Supporting code – Visualization

# **Section 3. Visualization**

3.1 One visualization should contain two histograms: one of ENTRIESn\_hourly for rainy days and one of ENTRIESn\_hourly for non-rainy days.

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

"""

Created on Sun Apr 19 10:41:34 2015

@author: jsahewal

"""

import pandas as pd

import matplotlib.pyplot as plt

import numpy

data = pd.read\_csv("C:\Users\jsahewal\Downloads\weather.csv")

rain\_days = data[data['rain'] == 1]

#print rain\_days['ENTRIESn\_hourly']

no\_rain\_days = data[data['rain'] == 0]

bins = numpy.linspace(0, 15000, 15)

#no\_rain\_days['ENTRIESn\_hourly'].plot(kind='hist', bins=bins, alpha=0.5, color='green')

#rain\_days['ENTRIESn\_hourly'].plot(kind='hist', bins=bins, alpha=0.5, color='blue')

no\_rain\_days['ENTRIESn\_hourly'].hist(bins=bins, alpha=0.5, color='green')

rain\_days['ENTRIESn\_hourly'].hist(bins=bins, alpha=0.5, color='blue')

plt.title('ENTRIESn\_hourly Histograms rain vs. no-rain')

plt.axis([0, 15000, 0, 70000])

plt.xlabel('ENTRIESn\_hourly')

plt.ylabel('frequency')

plt.legend(['rain', 'no-rain'])

plt.show()

3.2 One visualization can be more freeform. You should feel free to implement something that we discussed in class (e.g., scatter plots, line plots) or attempt to implement something more advanced if you'd like

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

"""

Created on Sun Apr 19 10:42:03 2015

@author: jsahewal

"""

import matplotlib.pyplot as plt

from pandas import \*

from ggplot import \*

from datetime import \*

from numpy import mean

data = read\_csv("C:\Users\jsahewal\Downloads\weather.csv")

data['Day'] = data['DATEn'].map(lambda x:datetime.strptime(x, '%Y-%m-%d').strftime('%w'))

agg\_day = data.groupby(['Day'], as\_index=False).aggregate(mean)

agg\_day.loc[agg\_day['Day']=='0', 'Day'] = '7'

agg\_day['Day'] = agg\_day['Day'].astype(int)

agg\_day.sort(['Day'], inplace=True)

agg\_hour = data.groupby(['Hour'], as\_index=False).aggregate(mean)

lables = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

bar = agg\_day.plot(x='Day',y='ENTRIESn\_hourly', kind='bar', color = "grey")

plt.title('Avg NYC Subway riders by day of week')

plt.xlabel('Day of the week')

plt.ylabel('Avg Subway Entries')

plt.legend().set\_visible(False)

bar.set\_xticklabels(lables)

plot\_hour = ggplot(agg\_hour, aes(x='Hour', y='ENTRIESn\_hourly')) + geom\_bar(stat="bar") + geom\_line() + ggtitle('Avg NYC Subway riders by hour of the day') + xlab('hour of the day') + ylab('Avg Subway Entries')

print plot\_hour

# **Section 4. Conclusion**

4.2 What analyses lead you to this conclusion? You should use results from both your statistical tests and your linear regression to support your analysis.

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

import numpy as np

import pandas

import scipy

import statsmodels.api as sm

"""

In this optional exercise, you should complete the function called

predictions(turnstile\_weather). This function takes in our pandas

turnstile weather dataframe, and returns a set of predicted ridership values,

based on the other information in the dataframe.

In exercise 3.5 we used Gradient Descent in order to compute the coefficients

theta used for the ridership prediction. Here you should attempt to implement

another way of computing the coeffcients theta. You may also try using a reference implementation such as:

http://statsmodels.sourceforge.net/devel/generated/statsmodels.regression.linear\_model.OLS.html

One of the advantages of the statsmodels implementation is that it gives you

easy access to the values of the coefficients theta. This can help you infer relationships

between variables in the dataset.

You may also experiment with polynomial terms as part of the input variables.

The following links might be useful:

http://en.wikipedia.org/wiki/Ordinary\_least\_squares

http://en.wikipedia.org/w/index.php?title=Linear\_least\_squares\_(mathematics)

http://en.wikipedia.org/wiki/Polynomial\_regression

This is your playground. Go wild!

How does your choice of linear regression compare to linear regression

with gradient descent computed in Exercise 3.5?

You can look at the information contained in the turnstile\_weather dataframe below:

https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile\_data\_master\_with\_weather.csv

Note: due to the memory and CPU limitation of our amazon EC2 instance, we will

give you a random subset (~10%) of the data contained in turnstile\_data\_master\_with\_weather.csv

If you receive a "server has encountered an error" message, that means you are hitting

the 30 second limit that's placed on running your program. See if you can optimize your code so it

runs faster.

"""

def predictions(weather\_turnstile):

"""

#

# Your implementation goes here. Feel free to write additional

# helper functions

#

"""

#X = weather\_turnstile[['rain', 'precipi', 'fog', 'meantempi', 'Hour']]

X = weather\_turnstile[['rain']]

units\_info = pandas.get\_dummies(weather\_turnstile['UNIT'], prefix='unit')

X = X.join(units\_info)

Y = weather\_turnstile['ENTRIESn\_hourly']

X = sm.add\_constant(X)

# I wanted to loop through the model few times but CPU time exceeded 30 sec limit

model = sm.OLS(Y,X)

results = model.fit()

prediction = results.predict(X)

print results.params

return prediction