ELEC0033 Data Analytics Report

Climate Data Analysis using Python.

Author's Student ID: 22216907

EEE department, University College London, zceefji@ucl.ac.uk

Abstract: This report describes the task of analyzing climate data using Python. The task includes preliminary analysis (data understanding, data cleaning and data statistics), outliers and data inference. By analyzing the data, I find the correlation between different features of the data and infer the probability of having moderate to heavy rain as a function of the cloud cover index. Based on the significant correlations detected, I create a multivariate model to predict the photovoltaic production (PV production). This report improves my Python programming skills and data analysis and inference skills.

CCS CONCEPTS • Data processing and analysis• Correlation and Pattern Inference • Python Programming

Additional Keywords and Phrases: Climate data Analysis, Correlation Analysis, Outlier Detection

Reference:

Court, A. (1949). How hot is Death Valley? Geographical Review, 39, 214-220.

El Fadli, K., Cerveny, R. S., Burt, C. C., Eden, P., Parker, D., Brunet, M., Peterson, T. C., Mordacchini, G., Pelino, V., Bessemoulin, P., Stella, J. L., Driouech, F., Abdel wahab, M. M., & Pace, M. B. (2012). World Meteorological Organization Assessment of the Purported World Record 58ºC Temperature Extreme at El Azizia, Libya (13 September 1922). Bulletin of the American Meteorological Society. doi: http://dx.doi.org/10.1175/BAMS-D-12-00093.1.

1 PRELIMINARY ANALYSIS

This part describes how I carried out my study in Task 1, which analyzes the statistical information of overall data for further processing.

1.1 Data Understanding

First of all, I import the NumPy and Pandas package using `import numpy as np` and `import pandas as pd`. Store the data from "weather-denmark-resampled.pkl" file into `data` variable and print the length and description of the data using code below:

```
data = pd.read_pickle('weather-denmark-resampled.pkl')

ori = len(data) # to record the total number of data

print("the original data has", ori,"lines")

print('data:\n',data.describe())
```

By reading the description of table, we may acquire the basic information:

1. There are *333110 lines* of data in total. The format of the data is a Pandas *DataFrame* object in Python, as indicated by the presence of the index (DateTime) and column headers (city names and weather measurements).
2. This is a time series dataset containing weather data for the 5 cities of *Aalborg, Aarhus, Esbjerg, Odense and Roskilde*. The dataset includes 4 different measures: t*emperature,*

pressure, wind speed, and wind direction measurements for each city, recorded hourly from March 1, 1980, to March 1, 2018.

3. The data is in a *tabular forma*t with three columns representing each city, and each column contains sub-columns for the different weather measurements. The data is organized by rows, where each row represents a specific date and time, and the weather measurements for each city are recorded for that particular time. More information could be found in the table below.

```
the original data has 333110 lines
data:
              Aalborg                                                \
                 Temp       Pressure      WindSpeed        WindDir
count   333110.000000  333110.000000  333110.000000  333110.000000
mean         8.323675    1012.743473       4.867406     192.307074
std          6.986639      11.690186       2.793941      88.071567
min        -25.000000     951.900000       0.000000      10.000000
25%          3.100000    1005.700000       2.666667     116.666667
50%          8.100000    1013.400000       4.600000     210.000000
75%         13.600000    1020.500000       6.700000     260.000000
max         30.800000    1050.800000      32.900000     360.000000

               Aarhus                                                \
                 Temp       Pressure      WindSpeed        WindDir
count   333110.000000  333110.000000  333110.000000  333110.000000
mean         8.290577    1013.352071       4.036376     201.261096
std          7.027572      11.277480       2.549404      82.166840
min        -24.300000     955.500000       0.000000      10.000000
25%          3.000000    1006.600000       2.100000     140.000000
50%          8.000000    1014.000000       3.600000     213.333333
75%         13.500000    1020.800000       5.600000     270.000000
max         30.900000    1050.000000      33.400000     360.000000

              Esbjerg                                                \
                 Temp       Pressure      WindSpeed        WindDir
count   333109.000000  332070.000000  333109.000000  333109.000000
mean         8.537116    1013.131439       4.892615     201.758338
std          6.743867      10.904699       2.681328      87.880378
min        -27.000000     959.300000       0.000000      10.000000
25%          4.000000    1006.954601       2.933333     126.666667
50%          8.333333    1014.127073       4.516667     216.666667
75%         13.582857    1019.861904       6.533333     273.703704
max         54.000000    1049.300000      39.100000     360.000000

               Odense                                                \
                 Temp       Pressure      WindSpeed        WindDir
count   333110.000000  333110.000000  333110.000000  333110.000000
mean         8.802755    1013.805596       4.848788     195.840053
std          6.924723      10.958942       2.768103      83.739036
min        -22.500000     959.700000       0.000000      10.000000
25%          3.700000    1007.300000       2.766667     126.666667
50%          8.600000    1014.433333       4.600000     210.000000
75%         13.950000    1021.000000       6.635696     260.000000
max         49.900000    1048.900000      62.521795     360.000000

             Roskilde
                 Temp       Pressure      WindSpeed        WindDir
count   333109.000000  332346.000000  333109.000000  333109.000000
mean         8.264180    1012.839357       4.835396     202.708912
std          7.124592      11.739851       2.755634      86.188538
min        -21.833333     959.800000       0.000000      10.000000
25%          3.000000    1006.100000       2.766667     130.000000
50%          8.000000    1013.833333       4.433333     220.000000
75%         13.700000    1020.700000       6.574359     270.000000
max         32.000000    1048.100000      25.000000     360.000000
```

