Metro ridership during rainy days versus non rainy days

Federico A. Todeschini

April 29, 2015

0 References

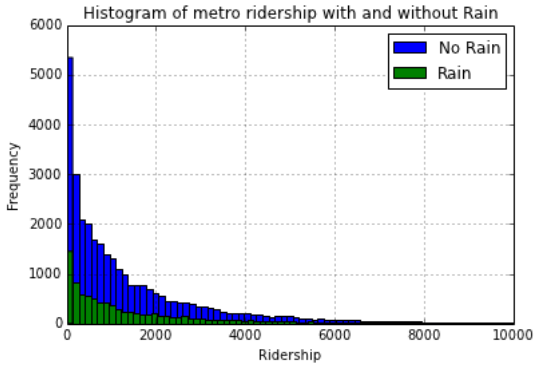
- http://en.wikipedia.org/wiki/Mann%E2%80%93Whitney_U_test
- http://statsmodels.sourceforge.net/stable/
- http://blog.yhathq.com/posts/aggregating-and-plotting-time-series-in-python.html
- https://python.g-node.org/python-summerschool-2009/python_code_in_latex

1 Statistical Test

1.1 In order to answer the question on whether more people used the subway on a rainy day, we performed a Mann-Whitney test. In this case the null hypothesis was that the both the number of entries per hour during rain and during no rain were from the same population and as such the probability of an observation from the rain population exceeding and observation from the non rain population equals the probability of an observation from the non rain population exceeding an observation from the rain population. We used a two sided test, since we had no a priori knowledge on the median of the two variables and as a consequence the difference of the two means could take any direction. Our choice for the critical p value was 0.05.

1.2 The Mann-Whitney test is a nonparametric test that allows two groups or conditions or treatments to be compared without making the assumption that values are normally distributed. In order to see whether the variables were normally distributed, we first draw and histogram of the number of entries per hour both for the cases where it was raining and when it was not raining, which we can see in graphic 1.

Figure 1: Metro ridership with and without rain



According to the aforementioned graphic, the distribution of the variables do not seem to come from a normal distribution. Because of that we run the Shapiro-Wilk test to know what was the likelihood that the number of entries per hour was obtained from a normal distribution. Since the student test is applicable

only certain assumptions (normal distribution) but is more efficient than the Mann-Whitney test, we wanted to make sure which one to use.

Due to the evidence at hand and the fact that we had enough observations, we performed the the non parametric Mann-Whitney test

1.3 In table 1 we have the mean value of the metro ridership during rainy days and non rainy days, and the p-value from the Mann-Whitney test.

Table 1: Mean Ridership during rainy and non rainy days

Mean Rain	Mean No Rain	Mann Whitney p value
2028.20	1845.54	5.482139e-06

1.4 From table 1 we see that the mean ridership per hour during rainy moments is larger than during non rainy moments. Now, this could be due just by chance, that is, an unfortunate sample. The result from the Mann Whitney test tell us that the probability of observing such extreme difference between the two distribution is very low. As a consequence, we can reject the null hypothesis that the two distributions are the same in favor of the alternative hypothesis.

2 Linear Regression

2.1 The approach used to compute the coefficients theta and produce prediction for ENTRIESn_hourly in my regression model was OLS using Statsmodels.

2.2 Among the input variables used to estimate the model were

- rain
- precipi
- fog
- tempi
- pressurei
- wspdi
- station
- day

- hour

Weekday is a dummy variable that indicates whether it is a weekday or not. The variable hour was transformed into dummy variables for each time of the day and the variable day was transformed into dummy variables indicating the day of the week. Additionally the station is also used as an input and it is transformed into dummy variables. We have also created another variables from precipi. One precipi2, which is precipi squared and another, rain alt, which is a dummy variable being one when precipi is greater than zero.

As such, the estimated model was:

$$Ridership_{sht} = \alpha + \beta_1 rain_{sht} + \beta_3 precipi_{sht} + \beta_3 precipi^2_{sht} + \beta_4 tempi_{sht} + \beta_5 pressurei_{sht} + \beta_6 wspdi_{sht} + \mu_s + \rho_h + \rho_t + \epsilon_{sht} \quad (1)$$

The parameters in equation 1 are the following:

