Note to the Grader:

Dear grader,

I have completed all the problem sets (1-5) but with the recent change in core curriculum Linear regression 3.5 solution has disappeared. I am adding the solution below:

```
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
    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):
    """
    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
        Predicted_values = np.dot(features,theta)
        theta = theta - (alpha/m) * np.dot((Predicted_values - values), features)

        cost = compute_cost(features, values, theta)
    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', 'Hour','meantempi','fog','precipi']]

# 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 = 100 # 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)
    print theta_gradient_descent  ## prints out theta/weights/coefficients
    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 )
```