FIGURE 1: Description and summarize of the original data.

By using the describe method, we may attain the mean, standard, maximum and minimum value of the data.

1.2 Data Cleaning

This step is designed for missing data processing. Using `df.isnull().values.any()` property, I first check to see if there are any missing values. Since there is a missing value (the code returns `True`), I use the `df.dropna()` function in the library to remove rows containing *NaN* from the data frame. The specific code is:

```
df.dropna(axis=0,how='any',inplace=True)
```

Where `axis=0` specifies the how the data would be deleted (that is, along the row direction); `how='any'` specifies that a line is deleted if it has any *NaN* values in it; `inplace=True` specifies that the modification is made in the original data frame, rather than returning a new data frame. Therefore, this code deletes all lines containing any *NaN* values in the original data frame. Here is the flow of the code:

Data Cleaning Algorithm
check whether there is missing values remove all rows concluding missing values

After checking the difference between original number of lines and lines after deleting, 1040 lines with missing value have been cleaned successfully by using build in function.

1.3 Data Statistics

This step is designed to describe the data with graphical visualization, and I am going to provide reflections toward anomalies based on the data description.

Initially, I imported `matplotlib.pyplot` to generate the plot diagram. The basic idea to visualize data in plot diagram is store data of corresponding X axis and Y axis into two lists. Since the question requires to draw a plot of temperature versus time in May 2006 for the city of Odense, I initially create two list `Odense_time_list` and `Odense_temp_list`. Using for loop to append time and temperate value to two list respectively.

Since we only need data from May 2006, we need to do a second round of cleaning of the extracted temperature and time data. The basic idea is to convert the `datetime.datetime` format time in the data frame to the class of `pandas._libs.tslibs.timestamps.Timestamp` for easier comparison. The method of conversion is the `pd.Timestamp()` method through `from datetime import datetime`. Using the for loop again to check whether the value in `Odense_time_list` is in May 2006. If the value of time does in the period in May 2006, we may append the time and temperature into `print_time_x` and `print_time_y` list at the same time because time and temperature are corresponding in `Odense_time_list` and `Odense_temp_list`.

At last, I use `np.array()` to convert the list to a NumPy array and `plt.scatter()` to plot the scatter plots. `plt.show()` is used to display the plot but doesn't modify the plot. I unified the font of title, label and ticks into *Times New Roman*, and displayed the horizontal time as a scale every 4 days and set the diagonal representation to make it more beautiful. Here is the flow of the code:

Data Visualization Algorithm

```
retrieve temperature data for the city of Odense
for every row saved in Odense data
        save the temperature data and the exact time of each data point to two lists respectively
set the start and end point of May 2006, define the time using string and convert it into timestamp for comparison
create two new list for second round of data cleaning to store x-axis and y-axis values
for every value in time list
        if the time value is in the May 2006
                save the value of the same index of time and temperature list into new one
converts the second-round cleaning time and temperature lists to a NumPy array
set the style of the plot diagram
draw the scatter plot of temperature vs time
draw the plot
```