- α is a constant which reflects the total number of riders that are predicted if all the other variables are set to zero.
- β is the coefficient of rain, which is a dummy variable that takes a value of one on the days that rained on a particular station. Therefore, for each station rain has the same value throught out the day. Therefore a positive β_1 would indicate that on the days that it rained, on average more people raided the metro.
- β_2 is the coefficient of precipi, which is in turns is the precipitation in inches at the time and location. Therefore, this variable does change on station and time. We include this variable because we believe that the ammount of rain might change the number of riders and not just the fact that has rained. A positive β_2 here would be signalling that increments in the number of inches rained increased the number of raiders. Because the units in this case are to high, we multiplied the variable by 100 so that the coefficient will be decreased in exactly the same proportion
- β_3 is the coefficient for the square of (modified) precipi. We are assuming therefore the possibility of a non linear relationship between the ammount that has rained and metro ridership.
- β_4 is the coefficient for tempi, which measures the temperature in F at the time and location. Here we are assuming that other conditions may be correlated with rain and at the same turn, they affect ridership. For instance, it could be that during low temperatures people rides the metro more, as they want to avoid catching a cold from walking. But rain may happen more during low temperatures and therefore, we should include temperature as well.
- β_5 is the coefficient for the barometric pressure in inches Hg at the time and location, represented by pressurei. The line of reasoning for temperature applies here.
- β_6 is the coefficient for the wind speed in mph at the time and location, represented by wspdi.
- μ_s are station dummies that are included to capture the effects that are fixed at the level of the station. The line of reasoning for temperature applies here.

- ρ_h are hourly dummies and we include them in order to capture the different seasonality during the day, since a lot more people take the metro during peak our but very few at, say 11pm.
- ρ_t are weekday dummies and we include them in order to capture the seasonality during the week, since less people take the metro during weekends.
- ϵ_{sht} is the error term, that is all things that are not captured by our model

The results of this model is that rain **decreases** ridership in 72 persons per hour, a result that is statistically different from zero, which leads us to reject the null hyptohesis that ridership is not affected by rain. In table 2 we show the results for the different specification of the statistical analysis

Table 2: Results from the different OLS estimations

Variable	Baseline model	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7
$rain$	182.65***		308.29***	404.60***	241.96***	252.85***	232.09*	38.51
$rain\ alt$		-153.43***						
$precipi$				-78.19***	-70.38***	-79.99***	-161.41***	
$precipi^2$				1.32	0.88	1.23	3.28***	
fog			-333.95**	100.4	162.43	186.23	708.68***	
$tempi$			35.10***	33.93	32.07***	35.50***	27.00***	
$pressurei$			-178.85	-123.61	-459.73***	-253.8515***	-185.13***	
$wspdi$			37.26***	37.74*	25.03***	31.21***	85.48***	
$weekday$					917.74***			
Hour dummies	No	No	No	No	No	No	No	Yes
Day dummies	No	No	No	No	No	Yes	Yes	Yes
Station dummies	No	No	No	No	No	No	Yes	Yes
R^2	0.01	0.00	0.013	0.015	0.034	0.038	0.372	0.494

*,** and *** signifcant at 10%, 5% and 1% respectively

2.3 Weekday is included because it is a variable that is correlated with metro ridership, as more people uses the metro during labor days. As such, if in our sample rain mostly happens during sunday, shall we not include this variable we might conclude that ridership is not affected by rain. For the same reason we included the station and the hour of the day. The variable precipitation as its square was included because it is possible that the relationship between rain and ridership is not linear. The variables fog, tempi, pressurei and wspdi were included on an intuitive basis, in the sense that atmospheric conditions might affect ridership and at the same time are likely to be correlated with rain.

2.4 The coefficients for the non dummy variables in our the model 3 estimation, that includes precipi, precipi2, fog, tempi, pressurei and wspdi and weekday are shown in table 2 along with the R^2.

2.5 The largest estimated model's R^2 is 0.491. The R^2 is the proportion of the total variance that of ENTRIESn_horly explained by linear variation from the selected explanatory variables. However, that particular score of the R^2 comes from a model that includes dummy variables for each station, each day of the week

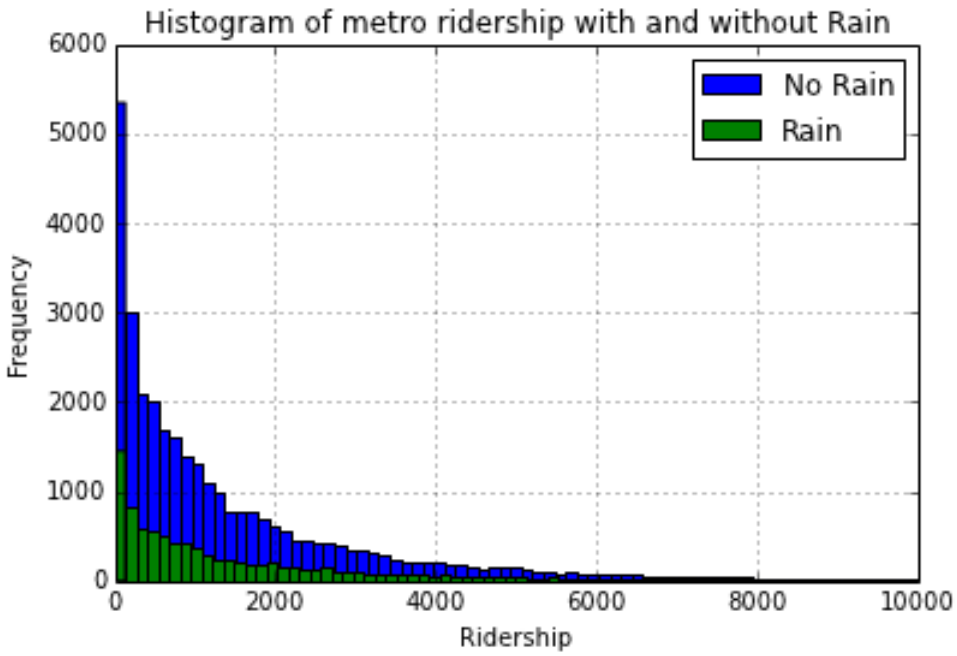
and each hour of the day, so it is probably 'inflated' in the sense that these type of variables usually drive up the R^2 without necessarily improving the prediction ability of the model.

