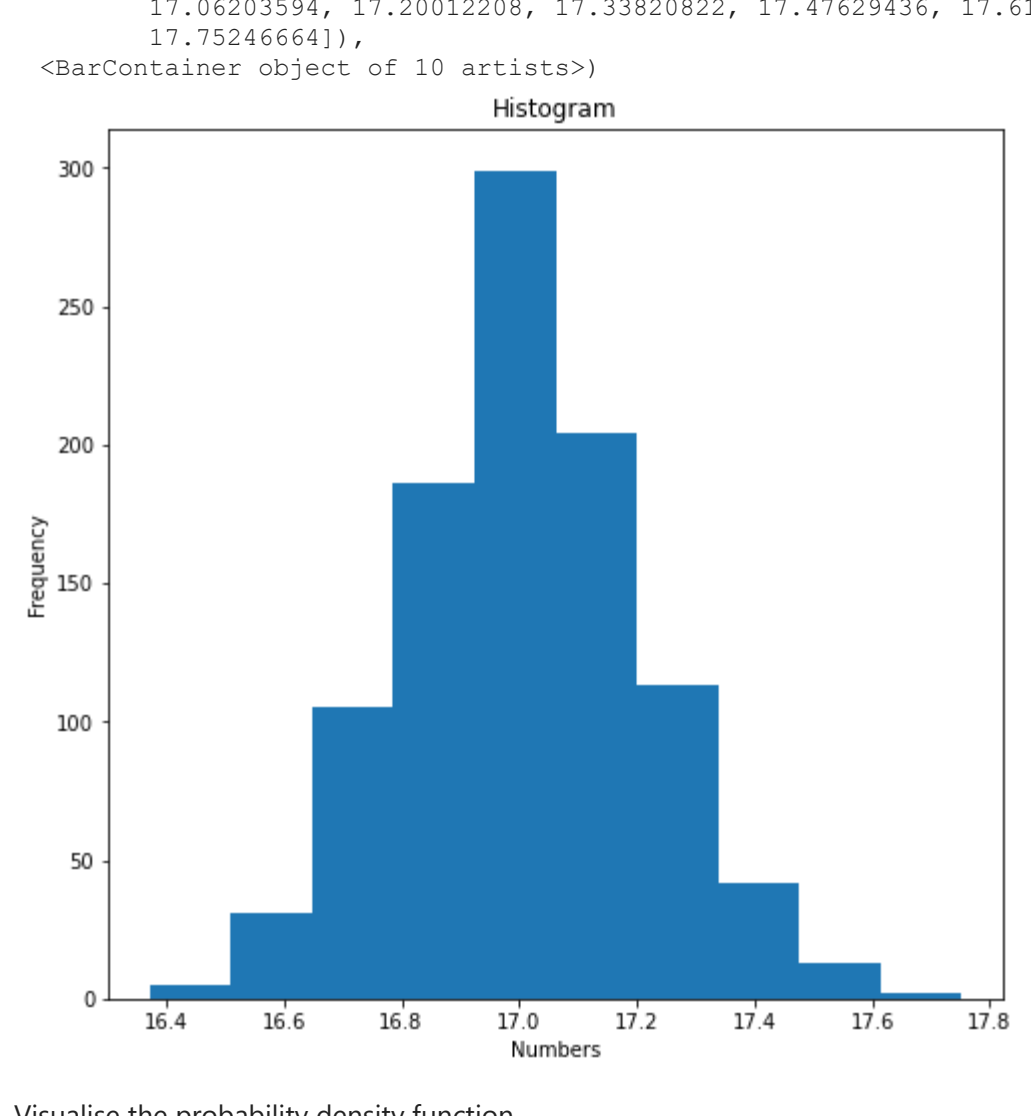
```
1.3808613987059744
```

Part 5 (2 points):

Visualise the dataset by using a histogram with 10 bins.

```
plt.figure(figsize=(8, 8))
plt.title("Histogram")
plt.xlabel("Numbers")
plt.ylabel("Frequency")
plt.hist(data, bins=10)
```

```
(array([  5.,  31., 105., 186., 299., 204., 113.,  42.,  13.,   2.]),
 array([16.37160524, 16.50969138, 16.64777752, 16.78586366, 16.9239498 ,
        17.06203594, 17.20012208, 17.33820822, 17.47629436, 17.6143805 ,
        17.75246664]),
 <BarContainer object of 10 artists>)
```



Visualise the probability density function.

