HKUSPACE COMMUNITY COLLEGE

CCIT4088 PROGRAMMING FOR DATA SCIENCE 2019/2020

Individual Project Report

Student name: Chow Cheuk Yiu

Student number: 20119953

Chow Cheuk Yiu
20119953

Introduction

This project analyses 5 months of meteorological data retrieved from Hong Kong Observatory's website (hko.gov.hk). The months of January 2015 to 2019 are chosen and extracted from the webpage then put into pandas in Python 3.7 to be cleaned and analysed. Jupyter notebook is the application used. Libraries used in this project include pandas, matplotlib, seaborn. Pandas is used to organise the data into data frames so that they can be manipulated easily. Matplotlib, seaborn and plotly are data visualisation libraries that I used to plot various graphs to show the relationships between the data.

Dataset

The dataset is 5 months of daily meteorological data of January 2015, 2016, 2017, 2018 and 2019. The raw html table contains 12 columns. These are: Day, Mean Pressure, Absolute Daily Max Air Temperature, Mean Air Temperature, Absolute Daily Min Air Temperature, Mean Dew Point, Mean Relative Humidity, Mean Amount of Cloud, Total Rainfall, Total Bright Sunshine, Prevailing Wind Direction, and Mean Wind Speed.

Data Collection and Cleaning

The data is collected from hko.gov.hk by saving them as html files and reading them into Jupyter notebook using pandas' "read_html" method. The first table of each html file is the daily weather data we are using. I assign each corresponding table to a variable called j201x, 'j' for January, 201x for the year. Each dataframe has 31 rows of daily data plus 2 extra rows at the end for the "Mean/Total" and "Normal" values of that month.

```
In [1]: import pandas as pd
```

```
In [2]: j2015 = pd.read_html('Daily Extract 2015.html')[0]
        j2016 = pd.read_html('Daily Extract 2016.html')[0]
        j2017 = pd.read_html('Daily Extract 2017.html')[0]
        j2018 = pd.read_html('Daily Extract 2018.html')[0]
        j2019 = pd.read_html('Daily Extract 2019.html')[0]
```

The column headers are organised as a MultiIndex, i.e.:

```
In [245]: j2015
```

Out[245]:

Day	Hong Kong Observatory									King's Park	Waglan Island^	
Day	Mean Pressure (hPa)	Air Temperature			Mean Dew Point (deg. C)	Mean Relative Humidity (%)	Mean Amount of Cloud (%)	Total Rainfall (mm)	Total Bright Sunshine (hours)	Prevailing Wind Direction (degrees)	Mean Wind Speed (km/h)	
Day	Mean Pressure (hPa)	Absolute Daily Max (deg. C)	Mean (deg. C)	Absolute Daily Min (deg. C)	Mean Dew Point (deg. C)	Mean Relative Humidity (%)	Mean Amount of Cloud (%)	Total Rainfall (mm)	Total Bright Sunshine (hours)	Prevailing Wind Direction (degrees)	Mean Wind Speed (km/h)	
0	01	1024.5	19.2	16.0	13.7	8.3	61	16	0.0	9.5	80	24.6
1	02	1025.0	17.1	15.2	13.1	10.0	71	21	0.0	9.5	70	32.9

As such, we need to rename the headers so we can call the columns easily in the future:

```
In [4]: cols = ['Day','Mean Pressure (hPa)','Air Temp Absolute Daily Max (C)','Air Temp Mean (C)',
                'Air Temp Absolute Daily Min (C)','Mean Dew Point (C)','Mean Relative Humidity (%)',
                'Mean Amount of Cloud (%)','Total Rainfall (mm)',
                'Total Bright Sunshine (hours)','Prevailing Wind Direction (degrees)','Mean Wind Speed (km/h)']
```

This is the list of column names that will replace the current headers. Information about which observatory the data is collected from is discarded as it is irrelevant to our analysis.

After renaming the columns of j2015 to 'cols', here are the datatypes of each column. The entries 'Day' column are string objects because of "Mean/Total" and "Normal" in the last two rows. We will have to do some data cleaning and change it into integers. "Total Rainfall (mm)" also contains string objects because there are 'trace' entries. The rest of the columns are either floating points or integers and can be readily manipulated without performing any data cleaning.

```
In [253]: j2015.columns = cols
          j2015.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33 entries, 0 to 32
Data columns (total 12 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Day                                 33 non-null     object
 1   Mean Pressure (hPa)                 33 non-null     float64
 2   Air Temp Absolute Daily Max (C)     33 non-null     float64
 3   Air Temp Mean (C)                   33 non-null     float64
 4   Air Temp Absolute Daily Min (C)     33 non-null     float64
 5   Mean Dew Point (C)                  33 non-null     float64
 6   Mean Relative Humidity (%)          33 non-null     int64
 7   Mean Amount of Cloud (%)            33 non-null     int64
 8   Total Rainfall (mm)                 33 non-null     object
 9   Total Bright Sunshine (hours)       33 non-null     float64
 10  Prevailing Wind Direction (degrees) 33 non-null     int64
 11  Mean Wind Speed (km/h)              33 non-null     float64
dtypes: float64(7), int64(3), object(2)
memory usage: 3.2+ KB
```

Trace entries under the Total Rainfall column denote rainfall of less than 0.05mm for that particular day. If we want to analyse this data later, we have to do some data cleaning. I decided to change all trace entries to 0.05 because it is a good and simple estimation instead of ignoring them and changing them all to 0s.