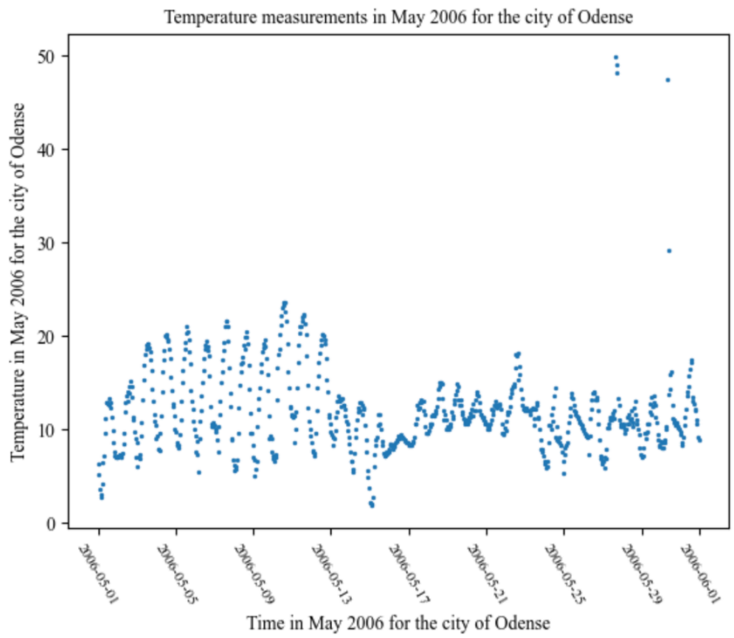


FIGURE 2: Temperature measurements in May 2006 for the city of Odense

According to the data description in FIGURE 1, there exits some anomalies for the data. For example, the highest temperature in Esbjerg, Denmark is 54.000 Celsius degrees while Court (1949) and El Fadli et al. (2012) points the highest temperature in the earth is only 56.7 Celsius degrees from 1911. It is quite impossible for Denmark, a northern European country to attain such high temperature. Also, according to the FIGURE 2, we can see that there are 5 points not on the curve, among which their temperature is significantly higher than other data, which is not reliable.

We may handle these anomalies in the next section.

2 OUTLIERS

In this section, we describe the steps taken to identify and handle outliers in the data. Outliers are observations that significantly deviate from the majority of the data points and can have a strong impact on statistical analyses. To identify outliers, we first calculate the z-score for each data point and use linear interpolation if z-score is out of the allowable range. Additionally, we visually inspected the data using scatterplots and boxplots to identify any further outliers that may have been missed by the z-score method. Overall, these pre-processing steps helped to ensure that our analyses were not affected by outliers and that our results were reliable.

2.1 Z-score outlier detection

Z-score is a standardization method used to measure how far a data point deviates from the mean. It is based on the sample mean and standard deviation and calculates the deviation of each data point from the mean. The deviation is then divided by the standard deviation to obtain the Z-score value. The formula for calculating Z-score is:

$$Z - score \ = \ (x \ - \ \mu) \, / \, \sigma \quad (1)$$

Here, x is the value of a data point, μ is the sample mean, and σ is the sample standard deviation. A higher Z-score indicates a greater deviation from the mean. By computing Z-score, we can identify outlier values in the data. We considered data points with a z-score greater than 3 or less than -3 as outliers, as they are more than three standard deviations away from the mean. These outliers were then replaced with the mean value of the data. To detect whether there is any outlier, I import the `scipy` package from the `stats`. Using code below to calculate the z-score:

```
z_score = np.abs(stats.zscore(y))

outlier_indices = np.where(z_score > 3) # set threshold to 3
```

In the code above, `zcore()` function is used to detect the z-score of each value in y and `abs()` is used to calculate the absolute value (which measures the distance between a data point and mean). `np.where(z-score > 3)` is used to save all the index whose z-score larger than 3 into the `outlier_indices`.

2.2 Linear interpolation

We removed any identified outliers by replacing them with linearly interpolated values. Linear interpolation is a method for estimating the value of a function between two known values. It assumes that the function is linear between those two points, so it draws a straight line between them and calculates the value at the point of interest based on that line.

For example, suppose we have two data points (x1, y1) and (x2, y2), and we want to estimate the value of the function f(x) at some point x0 between x1 and x2. Linear interpolation would first calculate the slope of the line connecting (x1, y1) and (x2, y2):

$$m = (y2 - y1) / (x2 - x1) \quad (2)$$

Then it would use that slope to find the y-value of the line at x0:

$$y0 = y1 + m * (x0 - x1) \quad (3)$$

This gives us an estimate of the value of f(x) at x0 based on the assumption that it is a linear function between x1 and x2. It should be noted that we shouldn't use the linear interpolation method for the first and the last value.

