Code:

C1_ Mann-Whiteney-U Test

```
 def mann_whitney_plus_means(t):
    u=scipy.stats.mannwhitneyu(YesRain['ENTRIESn_hourly'], NoRain['ENTRIESn_hourly'],
   use_continuity=True)[0]
   m_u = len(YesRain['ENTRIESn_hourly'])*len(NoRain['ENTRIESn_hourly'])/2
      sigma_u = np.sqrt ( len(YesRain['ENTRIESn_hourly']) *len(NoRain['ENTRIESn_hourly'])
      *(len(YesRain['ENTRIESn_hourly'])+len(NoRain['ENTRIESn_hourly'])+1)/12)
   z = (u - m_u)/sigma_u
      p=pval = 2*scipy.stats.norm.cdf(z)
   print p
```

C1_Welsh's t-Test

```
def ttest(t):
   ttest_1=scipy.stats.ttest_ind(YesRain['ENTRIESn_hourly'], NoRain['ENTRIESn_hourly'], equal_var=False)
   if (ttest_1[1]<0.05):
   print ttest_1
      return (False, ttest_1)
   else:
      return (True, ttest_1)

ttest(t)
```

C2_OLS

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

print time.ctime()

#read csv file
t = pandas.read_csv('turnstile_weather_v2.csv')
features = t[['pressurei', 'precipi', 'tempi', 'weekday', 'hour']]
#dummy_units = pandas.get_dummies(t['UNIT'], prefix='unit')
#features = features.join(dummy_units)
features = sm.tools.tools.add_constant(features)
values = t['ENTRIESn_hourly']

mod = sm.regression.linear_model.OLS(values,features)
```

```
res = mod.fit()
print res.summary()
C2_Gradient Decent


import numpy as np
import pandas
import scipy
import scipy.stats
import time
import matplotlib.pyplot as plt
import sys
import csv
from ggplot import *

print time.ctime()

#read csv file
t = pandas.read_csv('turnstile_weather_v2.csv')


#gradient decent
def normalize_features(t):
    #Normalize the features in the data set.
    mu = t.mean()
    sigma = t.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.")
    t_normalized = (t - t.mean()) / t.std()

    return t_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 = []

    for i in range(num_iterations):
```

```
        predicted_values=np.dot(features,theta)
        theta= theta- alpha/m *np.dot((predicted_values-values), features)
        cost=compute_cost(features, values, theta)
        cost_history.append(cost)
    return theta, pandas.Series(cost_history)

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)

def predictions(t):

    # Select Features (try different features!)
    features = t[['rain', 'meantempi', 'weekday', 'hour']]

    # Add UNIT to features using dummy variables
    dummy_units = pandas.get_dummies(t['UNIT'], prefix='unit')
    features = features.join(dummy_units)

    # Values
    values = t['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 = plot_cost_history(alpha, cost_history)
```

```
    predictions = np.dot(features_array, theta_gradient_descent)
    print gradient_descent(features_array, values_array, theta_gradient_descent, alpha, num_iterations)

    #print features_array


predictions(t)
C3_ Histogram:
YesRain =  t[t['rain']==1]
NoRain =  t[t['rain']==0]
with_rain = YesRain['ENTRIESn_hourly']
without_rain = NoRain['ENTRIESn_hourly']
plt.hist(with_rain.values, bins=20, histtype='step', color='b', label='Yes Rain')
plt.hist(without_rain.values, bins=20, histtype='step', color='r',label='No Rain')
plt.title("Histogram of ENTRIESn_hourly")
plt.xlabel("hourly entries")
plt.ylabel("Frequency")
plt.legend()
plt.show()

C3_Scatter Plot:
busy=t[t['UNIT']=='R084']
plt.scatter(busy['hour'], busy['ENTRIESn_hourly'])
plt.axhline(1537)
plt.title("Entries by hour at the busiest station (R084)")
plt.xlabel("Hour")
plt.ylabel("Entries hourly")
plt.legend()
plt.show()
```