Total Rainfall (mm)
Trace
0.3
5.6
Trace
46.7
Trace

```
In [5]: def trace(value):
            if value == 'Trace':
                return 0.05
            else:
                return value
```

Although a function would be a more systematic approach if our dataset is bigger, I decided to use a for loop for data cleaning instead because our dataset is small. The for loop will achieve the same result as a function. First, I create a list called 'list' with our 5 dataframes inside, then the for loop will loop over the dataframes and clean the data inside. The dataframe named 'average' contains the mean values of each month so we can reference them easily later on.

```
In [6]: # the 'average' dataframe will contain all the mean values of the 5 months
        average = pd.DataFrame()

        list = [j2015,j2016,j2017,j2018,j2019]
        for x in list:
            # rename multiindex header to 'cols'
            x.columns = cols
            # select the "Mean/Total" row of each df and assign to "mean"
            mean = x.iloc[-2]
            # append mean to the df called "average"
            average = average.append(mean)
            # apply the function 'trace' to 'Total Rainfall' column to eliminate 'trace' entries
            x['Total Rainfall (mm)'] = x['Total Rainfall (mm)'].apply(trace)
            # now that 'Total Rainfall' doesn't contain any strings, we can convert the column into floating points
            x['Total Rainfall (mm)'] = x['Total Rainfall (mm)'].astype('float64')
            # drop the last 2 rows ("Mean/Total" and "Normal")
            x.drop(x.tail(2).index,inplace=True)
            # now that all the entries in the 'Day' column are integers (1-31), we can convert it into int datatype
            x.Day = x.Day.astype('int64')
            # set 'Day' as the new index
            x.set_index('Day',inplace=True)
```

This is the new j2015 - j2019 dataframe after performing data cleaning:

```
In [256]: j2017.head(5)
```

Out[256]:

Day	Mean Pressure (hPa)	Air Temp Absolute Daily Max (C)	Air Temp Mean (C)	Air Temp Absolute Daily Min (C)	Mean Dew Point (C)	Mean Relative Humidity (%)	Mean Amount of Cloud (%)	Total Rainfall (mm)	Total Bright Sunshine (hours)	Prevailing Wind Direction (degrees)	Mean Wind Speed (km/h)
1	1021.7	20.8	19.2	18.4	15.6	80	72	0.0	4.6	60	34.2
2	1020.2	23.3	20.2	18.4	16.7	81	28	0.0	9.3	70	17.6
3	1019.8	21.3	20.0	18.9	17.1	83	56	0.0	3.8	70	26.1
4	1018.7	21.7	19.9	18.7	16.3	80	51	0.0	6.3	70	27.7
5	1016.9	23.4	21.1	18.9	17.5	80	61	0.0	1.7	40	14.3

The ‘Mean/Total’ row is dropped and moved to ‘average’ dataframe while the ‘Normal’ row is discarded because we are not using that data.

```
In [258]: j2017.tail(3)
```

Out[258]:

Day	Mean Pressure (hPa)	Air Temp Absolute Daily Max (C)	Air Temp Mean (C)	Air Temp Absolute Daily Min (C)	Mean Dew Point (C)	Mean Relative Humidity (%)	Mean Amount of Cloud (%)	Total Rainfall (mm)	Total Bright Sunshine (hours)	Prevailing Wind Direction (degrees)	Mean Wind Speed (km/h)
29	1016.6	21.5	19.1	18.1	17.0	88	88	2.4	0.6	50	28.2
30	1018.2	23.4	20.2	17.4	18.4	90	88	1.2	0.6	40	22.8
31	1020.2	17.6	16.7	15.9	14.6	87	88	0.5	0.0	70	34.3

