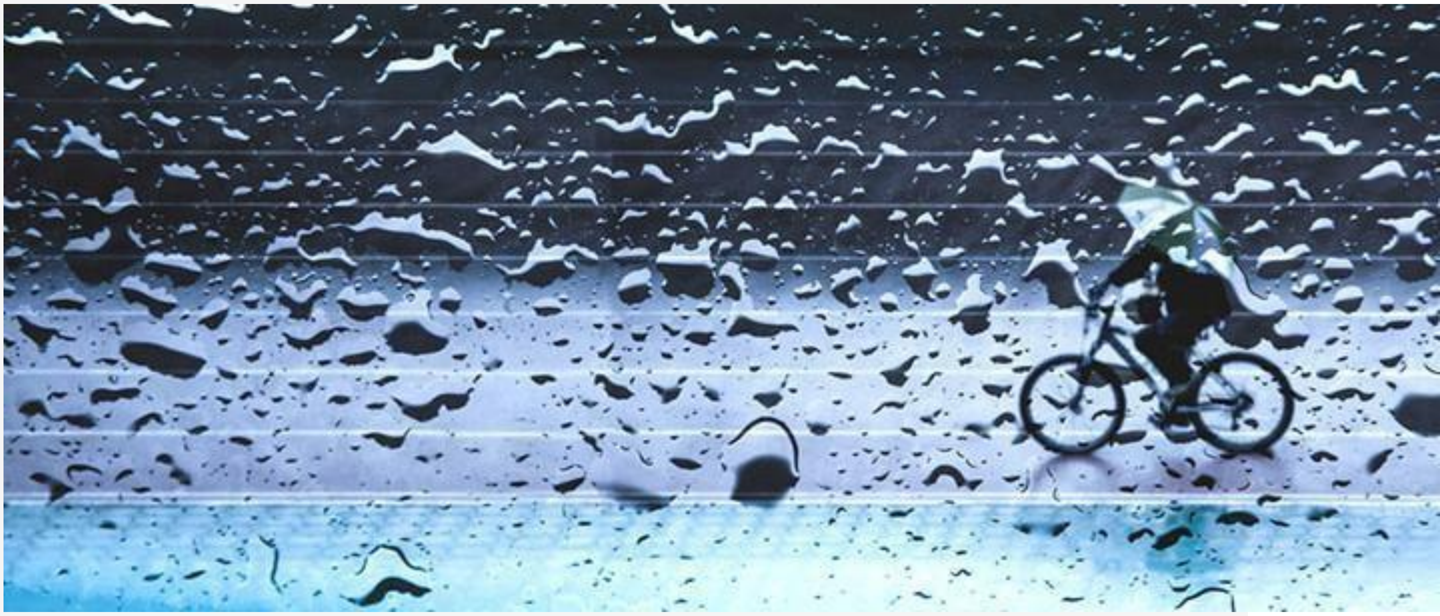
Experiment No:12

Aim:Locatedataset(e.g.,sample_weather.txt)forworkingonweatherdatawhichreadsthetext input files and findsaverage for temperature, dew pointand wind speed.

Theory: AnalysisofWeatherdatausingPandas,Python,andSeaborn:



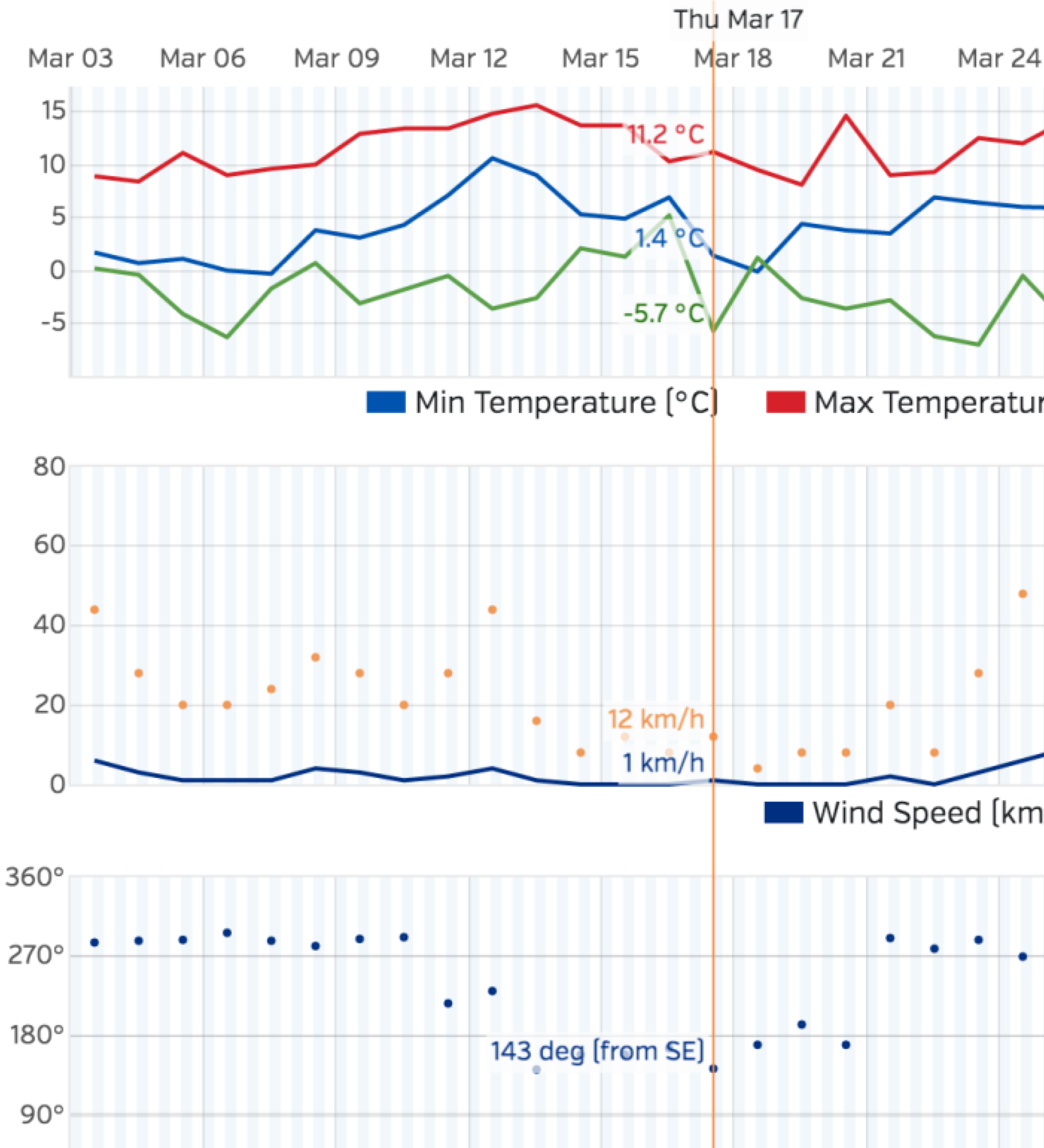
The most recent poston this site was an analysis of how often people cycling to work actuallygetrainedonin differentcitiesaround theworld.You cancheckit outhere.
The analysis was completed using data from the Wundergroundweather website, Python,specifically the Pandas and Seabornlibraries. In this post, I will provide the Python code toreplicatetheworkandanalyseinformationforyourowncity.Duringtheanalysis,IusedPythonJupyternotebookstointeractivelyexploreandcleansedata;there'sasimplesetupifyouelecttouse something like the Anaconda Python distributionto install everything you need.
If you want to skip data downloading and scraping, all of the data I used is available to downloadhere.

ScrapingWeatherData

Wunderground.comhasa"PersonalWeatherStation(PWS)"networkforwhich fantastic historical weather data is available – covering temperature, pressure, wind speed anddirection,andofcourserainfallinmm–allavailableonaper-minutelevel.Individualstationscan be examined at specific URLS, for example herefor station "IDUBLIND35".
There'snoofficialAPIforthePWSstationsthatIcouldsee,butthereisaverygoodAPIforforecast data. However,CSV format data with hourly rainfall, temperature, and pressureinformation can be downloaded from the website with some simple Python scripts.