2.6 The value obtained means that we are able to explain about 50% of the variance. However, a large part of the explanatory analysis of our model is possibly inflated by the fact that we are using station dummies as well as other variables that are possibly highly colinear. This means that the variance covariance matrix from our model should not be trusted blindly. So, would I trust the predictions made by the model? In terms of the rain coefficient, besides some problems with the data and with the statistical model that are covered in section 5, we don't have anything to say. But in terms of the power of our predictions, I find the R^2 to be too high relative to the case without those dummy variables and therefore I wouldn't trust the model in terms of its predictive capacity on metrro ridership by hour and I would suggest a different statistical analysis (more in section 5) for that.

3 Visualization

3.1 Histrogram of ridership for rainy and non rainy days In figure 2 we can see the histogram for the ridership for rain from 0 to 10,000 riders per hour. As we can see, on those days when it rain, ridership increases creating very extreme values with a very low frequency.

Figure 2: Metro ridership with and without rain



3.2 Boxplot of ridership by day of the week In figure 3 we can see how the variabiliy of metro ridership by day of the week using a boxplot graph. This chart, show us both the different quantiles of the distribution plus the outliers.

Figure 3: Metro ridership by day of the week

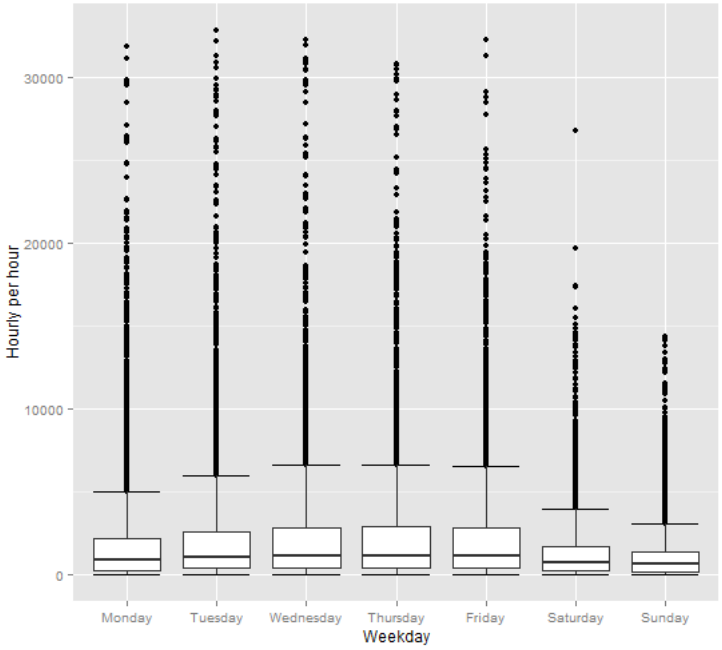


Figure 4 shows boxplots by weekday and whether it has rained or not. We can see that it is mostly the outliers from the working days that are driving the results in the mean test and in the OLS regression

Finally in figure 5 we can see the total ridership per day during May at the different hours of the day. In this graphic we can see the huge stationality that the data has.

4 Conclusion

4.1 According to the analysis done our main conclusion is that rain affects metro ridership. That is, during the day that rain is recorded, on average more people uses the metro. However, if we take into account some particularities of the different location where rain was recorded and we consider the hour when it rained, then the result is modified. In that case, we could say that rain makes more people use the metro but they use it less while it's raining, probably because less people goes out or because some specific locations that are usually too crowded and they are even more crowded when it rains.

4.2 The Mann Whitney statistical test rejects the null hypothesis that the distributions of metro ridership by hour are the same during rainy and non rainy days in favor of the alternative hypothesis, in this case that

Figure 4: Metro ridership by weekday and rain

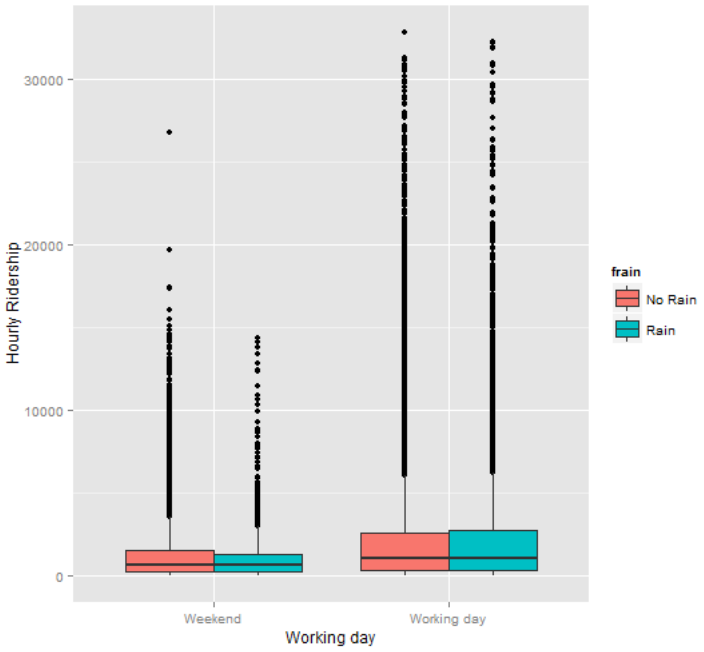
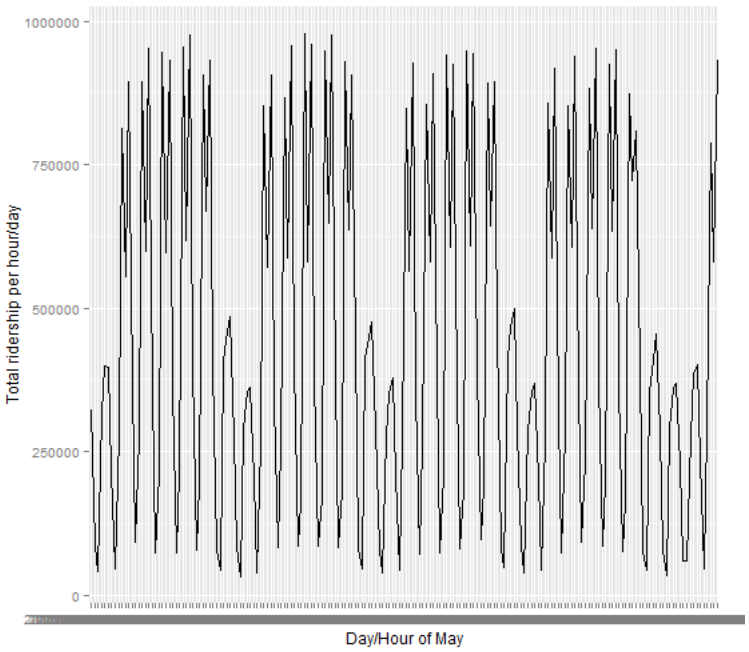


Figure 5: Total daily metro ridership



metro ridership is larger during rainy days. However, the histogram as well as the boxplot show that the two distribution are somehow similar, with no rain actually having larger values than rain, except at very extreme values. In the histogram, for instance, we can see that rainy days have a much larger values although a very low frequency as the proportion of rainy days in the sample is quite low (22%)

The very basic regression analysis, that is, those without dummy variables at the station level, confirms the idea from the Mann Whitney test and show that on average more people rides the metro on rainy days. For instance, according to model 4 from table 2 we could state that during rainy days on average 241 more people rides the metro hourly per station. Taking into consideration that the average number of riders per hour is about 1887, it would account for a 13% increment.

When we include dummies at the hourly level (model 7 from table 2), the number of riders per hour at the station *is no longer significant*, although the sign remains the same, but the fact that the null hypothesis can't be rejected in this case could be a consequence of a variance covariance matrix that does satisfies the normality assumptions, since we are facing a high colinearity among the explanatory variables (for instance, the probability of rain is not distributed uniformly through out the day and that could mean that the hour at which is raining and metro ridership by hour are correlated). However, if instead of using rain as the explanatory variable we used a dummy variable being one if it has rained during a specific hour at a certain station the results changes and now we have a reduction of -153 riders per hour. This results tell us that even though on average more people rides the metro during rainy days, at the time it's actually raining, less people are out in the street and therefore less of them are taking the metro.

Therefore, our conclusion from the statistical analysis performed is that rains affects ridership positively. However, while it's raining less people rides the metro. A potential explanation to this finding is that based on the forecast from the previous day, some people decides not to use their usual mean of transportation (like bike or motorbike) and decide to go by metro.

As a final note, just mentioning that OLS and univariate tests such as the Mann Whitney are not necessarily well suited to the presence of outliers, which as we previously stated, seems to be what is driving some of the results, so I word of caution on the use of the estimated.