Linear Algebra in Practice

Task 1

Initialize matrix A

```
import numpy as np
A = np.array([
    [2, 5, 1],
    [4, 3, 7],
    [1, 3, 2]
])
```

Determinant

In [53]:
```
np.linalg.det(A)
```
Out[53]: -26.000000000000014

Trace

In [54]:
```
A.trace()
```
Out[54]: 7

Inverse of matrix

In [55]:
```
#inverse
np.linalg.inv(A)
```
Out[55]:
```
array([[ 0.57692308,  0.26923077, -1.23076923],
       [ 0.03846154, -0.11538462,  0.38461538],
       [-0.34615385,  0.03846154,  0.53846154]])
```

Task 2

Find the product P of the matrices B and C by using the Python function for matrix multiplication. Display the result on the screen

In [16]:
```
B = np.array([
    [4, 7, 2],
    [3, 2, 5],
    [6, 4, 3]
])

C = np.array([
    [3, 1, 9],
    [7, 5, 6],
    [2, 1, 1]
])
```

Multiplication of matrix B and C

In [17]:
```
A * B
```
Out[17]:
```
array([[ 8, 35,  2],
       [12,  6, 35],
       [ 6, 12,  6]])
```

Task 3:

Represent the system of linear equations by using matrices.

3x + 2y - z = 25

2x - y + 4z = 19

4x - 2y + 3z = 18

A		vars		b
$$\begin{bmatrix} 3 & 2 & -1 \\ 2 & -1 & 4 \\ 4 & -2 & 3 \end{bmatrix}$$	$$	$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$	=	$$\begin{bmatrix} 25 \\ 19 \\ 18 \end{bmatrix}$$

Task 4:

Algebraic steps for solving the system of linear equations from Task 3 by using matrix notation.

$A=\begin{bmatrix} 3 & 2 & -1 \\ 2 & -1 & 4 \\ 4 & -2 & 3 \end{bmatrix}$$

$b=\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$Z = \begin{bmatrix} 25 \\ 19 \\ 18 \end{bmatrix}$$

Ab = Z

A^-^{1}b = ZA^-^{1}

b = ZA^-^{1}

Task 5:

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

In [56]:
```
#initialize matrix A
A = np.array([
    [3, 2, -1],
    [2, -1, 4],
    [4, -2, 3]
])

#find the inverse of matrix A
invA = np.linalg.inv(A)

#print to view
print(invA)
```
```
[[ 0.14285714  0.11428571  0.2       ]
 [ 0.28571429  0.37142857 -0.4       ]
 [ 0.          0.4        -0.2       ]]
```

In [56]:
```
#initialize matrix b
b = np.array([
    [25],
    [19],
    [18]
])

#print to view
print(b)
```
```
[[25]
 [19]
 [18]]
```

In [57]:
```
x, y, z = np.dot(invA, b)
print(x, y, z)
```
```
[5.] [7.] [4.]
```

Data Visualisation with Python

Data Visualisation with Python

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 [ ]:

In [ ]:

Task 2

Average monthly temperatures of three cities of your choice

In [58]:
```
#Path to the csv file
path = "C:\\Users\\joshua\\Desktop\\ASSIGNMENTS\\5582744_1500739164_ProgrammingFundamentals\\Programming Fundan

#import pandas
import pandas as pd

#read the heatMap.csv dataset using pandas and transpose
heat = pd.read_csv(path, index_col=0).transpose()

#View the first five variables
heat.head()
```
Out[58]:

City	Phoenix	Little Rock	Sacramento	Denver	Hartford	Dover	Tallahassee
1	6	4	9	-2	-2	1	15
2	8	6	11	1	1	2	17
3	11	12	13	5	4	6	19
4	16	17	15	9	10	12	22
5	21	21	18	14	15	17	25

In [59]:
```
#average Little Rock
print("Little Rock mean: ", heat["Little Rock"].mean())
```
```
Little Rock mean:  16.416666666666668
```