Graphs | Table

Weather History Graph
March 3, 2016 - April 2, 2016



Wundergroundhaveanexcellentsitewithinteractivegraphstolookatweatherdataonadaily,monthly, andyearlylevel.DataisalsoavailabletodownloadinCSVformat,whichisgreatfordata science purposes.

```
import requests

import pandas as pd

fromdateutilimportparser,rrule

fromdatetimeimportdatetime,time, date

import time

defgetRainfallData(station,day,month,year):

"""

Functiontoreturnadataframeofminute-levelweatherdataforasingleWundergroundPWSstation.

Args:

station(string):StationcodefromtheWundergroundwebsiteday

(int): Dayof month forwhich data isrequested

month(int):Monthforwhichdataisrequestedyea

r (int): Year for which data is

requestedReturns:

Pandas Dataframe with weather data for specified station and

date."""

url
="http://www.wunderground.com/weatherstation/WXDailyHistory.asp?ID={station}&amp;day=
{day}&amp;month={month}&amp;year={year}&amp;graphspan=day&amp;format=1"

full_url=url.format(station=station,day=day,month=month,year=year)#R

equest data from wunderground data

response=requests.get(full_url,headers={'User-
agent':'Mozilla/5.0(WindowsNT6.1)AppleWebKit/537.36(KHTML,likeGecko)Chrome
/41.0.2228.0Safari/537.36'})
```

```
print("Workingon{}".format(station))d

ata[station] = []

for date in dates:

# Print period status update messages

if date.day % 10 == 0:

print("Workingondate:{}forstation{}".format(date,station))don

e = False

while done == False:

try:

weather_data=getRainfallData(station,date.day,date.month,date.year)do

ne= True

except ConnectionError as e:

#MaygetratelimitedbyWunderground.com,backoffifso.print

("Got connection error on {}".format(date))

print("Willretryin{}seconds".format(backoff_time))ti

me.sleep(10)

# Add each processed date to the overall

datadata[station].append(weather_data)

# Finally combine all of the individual days and output to CSV for

analysis.pd.concat(data[station]).to_csv("data/{}_weather.csv".form
```

```
monthly.replace({"Rainy": {True: "Wet", False:"Dry"}},

inplace=True)monthly['month_name']=monthly['month'].apply(lambdax:calendar.mo

nth_abbr[x])# Get aggregate stats for each day in the dataset on rain in general - for

heatmaps.rainy_days = data.groupby(['day']).agg({

"rain": {"rain": lambda x: (x &gt;

0.0).any(),"rain_amount": "sum"},

"total_rain": {"total_rain":

"max"},"get_wet_cycling":{"get_wet_cycling"

:"any"}

})

# clean up the aggregated data to a more easily analysed

set:rainy_days.reset_index(drop=False,inplace=True)#removethe'day'astheindex

rainy_days.rename(columns={"":"date"},inplace=True)#Theoldindexcolumndidn'thaveaname
- add "date" as name

rainy_days.columns = rainy_days.columns.droplevel(level=0) # The aggregation left us with
amulti-index

# Remove the top level of this index.

rainy_days['rain']=rainy_days['rain'].astype(bool)#Changethe"rain"columntoTrue/Falsevalue
s

#Addthenumberofrainyhoursperdaythistotherainy_daysdataset.temp =

data.groupby(["day", "hour_of_day"])['raining'].any()

temp =

temp.groupby(level=[0]).sum().reset_index()temp.rename(colum

ns={'raining': 'hours_raining'},

inplace=True)temp['day']=temp['day'].apply(lambdax:x.to_datet

ime().date())

(rainy_days), station)
```

```
print "It was wet while cycling {} working days of {} at
{}".format(wet_cycling['get_wet_cycling'].sum(),

len(wet_cycling),

station)

print "You get wet cycling {} % of
thetime!!".format(wet_cycling['get_wet_cycling'].sum()*1.0*100/len(wet_cyclin
g))
```

At this point, we have two basic data frames which we can use to visualise patterns for the citybeing analysed.