5 Reflection

5.1 A very first problem that we have with our dataset is that the variable of interest, metro ridership by hour, is probably not stationary. That means, in a very plane manner, that the variable grows in time and that can affect the relationship between any variable and ridership. In that sense, it would be useful to put in context, for instance, something like the ratio of ridership to the total population. A second shortcoming of the dataset is that during the year there are rainy seasons and there are seasons without rain. That means that rain will behave very differently during different moments of the year and we should try to remove that. The same happens with ridership, although in a worst manner. If we would be taking our information on August and September and it happens to be the case that september is the rainy moment of the year while August is the holiday month, everything else being equal we would wrongly conclude that rain negatively affects ridership.

Another interesting thing would be to know whether on that day there was something important going on (some accident, a parade, etc) that would have driven more people into the metro irrespectively. Last but definitely not least, people usually try to forecast what will be the weather during the following day. In that sense, if let's say the forecast is rain then they will be equipped with an umbrella such that rain shouldn't be a problem. Therefore, what will probably affects ridership is unexpected rain. Following on that argument, we would like to incorporate to the dataset the weather forecast from the previous day.

5.1 Since the outliers in this dataset play a major role in driving the results, maybe using linear models it's not the best way to answer the question of whether people ride more the metro during rainy days. One potential way of dealing with this issue would be to model rainy days and non rainy days separately, as if they were two different process. In the case of the Titanic, we did something in that line of reasoning as we tried to separate the different people on the ship. Alternatively, we could try to get rid of the outliers using a more robust model, like quantile regression.

6 Code for the Problem Sets

6.1 PS2

```
import pandas
import pandasql


def num_rainy_days(filename):
    weather_data = pandas.read_csv(filename)

    q = """
    SELECT
    count(date)
    FROM
    weather_data
    WHERE
    cast(rain as integer) = 1
    """

    #Execute your SQL command against the pandas frame
    rainy_days = pandasql.sqldf(q.lower(), locals())
    return rainy_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;
    """

    #Execute your SQL command against the pandas frame
    foggy_days = pandasql.sqldf(q.lower(), locals())
    return foggy_days



import pandas
import pandasql

def avg_weekend_temperature(filename):
    weather_data = pandas.read_csv(filename)

    q = """
    SELECT
    avg(meantempi)
    FROM
    weather_data
    WHERE
    cast(strftime('%w', date) as integer) = 0 OR cast(strftime('%w', date) as integer) = 6;
    """

    #Execute your SQL command against the pandas frame
    mean_temp_weekends = pandasql.sqldf(q.lower(), locals())
    return mean_temp\_weekends
```

```
import pandas
import pandasql

def avg_min_temperature(filename):
    weather_data = pandas.read_csv(filename)

    q = """
    SELECT
    avg(mintempi)
    FROM
    weather_data
    WHERE
    rain = 1 and cast(mintempi as integer)>55;
    """

    #Execute your SQL command against the pandas frame
    avg_min_temp_rainy = pandasql.sqldf(q.lower(), locals())
    return avg_min_temp_rainy


import csv

def fix_turnstile_data(filenames):
    for name in filenames:
        f_in = open(name, 'r')
        f_out = open('updated_' + name, 'w')

        reader_in = csv.reader(f_in, delimiter=',')
        writer_out = csv.writer(f_out, delimiter=',')

        #reader_in.next()
        for line in reader_in:
            primero = line[0]
            segundo = line[1]
            tercero = line[2]
            for row in range((len(line)-3)/5):
                linea = [primero, segundo, tercero, line[3+5*row], line[4+5*row], line[5+5*row],
                writer_out.writerow(linea)
        f_in.close()
        f_out.close()
```

```
import csv
def create_master_turnstile_file(filenames, output_file):
    with open(output_file, 'w') as master_file:
        master_file.write('C/A,UNIT,SCP,DATEn,TIMEn,DESCn,ENTRIESn,EXITSn\n')
        for filename in filenames:
            f_in = open(filename, 'r')
            reader_in = csv.reader(f_in, delimiter=',')
            for linea in reader_in:
                a = ','.join(linea)
                a = a+'\n'
                master_file.write(a)
            f_in.close()


import pandas
import pandasql
import csv

def filter_by_regular(filename):
    turnstile = pandas.read_csv(filename)
    turnstile_data = pandas.DataFrame(turnstile)
    turnstile_data = turnstile_data[turnstile_data.DESCn == 'REGULAR']

    return turnstile_data


import pandas
import pandasql
import csv

def filter_by_regular(filename):
    turnstile = pandas.read_csv(filename)
    turnstile_data = pandas.DataFrame(turnstile)
    turnstile_data = turnstile_data[turnstile_data.DESCn == 'REGULAR']

    return turnstile_data
```

