Data Analyst Nanodegree

Project 1

Analyzing the NYC Subway Dataset

Michal Mašika

Content

1. Introduction

Demand forecasting is one of the most important elements in the analysis of transportation system. In this project we create a simple statistical model which should help us to predict the usage of the metro in New York City (NYC) at any given time. In particular, we are interested in examining the question, whether weather conditions (e.g. rain) influence the demand for subway.

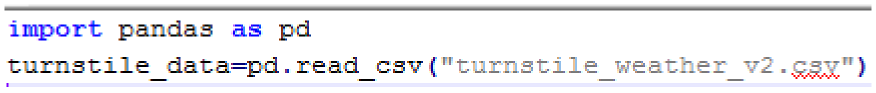
Answering these questions is important as it might help transportation company (here Metro Transit Authority in NYC) to make better investments with respect to the metro fleet, the railroad as well as the staff and so to improve the quality.

This project is the first project of Data Analyst Nanodegree. It is basically the final project of Introduction to Data Science course and is structured according to the instructions. Section 2 deals with answering of short questions. In the section 3 you can find my code to the problem sets 2 to 5 of the course as well as some more comments. Section 4 provides some additional analysis to both previous sections. The last part contains all used references.

2. Short Questions

This section provides answers on given short questions. Before we can answer these questions we have to get and clean the data. Section 3.1 (Problem Set 2) deals with data wrangling and provides the code we used.

However, we used the improved csv file (turnstile_weather_v2.csv) provided by Udacity. In the very first step we have to import this file to python. To this end we use pandas library (the code is provided below).



```
import pandas as pd
turnstile_data=pd.read_csv("turnstile_weather_v2.csv")
```

2.1. Statistical Test

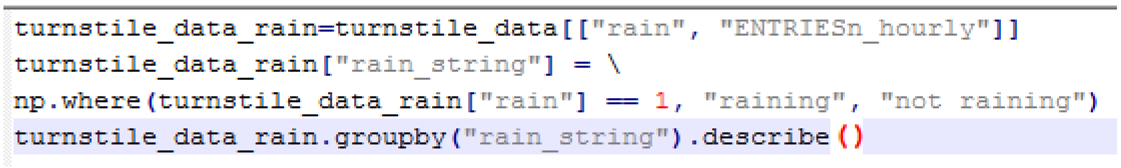
In this section we answer all short questions connected to the statistical test used (i.e. 1.1 to 1.4. from short questions docs).

In particular, we are interested if there is any difference in metro usage by different rainy conditions (i.e. between days with and without rain). So, in the very first step we can look at the distributions of hourly entries by rain and no-rain, where rain variable is an indicator (0 or 1) depending on whether rain occurred within the calendar day at the location.[1] Both histograms are provided in section 3.2 i). From that figure we see that **both distributions seem to be not-normal**. Indeed, they are skewed to the right. The normality test is provided

[1] In the section 4.1 we conduct the same analysis for additional rain variables constructed by means of precipitation rate as well as weather conditions.

below in this section. In addition, the figure shows that the pattern of metro usage doesn't change according to the rain.

Before we conduct the appropriate statistical test let's look at summary statistics of hourly entries by rain conditions.



```
turnstile_data_rain=turnstile_data[["rain", "ENTRIESn_hourly"]]
turnstile_data_rain["rain_string"] = \
np.where(turnstile_data_rain["rain"] == 1, "raining", "not raining")
turnstile_data_rain.groupby("rain_string").describe()
```

conditions	**n**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
not raining	33064	1845.54	2878.77	0	269	893	2197	32814
raining	9585	2028.20	3189.43	0	295	939	2424	32289

The summary statistics show that the samples differ in both the size and the variance (the number of observations is larger by days without rain than by with rain). Moreover, **the mean as well as all percentiles of hourly entries are higher when it is raining than when it is not raining.** So, there is an observable difference in metro usage. Nevertheless, the question remains whether this difference is statistically significant. To this end we conduct different statistical tests.

- *Which Statistical Test & Assumptions (i.e. 1.1. and 1.2. of short questions):*

There are several tests which might be helpful for mean comparison of two samples. In this project we are going to shortly discuss three of them. **Student's t test, Welch's t test and Mann-Whitney U test (M-W test)**. All three tests assume that the distributions are independent of each other.

Student's t test assumes that the data is normal distributed and the variances are the same in both samples. From the previous analysis it is very likely that both assumptions don't hold. Thus, this test is not appropriate here.

In contrary, **Welch's t test** doesn't assume the equality of variances but **assumes** that the **underlying distribution** of the data is **normal**. In the analysis above (figure and summary statistics) we have seen that the data is very likely not normal. The normality can be also statistically tested.

```
import pandas
import scipy.stats
(k2, pvalue) = scipy.stats.normaltest(turnstile_data["ENTRIESn_hourly"])
```

This test is based on D'Agostino and Pearson's test which examines whether a sample differs from a normal distribution. The null hypothesis is that all the values were sampled from a normal distribution. **Here, the P-value is 0.0 meaning that it is very likely that the data is not normal distributed.**[2]

Therefore, it is very problematic to use the Welch's t test as one of the underlying assumptions doesn't hold.

Mann-Whitney U test is a non-parametric test. It doesn't make any assumption about the underlying distribution of the data. From this perspective, it seems that this test is the most appropriate for our dataset.

Indeed, there is a large discussion in the literature under what condition which test should be used.[3] It is very often argued that based on central limit theorem if the data size is sufficiently large, we can use Welch's t test even though the data isn't normal.[4] The M-W test is still preferable, though as it doesn't make any assumptions about underlying distribution. The usage of M-W test can be problematic when the variances of the samples are not same. In this case Welch's t test outperforms the M-W test.

Thus, conducting of both tests can be problematic here. We are going to use the non-parametric test (i.e. **Mann-Whitney U test**).[5]

The code of conducting M-W U test can be found in section 3.2. ii). The underlying **null hypothesis** states that the **distributions** of both groups (hourly entries with and without rain) **are identical**. I.e. there is a 50 % probability that a value randomly selected from one population exceeds a value randomly selected from the other population. In the literature M-W test is often used to compare the means or medians of two independent distributions.

[2] Used references: http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.normaltest.html
http://www.graphpad.com/guides/prism/6/statistics/index.htm?stat_qa_normality_tests.htm.

[3] Some examples: Stonehouse and Forrester (1998), Ruxton (2006), Zimmerman and Zumbo (1993), Fenstad and Skovlund (2000), Fagerland and Sandvik (2009).

[4] Thus, we could use it here as our data size is large enough (N=42649).

[5] For robustness checks we also conduct Welch's t test. The results can be found in section 4.1.

Given this null hypothesis we are going to use **two-tail P value** as we only argue that both distributions are different (we haven't hypothesized if the entries are higher with rain than without rain or vice versa).

- *Results and their interpretation (i.e. 1.3. and 1.4. of short questions):*

The code provided in section 3.2 ii) usually returns means (with and without rain), U-statistic and p-value where p-value is one-sided. As we want to use two-tail p-value, we must multiply it by two.

Unfortunately, in our case no p-value is provided by using a standard code for M-W test. Therefore, we have to make use of the fact that for large samples, U is approximately normally distributed. Having known that, we can manually compute the p-value. The formula and code are contained in section 3.2 ii).

Having done that we get following results: **Mean of entries on days with rain is equal to 2028.196, mean of entries on days without rain is 1845.539 and the computed p-value is 5.48 * 10^-6**. So, we can **reject the null hypothesis** at **significance level of 0.01%** (as 5.48 * 10^-6 < 0.0001) that both distributions (hourly entries at rainy days and hourly entries at no rainy days) are the same.[6]

Moreover, by using these results in combination with descriptive statistics shown in this section above, it is very likely that the metro usage on rainy days is higher than on days without rain. This can be confirmed by one-sided Welch's t test conducted in section 4.1. Linear regression in section 2.2 will examine this point even in more detail.

Section 4.1 provides additional tests (M-W test on other rainy variables as well as Welch's t test). The Welch's t test conducted on (daily) rain variable delivers very similar result to the result provided in this section (p-Value – 4.64*10^-7). This means that the means of both distributions are very likely different from each other.[7]

[6] Note, that the significance level is even smaller than 0.0001.

[7] By conducting both tests on variable "rain_time", which indicates, whether rain occurred at that point of time at the location, we can also reject null hypothesis. However, the descriptive statistics and the figure in section 4.3 indicate that there is a negative relationship. By using rain variable generated from the weather conditions variable we are not able to reject null hypothesis at 10% significance level, though.

2.2. Linear Regression

Both statistical tests and data visualization are very useful tools to get insights from our data. However, both of them can help us only partly to predict the demand for subway. In addition, these tools are mostly used to get a hint for relationship between 2 variables (max 3 variables in 3D figures).[8] If there are more variables having a potential influence on our dependent variable, other statistical tools might be more useful. In order to make predictions of hourly entries, we are going to use linear regression (ordinary least square). The formula for OLS is as follows:

$$y = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + .. + \theta_m x_m \ ,$$

where y is an output variable (also called dependent variable or explained variable). In this project hourly entries (***ENTRIESn_hourly***) constitute the dependent variable. x is set of inputs (also called independent variables or explanatory variables). Explanatory variables are basically factors which have an impact on dependent variable. Besides the intercept (***ones***) we **use** following factors as **explanatory variables**:

- ***weekday*** (indicator whether it is weekday or weekend) – figure in section 3.3 ii) suggests that weekdays are busier (i.e. more people use the subway) than weekends. Therefore, it might be useful to include this indicator into our regression.
- ***rain*** (indicator whether rain occurred during the day) - both the statistical test in 2.1 and figures in 2.3, 3.3 and 4.3 suggest that the hourly entries vary with rainy conditions. Other reason why include this variable into the model is that rain is the main variable of interest as we want to find out what is the impact of rain on the subway usage.
- ***weekdayrain*** (interaction term consisting of *rain* and *weekday*) - this variable captures the fact that the impact of rain on hourly entries might vary over different days (weekday vs. weekend). The intuition for this is as follows. When it is rainy during the weekend, people rather stay at home and therefore don't use the subway that much. During the weekdays we might observe a positive impact because the people are already out of their houses (due to jobs) and therefore if it is raining they are willing to use the subway more. They might substitute other vehicles.
- ***meantempi*** (the average daily temperature in Fahrenheit) – The higher temperature might cause that more people prefer to be outside (on the streets) than in the subway. Higher temperature might also leads to higher usage of other vehicles (car, bicycle,

[8] Note there are statistical tests considering more variables (e.g. F test).

bus). Moreover, nice weather at weekend might cause people to have short trips outside the city and therefore smaller subway usage. We also decided to include this variable in order to prevent omitted variable bias. There is a clear negative relationship between temperature and rain. So, in case temperature has an influence on hourly entries, the coefficient of rain variable will be biased due to omitted variable bias.[9]

- ***dummy variables (hour)*** – we can see in section 3.3 that the time has very likely an impact on the subway usage. Basically, there are peak hours where the subway is used more. We also observe that this impact is not linear. Therefore, it is better to use dummy variables for each hour, which capture different levels (i.e. each hour has different level on hourly entries). It is important to note that we don't create dummy variable for each hour to avoid perfect multicollinearity in our regression. We use midnight (*hour* = 0) as reference category.
- ***dummy variables (unit)*** – in the section 2.3 we observe that hourly entries vary over different stations. The same holds for different turnstile units. Intuitively, stations / units vary in their size, are located in different parts of city (very busy parts vs. not so busy parts), or more lines go through them. As we don't observe the size of the stations or the area, where stations are located, directly (we don't have this data), it is useful to use dummy variables which capture all this stations' (units') characteristics which might influence hourly entries. We decide to use dummy variables for turnstile units and not dummy variables for stations as this improves our coefficient of determination (R^2).[10] This is not that surprising as the number of unit dummies is larger than number of station dummies (239 to 206). Due to perfect multicollinearity issues we don't create a dummy variable for unit R012.[11]

There are also **other candidates** for **explanatory variables**, which we are finally **not** going to use:

- ***meanprecipi*** (the average amount of rain falling during a day) – we don't include this variable into our model as it is highly correlated with *rain* variable. Therefore, the inclusion of it would lead to potential multicollinearity problems and therefore the coefficients of interest might get unstable (i.e. the coefficients will be very sensitive to

[9] There is one disadvantage of considering temperature in the model, though. By including this variable the conditional number dramatically increases indicating multicollinearity or other numerical problems. We tested for multicollinearity by regressing meantempi on other independent variables. The R^2 was below 0.2 indicating no multicollinearity issues in the original regression.

[10] As robustness check we estimate the model with station dummies in section 4.2. We don't create the dummy variable for 34 ST-PENN station.

[11] Wissmann et al (2007) argue that the choice of reference category might have an effect on multicollinearity in the regression. To reduce the likelihood of multicollinearity, the category with highest frequency should be chosen as reference category (in our case unit R012 or the station 34 ST-PENN).

any changes in the model).[12] Moreover, the multicollinearity leads to higher variance of the coefficients and therefore it is more difficult to interpret the coefficients (Wissmann et al (2007)).

The only parameters not explained yet are θ's. These are the parameters we have to estimate. The goal of the regression is to find those parameters by minimizing the error (the difference between predictions and actual data). More precisely, we minimize the sum of squared differences (ordinary least squares). We can formalize it as follows:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{n}(predictions - actuals)^2 = \frac{1}{2m}\sum_{i=1}^{n}(\theta^T x - y)^2$$

There are several methods which we can use to minimize this function. One of the methods is gradient descent – this is a numerical method where we define a set of starting parameters and we iteratively update our parameters as long as we converge. We estimate our model by means of gradient descent in section 3.2 iii).

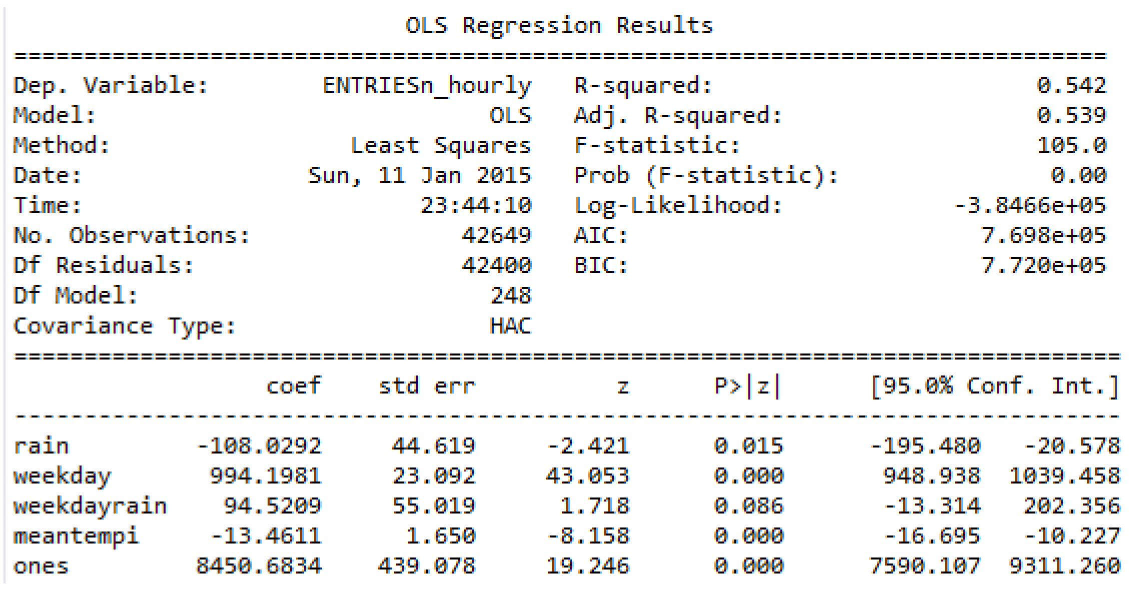
Here we use the OLS provided by statsmodels. Compared to gradient descent, this has at least two advantages. First, it is an analytical solution – in other words it is more precise. Second, it allows us to access the coefficient in very simple and appropriate way such that we can interpret them immediately. The code of the OLS model is provided in section 3.2 vi). Note, that we make use of heteroscedasticity and autocorrelation robust standard errors.