VisualisationusingPandasandSeaborn

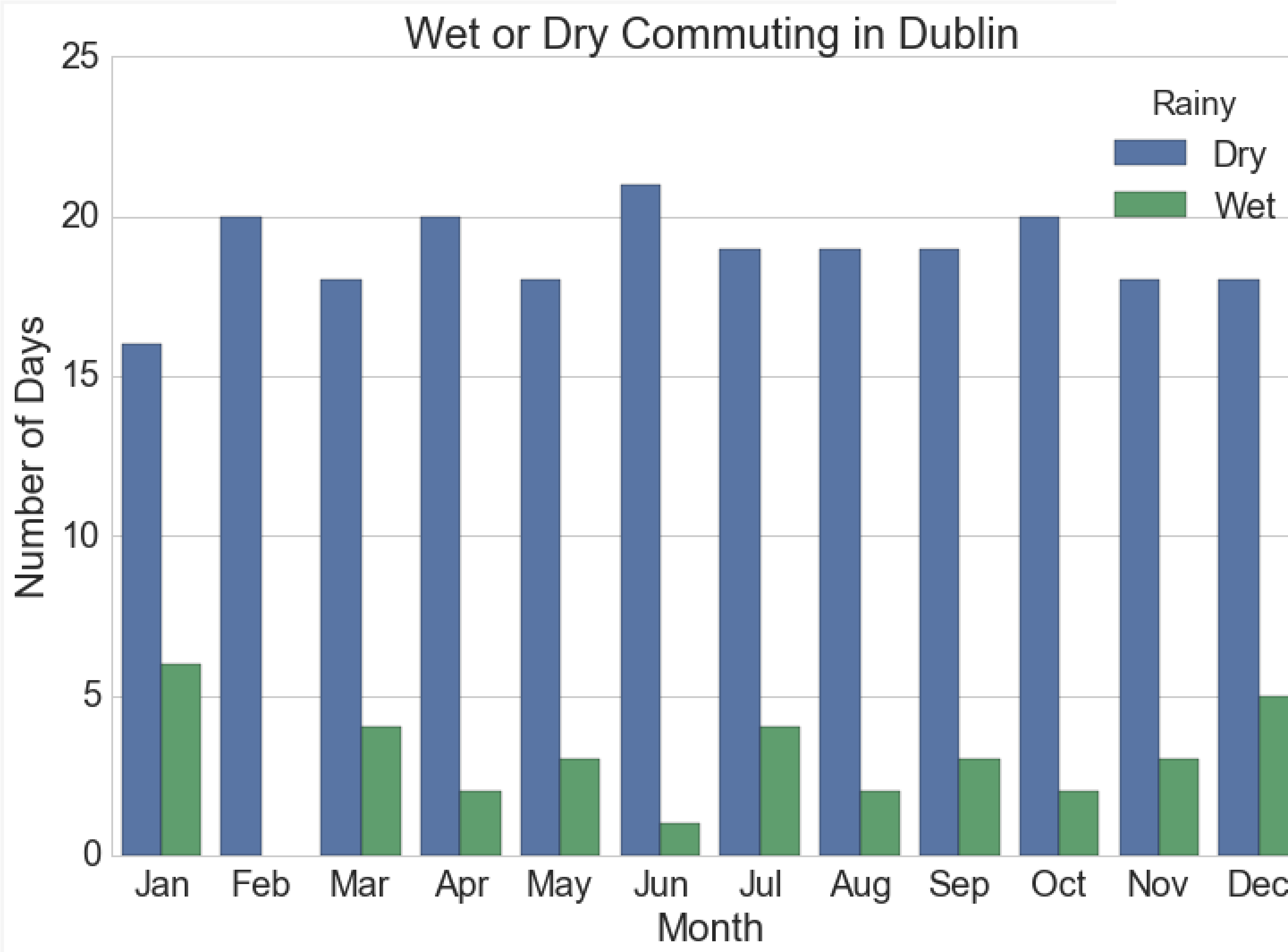
BarchartofMonthlyRainyCycles

The monthly summarised rainfall data is the source for this

chart.# Monthly plot of rainy days

```
plt.figure(figsize=(12,8))sns.set_style("w
hitegrid")sns.set_context("notebook",font
_scale=2)
```

```
sns.barplot(x="month_name",y="Days",hue="Rainy",data=monthly.sort_values(['month','
Rainy']))

plt.xlabel("Month")plt.ylabel

("NumberofDays")

plt.title("WetorDryCommutingin{}".format(station))
```



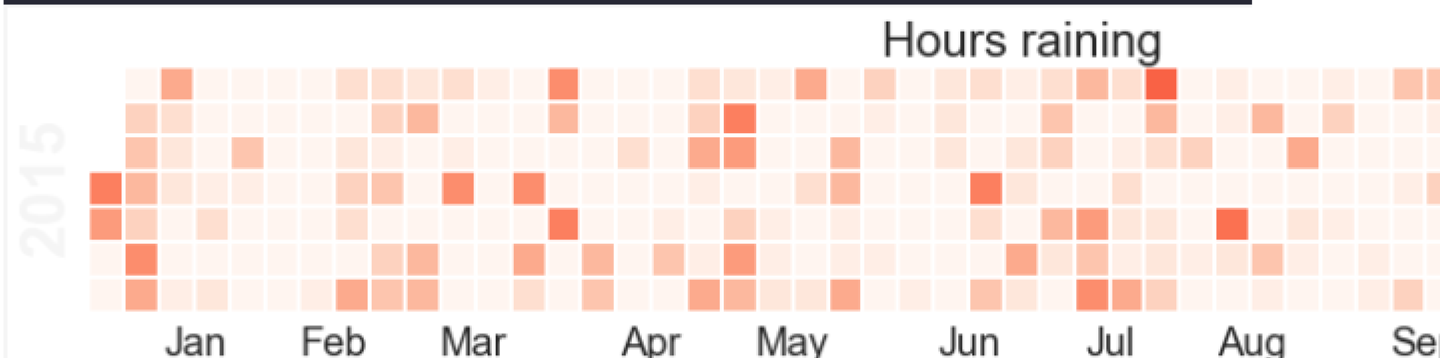
Number of days monthly when cyclists get wet commuting at typical work times in Dublin,Ireland.

Heatmaps of Rainfall and Rainy Hours per day

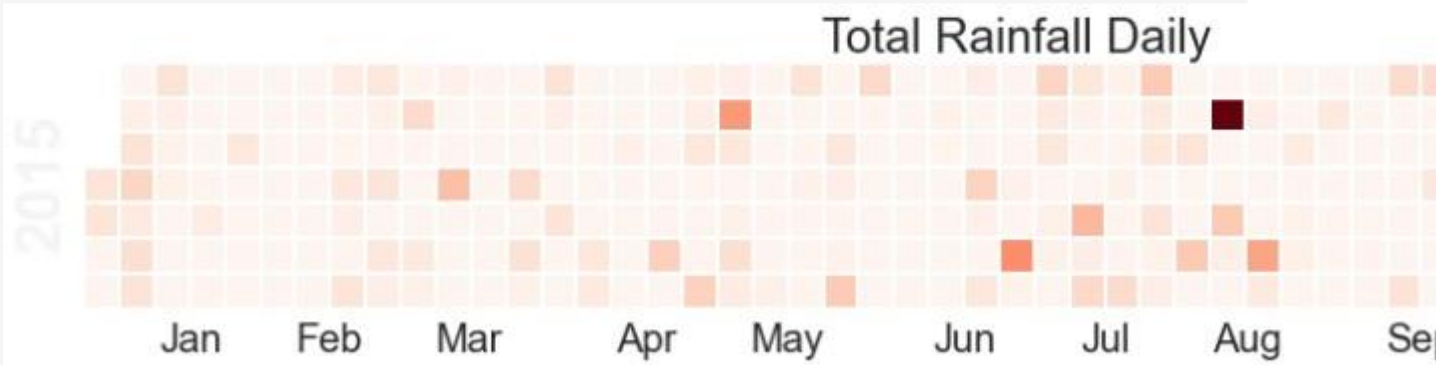
The heatmaps shown on the blog post are generated using the “calmap” python library,installable using pip. Simply import the library, and form a Pandas series with a DateTimeIndexandthelibrarytakescareoftherest.Ihadsomedifficultyherewithfontsizes,sohadtoincreasethesize of the plot overall to counter.

```
import calmap

temp =

rainy_days.copy().set_index(pd.DatetimeIndex(analysis['rainy_days']['date']))#temp.s

et_index('date',inplace=True)

fig, ax = calmap.calendarplot(temp['hours_raining'], fig_kws={"figsize":(15,4)})
```

```
plt.title("Hours raining")
fig, ax = calmap.calendarplot(temp['total_rain'],
fig_kws={"figsize":(15,4)})plt.title("TotalRainfall Daily")
```



The Calmap package is very useful for generating heatmaps. Note that if you have highlyoutlying points of data, these will skew your color mapping considerably – I'd advise removingor reducing them for visualisation purposes.



Comparison of Every City in Dataset

Tocompareeverycityinthedataset,summarystatsforeachcitywerecalculatedinadvanceandthen the plot was generated using the seaborn library. To achieve this as quickly as possible, Iwrapped the entire data preparation and cleansing phase described above into a single functioncalled “analyse data”, used this function on each city’s dataset, and extracted out the pieces ofinformation needed for the plot

.

Here’sthewrappedanalyse_datafunction:

```
defanalyse_station(data_raw,station):

"""

Functiontoanalyseweatherdataforaperiodfromoneweatherstation.Args:

data_raw(pd.DataFrame):PandasDataframemadefromCSVdownloadedfromwun
derground.com

station (String): Name of station being analysed (for

comments)Returns:

dict: Dictionary with analysis in keys:

data:Processedandcleanseddata

monthly:Monthlyaggregatedstatisticsonrainfalletc.

wet_cycling: Data on working days and whether you get wet or not

commutingrainy_days: Daily total rainfall for each day in dataset.

"""

#Givethevariablessomefriendliernamesandconverttypesasnecessary.data_r

aw['temp']=data_raw['TemperatureC'].astype(float)data_raw['rain'] =

data_raw['HourlyPrecipMM'].astype(float)data_raw['total_rain'] =

data_raw['dailyrainMM'].astype(float)data_raw['date']=data_raw['DateUT

C'].apply(parser.parse)
```

```
data_raw['humidity'] =

data_raw['Humidity'].astype(float)data_raw['wind_direction']=d

ata_raw['WindDirectionDegrees']data_raw['wind']=

data_raw['WindSpeedKMH']

# Extract out only the data we need.

data=data_raw.loc[:,['date','station','temp','rain','total_rain','humidity','wind']]

data = data[(data['date'] &gt;= datetime(2015,1,1)) &amp; (data['date']
&lt;=datetime(2015,12,31))]

# There's an issue with some stations that record rainfall ~-2500 where data is missing.

if (data['rain'] &lt; -500).sum() &gt; 10:

print("There's more than 10 messed up days for

{}".format(station))# remove the bad samples

data = data[data['rain'] &gt; -500]

# Assign the "day" to every date entry

data['day'] = data['date'].apply(lambda x: x.date())

# Get the time, day, and hour of each timestamp in the

datasetdata['time_of_day'] = data['date'].apply(lambda x:

x.time())data['day_of_week'] = data['date'].apply(lambda x:

x.weekday())data['hour_of_day'] =

data['time_of_day'].apply(lambda x: x.hour)# Mark the month for

each entry so we can look at monthly patternsdata['month'] =

data['date'].apply(lambda x: x.month)

# Is each time stamp on a working day (Mon-Fri)

data['working_day']=(data['day_of_week']&gt;=0)&amp;(data['day_of_week']&lt;=4)#Class

ifyintomorningoreveningtimes(assumingtravelbetween8.15-9amand5.15-6pm)

data['morning']=(data['time_of_day']&gt;=time(8,15))&amp;(data['time_of_day']&lt;=tim
e(9,0))
```

```
data['evening'] = (data['time_of_day'] &gt;= time(17,15)) &amp; (data['time_of_day']
&lt;=time(18,0))

# If there's any rain at all, mark

that!data['raining'] = data['rain']

&gt; 0.0

# You get wet cycling if its a working day, and its raining at the travel

times!data['get_wet_cycling'] = (data['working_day']) &amp; ((data['morning'] &amp;

data['rain']) |(data['evening'] &amp; data['rain']))

# Looking at the working days only:

wet_cycling=data[data['working_day']==True].groupby('day')['get_wet_cycling'].any()

wet_cycling =

pd.DataFrame(wet_cycling).reset_index()# Group by

month for display

wet_cycling['month'] = wet_cycling['day'].apply(lambda x: x.month)

monthly =

wet_cycling.groupby('month')['get_wet_cycling'].value_counts().reset_index()monthly.re

name(columns={"get_wet_cycling":"Rainy", 0:"Days"},

inplace=True)monthly.replace({"Rainy": {True: "Wet", False:"Dry"}},

inplace=True)monthly['month_name']=monthly['month'].apply(lambdax:calendar.mont

h_abbr[x])

# Get aggregate stats for each day in the

dataset.rainy_days = data.groupby(['day']).agg({

"rain": {"rain": lambda x: (x &gt;

0.0).any(),"rain_amount": "sum"}
```

```
plt.suptitle("What percentage of your cycles to work do you need a raincoat?",
y=1.05,fontsize=32)

plt.title("Based on Wundergroud.com weather data for 2015",

fontsize=18)plt.xticks(rotation=60)plt.savefig("images/city_comparison_wet_c

ommutes.png",bbox_inches='tight')
```