Data Science – Learning Management Questions

**Week Four – Question #1**

. import numpy as np

import pandas

import matplotlib.pyplot as plt

def entries\_histogram(turnstile\_weather):

'''

Before we perform any analysis, it might be useful to take a

look at the data we're hoping to analyze. More specifically, let's

examine the hourly entries in our NYC subway data and determine what

distribution the data follows. This data is stored in a dataframe

called turnstile\_weather under the ['ENTRIESn\_hourly'] column.

Let's plot two histograms on the same axes to show hourly

entries when raining vs. when not raining. Here's an example on how

to plot histograms with pandas and matplotlib:

turnstile\_weather['column\_to\_graph'].hist()

Your histogram may look similar to bar graph in the instructor notes below.

You can read a bit about using matplotlib and pandas to plot histograms here:

http://pandas.pydata.org/pandas-docs/stable/visualization.html#histograms

You can see the information contained within the turnstile weather data here:

https://s3.amazonaws.com/content.udacity-data.com/courses/ud359/turnstile\_data\_master\_with\_weather.csv

'''

plt.figure()

turnstile\_weather['...'] # your code here to plot a historgram for hourly entries when it is raining

turnstile\_weather['...'] # your code here to plot a historgram for hourly entries when it is not raining

return plt

**Week Four – Question #2**

**Please review and respond to the attached file – Question 2**

**Week Four – Question #3**

import numpy as np

import scipy

import scipy.stats

import pandas

def mann\_whitney\_plus\_means(turnstile\_weather):

'''

This function will consume the turnstile\_weather dataframe containing

our final turnstile weather data.

You will want to take the means and run the Mann Whitney U-test on the

ENTRIESn\_hourly column in the turnstile\_weather dataframe.

This function should return:

1) the mean of entries with rain

2) the mean of entries without rain

3) the Mann-Whitney U-statistic and p-value comparing the number of entries

with rain and the number of entries without rain

You should feel free to use scipy's Mann-Whitney implementation, and you

might also find it useful to use numpy's mean function.

Here are the functions' documentation:

http://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html

http://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html

You can look at the final turnstile weather data at the link below:

https://s3.amazonaws.com/content.udacity-data.com/courses/ud359/turnstile\_data\_master\_with\_weather.csv

'''

### YOUR CODE HERE ###

return with\_rain\_mean, without\_rain\_mean, U, p # leave this line for the grader

**Week Four – Question #4**

**Please review and respond to the attached file – Question 4**

**Week Four – Question #5**

import numpy as np

import pandas

from ggplot import \*

"""

In this question, you need to:

1) implement the compute\_cost() and gradient\_descent() procedures

2) Select features (in the predictions procedure) and make predictions.

"""

def normalize\_features(df):

"""

Normalize the features in the data set.

"""

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 value " + \

"in your model.")

df\_normalized = (df - df.mean()) / df.std()

return df\_normalized, mu, sigma

def compute\_cost(features, values, theta):

"""

Compute the cost function given a set of features / values,

and the values for our thetas.

This can be the same code as the compute\_cost function in the lesson #3 exercises,

but feel free to implement your own.

"""

# your code here

return cost

def gradient\_descent(features, values, theta, alpha, num\_iterations):

"""

Perform gradient descent given a data set with an arbitrary number of features.

This can be the same gradient descent code as in the lesson #3 exercises,

but feel free to implement your own.

"""

m = len(values)

cost\_history = []

for i in range(num\_iterations):

# your code here

return theta, pandas.Series(cost\_history)

def predictions(dataframe):

'''

The NYC turnstile data is stored in a pandas dataframe called weather\_turnstile.

Using the information stored in the dataframe, let's predict the ridership of

the NYC subway using linear regression with gradient descent.

You can download the complete turnstile weather dataframe here:

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

Your prediction should have a R^2 value of 0.40 or better.

You need to experiment using various input features contained in the dataframe.

We recommend that you don't use the EXITSn\_hourly feature as an input to the

linear model because we cannot use it as a predictor: we cannot use exits

counts as a way to predict entry counts.

Note: Due to the memory and CPU limitation of our Amazon EC2 instance, we will

give you a random subet (~15%) of the data contained in

turnstile\_data\_master\_with\_weather.csv. You are encouraged to experiment with

this computer on your own computer, locally.

If you'd like to view a plot of your cost history, uncomment the call to

plot\_cost\_history below. The slowdown from plotting is significant, so if you

are timing out, the first thing to do is to comment out the plot command again.

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. Try using a

smaller number for num\_iterations if that's the case.

If you are using your own algorithm/models, see if you can optimize your code so

that it runs faster.

'''

# Select Features (try different features!)

features = dataframe[['rain', 'precipi', 'Hour', 'meantempi']]

# 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):

"""This function is for viewing the plot of your cost history.

You can run it by uncommenting this

plot\_cost\_history(alpha, cost\_history)

call in predictions.

If you want to run this locally, you should print the return value

from this function.

"""

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 )