The table below provides the results of our estimation. The results indicate very strong positive impact of weekdays on the subway usage. Basically, the number of hourly entries on weekdays is larger by 994 than this number at weekends. This effect is significant at 1% significance level. Moreover, the temperature negatively affects the hourly entries (significant at 1%). This goes in line with our hypothesis that by higher temperature people don't prefer to sit in the underground. Finally, rain has a negative impact on weekends but positive impact during the week. This also goes in line with our expectation. While the first effect is significant at 5% significance level, the second is significant only at 10% significance level. As mentioned above, section 4.2 provides two robustness checks to this estimation (usage of precipitation rate as well as station dummies). The results in section 4.2 are consistent with these findings.

[12] In the section 4.2 we estimate this model with meanprecipi instead .

There are different ways how to measure the **goodness of the test**. First, we can **plot the residuals** and look if they are normally distributed with mean = 0 and some finite variance. We plot the residuals in section 3.2 iv).[13] Second we can look at the **coefficient of determination (R^2)**. This coefficient is defined as the share of the dependent variable variation that is explained by our model.[14] In our case this **coefficient is 54.2 %** which is higher than R^2 gained by using gradient descent. At the first glance our R^2 is pretty high as we try to predict human behavior. It is important to note, however, that R^2 on its own can't say anything how good or bad is our model. This coefficient is not able to determine whether the coefficients of interest are biased. In addition, this coefficient doesn't indicate if our model is adequate.

In general, the evaluation of R^2 depends on our goals. If we are primary interested on relationship between two variables (e.g. the impact of rain on hourly entries), the size of R^2 is not that important as long as the coefficients of predictors are statistically significant and not biased. On the other hand if our primary goal is to make good predictions, R^2 should be large enough. The appropriate size of R^2 depends than on what failure are we willing to accept. In summary, the goodness of the test should be always evaluated by looking at more measures (R^2, residuals plot, t statistics, F statistics, …).[15]



```
                            OLS Regression Results
==============================================================================
Dep. Variable:         ENTRIESn_hourly   R-squared:                       0.542
Model:                             OLS   Adj. R-squared:                  0.539
Method:                  Least Squares   F-statistic:                     105.0
Date:                 Sun, 11 Jan 2015   Prob (F-statistic):               0.00
Time:                         23:44:10   Log-Likelihood:            -3.8466e+05
No. Observations:                42649   AIC:                         7.698e+05
Df Residuals:                    42400   BIC:                         7.720e+05
Df Model:                          248
Covariance Type:                   HAC
==============================================================================
                 coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
rain        -108.0292     44.619     -2.421      0.015      -195.480   -20.578
weekday      994.1981     23.092     43.053      0.000       948.938  1039.458
weekdayrain   94.5209     55.019      1.718      0.086       -13.314   202.356
meantempi    -13.4611      1.650     -8.158      0.000       -16.695   -10.227
ones        8450.6834    439.078     19.246      0.000      7590.107  9311.260
```

[13] For further discussion about residuals distribution please go to 3.2.

[14] Reference: http://blog.minitab.com/blog/adventures-in-statistics/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit

[15] For additional discussion about the goodness of our model see sections 2.4 and 2.5.

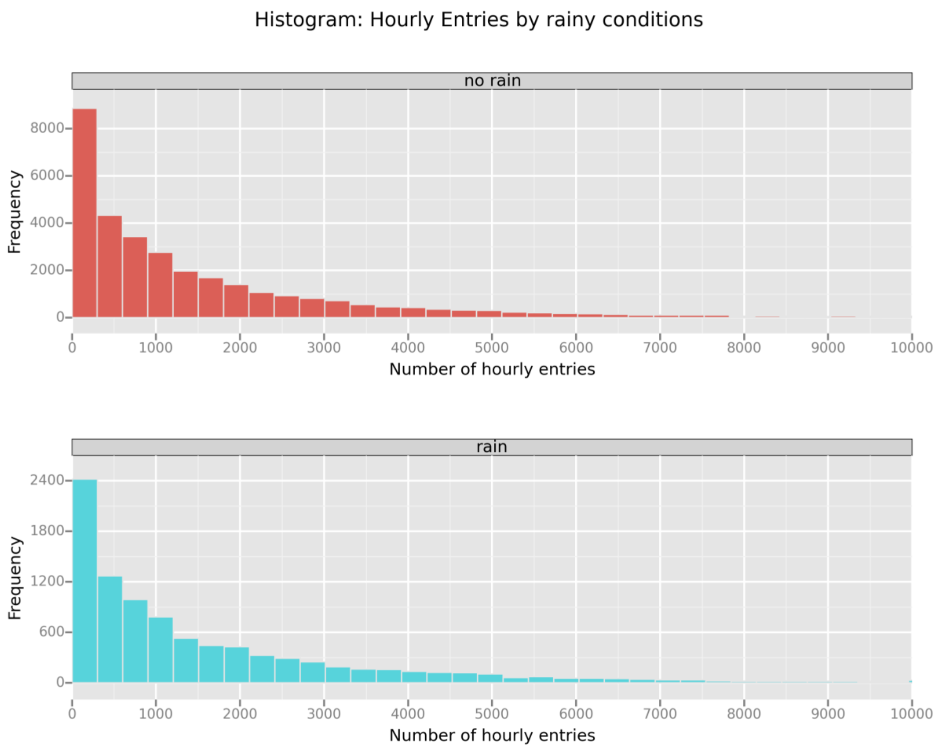
2.3. Visualization

One of the very important parts of data analysis is the data visualization. Good figures might provide us with very nice insights about the data. In this section we show two different figures that illustrate the relationship between subway usage and different variables.

2.3.1. Hourly Entries for rainy and non-rainy days (histograms)

This subsection presents the distribution of hourly entries for rainy and non-rainy days. Actually, this is the same figure as required for the very first part of problem set 3. We present this figure in section 3.2. i), where we make use of matplotlib. Therefore, in this section we create both histograms by means of ggplot.

```
import pandas as pd
from ggplot import *
import numpy as np

turnstile_data['Rain']=np.where(turnstile_data['rain']==1, 'rain', 'no rain')

gg1 = ggplot(turnstile_data,aes(x='ENTRIESn_hourly', fill='Rain'))+\
geom_histogram(binwidth=300) + xlab('Number of hourly entries') + \
ylab('Frequency') + ggtitle("Histogram: Hourly Entries by rainy conditions") + \
scale_x_continuous(breaks=(0,1000,2000,3000,4000,5000,6000,7000,8000,9000,10000),\
labels=(0,1000,2000,3000,4000,5000,6000,7000,8000,9000,10000),limits=(0,10000)) + \
facet_wrap('Rain')

print gg1
```

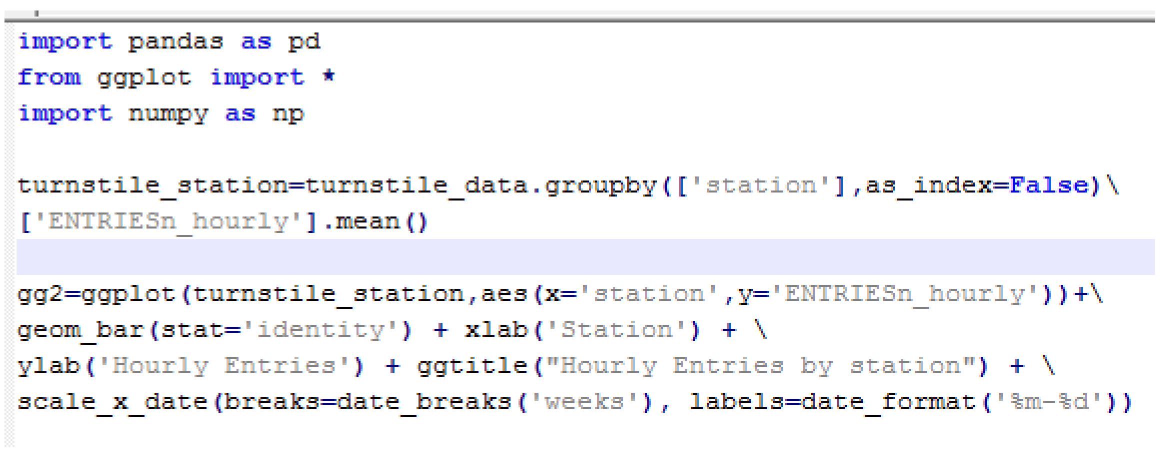


Note that for illustrational purposes we limited the x axis (0 to 10000). By this limitation we don't capture by our histograms all observations (around 2% of observations are not there).[16] From these histograms it is clear that the number of observations without rain is significantly larger than the number of observations with with rain. Moreover, both distributions are right skewed and very similar (i.e. with huge amount of data with smaller number of hourly entries). It seems that the pattern of people using subway in NYC doesn't change according to the rain.

2.3.2. Hourly Entries by Station

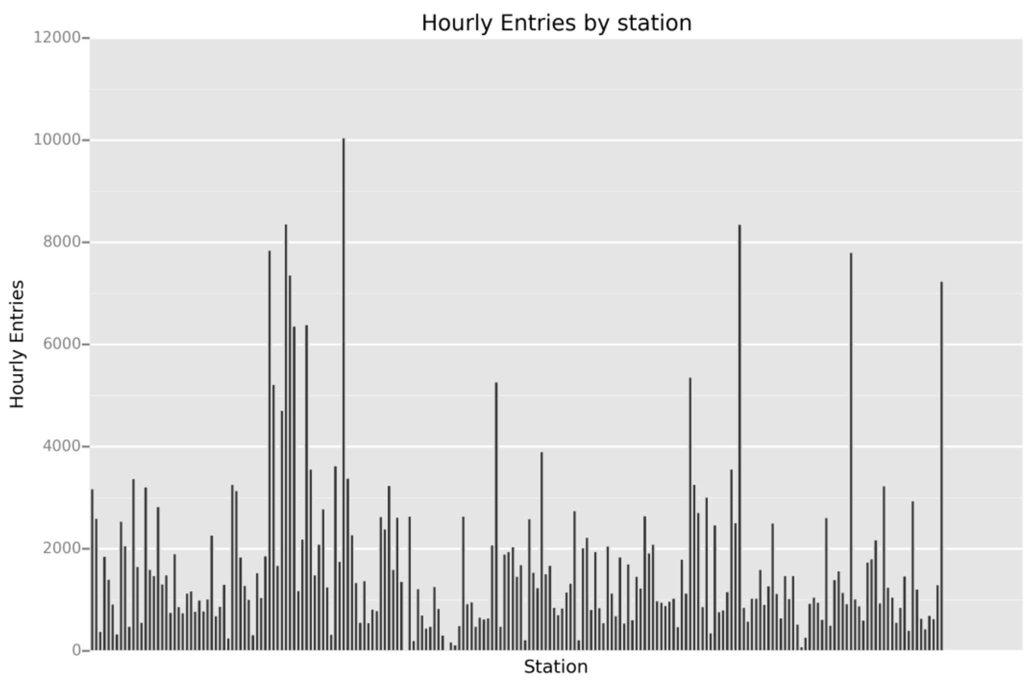
As mentioned above, data visualization might give us a hint which variables might be important for subway demand forecasting. In section 3.3 we present two figures which illustrate that hourly entries are strongly affected by both hour and weekday. In addition, we see there that rain also might have an influence at the subway usage. Another candidate influencing the subway usage is the unit or the station as the stations vary in their size or in the number of lines going through these stations. In order to see how the stations affect hourly entries, we plot a bar plot with mean hourly entries by station.[17]

[16] One possibility, how to capture all observations, would be log transformation of hourly entries variable.
[17] Note that the last line in code is there only for the purpose of hiding x tick labeling.



```
import pandas as pd
from ggplot import *
import numpy as np

turnstile_station=turnstile_data.groupby(['station'],as_index=False)\
['ENTRIESn_hourly'].mean()

gg2=ggplot(turnstile_station,aes(x='station',y='ENTRIESn_hourly'))+\
geom_bar(stat='identity') + xlab('Station') + \
ylab('Hourly Entries') + ggtitle("Hourly Entries by station") + \
scale_x_date(breaks=date_breaks('weeks'), labels=date_format('%m-%d'))
```

From the chart below we can tell that there are huge differences among stations. Some stations are very busy and some not. We get very similar picture if we examine the turnstile units. Therefore, it is important to capture this variation in our linear regression model. We do that by including dummy variables for each unit. Section 4.3 provides some more useful data visualizations (e.g. distributions of hourly entries by weather conditions).



2.4. Conclusion

One of the main goals of this project was to determine whether more people use the subway when it is raining or when it is not raining. To this end we make use of different statistical tools. In sections 2.1 and 4.1 we conduct two different statistical tests. Mann-Whitney U test indicates that the distributions of hourly entries on days with rain and without rain are very likely to be different. This test in combination with descriptive statistics indicates that the subway is busier on days with rain. We can confirm this finding by conducting two-sided and one-sided Welch's t test in section 4.1.[18] Thus, based on our statistical tests we conclude that it is very likely that more people use the subway when it is raining than when it is not raining.

We examine this relationship in section 2.2 in more detail by using linear regression. The results from our linear regression suggest that we have to differentiate between weekdays and weekends. While on weekdays we can observe the same relationship (statistically significant at 10 % significance level), at weekends we observe contrary relationship meaning that the subway is busier when it is not raining (statistically significant at 1% significance level).[19]

2.5. Reflection

There might be different shortcomings of analysis conducted in this project. First, it is likely that the weather variables contain a measurement error. We observe some inconsistencies in our data. One example of those inconsistencies is that there are observations with rain=1 but no precipitation rate (meanprecipi=0) and vice versa. If there is a measurement error in our explanatory variables, our estimated coefficient will be biased and inconsistent and therefore our conclusion about the subway usage on rainy days might be not correct (Chen et al. (2007)). Moreover, it is also questionable whether it is not better to use weather forecasts. The reasoning is pretty simple. If the transportation companies should use our model for better planning, it will be for them more useful, if our model would be based on forecasts rather than on actual weather conditions. Besides the discussed problems with the dataset, there might be diverse problems with statistical tests used in sections 2.1 and 4.1. We have already discussed those in section 2.1.

The residual plot indicates that our model underestimate the actual data and therefore it is likely that our model is biased. One of the possible sources of this bias is an omitted variable.

[18] We discuss the advantages and disadvantages of both tests in section 2.1.
[19] We conduct some robustness checks in section 4.2. The results of estimation in section 4.2 are in line with those findings.

The demand for subway also depends on demand for other transport vehicles (e.g. car, bus, bicycle …). Since the usage of other transport vehicles is very likely to be correlated with weather conditions (e.g. rain), the omission of those variables will lead to biased coefficients and therefore to wrong conclusions. Moreover, the usage of many dummy variables might lead to multicollinearity issues and therefore to unstable coefficients (Wissmann et al. (2007)). Indeed, we observe pretty high conditional number which might be an indicator for multicollinearity issues.

Finally, the usage of linear model is also questionable. Indeed, there are more profound models in economic literature used for demand estimation. Situations where the customer have more options (here – different transport vehicles) are pretty intuitively modelled by means of nested logit models or random coefficient models (McFadden (1974), Berry et al. (1995)). Those models are however more data intensive.