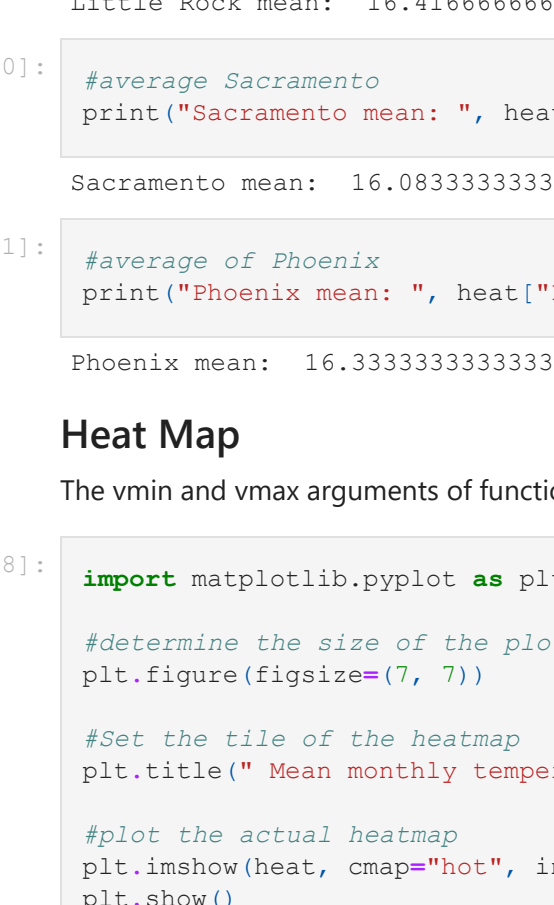
In [60]:
```
#average Sacramento
print("Sacramento mean: ", heat["Sacramento"].mean())
```
```
Sacramento mean:  16.083333333333332
```

In [61]:
```
#average of Phoenix
print("Phoenix mean: ", heat["Phoenix"].mean())
```
```
Phoenix mean:  16.333333333333332
```