All columns are now either floating points or integers:

```
In [257]: j2018.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 31 entries, 1 to 31
Data columns (total 11 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Mean Pressure (hPa)                 31 non-null     float64
 1   Air Temp Absolute Daily Max (C)     31 non-null     float64
 2   Air Temp Mean (C)                   31 non-null     float64
 3   Air Temp Absolute Daily Min (C)     31 non-null     float64
 4   Mean Dew Point (C)                  31 non-null     float64
 5   Mean Relative Humidity (%)          31 non-null     int64
 6   Mean Amount of Cloud (%)            31 non-null     int64
 7   Total Rainfall (mm)                 31 non-null     float64
 8   Total Bright Sunshine (hours)       31 non-null     float64
 9   Prevailing Wind Direction (degrees) 31 non-null     int64
 10  Mean Wind Speed (km/h)              31 non-null     float64
dtypes: float64(8), int64(3)
memory usage: 2.9 KB
```

This is the ‘average’ dataframe after configuring the index:

```
In [7]: average.index = ['Jan 2015','Jan 2016','Jan 2017','Jan 2018','Jan 2019']
```

```
In [8]: average.drop('Day',axis=1,inplace=True)
```

```
In [9]: average
```

Out[9]:

	Air Temp Absolute Daily Max (C)	Air Temp Absolute Daily Min (C)	Air Temp Mean (C)	Mean Amount of Cloud (%)	Mean Dew Point (C)	Mean Pressure (hPa)	Mean Relative Humidity (%)	Mean Wind Speed (km/h)	Prevailing Wind Direction (degrees)	Total Bright Sunshine (hours)	Total Rainfall (mm)
Jan 2015	18.9	14.5	16.4	45.0	11.2	1021.2	72.0	24.3	50.0	198.8	41.7
Jan 2016	17.8	14.4	16.0	79.0	13.0	1020.4	83.0	29.4	60.0	67.1	266.9
Jan 2017	20.6	17.0	18.5	66.0	14.7	1019.8	79.0	26.4	70.0	145.1	7.8
Jan 2018	18.5	14.1	16.1	69.0	11.7	1018.4	77.0	29.5	60.0	136.1	62.2
Jan 2019	20.4	16.4	18.1	68.0	13.7	1021.3	76.0	22.8	60.0	133.3	4.7

Basic analysis

To calculate the monthly mean air temperature, we call the '.mean()' method on the 'Air Temp Mean (C)' column for each respective month, using a for loop. To verify the answers, we call the respective column on the 'average' dataframe:

```
In [267]: # Monthly mean air temperature
          year = 2015
          for x in list:
              print('Jan',year,':',x['Air Temp Mean (C)'].mean())
              year += 1
```

```
Jan 2015 : 16.448387096774194
Jan 2016 : 16.025806451612905
Jan 2017 : 18.558064516129033
Jan 2018 : 16.061290322580646
Jan 2019 : 18.13548387096774
```

```
In [264]: # Verify
          average['Air Temp Mean (C)']
```

```
Out[264]: Jan 2015    16.4
          Jan 2016    16.0
          Jan 2017    18.5
          Jan 2018    16.1
          Jan 2019    18.1
          Name: Air Temp Mean (C), dtype: float64
```

Comparing the answers, they are correct except for a minor error in the value for Jan 2017, where the official value from the dataset is rounded to 18.5 but our calculation showed 18.558 (which would be rounded to 18.6 with 1 dp). This error is negligible.

To calculate the monthly total rainfall for each of the months, we use the '.sum()' method on the 'Total Rainfall (mm)' column with a for loop. To verify the answers, we call the respective column on the 'average' dataframe.

```
In [270]: # Monthly total rainfall
          year = 2015
          for x in list:
              print('Jan',year,':',x['Total Rainfall (mm)'].sum())
              year += 1
```

```
Jan 2015 : 41.99999999999999
Jan 2016 : 267.1
Jan 2017 : 7.95
Jan 2018 : 62.45
Jan 2019 : 5.000000000000001
```

```
In [272]: # Verify
          average['Total Rainfall (mm)']
```

```
Out[272]: Jan 2015     41.7
          Jan 2016    266.9
          Jan 2017      7.8
          Jan 2018     62.2
          Jan 2019      4.7
          Name: Total Rainfall (mm), dtype: object
```

There are some differences comparing our calculations with the official numbers from the dataset. This is because we changed all the trace entries to 0.05mm of rainfall and thus the calculated rainfall differs slightly from the actual numbers. However, the differences are quite small and not significant at all.

To calculate the monthly mean air temperature range of each months, we first have to find out the daily air temperature range. Using a for loop, we create a new column called 'Daily Air Temp range' that is equal to the 'Air Temp Absolute Daily Max' column minus 'Air Temp Absolute Daily Min' column. Then, we call the '.mean()' method on this new column and print the respective values of each months.

```
In [14]: # Monthly mean air temperature range

         year = 2015
         for x in list:
             # daily air temp range = absolute daily max - absolute daily min
             x['Daily Air Temp Range'] = x['Air Temp Absolute Daily Max (C)']- x['Air Temp Absolute Daily Min (C)']
             # monthly mean air temp range = average of daily air temp range in that month
             print('Jan',year,':',x['Daily Air Temp Range'].mean())
             year += 1
```

```
Jan 2015 : 4.416129032258064
Jan 2016 : 3.3741935483870966
Jan 2017 : 3.5419354838709696
Jan 2018 : 4.374193548387097
Jan 2019 : 3.954838709677419
```

Exploratory analysis

Mean Air Temperature Trend

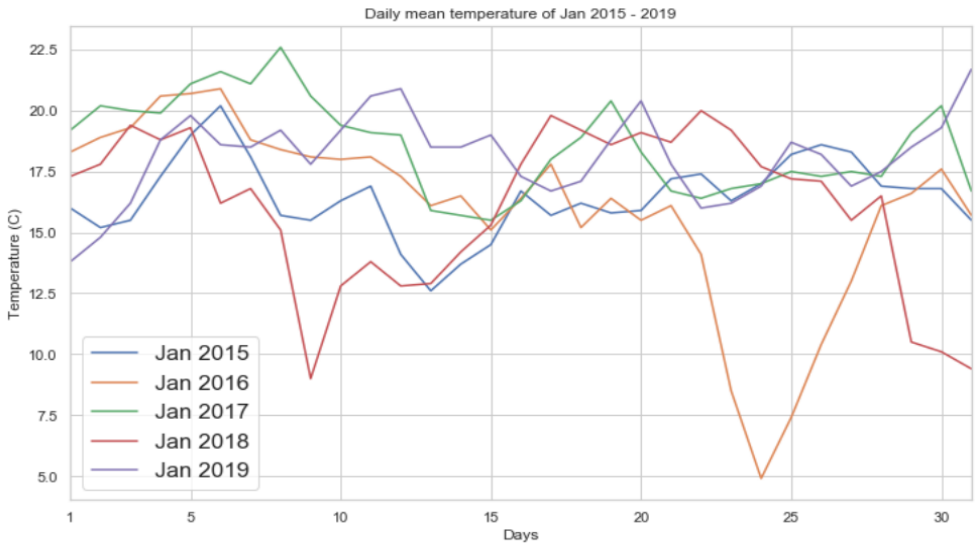
Import matplotlib and seaborn libraries for data visualisation, then set matplotlib to 'inline'.

```
In [273]: import matplotlib.pyplot as plt
          import seaborn as sns
          %matplotlib inline
```

Using matplotlib, we can plot the daily air temperature mean of each month and represent them with different coloured lines:

```
In [604]: fig,ax = plt.subplots(figsize=(12,7))

          ax.plot(j2015['Air Temp Mean (C)'],label='Jan 2015')
          ax.plot(j2016['Air Temp Mean (C)'],label='Jan 2016')
          ax.plot(j2017['Air Temp Mean (C)'],label='Jan 2017')
          ax.plot(j2018['Air Temp Mean (C)'],label='Jan 2018')
          ax.plot(j2019['Air Temp Mean (C)'],label='Jan 2019')

          ax.set_title('Daily mean temperature of Jan 2015 - 2019')
          ax.set_xlabel('Days')
          ax.set_ylabel('Temperature (C)')
          ax.set_xlim(1,31)
          ax.xaxis.set_ticks([1,5,10,15,20,25,30])
          ax.legend(loc='lower left',prop={'size': 17})
```

```
Out[604]: <matplotlib.legend.Legend at 0x1948136c6d8>
```



From this figure, we can see the trend of air temperature from the start to the end of the month of January from 2015 to 2019. There is no obvious trend in daily mean temperature. An interesting observation is that for all the Januaries except for 2015, the month begins with an increase in temperature, then end with a temperature drop on the last day except for 2019. We can see significant drops in temperature near the end of Jan 2016 where it was 5 degrees Celsius on the 24th, and on the 9th of January 2018, the temperature dropped to 9 degrees Celsius. Other than these anomalies the temperature of January across the years of 2015 to 2019 generally hover between 15 to 20 degrees Celsius.

Next, we can look at the distribution and range of daily mean temperatures of each month. First, we concatenate the 'Air Temp Mean' columns from each month into a single dataframe called 'mean_air_temp':