3. Problem Sets 2 - 5

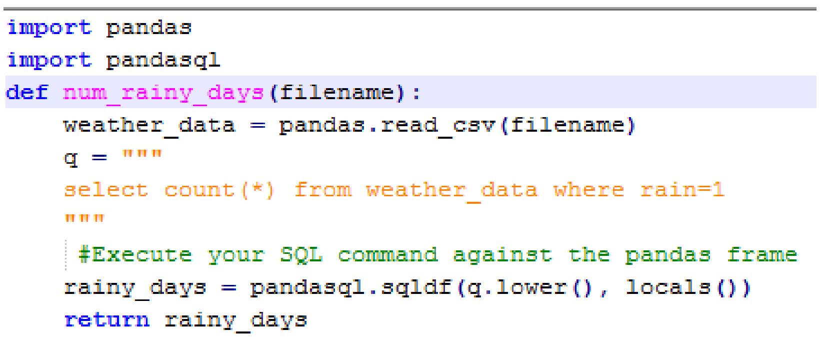
In the following we provide a code for the problem sets 2 – 5 of the Introduction to Data Science class. Moreover, for each problem set we also provide a short description.

3.1. Problem Set 2 (Wrangling Subway Data)

Lesson 2 of Intro to Data Science is dealing with data wrangling. In the following you can find very short description as well as code to each of eleven problems from the lesson 2. In general, all these problems show, how we can get, clean and manipulate our data. Problems i) to iv) are dealing with different types of SQL queries, which might help us to quickly check some characteristics of our data (e.g. number of rainy days). Problems v) and vi) illustrate how to get data which is not in very nice format and how to combine different data files. Problems vii) - xi) represent different data manipulations which are important for further analysis. At the end of this section I also include code how to join the subway and weather data.

i) *Number of Rainy Days*

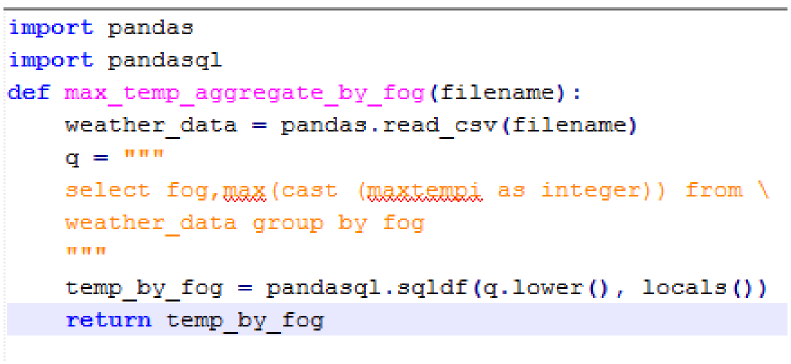
This function runs a SQL query and should return the number of days it rained.



```
import pandas
import pandasql
def num_rainy_days(filename):
    weather_data = pandas.read_csv(filename)
    q = """
    select count(*) from weather_data where rain=1
    """
    #Execute your SQL command against the pandas frame
    rainy_days = pandasql.sqldf(q.lower(), locals())
    return rainy_days
```

ii) *Temperature on Foggy and Non-foggy Days*

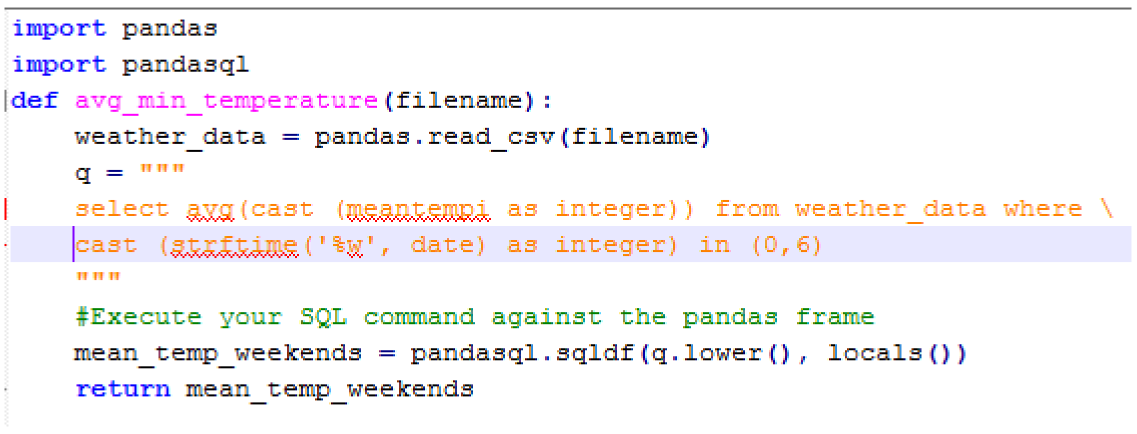
This function runs a SQL query and should return the maximum max temperature for both foggy and non-foggy days.



```
import pandas
import pandasql
def max_temp_aggregate_by_fog(filename):
    weather_data = pandas.read_csv(filename)
    q = """
    select fog,max(cast (maxtempi as integer)) from \
    weather_data group by fog
    """
    temp_by_fog = pandasql.sqldf(q.lower(), locals())
    return temp_by_fog
```

iii) *Mean Temperature on Weekends*

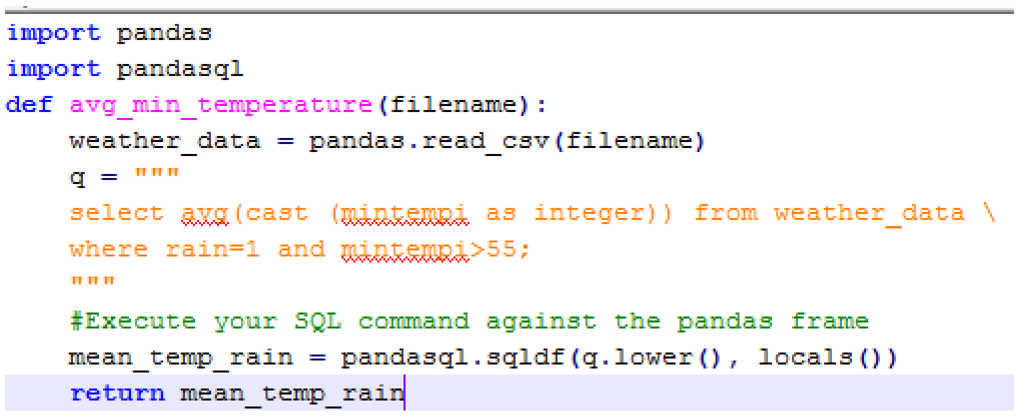
This function runs a SQL query and should return the average mean temperature on weekends.



```
import pandas
import pandasql
def avg_min_temperature(filename):
    weather_data = pandas.read_csv(filename)
    q = """
    select avg(cast (meantempi as integer)) from weather_data where \
    cast (strftime('%w', date) as integer) in (0,6)
    """
    #Execute your SQL command against the pandas frame
    mean_temp_weekends = pandasql.sqldf(q.lower(), locals())
    return mean_temp_weekends
```

iv) *Mean Temperature on Rainy Days*

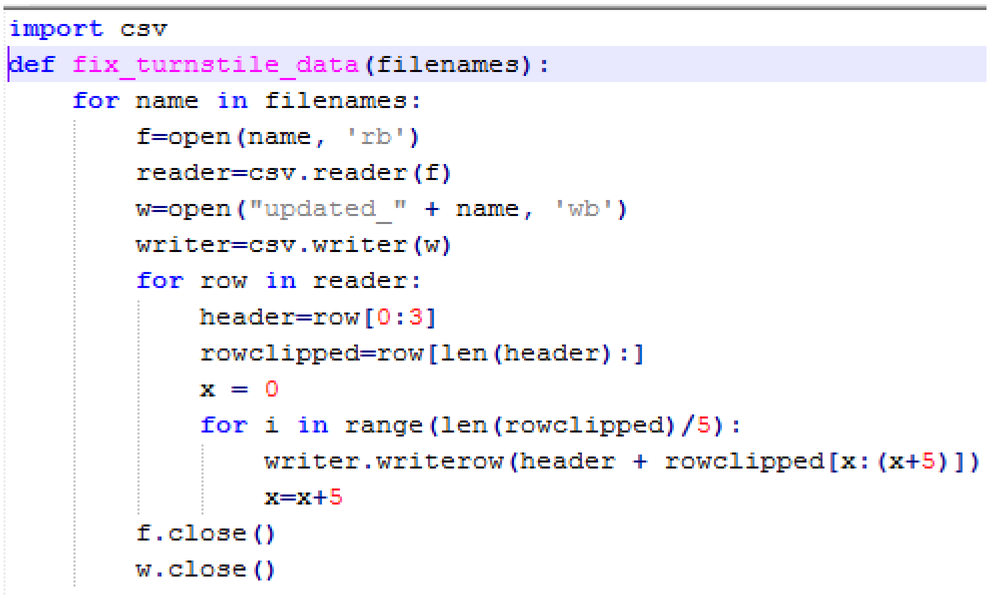
This function runs a SQL query and should return the average minimum temperature on rainy days where the minimum temperature is greater than 55 degrees.



```
import pandas
import pandasql
def avg_min_temperature(filename):
    weather_data = pandas.read_csv(filename)
    q = """
    select avg(cast (mintempi as integer)) from weather_data \
    where rain=1 and mintempi>55;
    """
    #Execute your SQL command against the pandas frame
    mean_temp_rain = pandasql.sqldf(q.lower(), locals())
    return mean_temp_rain
```

v) *Fixing Turnstile Data*

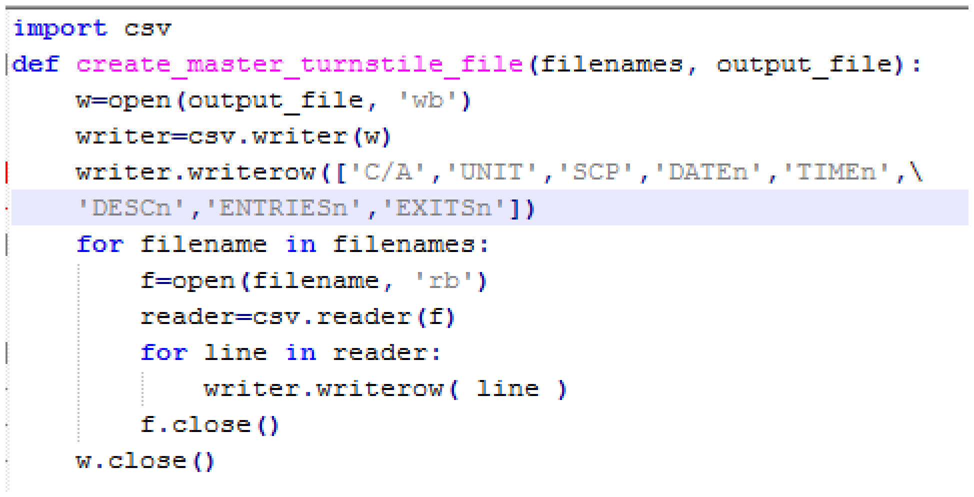
This function should change the format of the data such that we can easily analyze the data. Moreover, our changes should be saved in file (updated_name of input file.txt).



```
import csv
def fix_turnstile_data(filenames):
    for name in filenames:
        f=open(name, 'rb')
        reader=csv.reader(f)
        w=open("updated_" + name, 'wb')
        writer=csv.writer(w)
        for row in reader:
            header=row[0:3]
            rowclipped=row[len(header):]
            x = 0
            for i in range(len(rowclipped)/5):
                writer.writerow(header + rowclipped[x:(x+5)])
                x=x+5
        f.close()
        w.close()
```

vi) *Combining Turnstile Data*

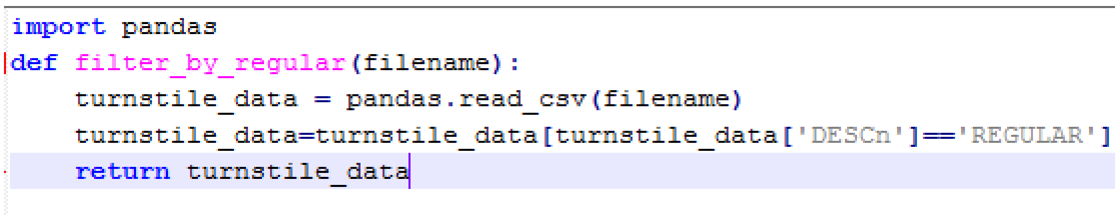
This function combines different files such that the later ones are appended to the first one. The header is used only from the first one.



```
import csv
def create_master_turnstile_file(filenames, output_file):
    w=open(output_file, 'wb')
    writer=csv.writer(w)
    writer.writerow(['C/A','UNIT','SCP','DATEn','TIMEn',\
    'DESCn','ENTRIESn','EXITSn'])
    for filename in filenames:
        f=open(filename, 'rb')
        reader=csv.reader(f)
        for line in reader:
            writer.writerow( line )
        f.close()
    w.close()
```

vii) *Filtering Irregular Data*

This function should filter only for observations where the ‘DECSn’ column has the value ‘REGULAR’

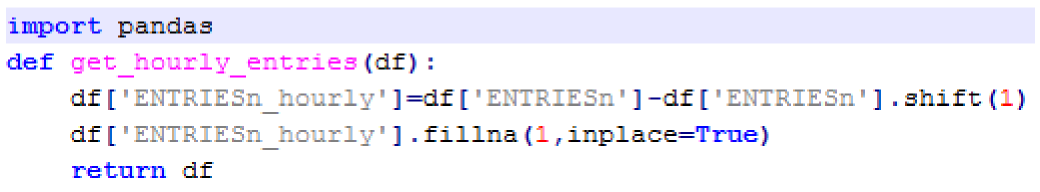


```
import pandas
def filter_by_regular(filename):
    turnstile_data = pandas.read_csv(filename)
    turnstile_data=turnstile_data[turnstile_data['DESCn']=='REGULAR']
    return turnstile_data
```

viii) *Get Hourly Entries*

This function should change the cumulative entries to the hourly entries per C/A, Unit and SCP where C/A is Control Area, Unit is a remote unit for the station and

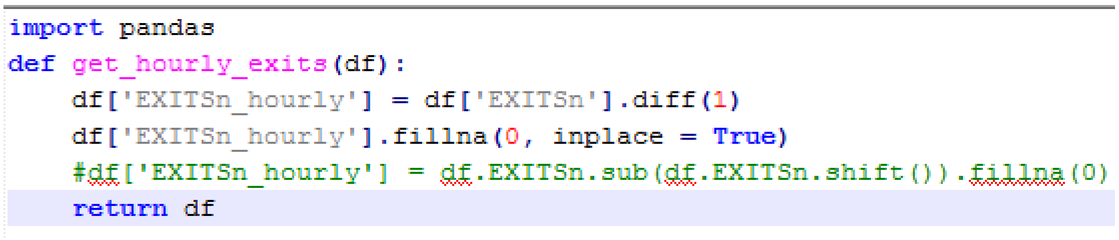
SCP means subunit channel position (which represents a specific address for that device).



```
import pandas
def get_hourly_entries(df):
    df['ENTRIESn_hourly']=df['ENTRIESn']-df['ENTRIESn'].shift(1)
    df['ENTRIESn_hourly'].fillna(1,inplace=True)
    return df
```

ix) *Get Hourly Exits*

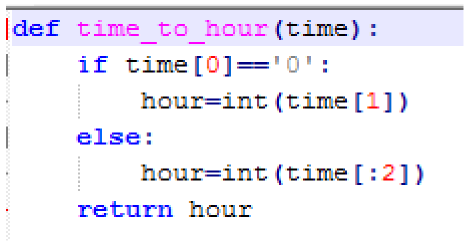
This function should change the cumulative exits to the hourly exits per C/A, Unit and SCP where C/A is Control Area, Unit is a remote unit for the station and SCP means subunit channel position (which represents a specific address for that device).



```
import pandas
def get_hourly_exits(df):
    df['EXITSn_hourly'] = df['EXITSn'].diff(1)
    df['EXITSn_hourly'].fillna(0, inplace = True)
    #df['EXITSn_hourly'] = df.EXITSn.sub(df.EXITSn.shift()).fillna(0)
    return df
```

x) *Change Time to Hour*

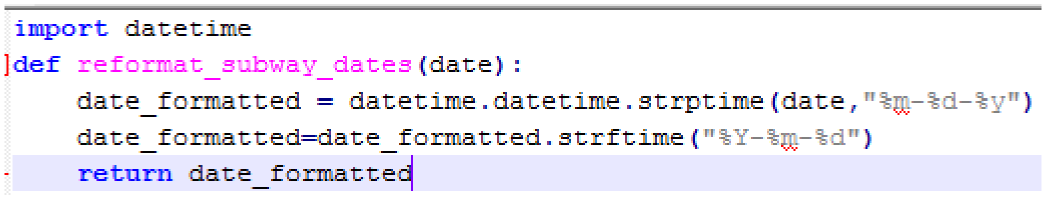
This function should map time in given format into a particular hour. This helps us to create a new column with hour in my dataset.



```
def time_to_hour(time):
    if time[0]=='0':
        hour=int(time[1])
    else:
        hour=int(time[:2])
    return hour
```

xi) *Reformat Subway Date*

This function should change date from one format to another format. This helps us to join the subway data with weather data as the dates in both datasets are in different formats.



```
import datetime
def reformat_subway_dates(date):
    date_formatted = datetime.datetime.strptime(date,"%m-%d-%y")
    date_formatted=date_formatted.strftime("%Y-%m-%d")
    return date_formatted
```

All the previous steps were necessary to prepare subway data for further analysis. We store this data into dataframe named subway_final. Moreover, we process weather data from Weather Underground and store it under weather_final. In very last step we can join these two datasets. The code for this joining you can find below. It is important to note that we use only such observations that occur in both datasets. To better see what observations are only in one or other datasets we can use outer joining.

```
import pandas
final_data=subway_final.merge(weather_final, left_on='DATEn', right_on='date')
```

3.2. Problem Set 3 (Data Analysis)

After data cleaning we can start with data analysis. This section presents the solutions for problem set 3 of Intro to Data Science class.

i) *Exploratory data analysis – histogram plotting*

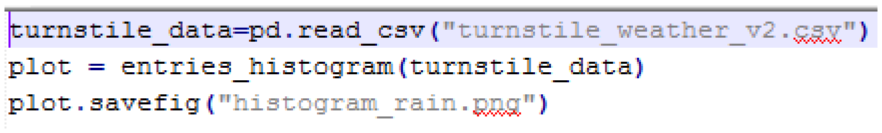
This function should plot two histograms of hourly entries by rain conditions on the same axes.[20]

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
def entries_histogram(turnstile_weather):
    plt.figure()
    turnstile_weather['ENTRIESn_hourly'][turnstile_weather['rain']==0].\
    hist(bins=100,stacked=True,color='g', alpha=0.7,label='No Rain')
    turnstile_weather['ENTRIESn_hourly'][turnstile_weather['rain']==1].\
    hist(bins=100,stacked=True,color='b', alpha=0.4,label='Rain')
    plt.title("Hourly Entries by weather conditions (rain/no rain)")
    plt.xlabel("Hourly Entries")
    plt.xlim([0,10000])
    plt.ylabel("Frequency")
    plt.legend()
    return plt
```

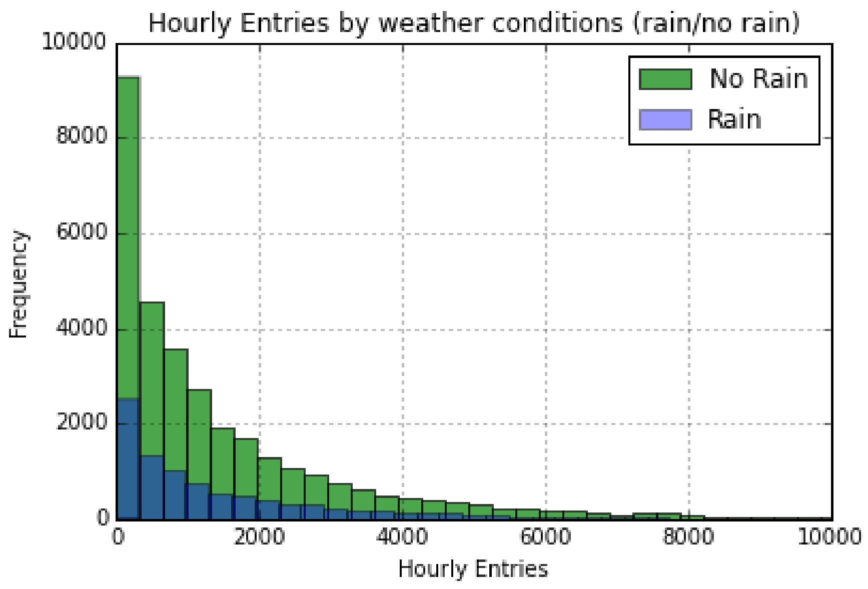
The figure can be seen by using following code. In the figure we can see that the number of observations without rain is significantly larger than the number of observations with rain. Moreover, both distributions are right skewed and very similar (i.e. with huge amount of data with smaller number of hourly entries). In

[20] Note that for illustrative purpose we use a limitation on x axes (from 0 to 10000). I.e. some data (less than 10% of all data) is not used here.

other words the pattern of people using subway in NYC doesn't change according to the rain.



```
turnstile_data=pd.read_csv("turnstile_weather_v2.csv")
plot = entries_histogram(turnstile_data)
plot.savefig("histogram_rain.png")
```



In the next step we can examine whether the demand for NYC subway is larger by rain than by no rain. In order to statistically test this, we are going to use Mann-Whitney U test (see problem ii)). For the discussion which test is more appropriate see section 2.1.[21]

ii) *Mann-Whitney U Test*

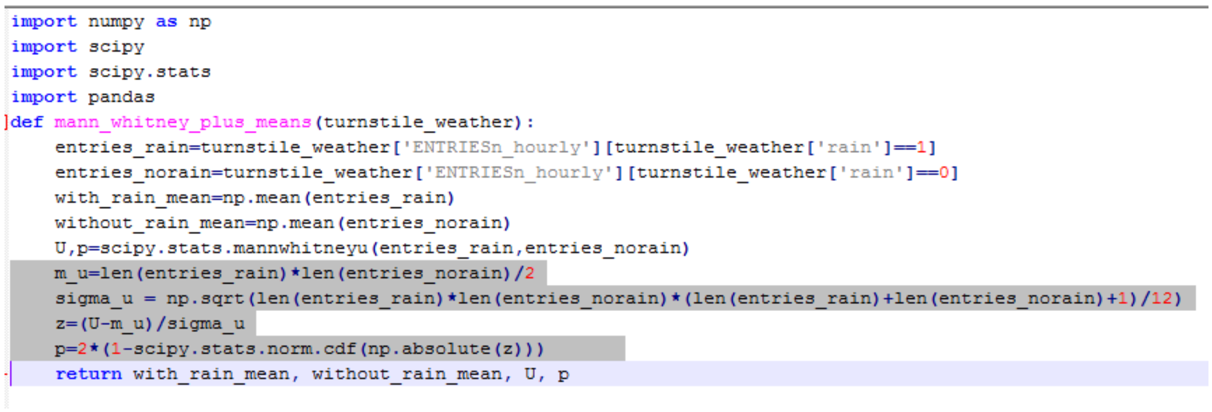
This function conducts the Mann Whitney U-test on hourly entries by different rainy conditions. It should return means of entries (with and without rain) as well as the U-statistic and P-value of the test. However, there is one problem with M-W test implementation in python. If we use it on improved dataset, no p value will be returned (p = nan).[22] Therefore, we use the fact that for large samples, U is approximately normal distributed. I. e. we can compute standardized value and from it the P-value. The code of the implementation is provided below. The manual computation of p value is highlighted. It is important to note that the M-W

[21] We also conduct the Welch's t test. The results of this test can be found in 4.1.

[22] By using it on original data we get following results: mean_rain=1105.4463767458733, mean_norain=1090.278780151855, U=1924409167.0, P-value = 0.019309634413792565, indicating that the distributions are different at 5 % significance level (2*p-Value) must be computed.

test implementation in python as well as our computation by means of standardized value returns only one-sided p value. Therefore we multiply this value by 2.

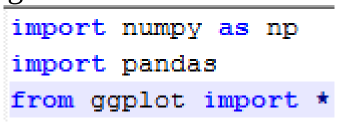
$$z = \frac{U - \mu_u}{\sigma_u}, \mu_u = \frac{n_1 n_2}{2}, \sigma_u = \frac{n_1 n_2 (n_1 + n_2 + 1)}{12}$$



```
import numpy as np
import scipy
import scipy.stats
import pandas
def mann_whitney_plus_means(turnstile_weather):
    entries_rain=turnstile_weather['ENTRIESn_hourly'][turnstile_weather['rain']==1]
    entries_norain=turnstile_weather['ENTRIESn_hourly'][turnstile_weather['rain']==0]
    with_rain_mean=np.mean(entries_rain)
    without_rain_mean=np.mean(entries_norain)
    U,p=scipy.stats.mannwhitneyu(entries_rain,entries_norain)
    m_u=len(entries_rain)*len(entries_norain)/2
    sigma_u = np.sqrt(len(entries_rain)*len(entries_norain)*(len(entries_rain)+len(entries_norain)+1)/12)
    z=(U-m_u)/sigma_u
    p=2*(1-scipy.stats.norm.cdf(np.absolute(z)))
    return with_rain_mean, without_rain_mean, U, p
```

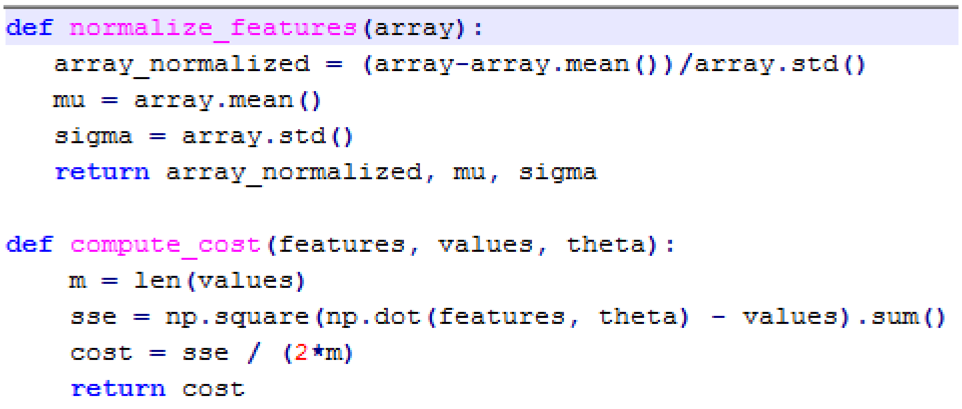
iii) *Linear Regression (Gradient Descent)*

In this section we implement linear regression by means of gradient descent. In order to conduct linear regression, we have to specify the independent variables (i.e. features) x first. We use following variables as independent variables: rain, weekday, interaction term consisting of rain and weekday, meantempi, dummy variables for hour as well as dummy variables for turnstile units. We explain the selection of these variables in section 2.2. To conduct gradient descent it is important to import following three libraries:



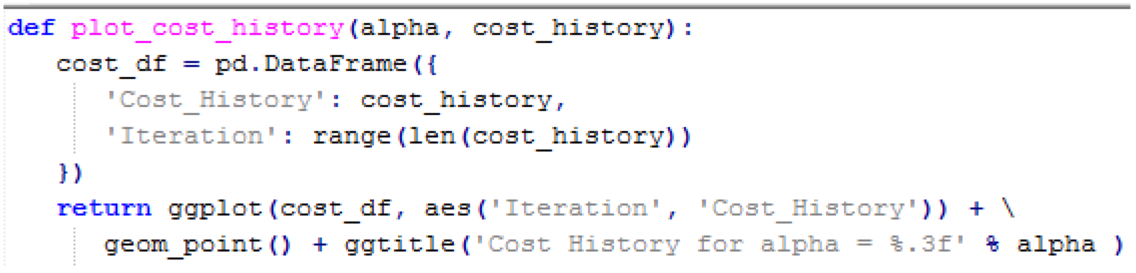
```
import numpy as np
import pandas
from ggplot import *
```

Moreover, it is necessary to implement some helper functions. Normalize_feature basically increase the efficiency of our gradient descent. Compute_cost defines the cost function, which should be minimized.



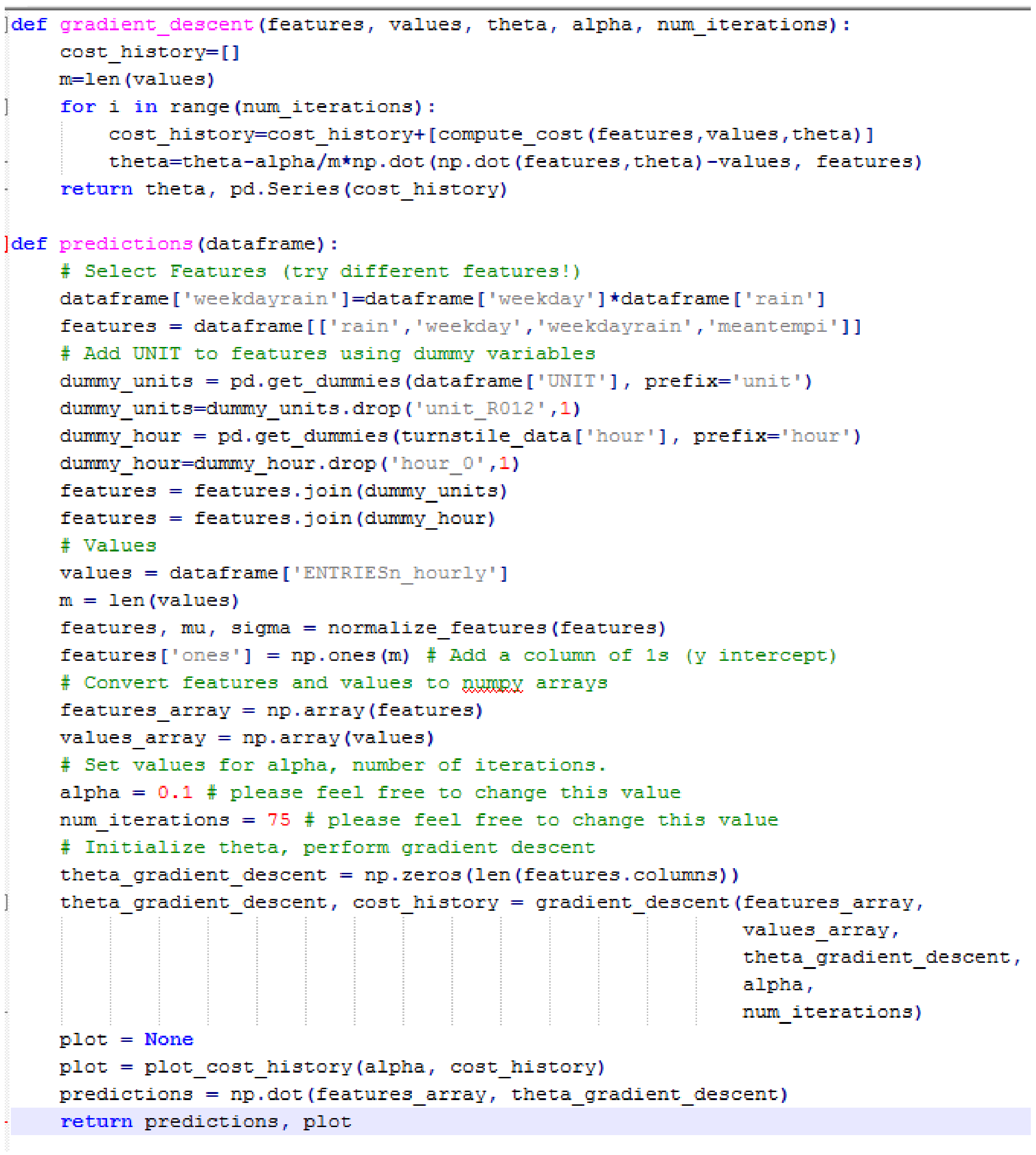
```
def normalize_features(array):
   array_normalized = (array-array.mean())/array.std()
   mu = array.mean()
   sigma = array.std()
   return array_normalized, mu, sigma

def compute_cost(features, values, theta):
    m = len(values)
    sse = np.square(np.dot(features, theta) - values).sum()
    cost = sse / (2*m)
    return cost
```

In addition, we examine whether the cost function decreases with every interaction. To this end we create a function plot_cost_history.



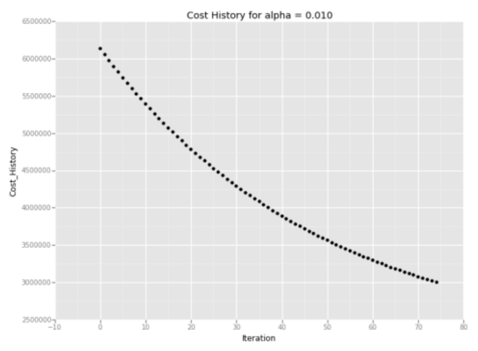
```
def plot_cost_history(alpha, cost_history):
    cost_df = pd.DataFrame({
        'Cost_History': cost_history,
        'Iteration': range(len(cost_history))
    })
    return ggplot(cost_df, aes('Iteration', 'Cost_History')) + \
        geom_point() + ggtitle('Cost History for alpha = %.3f' % alpha )
```

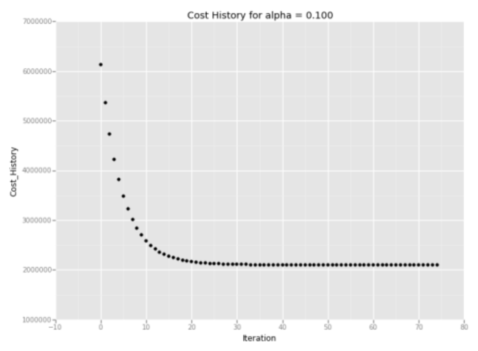
The code for the gradient descent and the predictions looks like as follows.

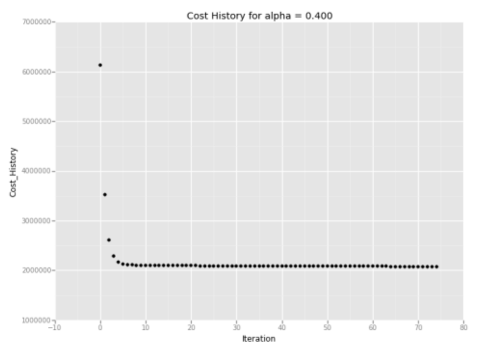


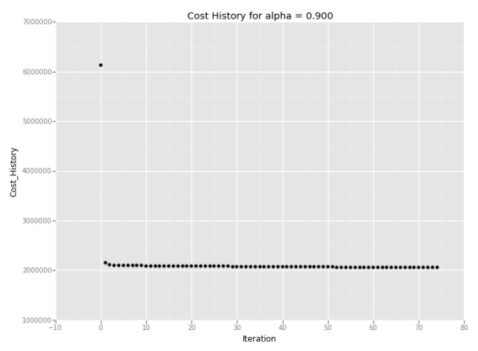
```
def gradient_descent(features, values, theta, alpha, num_iterations):
    cost_history=[]
    m=len(values)
    for i in range(num_iterations):
        cost_history=cost_history+[compute_cost(features,values,theta)]
        theta=theta-alpha/m*np.dot(np.dot(features,theta)-values, features)
    return theta, pd.Series(cost_history)

def predictions(dataframe):
    # Select Features (try different features!)
    dataframe['weekdayrain']=dataframe['weekday']*dataframe['rain']
    features = dataframe[['rain','weekday','weekdayrain','meantempi']]
    # Add UNIT to features using dummy variables
    dummy_units = pd.get_dummies(dataframe['UNIT'], prefix='unit')
    dummy_units=dummy_units.drop('unit_R012',1)
    dummy_hour = pd.get_dummies(turnstile_data['hour'], prefix='hour')
    dummy_hour=dummy_hour.drop('hour_0',1)
    features = features.join(dummy_units)
    features = features.join(dummy_hour)
    # Values
    values = dataframe['ENTRIESn_hourly']
    m = len(values)
    features, mu, sigma = normalize_features(features)
    features['ones'] = np.ones(m) # Add a column of 1s (y intercept)
    # Convert features and values to numpy arrays
    features_array = np.array(features)
    values_array = np.array(values)
    # Set values for alpha, number of iterations.
    alpha = 0.1 # please feel free to change this value
    num_iterations = 75 # please feel free to change this value
    # Initialize theta, perform gradient descent
    theta_gradient_descent = np.zeros(len(features.columns))
    theta_gradient_descent, cost_history = gradient_descent(features_array,
                                                            values_array,
                                                            theta_gradient_descent,
                                                            alpha,
                                                            num_iterations)
    plot = None
    plot = plot_cost_history(alpha, cost_history)
    predictions = np.dot(features_array, theta_gradient_descent)
    return predictions, plot
```

As you can see in the code, we can define learning rate α. Setting this too small will cause us to converge very slowly. In contrary, setting it too large increases the speed of finding true values but might cause us to diverge. The following figures show the relationship between cost function and different α‘s when the number of iteration is 75. We can see there that by α=0.01 we are not able to converge.



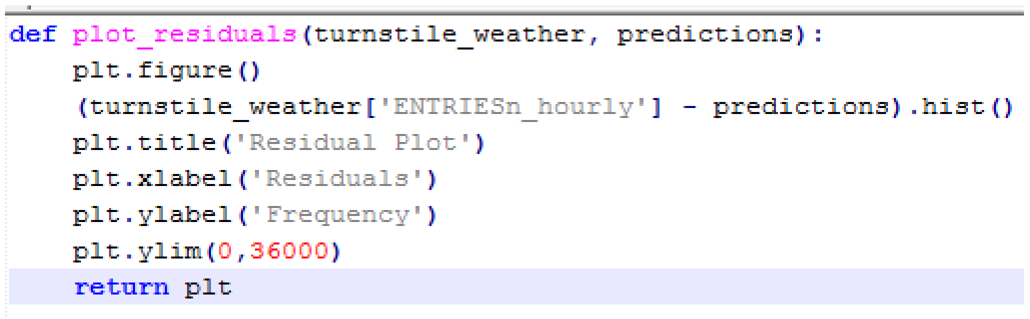






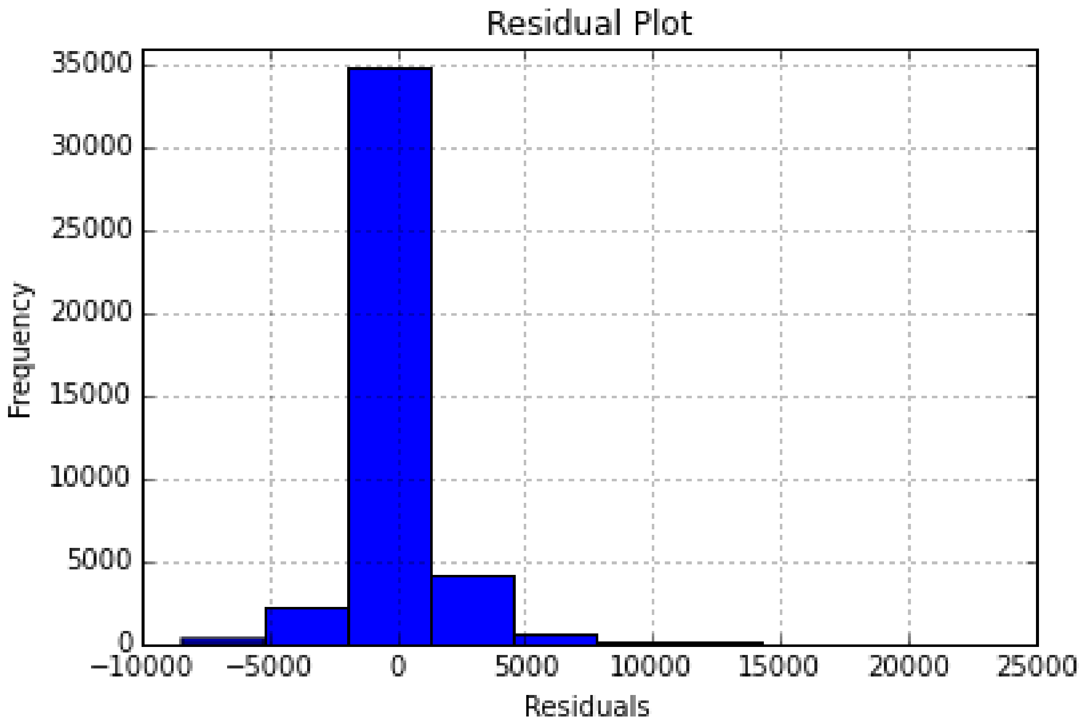
iv) *Plotting Residuals*

This function provides a histogram of the residuals from previous regression. Plotting the distribution of residuals is one of the possibilities how to examine the goodness of our regression. A good model is characterized by normal distributed residuals with mean of zero and some finite variance.



```
def plot_residuals(turnstile_weather, predictions):
    plt.figure()
    (turnstile_weather['ENTRIESn_hourly'] - predictions).hist()
    plt.title('Residual Plot')
    plt.xlabel('Residuals')
    plt.ylabel('Frequency')
    plt.ylim(0,36000)
    return plt
```

At the first glance the figure below indicates that the residuals are relatively normal distributed. However, if we examine it in more detail, we can see that positive numbers appear more often. In other words our model tends to predict lower values more often (i.e. underestimation).[23] This means that there might be still a space for improvement in our model by including additional explanatory variable(s).[24]



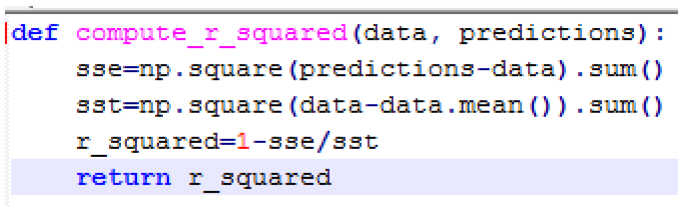
v) *Computing R^2*

Another possibility how to determine the goodness of our model is to compute the coefficient of determination, which measures how much variability in the data is explained by our model. To this end we compute R squared according to this formula.

$$R^2 = 1 - \frac{\sigma^2_{errors}}{\sigma^2_{data}} = 1 - \frac{\sum_{i=1}^{n}(y_i - f_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

By using this formula, we come to the conclusion that our model identifies 51.9 % of the variation in the data. This is smaller than the case where we use OLS solution provided by statsmodels.

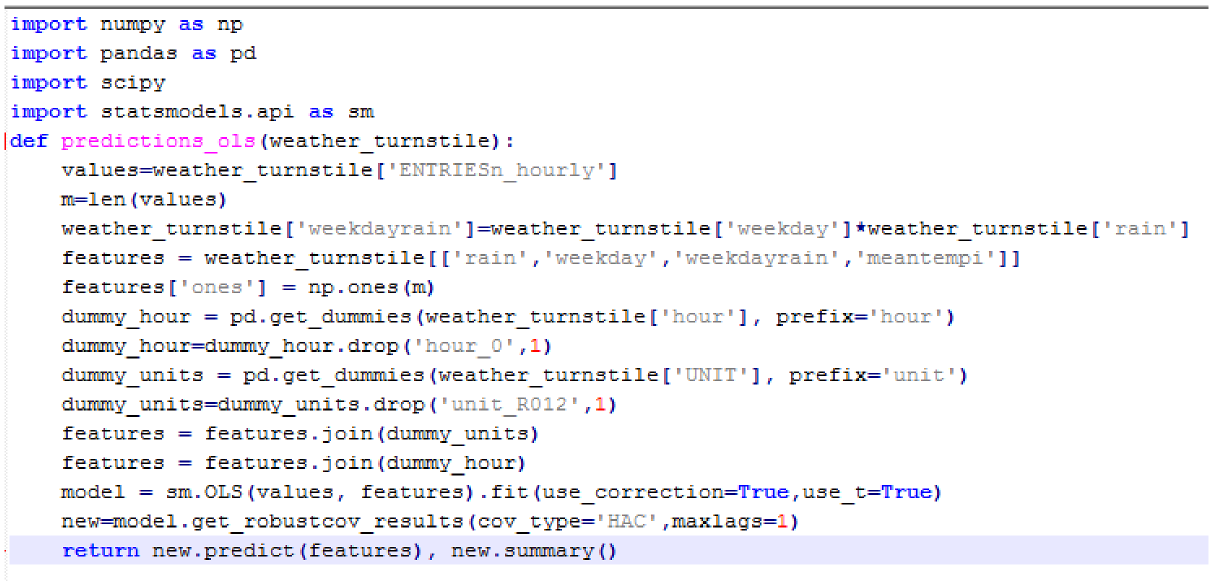
[23] By conducting the normality test we reject null hypothesis meaning that the residuals are very likely not normal distributed.
[24] Reference: http://blog.minitab.com/blog/adventures-in-statistics/why-you-need-to-check-your-residual-plots-for-regression-analysis



```python
def compute_r_squared(data, predictions):
    sse=np.square(predictions-data).sum()
    sst=np.square(data-data.mean()).sum()
    r_squared=1-sse/sst
    return r_squared
```

vi) *Linear Regression by using OLS*

While the gradient descent constitutes a numerical solution for our optimization problem, we can also use an analytical solution. There is a direct implementation of OLS in statsmodels library. The advantage of this model is that we get easy access to the coefficients so that we can immediately interpret them. The results are provided and interpreted in section 2.2.



```python
import numpy as np
import pandas as pd
import scipy
import statsmodels.api as sm
def predictions_ols(weather_turnstile):
    values=weather_turnstile['ENTRIESn_hourly']
    m=len(values)
    weather_turnstile['weekdayrain']=weather_turnstile['weekday']*weather_turnstile['rain']
    features = weather_turnstile[['rain','weekday','weekdayrain','meantempi']]
    features['ones'] = np.ones(m)
    dummy_hour = pd.get_dummies(weather_turnstile['hour'], prefix='hour')
    dummy_hour=dummy_hour.drop('hour_0',1)
    dummy_units = pd.get_dummies(weather_turnstile['UNIT'], prefix='unit')
    dummy_units=dummy_units.drop('unit_R012',1)
    features = features.join(dummy_units)
    features = features.join(dummy_hour)
    model = sm.OLS(values, features).fit(use_correction=True,use_t=True)
    new=model.get_robustcov_results(cov_type='HAC',maxlags=1)
    return new.predict(features), new.summary()
```

3.3. Problem Set 4 (Data Visualization)

As mentioned in section 2.3, data visualization is very important as it might provide us with useful insights about our data. In this section we show two figures created for solving the problem set 4 of Intro to Data Science class. The code, how we created those figures, is also provided. Section 4.3 provides some additional figures.

i) *Hourly Entries by hour and rain conditions.*

One of the possible determinants of metro usage can be the hour. There are peak hours during the workdays. These are basically the hours when people travel to and from their work. Can we confirm such a pattern by our data? To this end we create a figure showing the metro usage variation by hour and rain conditions. The code for this can be found below:

```python
import pandas as pd
from ggplot import *
import numpy as np

def plot_weather_data(turnstile_weather):
    by_hour_mean=turnstile_weather.groupby(['hour','rain'],\
    as_index=False)['ENTRIESn_hourly'].mean()
    by_hour_mean['Rain']=np.where(by_hour_mean['rain']==1, 'rain', 'no rain')

    plot = ggplot(by_hour_mean,aes(x='hour',y='ENTRIESn_hourly', fill='Rain')) + \
    geom_bar(stat='identity') + \
    xlab('Hour') + ylab('Number of Entries') + \
    ggtitle("NYC Tube Entries by hour and rain conditions") + \
    scale_x_discrete(breaks=(0,4,8,12,16,20),\
    labels=(0,4,8,12,16,20),limits=(0,23)) + \
    scale_y_continuous(limits=(0,4000)) + facet_wrap('Rain')
    return plot
```

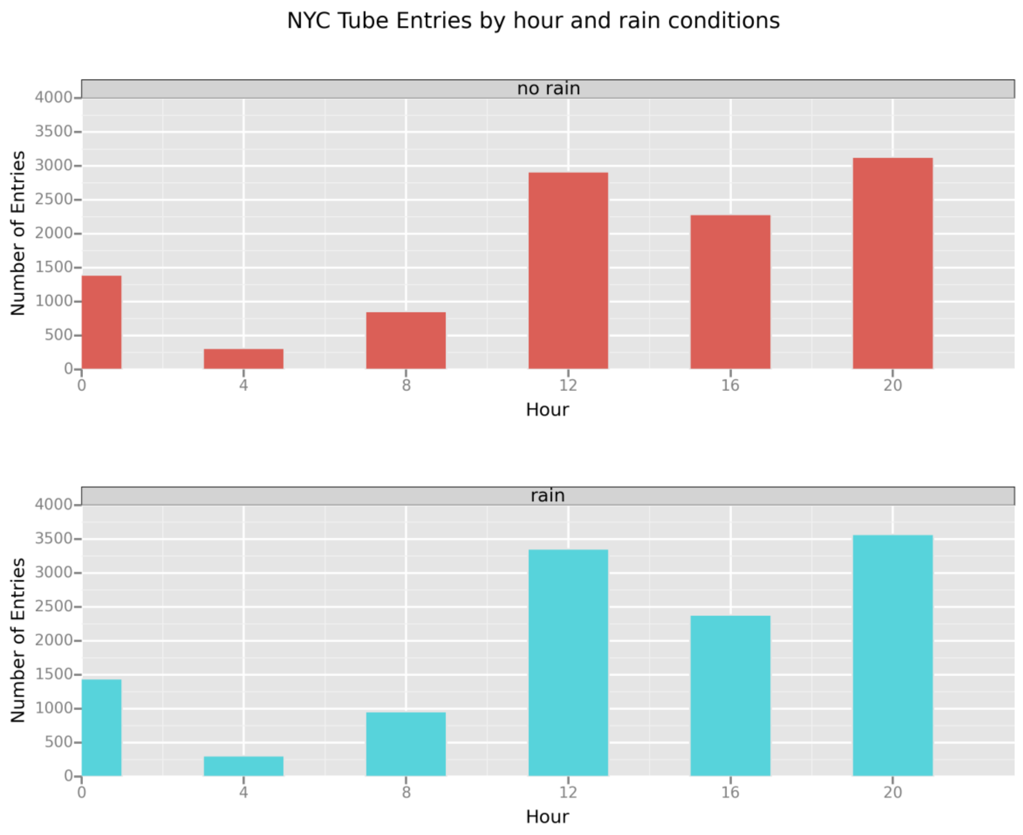
By looking at the figure we can observe following things:

First, the demand for subway strongly varies by examined hour with the peaks at 12 and 20. This goes in line with our “peak-hours” story. In the morning between 7 and 10 many people have to go to the work and use very likely the subway. The same thing happens in the afternoon between 4 and 8 when people go from their work. Note that the numbers for 12 hour might be a little bit biased (overestimated) as many observations for the hour 8 are not provided.

Second, there is also a difference between rainy and non-rainy days. The subway is more used by rainy days. The bars in the second figure are a little bit higher.

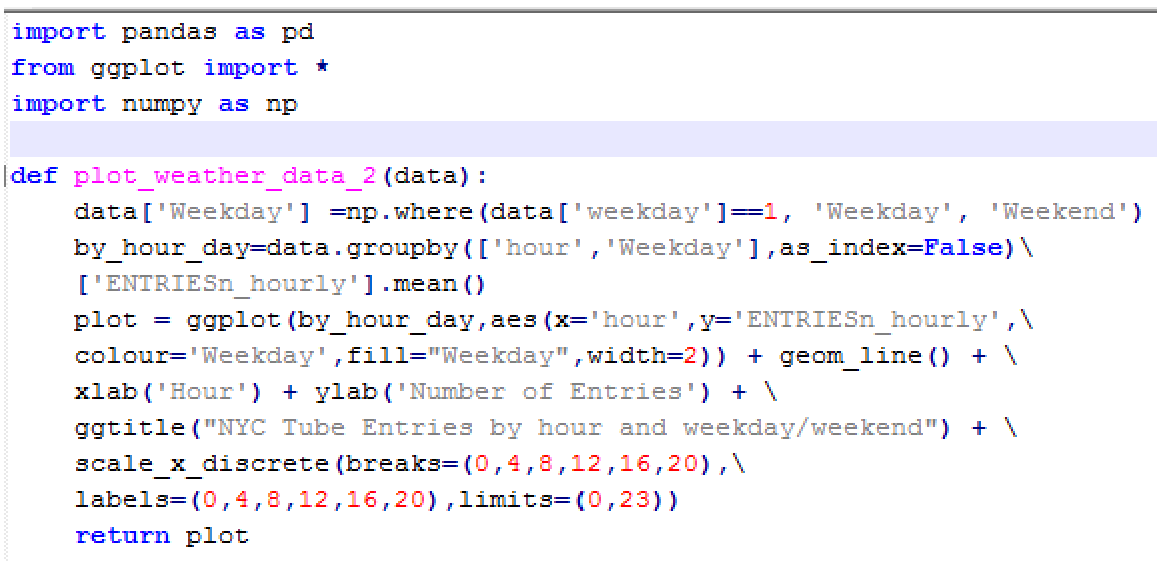
Third, the impact of hour is much stronger than the impact of rainy conditions.

Finally, the hourly pattern of metro usage doesn’t change according to the rain.



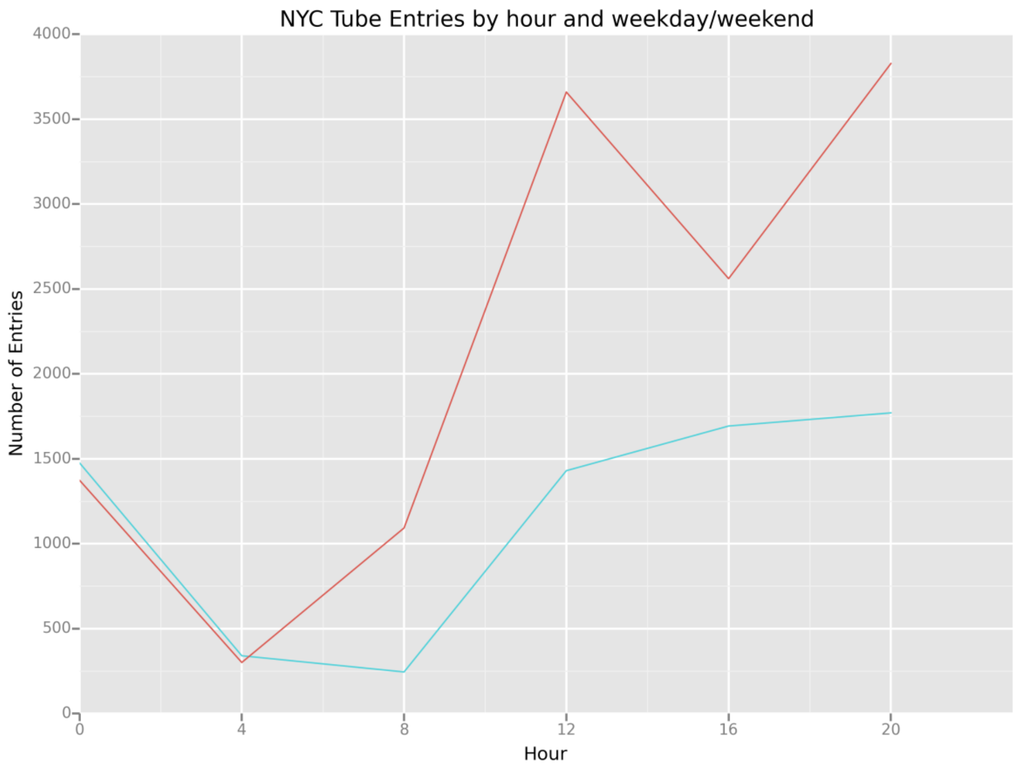
ii) *Hourly entries by hour and weekday/weekend*

This function should return a line plot which illustrates the average hourly entries by hour (i.e. 0, 4, 8, 12, 16, 20) and variable indicating whether it is weekend or not. Both the code and the figure can be found below.



```
import pandas as pd
from ggplot import *
import numpy as np

def plot_weather_data_2(data):
    data['Weekday'] =np.where(data['weekday']==1, 'Weekday', 'Weekend')
    by_hour_day=data.groupby(['hour','Weekday'],as_index=False)\
    ['ENTRIESn_hourly'].mean()
    plot = ggplot(by_hour_day,aes(x='hour',y='ENTRIESn_hourly',\
    colour='Weekday',fill="Weekday",width=2)) + geom_line() + \
    xlab('Hour') + ylab('Number of Entries') + \
    ggtitle("NYC Tube Entries by hour and weekday/weekend") + \
    scale_x_discrete(breaks=(0,4,8,12,16,20),\
    labels=(0,4,8,12,16,20),limits=(0,23))
    return plot
```

While the red line illustrates mean hourly entries during the workdays, average hourly entries at the weekend are represented by the blue line. The figure shows again that given hour is an important determinant for the subway usage. Moreover, we see that more people use the subway during the workdays than at the weekend.[25] Finally, the hourly pattern of entries differs between weekdays and weekends. At weekend there are no clear peak hours.



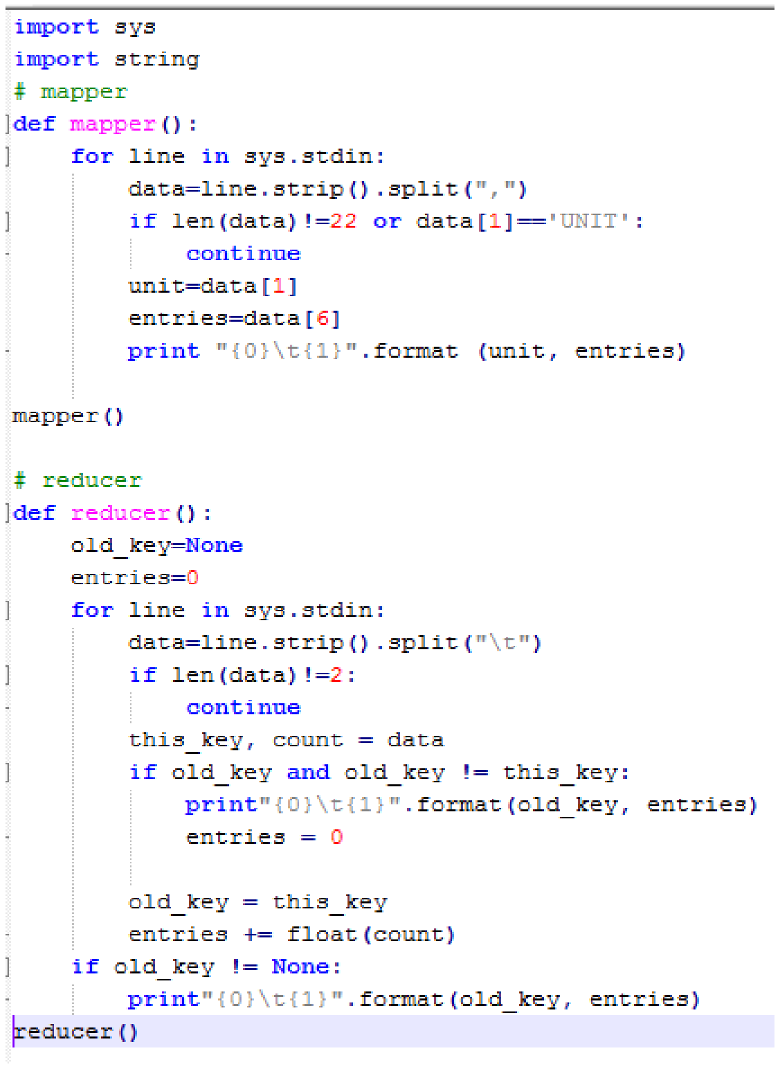
[25] In section 4.3 we show how the number of entries is affected by day of the week.

3.4. Problem Set 5 (Map Reduce)

Map and Reduce are very important techniques by dealing with big data. This section provides the code and short description to problem set of map and reduce lesson of Intro to Data Science class.

i) *Ridership per unit*

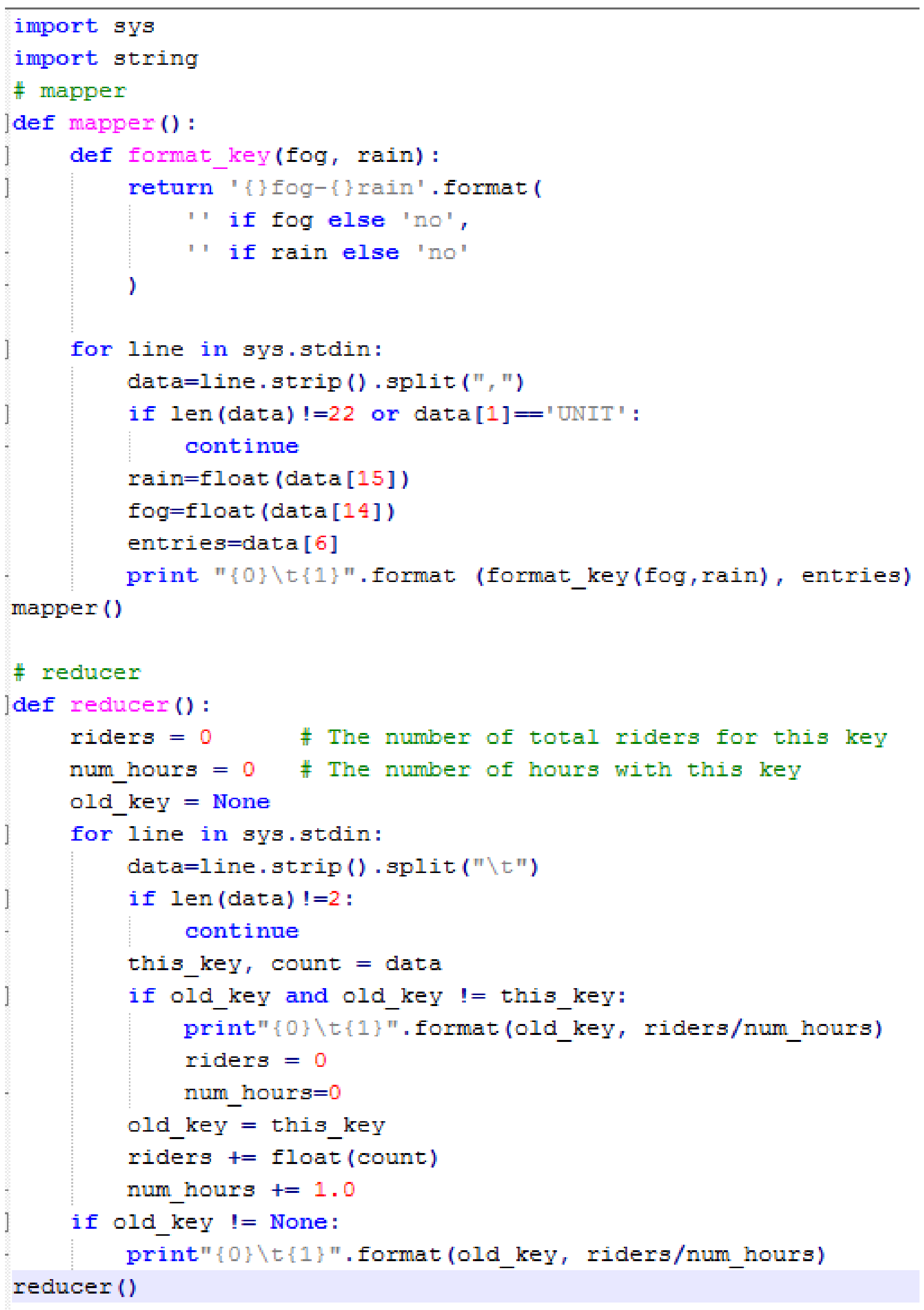
This function should compute for each UNIT the amount of entries. To this end we use map and reduce techniques. For each line of input, the mapper output should print the UNIT as the key, the number of hourly entries as the value, and separate the key and the value by a tab. The reducer should then print one line per UNIT along with the total number of hourly entries separated by a tab.



```
import sys
import string
# mapper
def mapper():
    for line in sys.stdin:
        data=line.strip().split(",")
        if len(data)!=22 or data[1]=='UNIT':
            continue
        unit=data[1]
        entries=data[6]
        print "{0}\t{1}".format (unit, entries)

mapper()

# reducer
def reducer():
    old_key=None
    entries=0
    for line in sys.stdin:
        data=line.strip().split("\t")
        if len(data)!=2:
            continue
        this_key, count = data
        if old_key and old_key != this_key:
            print"{0}\t{1}".format(old_key, entries)
            entries = 0

        old_key = this_key
        entries += float(count)
    if old_key != None:
        print"{0}\t{1}".format(old_key, entries)
reducer()
```

ii) *Ridership by weather type*

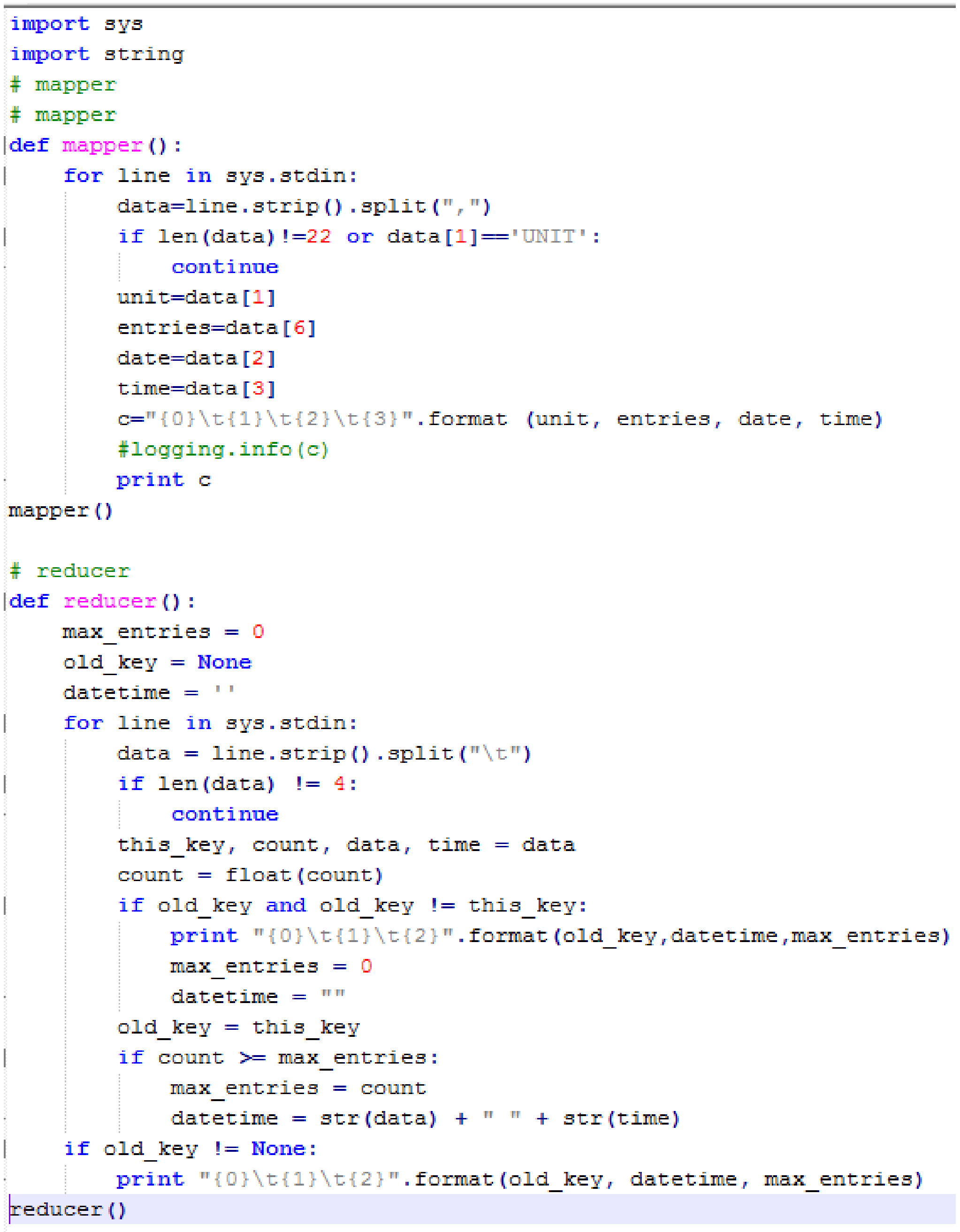
This function should compute the average value of hourly entries for different rainy/fog conditions. To this end we use map and reduce techniques. For each line of input, the mapper output should print the weather type (e.g. fog-norain) as the key, the number of hourly entries as the value, and separate the key and the value by a tab. The reducer should then print one line per weather type along with the average number of hourly entries, separated by a tab.



```
import sys
import string
# mapper
def mapper():
    def format_key(fog, rain):
        return '{}fog-{}rain'.format(
            '' if fog else 'no',
            '' if rain else 'no'
        )

    for line in sys.stdin:
        data=line.strip().split(",")
        if len(data)!=22 or data[1]=='UNIT':
            continue
        rain=float(data[15])
        fog=float(data[14])
        entries=data[6]
        print "{0}\t{1}".format (format_key(fog,rain), entries)
mapper()

# reducer
def reducer():
    riders = 0       # The number of total riders for this key
    num_hours = 0    # The number of hours with this key
    old_key = None
    for line in sys.stdin:
        data=line.strip().split("\t")
        if len(data)!=2:
            continue
        this_key, count = data
        if old_key and old_key != this_key:
            print"{0}\t{1}".format(old_key, riders/num_hours)
            riders = 0
            num_hours=0
        old_key = this_key
        riders += float(count)
        num_hours += 1.0
    if old_key != None:
        print"{0}\t{1}".format(old_key, riders/num_hours)
reducer()
```

iii) *Busiest hour*

This function should determine for each UNIT the busiest date and time. To this end we use map and reduce techniques. For each line of input, the mapper output should print the UNIT, ENTRIESn_hourly, DATEn and TIMEn columns, separated by a tab. The reducer should then print one line per UNIT along with the busiest date and time (grouped and separated by a space) and the number of entries corresponding to this datetime.



```
import sys
import string
# mapper
# mapper
def mapper():
    for line in sys.stdin:
        data=line.strip().split(",")
        if len(data)!=22 or data[1]=='UNIT':
            continue
        unit=data[1]
        entries=data[6]
        date=data[2]
        time=data[3]
        c="{0}\t{1}\t{2}\t{3}".format (unit, entries, date, time)
        #logging.info(c)
        print c
mapper()

# reducer
def reducer():
    max_entries = 0
    old_key = None
    datetime = ''
    for line in sys.stdin:
        data = line.strip().split("\t")
        if len(data) != 4:
            continue
        this_key, count, data, time = data
        count = float(count)
        if old_key and old_key != this_key:
            print "{0}\t{1}\t{2}".format(old_key,datetime,max_entries)
            max_entries = 0
            datetime = ""
        old_key = this_key
        if count >= max_entries:
            max_entries = count
            datetime = str(data) + " " + str(time)
    if old_key != None:
        print "{0}\t{1}\t{2}".format(old_key, datetime, max_entries)
reducer()
```

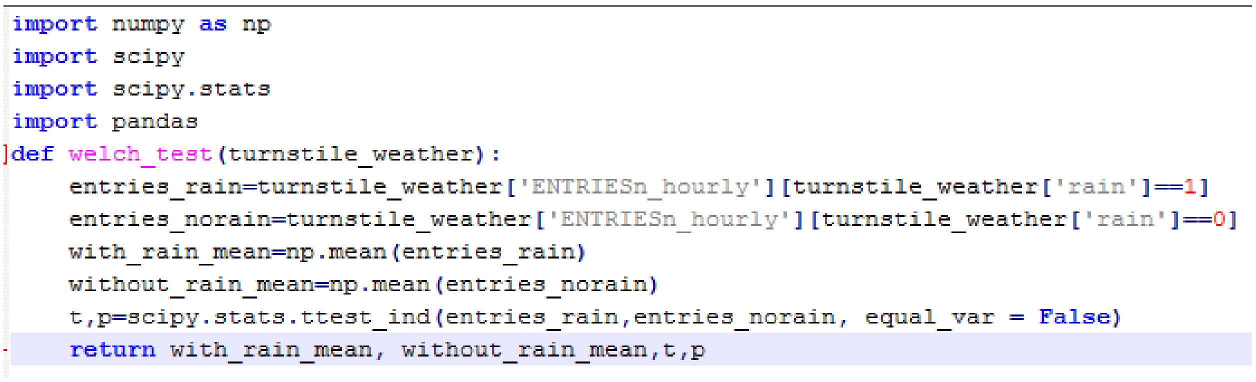
4. Short Questions – Complementary Analysis

In this section we provide code for answering of short questions (when not previously provided) as well as some additional analysis.

4.1. Statistical Test

In the main text we conduct Mann-Whitney U test on rain variable indicating if it was raining within the day at the location. In this section we conduct Welch's t test on this variable, as this might be superior to the M-W test by unequal variances.[26] Moreover, we generate two additional rainy variables and conduct the same tests on them.

We start with Welch's t test conducted on original rain variable. The code looks like as follows.



```
import numpy as np
import scipy
import scipy.stats
import pandas
def welch_test(turnstile_weather):
    entries_rain=turnstile_weather['ENTRIESn_hourly'][turnstile_weather['rain']==1]
    entries_norain=turnstile_weather['ENTRIESn_hourly'][turnstile_weather['rain']==0]
    with_rain_mean=np.mean(entries_rain)
    without_rain_mean=np.mean(entries_norain)
    t,p=scipy.stats.ttest_ind(entries_rain,entries_norain, equal_var = False)
    return with_rain_mean, without_rain_mean,t,p
```

The null hypothesis of this test states that the means of both distributions are not different. By conducting this test we get p-value (for two-sided test) equal to 4.64*10^-7, indicating rejecting the null hypothesis at very small significance level (for example 0.01%). Thus, we can conclude that the means are very likely to be different. Moreover, p-value together with the t statistic (t=5.04) can be used for the one-sided test whether mean by days with rain is larger than mean by days without rain. Basically, we have to divide the p-value by 2. From that we can conclude that the mean_rain is significantly larger than mean_norain at 0.01% significance level.

- *Rain_time*

  This variable indicates whether rain occurred at that time at the location. In order to construct this variable, we use precipitation rate (precipi). Whenever precipi > 0 we argue that it was raining (i.e. raini_time=1).

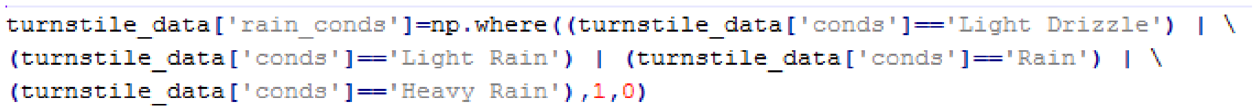
[26] This issue was discussed in the section 2.1.

The results of the tests indicate that there are differences in mean of both distributions at 1 % significance level.

Test	mean_norain	mean_rain	P-value (two-tailed)
Mann-Whitney U test	1896.742	1743.309	0.0002
Welch’s t test	1896.742	1743.309	0.0026

- *Rain_conditions*

  This variable is constructed by means of conds variable. If the conds variable indicates light drizzle, light rain, rain or heavy rain, the rain_conds will be 1, otherwise 0. We generate this variable by using of the following code.



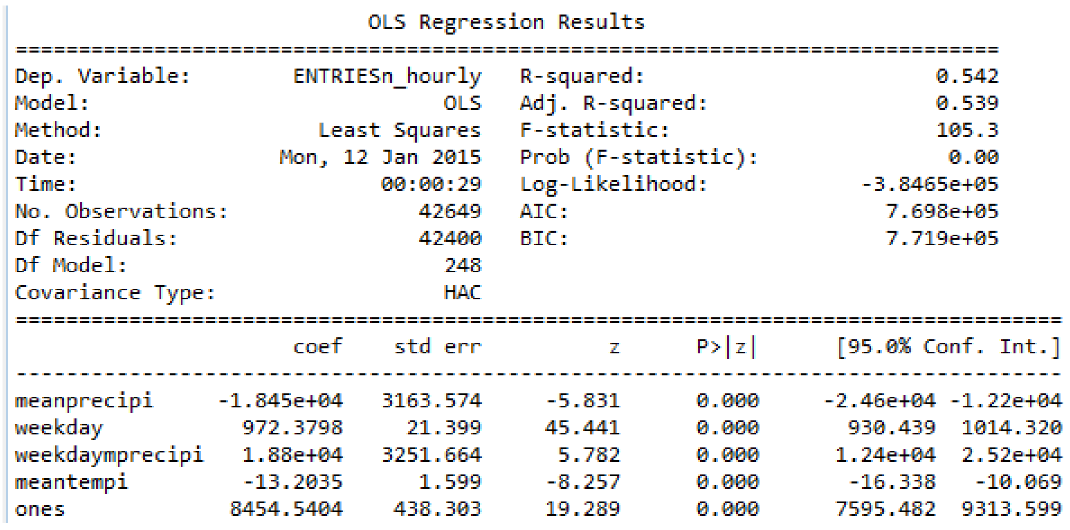
```
turnstile_data['rain_conds']=np.where((turnstile_data['conds']=='Light Drizzle') | \
(turnstile_data['conds']=='Light Rain') | (turnstile_data['conds']=='Rain') | \
(turnstile_data['conds']=='Heavy Rain'),1,0)
```

The results of the tests can be seen in the table below. While the Welch’s test indicates that there are differences in means of both distributions at 1 % significance level, by using Mann-Whitney U test we are not able to reject null hypothesis at the 10 % significance level. So, we cannot confirm that the means are statistically different.

Test	mean_norain	mean_rain	P-value (two-tailed)
Mann-Whitney U test	1896.472	1774.807	0.1093
Welch’s t test	1896.472	1774.807	0.0082

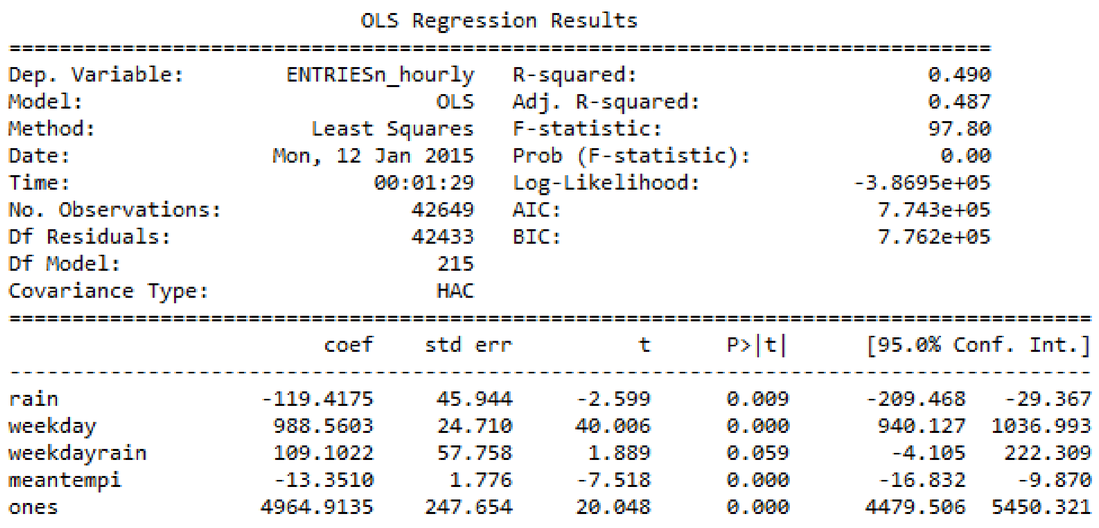
4.2. Linear Regression

This section provides two robustness checks to the original estimation. First, instead of rain variable we use the precipitation rate. The results can be seen in the table below. We conclude that the results are similar to findings from section 2.2. So, the precipitation rate has a negative impact on the subway usage at weekends but a positive impact during weekdays. Both effects are significant at 1%.



```
                            OLS Regression Results
==============================================================================
Dep. Variable:         ENTRIESn_hourly   R-squared:                       0.542
Model:                             OLS   Adj. R-squared:                  0.539
Method:                  Least Squares   F-statistic:                     105.3
Date:                 Mon, 12 Jan 2015   Prob (F-statistic):               0.00
Time:                         00:00:29   Log-Likelihood:            -3.8465e+05
No. Observations:                42649   AIC:                         7.698e+05
Df Residuals:                    42400   BIC:                         7.719e+05
Df Model:                          248
Covariance Type:                   HAC
===================================================================================
                      coef    std err          z      P>|z|      [95.0% Conf. Int.]
-----------------------------------------------------------------------------------
meanprecipi     -1.845e+04   3163.574     -5.831      0.000     -2.46e+04 -1.22e+04
weekday           972.3798     21.399     45.441      0.000       930.439  1014.320
weekdaymprecipi   1.88e+04   3251.664      5.782      0.000      1.24e+04  2.52e+04
meantempi         -13.2035      1.599     -8.257      0.000       -16.338   -10.069
ones             8454.5404    438.303     19.289      0.000      7595.482  9313.599
```

Second, we use station dummies instead of unit dummies. Also the results from this estimation confirm our previous findings. R^2 is here lower than in the original estimation.



```
                            OLS Regression Results
==============================================================================
Dep. Variable:         ENTRIESn_hourly   R-squared:                       0.490
Model:                             OLS   Adj. R-squared:                  0.487
Method:                  Least Squares   F-statistic:                     97.80
Date:                 Mon, 12 Jan 2015   Prob (F-statistic):               0.00
Time:                         00:01:29   Log-Likelihood:            -3.8695e+05
No. Observations:                42649   AIC:                         7.743e+05
Df Residuals:                    42433   BIC:                         7.762e+05
Df Model:                          215
Covariance Type:                   HAC
=====================================================================================
                        coef    std err          t      P>|t|      [95.0% Conf. Int.]
-------------------------------------------------------------------------------------
rain               -119.4175     45.944     -2.599      0.009      -209.468   -29.367
weekday             988.5603     24.710     40.006      0.000       940.127  1036.993
weekdayrain         109.1022     57.758      1.889      0.059        -4.105   222.309
meantempi           -13.3510      1.776     -7.518      0.000       -16.832    -9.870
ones               4964.9135    247.654     20.048      0.000      4479.506  5450.321
```

4.3. Visualization

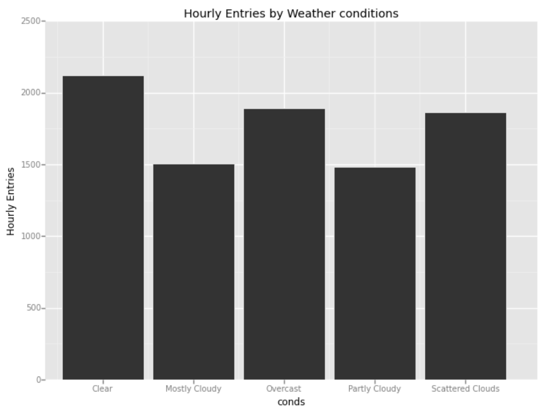
In this section we provide some more figures which are complementary to the previous analysis. In particular, we are interested in how the metro usage varies with different weather conditions as well as over different days.

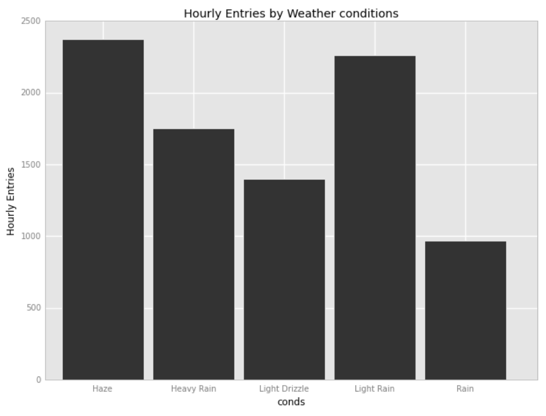
4.3.1. Hourly Entries and Weather Conditions

We might expect that if the weather conditions are not so good, people are willing to use the subway more. Figures from sections 2.3 and 3.3 suggest that the subway is a little bit more used by days with rain than days without rain. In this subsection we want to examine the relationship between subway usage and weather even further. In the following we illustrate the relationship between hourly entries and different weather conditions (e.g. mostly clouded), rain (different definitions of rain), precipitation rate as well as the temperature.

i) Weather Conditions

The figure below illustrates the average hourly entries by different weather conditions. The figures indicate that the busiest days are days with haze or light rain. It is a surprising that the days with rain or light drizzle are not so busy.[27] The figure also suggests (contrary to our expectation) that heavier rain doesn't necessary lead to higher demand for subway in NYC. To examine the impact of rain in more detail we create three different rain variables and look at the relationship between them and subway usage. Moreover, we also look below at the relation between precipitation rate and hourly entries.





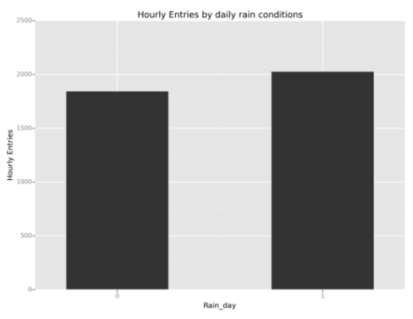
ii) Rain

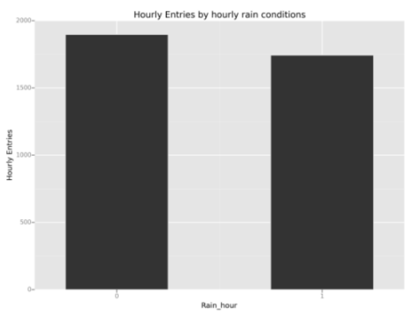
As mentioned above, there are several possibilities how to define the "rain" variable. First, we can use the given indicator whether the rain occurred within the calendar day. Second, we can use actual precipitation rate to construct actual rain indicator. Actual precipitation rate larger than zero is used as an indicator for

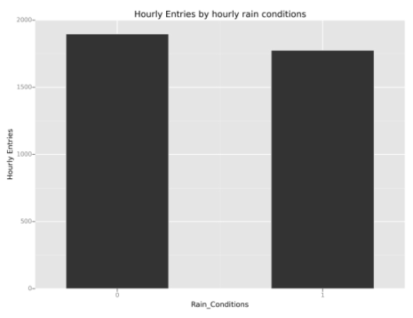
[27] This might be given by relatively small number of observations by rain and heavy rain as well as different definition of rain conditions to the original rain variable.

actual rain. Third, we can make use of the given “conds” variable. Options “Light Drizzle”, “Light Rain”, ”Rain” and “Heavy Rain” are used as an indicator for rain. The figures below illustrate the relationship between hourly entries and different rain variables, where 0 means no rain and 1 means rain. Only one figure (figure in the left upper corner) confirms our previous findings, that the subway is busier by rainy conditions. In this figure we use the daily indicator.

This might have different reasons. First, the actual data are not that precise. Actually, we can see that there are many “rainy-conds” with zero precipitation rate and situation where the precipitation rate is larger than 1 but the conditions are defined as “Overcast. Second, people don’t use subway according to immediate weather conditions but rather according to more general conditions (today is a rainy/sunny day).

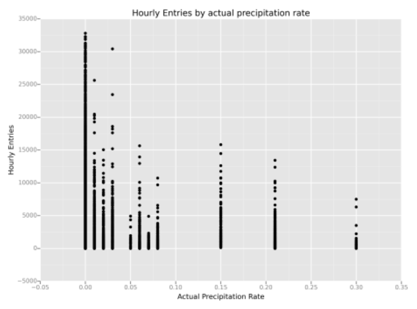


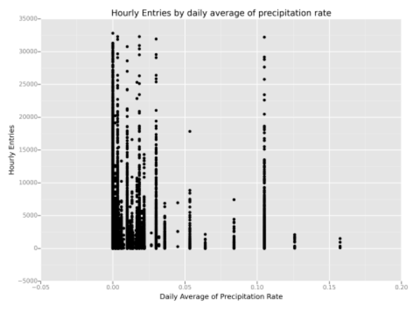




iii) Precipitation Rate

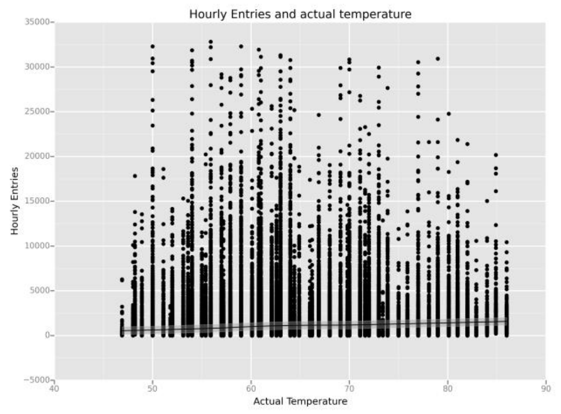
In the previous subsection we saw that there is no clear picture about how the rain influences the demand for subway. It depends on, if we use actual or daily measures. In this subsection we make very similar exercise with precipitation rate. It seems that higher precipitation rate doesn’t lead to higher usage of the subway. This can be clearly seen in the figure with actual precipitation rate (figure on the left hand side). The picture is not that clear, if we look at the average daily precipitation rate.

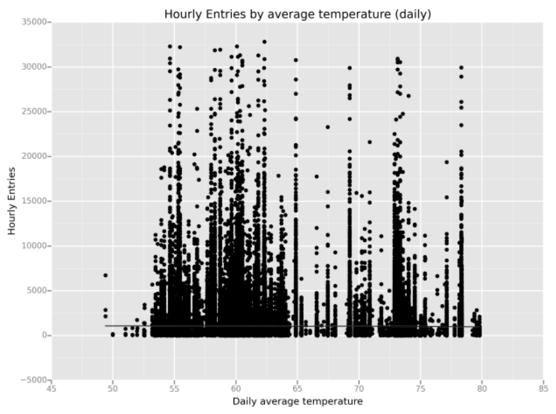




iv) Temperature

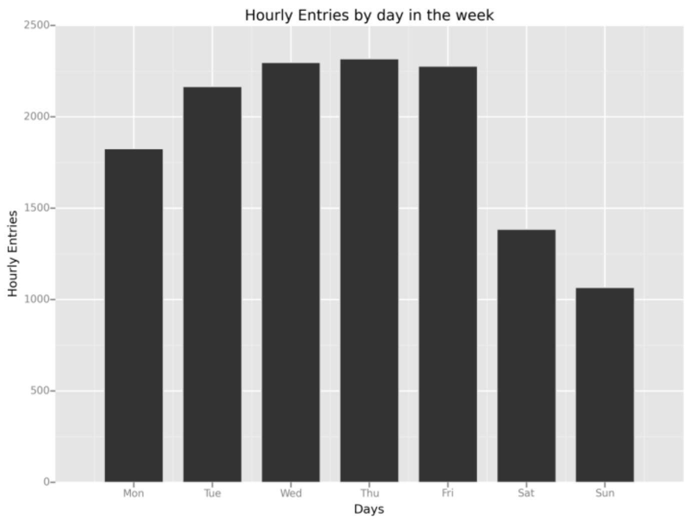
In this section we visualize the relation between temperature (again the actual and the daily average). In general, by considering the following scatter plots, we can’t realize any clear relationship. However, the loess curve suggests that there might be a slight positive relationship (at least for the actual temperature).





4.3.2. Metro Usage over days

We already saw that the subway is significantly busier on weekdays than at weekend. The natural question is, whether there is a significant variation over single days. The figure confirms our previous finding that weekend is less busy than weekdays. There are not so big differences between single weekdays.



5. References

- **P-S 2.1-2.4:** https://dev.mysql.com/doc/refman/5.1/en/counting-rows.html
- **P-S 2.5:** https://piazza.com/class/i23uptiifb6194?cid=116
- **P-S 2.11** http://docs.python.org/2/library/datetime.html#datetime.datetime.strptime

- **Merging of two datasets:** http://pandas.pydata.org/pandas-docs/version/0.15.2/
- **P-S 3.1:** http://pandas.pydata.org/pandas-docs/stable/visualization.html#histograms
- **Normality test**
  http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.normaltest.html
  http://www.graphpad.com/guides/prism/6/statistics/index.htm?stat_qa_normality_tests.htm

- **Discussion about what statistical test is appropriate for mean comparison:**
  Fagerland M. W. and Sandvik L. (2009), The Wilcoxon–Mann–Whitney test under scrutiny, Statistics in Medicine 28(10), 1487 – 1497.

  Ruxton G. D. (2006), The unequal variance t-test is an underused alternative to Student's t-test and the Mann-Whitney U test, Behavioral Ecology 17 (4), 688-690 .

Skovlund E. and Fenstad G. U. (2001). Should we always choose a nonparametric test when comparing two apparently nonnormal distributions? Journal of Clinical Epidemiology 54(1) , 86 – 92.

Stonehouse J. M and Forester G. J. (1998), Robustness of the t and U tests under combined assumption violations. Journal of Applied Statistics 25(1), 63-74.

Zimmerman D. W. and Zumbo B. D. (1993), Rank transformations and the power of the Student *t* test and Welch *t'* test for non-normal populations with unequal variances. Canadian Journal of Experimental Psychology/Revue canadienne de psychologie expérimentale, Vol 47(3), 523-539.

- **P-S 3.2:**
  http://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html
  http://graphpad.com/guides/prism/6/statistics/index.htm?how_the_mann-whitney_test_works.htm
  http://stats.stackexchange.com/questions/116315/problem-with-mann-whitney-u-test-in-scipy

- **OLS:** http://stackoverflow.com/questions/19991445/run-an-ols-regression-with-pandas-data-frame
- **Ploting Residuals:** http://www.itl.nist.gov/div898/handbook/pri/section2/pri24.htm

- **Conclusion and Reflection:**
  Berry, S., J. Levinsohn, and A. Pakes (1995). Automobile prices in market equilibrium. Econometrica 63.841-890.

  Chen, X., Hong H. and D. Nekipelov (2007): Measurement Error Models, survey prepared for Journal of Economic Literature.

  McFadden, D. (1974), The Measurement of Urban Travel Demand, Journal of Public Economics 3, 303-328.

  Wissmann, M., Toutenburg, H. and Shalabh (2007), Role of Categorical Variables in Multicollinearity in Linear Regression Model, Technical Report Number 08, Department of Statistics, University of Munich