If the length of `outlier_indices` is not 0, it suggests that there do exist outliers. And we may use the for loop to reassign value using the math method we mentioned above by checking each index except the first and last value in the list.

2.3 Inspect the data using visualization

Based on the code of Part 1, after linear interpolation, the outliers may still exists, so we need to detect the z-score outliers again and again until no outliers found and we could generate a plot diagram again to see if there is any anomalies. The basic flow of the program is in the follow:

Outlier detection Algorithm

```
detect the z-score of each value in the list
set the threshold of z-score and store the index of  outliers to a list
while z-score outliers exists (length of list of index doesn't equal to 0)
        for each index in the list of outliers
```

```
                    if the index is not 0 or len(list)-1
                              list[index] = (list[index-1] + list[index+1]) / 2
          check whether the z-score outliers still exists
set the style of the plot diagram
draw the scatter plot of temperature vs time
draw the plot diagram
```

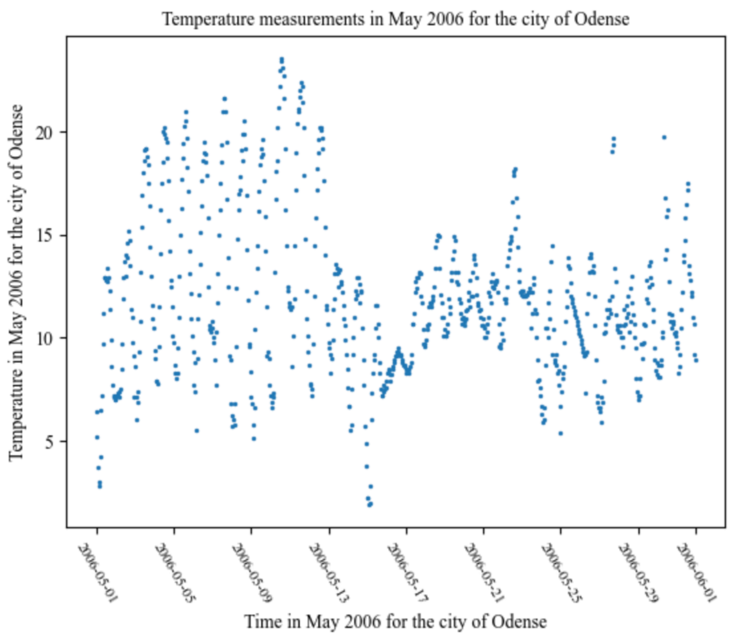


FIGURE 3: Temperature measurements in May 2006 for the city of Odense after tackling outliers pre-processing

After 4 turns of outliers' detection and adjust, we finally attain the plot diagram above, which is no more value with z-score larger than 3. Obviously, the data distribution is more consistent and centralized, and there are no extremely high temperature values. Although linear interpolation can help us to predict the value, it is not completely reliable.

3 DATA INFERENCE

In this section, we continue our data exploration and inference by investigating the correlations between different features in the dataset.

3.1 Data Correlation

In this section, we want to find the correlation between different features in the df_perth.pkl, a data file records the climate data of a city in Australia. We first import the data and print the data to get a basic understanding of the data.

```
                     temp  pressure  relative humidity  wind speed  \
DateTime
2005-01-01 00:00:00  24.7      1015                 68         3.3
2005-01-01 01:00:00  23.7      1015                 73         2.8
2005-01-01 02:00:00  23.1      1015                 70         3.3
2005-01-01 03:00:00  22.5      1015                 76         3.6
2005-01-01 04:00:00  22.0      1015                 75         2.6
...                   ...       ...                ...         ...
2005-12-31 19:00:00  23.7      1013                 47         6.9
2005-12-31 20:00:00  21.1      1013                 61         6.0
2005-12-31 21:00:00  18.5      1013                 75         4.2
2005-12-31 22:00:00  16.0      1013                 83         3.5
2005-12-31 23:00:00  13.4      1013                100         3.5
```

```
                     cloud cover  precipitation  PV production  \
DateTime
2005-01-01 00:00:00            0            0.0              0
2005-01-01 01:00:00            0            0.0              0
2005-01-01 02:00:00            0            0.0              0
2005-01-01 03:00:00            0            0.0              0
2005-01-01 04:00:00            0            0.0              0
...                          ...            ...            ...
2005-12-31 19:00:00            1            0.0              1
2005-12-31 20:00:00            1            0.0              0
2005-12-31 21:00:00            1            0.0              0
2005-12-31 22:00:00            1            0.0              0
2005-12-31 23:00:00            1            0.0              0
```

```
                     diffuse radiation, tilt  solar azimuth
DateTime
2005-01-01 00:00:00                        0           -2.5
2005-01-01 01:00:00                        0          -19.1
2005-01-01 02:00:00                        0          -33.4
2005-01-01 03:00:00                        0          -45.5
2005-01-01 04:00:00                        0          -55.4
...                                      ...            ...
2005-12-31 19:00:00                        1           61.1
2005-12-31 20:00:00                        0           52.2
2005-12-31 21:00:00                        0           41.4
2005-12-31 22:00:00                        0           28.5
2005-12-31 23:00:00                        0           13.3
```

FIGURE 4: Perth Weather Data

According to the observation of the data set, we may need to find the correlation between 9 features, which is *temp, pressure, relative_humidity, wind_speed, cloud_cover, precipitation, PV_production, diffuse_radiation_tilt* and *solar_azimuth*.

The Strategy to detect the correlation is using p-value through `SciPy` library. For each pair of data, the `np.corrcoef()` function from the NumPy library is used to compute the correlation coefficient, which measures the strength and direction of the linear relationship between the two variables. The correlation coefficient is stored in the `corr_coef` variable.

Then, the `pearsonr()` function from the `SciPy` library is used to compute the Pearson correlation coefficient and p-value for the same pair of data. The Pearson correlation coefficient is a measure of the linear relationship between two variables and ranges between -1 and 1, with 0 indicating no correlation. The p-value indicates the probability of observing a correlation as strong as the one computed in the sample if there is no correlation in the population.

Finally, if the p-value is less than 0.05, the code prints a message indicating that the two data have a significant correlation. The message includes the names of the feature, which are stored in a dictionary called `data_dic`.

Feature correlation Algorithm

```
read the data from the 'df_perth.pkl' file and print it.
create a DataFrame from the data and obtain the respective arrays for each feature
create a list of arrays from the above feature arrays and  a dictionary to map the array created
for array of specific feature in the list of all feature arrays collected
        for array of specific feature in the rest of the list of all feature arrays collected
                compute the correlation coefficient
                test whether the observed correlations are significant
                compute Pearson correlation coefficient and p-value for two arrays
                if p-value < 0.05
                        print features have significant correlations
```

We find that:

Table 1: Significant correlations between features

	solar_azimuth	temp	pressure	relative_humidity	wind_speed	cloud_cover	precipitation	PV_production	diffuse_radiation_tilt
solar_azimuth	X	√	X	√	√	X	X	√	√
temp	√	X	X	√	√	√	√	√	√
pressure	X	X	X	√	X	√	√	√	√
relative_humidity	√	√	√	X	√	√	√	√	√
wind_speed	√	√	X	√	X	√	X	√	√
cloud_cover	X	√	√	√	√	X	√	√	√
precipitation	X	√	√	√	X	√	X	√	X
PV_production	√	√	√	√	√	√	√	X	√
diffuse_radition_tilt	√	√	√	√	√	√	X	√	X

3.2 Data Inference

According to the Table 1, we find that there are significant correlations between precipitation and cloud cover. We now focus on the correlation between precipitation and cloud cover. We want to infer the probability of having moderate to heavy rain (> 1 mm/h) as a function of the cloud cover index.

To predict whether it will rain on a given day, we use a simple machine learning model for prediction.

First, we choose the days with precipitation greater than or equal to 1 as rainy day, and mark rainy days as 1 and non-rainy days as 0.

Then the correlation coefficient between the cloud cover index and the rain column is calculated, and a scatter plot is drawn to visualize the relationship. By using `corr()` function, we calculate the correlation coefficient between the cloud cover index and the rainy column.

At last we try to use `LogisticRegressionr()` to create an instance of the Logistic Regression model from the `sklearn.linear_model` module. The code does a logistic regression model fit on the data to predict the probability of moderate to heavy rain based on the cloud cover index using `fit(x, y)`. Finally, the trained model is used to predict the probability of moderate to heavy rain under a given cloud cover index. The predicted results are generated by `model.predict_proba ()`.

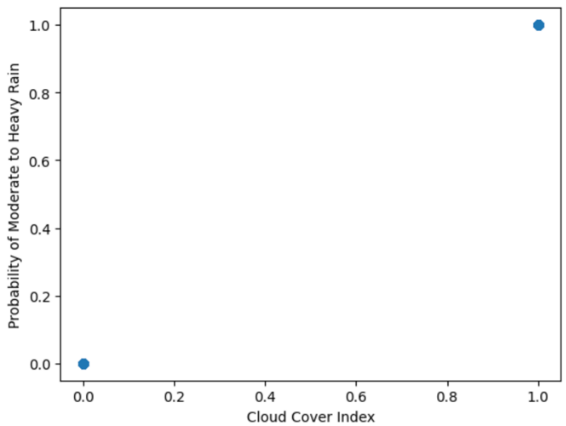
Rainy day inference Algorithm

import the liner_model from LogisticRegression library
select the rainy days if precipitation >= 1
calculate the correlation coefficient between the cloud cover index and the rainy column
scatter plot to visualize the relationship
fit a logistic regression model to predict the probability (use ravel() instead of reshape(-1,1))
predict the probability of moderate to heavy rain for a given cloud cover index (set cloud cover index as 0.7)

Probability of moderate to heavy rain for cloud cover index 0.7 equals to 0.49. We may also set the data of could cover and predicted probability into two arrays using:

```
cloud_cover = np.linspace(0, 1, num=100)

predicted_probs = model.predict_proba(cloud_cover.reshape(-1, 1))[:, 1]
```



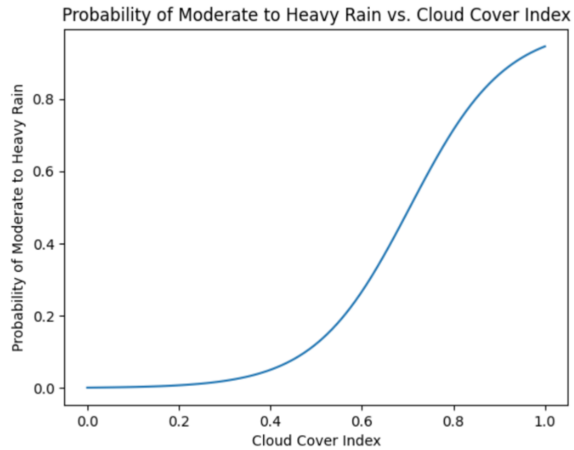


FIGURE 5: Correlation between Probability of Moderate to Heavy Rain and Cloud Cover Index

FIGURE 6: Probability of Moderate to Heavy Rain vs. Cloud Cover Index

Assume we want to predict the photovoltaic production (PV production) using multiple linear regression. According to the conslusion above, PV_production has significant correlations with temp, relatively_humidity, diffuse_radiation_tilt and solar_azimuth. (the relatively smaller p value)

We want to build a logistic regression model to predict the photovoltaic (PV) power generation. The four variables, temp, relatively_humidity, diffuse_radiation_tilt and solar_azimuth, are used as the input data (independent variables) to predict the PV production (dependent variable or output data). The four independent variables are merged into a DataFrame called variable, while y is a DataFrame containing the PV production variable. Model is a LogisticRegression object that fits the training data using the `fit()` method to obtain a well-trained model, `max_iter` is used to specify the maximum number of iterations for the model, and here it is set to a very large value.

```
x1 = diffuse_radiation_tilt.reshape(-1, 1)

x2 = solar_azimuth.reshape(-1, 1)

variable = df[['diffuse radiation, tilt','solar azimuth', 'relative humidity','temp']]

y = df[['PV production']].values.ravel()

model = LogisticRegression(max_iter=100000000)

model.fit(variable, y)
```

We successfully build a logistic regression model to predict the photovoltaic (PV) power generation then.

4 CONCLUSION

To sum up, this report uses Python programming language to conduct a comprehensive analysis of climate data. The study includes preliminary analysis, including data understanding, data cleaning, data statistics, and then outliers and data inference. The results show that there is a significant correlation between the various features of the data and the probability of moderate to heavy rain, which is a function of the cloud cover index. Based on the detected correlations, a multivariate model was developed to predict PV production.

Overall, this research helps the development of Python programming skills and data analysis and reasoning skills. The report also highlights potential challenges and limitations encountered during the study and provides directions for future research to improve the accuracy of the models developed.

REFERENCES

Court, A. (1949). How hot is Death Valley? Geographical Review, 39, 214-220.

El Fadli, K., Cerveny, R. S., Burt, C. C., Eden, P., Parker, D., Brunet, M., Peterson, T. C., Mordacchini, G., Pelino, V., Bessemoulin, P., Stella, J. L., Driouech, F., Abdel wahab, M. M., & Pace, M. B. (2012). World Meteorological Organization Assessment of the Purported World Record 58ºC Temperature Extreme at El Azizia, Libya (13 September 1922). Bulletin of the American Meteorological Society. doi: http://dx.doi.org/10.1175/BAMS-D-12-00093.1.

A APPENDICES

The appendix section saves the code of Course_weather_data from Jupyter Notebook.

A.1 Task I – Preliminary analysis

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sn

import scipy

from sklearn.metrics import mean_squared_error, mean_absolute_error

data = pd.read_pickle('weather-denmark-resampled.pkl')

ori = len(data) # to record the total number of data

print("the original data has", ori,"lines")

print('data:\n',data.describe())

df = pd.DataFrame(data)

# check whether there are missing values

print("Is there no value missing? ",df.isnull().values.any())

# remove all rows concluding missing values

df.dropna(axis = 0, how = 'any', inplace = True)

later = len(df)

print("successfully cleaned", (ori - later), "lines")

print(df.describe())
```

A.2 Task II – Outliers

```
import matplotlib.pyplot as plt

from datetime import datetime


odense = df['Odense'] # Pull data from Odense


# change the data type from series to list and get the exact time of each data

Odense_time_list = []

Odense_temp_list = []

for time, temp in odense['Temp'].items(): # Store time and temperature data separately

    Odense_time_list.append(time)

    Odense_temp_list.append(temp)


# set the time period we want

time_range_start = '2006-05-01 00:00:00' # time value in string

time_range_end = '2006-05-31 23:59:59'

timeArrayStart = datetime.strptime(time_range_start, "%Y-%m-%d %H:%M:%S") # convert the time from string to datetime

timeArrayEnd = datetime.strptime(time_range_end, "%Y-%m-%d %H:%M:%S")


# change the data type from <class 'datetime.datetime'> to <class 'pandas._libs.tslibs.timestamps.Timestamp'>

ts_timeArrayStart = pd.Timestamp(timeArrayStart)

ts_timeArrayEnd = pd.Timestamp(timeArrayEnd)


# select the data in May 2006

print_time_x = []

print_temp_y = []


for i in range(0,len(Odense_time_list)):

    if (Odense_time_list[i] >= ts_timeArrayStart) and (Odense_time_list[i] <= ts_timeArrayEnd): # if time in May 2006

        print_time_x.append(Odense_time_list[i])

        print_temp_y.append(Odense_temp_list[i])
```

```
#converts list to a NumPy array
x = np.array(print_time_x)
y = np.array(print_temp_y)

# set the style of the diagram (font style of title, label, x-axis and y-axis ticks style
and size of the scatter points)
plt.title('Temperature measurements in May 2006 for the city of Odense',
fontdict={'family':'Times New Roman', 'size':10})
plt.xlabel('Time in May 2006 for the city of Odense',family = 'Times New Roman', size = 10)
plt.ylabel('Temperature in May 2006 for the city of Odense',family = 'Times New Roman', size
= 10)
plt.yticks(fontproperties = 'Times New Roman', size = 10)
plt.xticks(fontproperties = 'Times New Roman', size = 8)

# draw the plot
plt.scatter(x, y, s = 2)
plt.xticks(rotation=300)
plt.show()

from scipy import stats

# use z-score method to detect outliers
z_score = np.abs(stats.zscore(y))
outlier_indices = np.where(z_score > 3) # set threshold to 3
count = 0

# replace outliers with linear interpolation
while len(outlier_indices[0]) > 0:
    count += 1
    for i in range(len(outlier_indices[0])):
        if outlier_indices[0][i] != 0 and outlier_indices[0][i] != len(y)-1:
            y[outlier_indices[0][i]] = (y[outlier_indices[0][i]-1] +
y[outlier_indices[0][i]+1]) / 2
    z_score = np.abs(stats.zscore(y))
```

```
    outlier_indices = np.where(z_score > 3)


# set the style of the diagram (font style of title, label, x-axis and y-axis ticks style
and size of the scatter points)
plt.title('Temperature measurements in May 2006 for the city of Odense',
fontdict={'family':'Times New Roman', 'size':10})
plt.xlabel('Time in May 2006 for the city of Odense',family = 'Times New Roman', size = 10)
plt.ylabel('Temperature in May 2006 for the city of Odense',family = 'Times New Roman', size
= 10)
plt.yticks(fontproperties = 'Times New Roman', size = 10)
plt.xticks(fontproperties = 'Times New Roman', size = 8)