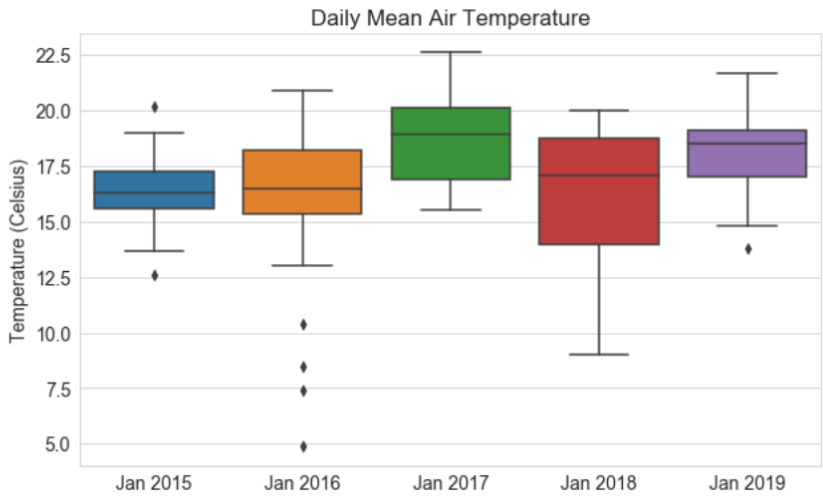
```
In [223]: j2015_atm = pd.DataFrame(j2015['Air Temp Mean (C)'])
          j2016_atm = pd.DataFrame(j2016['Air Temp Mean (C)'])
          j2017_atm = pd.DataFrame(j2017['Air Temp Mean (C)'])
          j2018_atm = pd.DataFrame(j2018['Air Temp Mean (C)'])
          j2019_atm = pd.DataFrame(j2019['Air Temp Mean (C)'])

In [234]: j2015_atm.rename(columns={"Air Temp Mean (C)": "Jan 2015"},inplace=True)
          j2016_atm.rename(columns={"Air Temp Mean (C)": "Jan 2016"},inplace=True)
          j2017_atm.rename(columns={"Air Temp Mean (C)": "Jan 2017"},inplace=True)
          j2018_atm.rename(columns={"Air Temp Mean (C)": "Jan 2018"},inplace=True)
          j2019_atm.rename(columns={"Air Temp Mean (C)": "Jan 2019"},inplace=True)

In [237]: mean_air_temp = pd.concat([j2015_atm,j2016_atm,j2017_atm,j2018_atm,j2019_atm],axis=1)

In [287]: mean_air_temp.head(5)
```

Out[287]:

Day	Jan 2015	Jan 2016	Jan 2017	Jan 2018	Jan 2019
1	16.0	18.3	19.2	17.3	13.8
2	15.2	18.9	20.2	17.8	14.8
3	15.5	19.3	20.0	19.4	16.2
4	17.3	20.6	19.9	18.8	18.8
5	19.0	20.7	21.1	19.3	19.8

Then we can use seaborn to plot a boxplot showing the temperature distributions of the months:

```
In [297]: plt.figure(figsize=(10,6))
          sns.boxplot(data=mean_air_temp)
          plt.title('Daily Mean Air Temperature')
          plt.ylabel('Temperature (Celsius)')

Out[297]: Text(0, 0.5, 'Temperature (Celsius)')
```



In this boxplot we ignore the time variable ('Day' column) and instead look at each month as a whole and compare them with other Januaries. We can observe an upward trend of temperature from 2015 to 2019, where the median of each month increases slightly every following year except for 2018 January. Though this correlation is not very strong since it is based on a very limited number of consecutive years. The IQR of January 2018 was significantly larger than other months, along with a longer lower whisker, indicating that the temperatures were quite a lot lower in that month in comparison. There were also a lot of outliers in January 2016, suggesting that the temperatures in that month were more irregular in which there were days where it was much colder than normal.

Correlation between daily max temp, daily min temp, and total bright sunshine hours

To analyse the relationship between daily max/daily mean temp and total bright sunshine hours, I merged these 3 columns into a separate dataframe for each month, then changed their column names so they can be easily called when plotting the graph:

```
In [405]: temp_sun15 = j2015[['Air Temp Absolute Daily Max (C)','Air Temp Absolute Daily Min (C)',
                              'Total Bright Sunshine (hours)']]
          temp_sun16 = j2016[['Air Temp Absolute Daily Max (C)','Air Temp Absolute Daily Min (C)',
                              'Total Bright Sunshine (hours)']]
          temp_sun17 = j2017[['Air Temp Absolute Daily Max (C)','Air Temp Absolute Daily Min (C)',
                              'Total Bright Sunshine (hours)']]
          temp_sun18 = j2018[['Air Temp Absolute Daily Max (C)','Air Temp Absolute Daily Min (C)',
                              'Total Bright Sunshine (hours)']]
          temp_sun19 = j2019[['Air Temp Absolute Daily Max (C)','Air Temp Absolute Daily Min (C)',
                              'Total Bright Sunshine (hours)']]

In [408]: list = [temp_sun15,temp_sun16,temp_sun17,temp_sun18,temp_sun19]
          for df in list:
              df.columns = ['Daily Max','Daily Min','Sunshine hours']
```

```
In [475]: temp_sun16.head()
```

Out[475]:

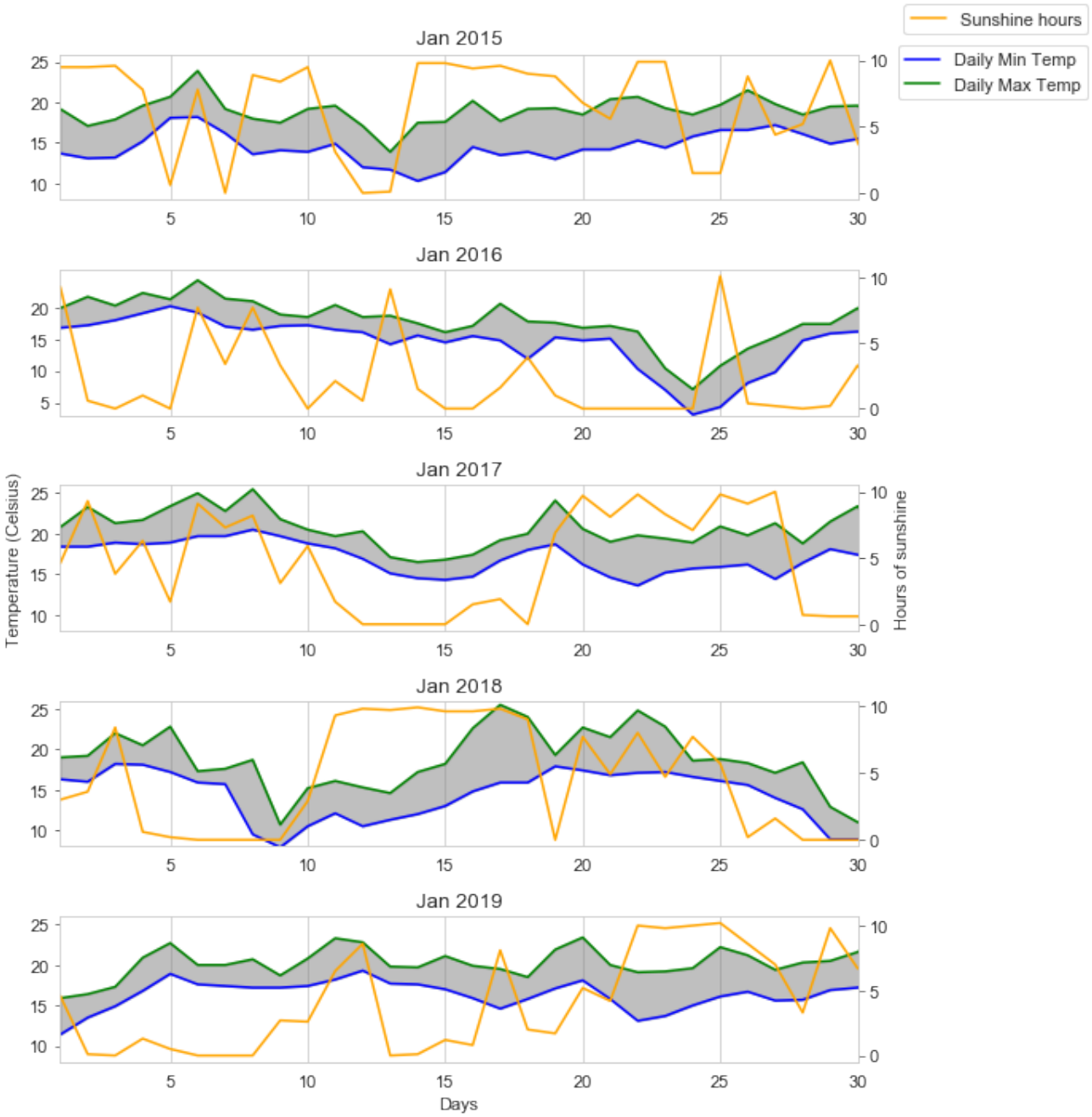
Day	Daily Max	Daily Min	Sunshine hours
1	19.9	16.8	9.3
2	21.7	17.2	0.6
3	20.3	18.0	0.0
4	22.3	19.1	1.0
5	21.3	20.2	0.0