Ex9 import pandas

```
def get_hourly_exits(df):
    df['EXITSn_hourly'] = df['EXITSn'] - df['EXITSn'].shift()
    df['EXITSn_hourly'].fillna(value = 0, inplace = True)
```

```
    return df


import pandas

def time_to_hour(time):
    hour = int(time[0:2])
    return hour


import pandas

def time_to_hour(time):
    hour = int(time[0:2])
    return hour


import datetime

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

6.2 PS 3

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

def entries_histogram(turnstile_weather):
    plt.figure()
    turnstile_weather['ENTRIESn_hourly'][turnstile_weather['rain']==0].hist(bins=240, facecolor=
    turnstile_weather['ENTRIESn_hourly'][turnstile_weather['rain']==1].hist(bins=240, facecolor=
    plt.xlabel("ENTRIESn_hourly")
    plt.ylabel("Frequency")
    plt.title("Histogram of ENTRIESn_hourly")
    plt.xlim(0, 6000)
    plt.legend()
    return plt
```

```
import numpy as np
import scipy
import scipy.stats
import pandas

def mann_whitney_plus_means(turnstile_weather):
    rain = turnstile_weather['ENTRIESn_hourly'][turnstile_weather['rain']==1]
    norain = turnstile_weather['ENTRIESn_hourly'][turnstile_weather['rain']==0]

    with_rain_mean = np.mean(rain)
    without_rain_mean = np.mean(norain)
    mwt = scipy.stats.mannwhitneyu(rain, norain)
    U = mwt[0]
    p = mwt[1]

    return with_rain_mean, without_rain_mean, U, p


import numpy as np
import pandas
from ggplot import *

def normalize_features(df):
    mu = df.mean()
    sigma = df.std()

    if (sigma == 0).any():
        raise Exception("One or more features had the same value for all samples, and thus could
                        "not be normalized. Please do not include features with only a single v
                        "in your model.")
    df_normalized = (df - df.mean()) / df.std()

    return df_normalized, mu, sigma

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

    return cost

def gradient_descent(features, values, theta, alpha, num_iterations):
```

```
    m = len(values)

    cost_history = []
    updated_theta = theta
    i = 0

    ############################
    ### YOUR CODE GOES HERE ###
    ############################
    for i in range(num_iterations):
        predicted_values = np.dot(features, updated_theta)
        updated_theta += alpha/m * np.dot((values - predicted_values), features)
        cost_history.append(compute_cost(features, values, updated_theta))
        i += 1

    theta = list(updated_theta)
    return theta, pandas.Series(cost_history)


def predictions(dataframe):
    # Select Features (try different features!)
    features = dataframe[['rain', 'precipi', 'Hour', 'fog']]

    # Add UNIT to features using dummy variables
    dummy_units = pandas.get_dummies(dataframe['UNIT'], prefix='unit')
    features = features.join(dummy_units)

    # 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
    # -------------------------------------------------
    # Uncomment the next line to see your cost history
    # -------------------------------------------------
    # plot = plot_cost_history(alpha, cost_history)
    #
    # Please note, there is a possibility that plotting
    # this in addition to your calculation will exceed
    # the 30 second limit on the compute servers.

    predictions = np.dot(features_array, theta_gradient_descent)
    return predictions, plot


def plot_cost_history(alpha, cost_history):
   cost_df = pandas.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 )



import numpy as np
import scipy
import matplotlib.pyplot as plt

def plot_residuals(turnstile_weather, predictions):

    plt.figure()
    (turnstile_weather['ENTRIESn_hourly'] - predictions).hist()
    return plt
```

```
import numpy as np
import scipy
import matplotlib.pyplot as plt
import sys

def compute_r_squared(data, predictions):
    SSReg = np.sum((data-predictions)**2)
    ## Alternatively
    ## SSReg = (data-predictions)**2).sum()
    SST = np.sum((data-np.mean(data))**2)
    r_squared = 1 - SSReg / SST


    return r_squared


# -*- coding: utf-8 -*-

import numpy as np
import pandas
import scipy
import statsmodels.api as sm
import statsmodels.formula.api as smf

def predictions(weather_turnstile):
    Y = weather_turnstile['ENTRIESn_hourly']
    X = weather_turnstile[['rain', 'precipi', 'Hour', 'fog', 'thunder']]
    X = sm.add_constant(X)
    dummy_units = pandas.get_dummies(weather_turnstile['UNIT'], prefix='unit')
    X = X.join(dummy_units)
    model = sm.OLS.from_formula('ENTRIESn_hourly ~ 1 + Hour + Hour*rain + fog*rain + rain + I(ra
fog + UNIT', weather_turnstile.reset_index())
    results = model.fit()
    print(model.fit().summary2())
    prediction = results.predict(X)



    return prediction
```

6.3 Problem Set 4

```
from pandas import *
from ggplot import *

def plot_weather_data(turnstile_weather):

    turnstile_df = DataFrame(turnstile_weather)
    turnstile_df['Date'] = to_datetime(turnstile_df['DATEn'])

    plot = ggplot(turnstile_weather, aes(x='Date', y='ENTRIESn_hourly')) + geom_boxplot() +
ggtitle('Ridership by day of the week') + xlab('Day') + ylab('Riders by hour')
    return plot
```

6.4 Problem Set 5

```
import sys
import string
import logging

from util import mapper_logfile
logging.basicConfig(filename=mapper_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')

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


mapper()
```

```
import sys
import logging
```

```
from util import reducer_logfile
logging.basicConfig(filename=reducer_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')

def reducer():
    '''
    Given the output of the mapper for this exercise, the reducer should PRINT
    (not return) one line per UNIT along with the total number of ENTRIESn_hourly
    over the course of May (which is the duration of our data), separated by a tab.
    An example output row from the reducer might look like this: 'R001\t500625.0'

    You can assume that the input to the reducer is sorted such that all rows
    corresponding to a particular UNIT are grouped together.

    Since you are printing the output of your program, printing a debug
    statement will interfere with the operation of the grader. Instead,
    use the logging module, which we've configured to log to a file printed
    when you click "Test Run". For example:
    logging.info("My debugging message")
    Note that, unlike print, logging.info will take only a single argument.
    So logging.info("my message") will work, but logging.info("my","message") will not.
    '''

    reg_count = 0
    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, reg_count)
            #logging.info("{0}\t{1}".format(old_key, reg_count))
            #logging.info(str(old_key)+"\t"+str(reg_count))
            reg_count = 0

        old_key = this_key
        reg_count += float(count)

    if old_key != None:
```

```
        print "{0}\t{1}".format(old_key, reg_count)
        #logging.info("{0}\t{1}".format(old_key, reg_count))
        #logging.info(str(old_key)+"\t"+str(reg_count))

reducer()



import sys
import string
import logging

from util import mapper_logfile
logging.basicConfig(filename=mapper_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')

def mapper():
    # Takes in variables indicating whether it is foggy and/or rainy and
    # returns a formatted key that you should output.  The variables passed in
    # can be booleans, ints (0 for false and 1 for true) or floats (0.0 for
    # false and 1.0 for true), but the strings '0.0' and '1.0' will not work,
    # so make sure you convert these values to an appropriate type before
    # calling the function.
    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
        #logging.info(data[14])
        #logging.info(data[15])
        #logging.info(format_key(data[14],data[15]))
        print "{0}\t{1}".format(format_key(float(data[14]), float(data[15])), data[6])
        #logging.info("{0}\t{1}".format(format_key(float(data[14]), float(data[15])), data[6]))
mapper()

import sys
import logging
```

```
from util import reducer_logfile
logging.basicConfig(filename=reducer_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')

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)
            #logging.info("{0}\t{1}".format(old_key, riders/num_hours))
            average_riders = 0
            riders = 0
            num_hours = 0
        old_key = this_key
        riders += float(count)
        num_hours += float(1)


    if old_key != None:
        print "{0}\t{1}".format(old_key, riders/num_hours)
        #logging.info("{0}\t{1}".format(old_key, riders/num_hours))

reducer()
```

```
import sys
import string
import logging

from util import mapper_logfile
logging.basicConfig(filename=mapper_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')
```

```
def mapper():
    """
    In this exercise, for each turnstile unit, you will determine the date and time
    (in the span of this data set) at which the most people entered through the unit.

    The input to the mapper will be the final Subway-MTA dataset, the same as
    in the previous exercise. You can check out the csv and its structure below:
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv

    For each line, the mapper should return the UNIT, ENTRIESn_hourly, DATEn, and
    TIMEn columns, separated by tabs. For example:
    'R001\t100000.0\t2011-05-01\t01:00:00'

    Since you are printing the output of your program, printing a debug
    statement will interfere with the operation of the grader. Instead,
    use the logging module, which we've configured to log to a file printed
    when you click "Test Run". For example:
    logging.info("My debugging message")
    Note that, unlike print, logging.info will take only a single argument.
    So logging.info("my message") will work, but logging.info("my","message") will not.
    """

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

import sys
import logging

from util import reducer_logfile
logging.basicConfig(filename=reducer_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')

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, date, time = data

        if old_key and old_key != this_key:
            print "{0}\t{1}\t{2}".format(old_key, datetime, max_entries)
            #logging.info("{0}\t{1}".format(old_key, reg_count))
            max_entries = 0
            datetime = ''

        old_key = this_key
        if float(count) > max_entries:
            max_entries = float(count)
            datetime = date + ' ' + time
        elif float(count) == max_entries and date[8:10] > datetime[8:10]:
            datetime = date + ' ' + time
        elif float(count) == max_entries and date[8:10] == datetime[8:10] and time[:2] > datetim
            datetime = date + ' ' + time


    if old_key != None:
        print "{0}\t{1}\t{2}".format(old_key, datetime, max_entries)
        #logging.info("{0}\t{1}".format(old_key, reg_count)

reducer()
```

6.5 Code used to answer the questions

```
import numpy as np
import scipy.stats
import scipy
import pandas
import pandasql
import matplotlib.pyplot as plt
from ggplot import *
import time
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

```
path = "C:\\Users\\federico\\Documents\\Formacio\\Udacity\\turnstile_weather_v2.csv"

def loadWeatherData(archivo):
    '''
    (path) -> Pandas Data Frame
    Returns a pandas dataframe
    '''
    data = pandas.read_csv(archivo)
    data_df = pandas.DataFrame(data)

    return data_df

def rainTest(dataframe):
    '''
    (Pandas Data Frame) -> tupple
    It takes the weather dataframe and performs the student and mann whitney test
    comparing the number of hourly entries with and without rain
    Returns the a tupple with the mean for rainy days, the mean for no rainy days, and the stati
    '''
    entries_rain = dataframe['ENTRIESn_hourly'][dataframe['rain']==1]
    entries_norain = dataframe['ENTRIESn_hourly'][dataframe['rain']==0]
    rain_mean = np.mean(entries_rain)
    norain_mean = np.mean(entries_norain)
    student = scipy.stats.ttest_ind(entries_rain, entries_norain, equal_var = False)
    t_student = student[0]
    p_student = student[1]
    mwt = scipy.stats.mannwhitneyu(entries_rain, entries_norain)
    U_mwt = mwt[0]
    p_mwt = mwt[1]*2 ##For two tails we have to multiply the pvalue

    return rain_mean, norain_mean, t_student, p_student, U_mwt, p_mwt

def rainHistogram(dataframe):
    '''
    (Pandas Data Frame) -> plot
    '''
    plt.figure()
    dataframe['ENTRIESn_hourly'][dataframe['rain']==0].hist(bins=240, facecolor='blue', label =
    dataframe['ENTRIESn_hourly'][dataframe['rain']==1].hist(bins=240, facecolor='green', label =
    plt.xlabel("Ridership")
    plt.ylabel("Frequency")
```

```
    plt.title("Histogram of metro ridership with and without Rain")
    plt.xlim(0, 10000)
    plt.legend()
    #plt.hist(bins=[0, 1000, 2000, 3000, 4000, 5000, 6000])
    return plt

def raidershipDay(dataframe):
    '''
    (Pandas Data Frame) -> plot
    '''
    #dummy_days = pandas.get_dummies(dataframe['day_week'], prefix='day')
    plt.figure()
    plt.scatter(dataframe['day_week'],dataframe['ENTRIESn_hourly'], alpha=0.5)
    plt.xlabel("Day")
    plt.ylabel("Ridership")
    plt.title("Scatterplot of day of the week and metro ridership")
    #plt.xlim(0, 10000)
    plt.legend()
    #plt.hist(bins=[0, 1000, 2000, 3000, 4000, 5000, 6000])
    return plt

def boxshipDay(dataframe):
    '''
    (Pandas Data Frame) -> plot
    '''
    plt = ggplot(dataframe, aes(x='day_week', y='ENTRIESn_hourly')) + geom_boxplot()

    return plt

def predictions(weather_turnstile):
    '''
    (Pandas Data Frame) -> object
    It takes the weather_turnstile data frame
    It creates dummy variables for station, days and hour.
    Modifies the precipitation variable to modify the unit of measure
    Computes the OLS regression of ENTRIESn_hourly (metro ridership by hour)
    on several explanatory variables and returns the prediction
    '''
    weather_turnstile['precipi_alt'] = weather_turnstile['precipi']*100
    weather_turnstile['precipi2_alt'] = weather_turnstile['precipi_alt']**2
    weather_turnstile['heavyrain'] = weather_turnstile['conds'] == 'Heavy Rain'
    weather_turnstile['rain_alt'] = 0
```

```
weather_turnstile['rain_alt'][weather_turnstile['precipi'] > 0] = 1
Y = weather_turnstile['ENTRIESn_hourly']
#X = weather_turnstile[['rain', 'precipi_alt', 'precipi2_alt', 'fog', 'tempi', 'pressurei',
X = weather_turnstile[['rain','fog', 'tempi', 'pressurei', 'wspdi']]
dummy_units = pandas.get_dummies(weather_turnstile['station'], prefix='station')
X = X.join(dummy_units)
dummy_days = pandas.get_dummies(weather_turnstile['day_week'], prefix='day')
X = X.join(dummy_days)
dummy_hour = pandas.get_dummies(weather_turnstile['hour'], prefix='hr')
X = X.join(dummy_hour)
X = sm.add_constant(X)
model = sm.OLS(Y,X)
results = model.fit()
prediction =results.predict(X)
results = model.fit()
print(model.fit().summary2())
prediction = results.predict(X)
return prediction
```