# draw the plot
plt.scatter(x, y, s = 2)
plt.xticks(rotation=300)
plt.show()


print("After", count, "turns of ouliters detection and adjust")
```

A.3 Task III – Correlation

```
from scipy.stats import pearsonr


# read the data
perth_data = pd.read_pickle('df_perth.pkl')
print(perth_data)


# attain the data list respectively
df = pd.DataFrame(perth_data)
temp = df['temp'].values
pressure = df['pressure'].values
relative_humidity = df['relative humidity'].values
wind_speed = df['wind speed'].values
cloud_cover = df['cloud cover'].values
precipitation = df['precipitation'].values
PV_production = df['PV production'].values
```

```
diffuse_radiation_tilt = df['diffuse radiation, tilt'].values
solar_azimuth = df['solar azimuth'].values

from sklearn.linear_model import LogisticRegression

# first of all select the rainy days if precipitation >= 1

rainy = []
cloud_cover_co = []

for i in range(0,len(precipitation)):
    if precipitation[i] > 1:
        rainy.append(1)  # set rainy days as 1
        cloud_cover_co.append(cloud_cover[i])
    else:
        rainy.append(0)  # set non-rainy days as 0
        cloud_cover_co.append(cloud_cover[i])



df_rainy = pd.DataFrame(rainy,columns= ['rainy'])
df_cloud_cover = pd.DataFrame(rainy,columns= ['cloud cover'])

# then we need to calculate the correlation coefficient between the cloud cover index and
the rainy column:
corr = df_cloud_cover['cloud cover'].corr(df_rainy['rainy'])
print(corr)

# scatter plot to visualize the relationship
plt.scatter(df_cloud_cover['cloud cover'], df_rainy['rainy'])
plt.xlabel('Cloud Cover Index')
plt.ylabel('Probability of Moderate to Heavy Rain')
plt.show()

# fit a logistic regression model to predict the probability
```

```
x = df_cloud_cover['cloud cover'].values.reshape(-1, 1)
y = df_rainy['rainy'].values.ravel()  # use ravel() instead of reshape(-1,1)

model = LogisticRegression()
model.fit(x, y)

# predict the probability of moderate to heavy rain for a given cloud cover index
cloud_cover_index = 0.7
predicted_proba = model.predict_proba([[cloud_cover_index]])[:, 1]  # extract probability of class 1
print(f"Probability of moderate to heavy rain for cloud cover index {cloud_cover_index}: {predicted_proba[0]:.2f}")
data_lst = [temp, pressure, relative_humidity, wind_speed, cloud_cover, precipitation, PV_production, diffuse_radiation_tilt, solar_azimuth]
data_dic = {0:'temp', 1:'pressure', 2:'relative_humidity', 3:'wind_speed', 4:'cloud_cover', 5:'precipitation', 6:'PV_production', 7:'diffuse_radiation_tilt', 8:'solar_azimuth' }

for i in range(len(data_lst)):
    for j in range(i+1, len(data_lst)):

        # Compute the correlation coefficient
        corr_coef = np.corrcoef(data_lst[i], data_lst[j])[0, 1]
        #print(np.corrcoef(df, df))

        # test whether the observed correlations are significant
        # Compute Pearson correlation coefficient and p-value for two arrays
        corr, p_value = pearsonr(data_lst[i], data_lst[j])

        if p_value < 0.05:
            print(data_dic[i], "has significant correlations with", data_dic[j])
# Set up the data
cloud_cover = np.linspace(0, 1, num=100)
predicted_probs = model.predict_proba(cloud_cover.reshape(-1, 1))[:, 1]

# Plot the data
```

```
plt.plot(cloud_cover, predicted_probs)
plt.xlabel('Cloud Cover Index')
plt.ylabel('Probability of Moderate to Heavy Rain')
plt.title('Probability of Moderate to Heavy Rain vs. Cloud Cover Index')
plt.show()

x1 = diffuse_radiation_tilt.reshape(-1, 1)
x2 = solar_azimuth.reshape(-1, 1)

variable = df[['diffuse radiation, tilt','solar azimuth', 'relative humidity','temp']]

y = df[['PV production']].values.ravel()

model = LogisticRegression(max_iter=100000000)
model.fit(variable, y)
```