We can now use matplotlib to plot these dataframes in one big diagram. I used plt.subplot to create a 5 row diagram that will each contain one month of data. There are three variables in each subplot: daily min temp, daily max temp, and sunshine hours, with 'Days' on the x-axis. Daily min and max temps are plotted on the left y-axis in blue and green respectively, while a twin axis is created on the right side to plot the hours of sunshine each day in orange. I used the fill_between method to fill in the area between max and min in grey colour to show the daily air temperature range.

```
In [474]: import numpy as np
          plt.figure(figsize=(10,10))

          ax0 = plt.subplot(511)
          ax1 = ax0.twinx()
          ax2 = plt.subplot(512,sharex=ax0,sharey=ax0)
          ax3 = ax2.twinx()
          ax4 = plt.subplot(513,sharex=ax0,sharey=ax0)
          ax5 = ax4.twinx()
          ax6 = plt.subplot(514,sharex=ax0,sharey=ax0)
          ax7 = ax6.twinx()
          ax8 = plt.subplot(515,sharex=ax0,sharey=ax0)
          ax9 = ax8.twinx()

          ax0.grid(axis='y')
          ax1.grid(False)
          ax2.grid(axis='y')
          ax3.grid(False)
          ax4.grid(axis='y')
          ax5.grid(False)
          ax6.grid(axis='y')
          ax7.grid(False)
          ax8.grid(axis='y')
          ax9.grid(False)
```

```
ax0.plot('Daily Min',data=temp_sun15,color='blue',lab
ax0.plot('Daily Max',data=temp_sun15,color='green',la
ax0.fill_between(temp_sun15.index,temp_sun15['Daily M
ax1.plot('Sunshine hours',data=temp_sun15,color='oran

ax2.plot('Daily Min',data=temp_sun16,color='blue')
ax2.plot('Daily Max',data=temp_sun16,color='green')
ax2.fill_between(temp_sun16.index,temp_sun16['Daily M
ax3.plot('Sunshine hours',data=temp_sun16,color='oran

ax4.plot('Daily Min',data=temp_sun17,color='blue')
ax4.plot('Daily Max',data=temp_sun17,color='green')
ax4.fill_between(temp_sun17.index,temp_sun17['Daily M
ax5.plot('Sunshine hours',data=temp_sun17,color='oran

ax6.plot('Daily Min',data=temp_sun18,color='blue')
ax6.plot('Daily Max',data=temp_sun18,color='green')
ax6.fill_between(temp_sun18.index,temp_sun18['Daily M
ax7.plot('Sunshine hours',data=temp_sun18,color='oran

ax8.plot('Daily Min',data=temp_sun19,color='blue')
ax8.plot('Daily Max',data=temp_sun19,color='green')
ax8.fill_between(temp_sun19.index,temp_sun19['Daily M
ax9.plot('Sunshine hours',data=temp_sun19,color='oran
```

This is the resulting diagram:



From this figure we can observe the general trend of daily max/min temperature and sunshine hours throughout the month of Jan of 2015 to 2019. There is not strong correlation between temperature and sunshine hours that is consistent in all of the months of January. For example, there seems to be a correlation in Jan 2018, where the temperature range and sunshine hours seemed to trend downwards together in the first 10 days of the month, then an upward increase is observed for both variables between day 10 and 17, and finally a sharp decrease in temperature and sunshine hours at the end of the month. There also seems to be a similarly weak correlation in Jan 2017. However, this trend is not consistent with Januaries of other years. For example, there is no visible correlation in 2015 and 2016. Jan 2016 seems to be an outlier with consistently dark days but sudden sunny days spread out. This could be related to the heavy amount of rain in that month (more on that later). Using the .corr() method in pandas, we can see that the relationship is generally weak and independent of each other:

In [509]:
```
display(temp_sun15[['Daily Max','Sunshine hours']].corr())
display(temp_sun15[['Daily Min','Sunshine hours']].corr())
```

	Daily Max	Sunshine hours
Daily Max	1.000000	0.211041
Sunshine hours	0.211041	1.000000

	Daily Min	Sunshine hours
Daily Min	1.000000	-0.294766
Sunshine hours	-0.294766	1.000000

[508]:
```
display(temp_sun18[['Daily Max','Sunshine hours']].corr())
display(temp_sun18[['Daily Min','Sunshine hours']].corr())
```

	Daily Max	Sunshine hours
Daily Max	1.000000	0.394662
Sunshine hours	0.394662	1.000000

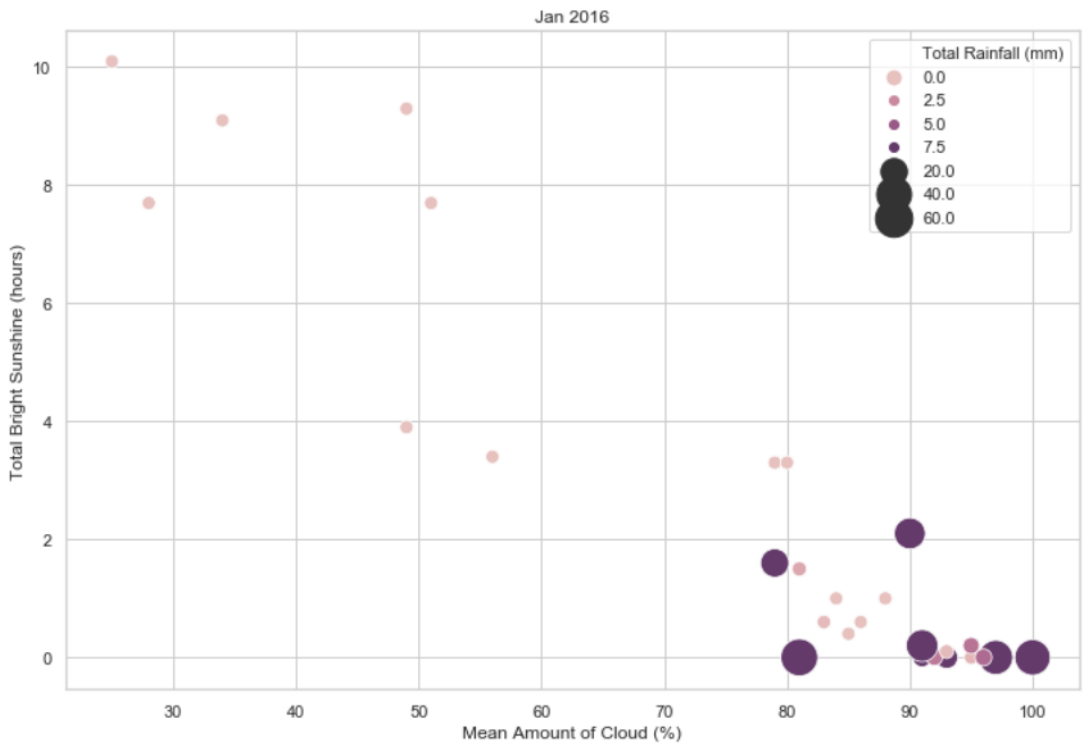
	Daily Min	Sunshine hours
Daily Min	1.000000	0.179411
Sunshine hours	0.179411	1.000000

Amount of Cloud vs. Hours of Sunshine

After being discouraged by the lack of correlation from my previous findings, I wanted to study something that will definitely have a strong correlation: the amount of cloud vs. the amount of sunshine, with an additional variable – total rainfall. Using seaborn we can plot a neat scatterplot with 'Mean Amount of Cloud (%)' on the x-axis, 'Total Bright Sunshine (hours)' on y-axis, and additionally 'Total Rainfall (mm)' as the 'hue' and 'size' argument in the scatterplot method. I used the data from Jan 2016 because it is the month with the heaviest rainfall out of the five.

In [577]:
```
plt.figure(figsize=(12,8))
cmap = sns.cubehelix_palette(dark=.3, light=.8, as_cmap=True)

ax = sns.scatterplot(x='Mean Amount of Cloud (%)',y='Total Bright Sunshine (hours)',
                     hue='Total Rainfall (mm)',hue_norm=(0,7),size='Total Rainfall (mm)',
                     sizes=(80, 600),palette=cmap,data=j2016)
ax.set_title('Jan 2016')
```

Out[577]:
```
<matplotlib.legend.Legend at 0x19486118668>
```



This figure shows that there is a strong negative relationship between the amount of cloud and total bright sunshine in one day. We can confirm this by looking at the .corr() method for that month:

```
In [586]: j2016[['Mean Amount of Cloud (%)','Total Bright Sunshine (hours)']].corr()
```

Out[586]:

	Mean Amount of Cloud (%)	Total Bright Sunshine (hours)
Mean Amount of Cloud (%)	1.000000	-0.934984
Total Bright Sunshine (hours)	-0.934984	1.000000

Obviously, this negative relationship of “more clouds = less sun” is self-evident. Furthermore is the extra variable that is the daily total rainfall which also has a strong correlation with those two labels. In days where there was very heavy rainfall – represented by the bigger datapoints with darker hue – there were very high amounts of cloud and very little hours of sunshine for that same day. Although these findings are not surprising, it is interesting to plot them out and show these relationships visually using Python.

Conclusion

To summarise this project, we performed some data cleaning and analysis on meteorological data from the January of five consecutive years 2015 to 2019, collected from the Hong Kong Observatory. In exploratory analysis, we first studied the trend of mean air temperature across the months. There is a weak correlation of temperature increase year on year. With the line plot we could see the significant temperature drops in 2016 and 2018. Secondly, we looked at the correlation between daily max/min air temperature and total bright sunshine hours. We visualised the data with a very large diagram showing the daily temperature and sunshine of every month. There was again a very weak trend between the two variables. Finally, we studied the relationship between amount of cloud and amount of sunshine, with a third variable of total rainfall. With the scatterplot we could see a very clear negative correlation between cloud and sunshine, while the amount of cloud positively correlates with heavy rainfall.

References

Seaborn documentation

https://seaborn.pydata.org/index.html

Matplotlib documentation

https://matplotlib.org/contents.html

Udemy Python for Data Science and Machine Learning Bootcamp course by Jose Portilla

https://www.udemy.com/course/python-for-data-science-and-machine-learning-bootcamp/learn/

Stackoverflow

https://www.stackoverflow.com