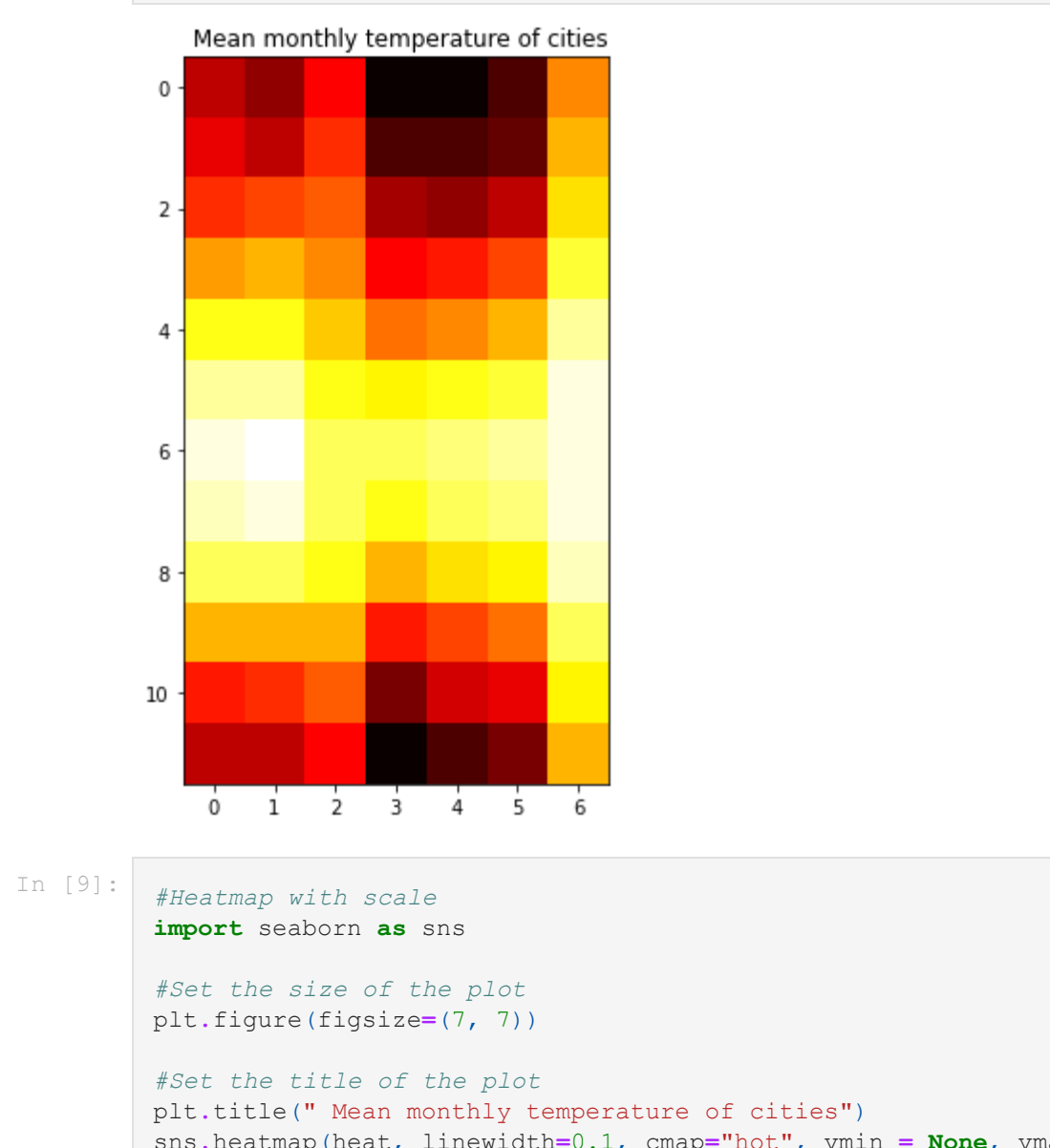
Heat Map

The vmin and vmax arguments of function *plt.imshow* allows users to specify a threshold.

In [8]:
```
import matplotlib.pyplot as plt

#determine the size of the plot
plt.figure(figsize=(7, 7))

#Set the tile of the heatmap
plt.title(" Mean monthly temperature of cities")

#plot the actual heatmap
plt.imshow(heat, cmap="hot", interpolation="nearest", vmin = None, vmax = None)
plt.show()
```



In [9]:
```
#Heatmap with scale
import seaborn as sns

#Set the size of the plot
plt.figure(figsize=(7, 7))

#Set the title of the plot
plt.title(" Mean monthly temperature of cities")
sns.heatmap(heat, linewidth=0.1, cmap="hot", vmin = None, vmax = None)
```
Out[9]: <AxesSubplot:title={'center':' Mean monthly temperature of cities'}, xlabel='City'>



Task 3

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

In [65]:
```
from pandas.plotting import parallel_coordinates
import pandas as pd
import matplotlib.pyplot as plt
```

In [11]:
```
pd.read_csv(path).head()
```
Out[11]:

	City	1	2	3	4	5	6	7	8	9	10	11	12
0	Phoenix	6	8	11	16	21	25	27	26	23	17	10	6
1	Little Rock	4	6	12	17	21	25	28	27	23	17	11	6
2	Sacramento	9	11	13	15	18	21	23	23	21	17	13	9
3	Denver	-2	1	5	9	14	20	23	21	17	10	3	-2
4	Hartford	-2	1	4	10	15	21	24	23	19	12	7	1

In [43]:
```
path = "C:\\Users\\joshua\\Desktop\\ASSIGNMENTS\\5582744_1500739164_ProgrammingFundamentals\\Programming Fundan
pd.read_csv(path).columns
```
Out[43]: Index(['City', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12'], dtype='object')

In [66]:
```
plt.figure(figsize=(8, 6))
plt.title("Temperature of cities vs month")
plt.xlabel("Months")
plt.ylabel("Temperature")

#the colors argument of function parallel coordinates allows you to specify different colors for the plot
parallel_coordinates(pd.read_csv(path), "City", color=["blue", "red", "green", "violet", "cyan", "yellow", "bla
```
Out[66]: <AxesSubplot:title={'center':'Temperature of cities vs month'}, xlabel='Months', ylabel='Temperature'>

In [ ]: