Analyzing the NYC Subway Dataset

Yigal Weinstein

December 4, 2015

Introduction

This project attempts to follow the outline provided in [0] and the project rubrick set forth in [0]. It is divided into two parts. The first part, below, are answers to the short questions in [0] while the second part contains the code and and notes regarding problem set 2,3, and 4 of the Intro to Data Science course [0].

Part I
Analyzing the NYC Subway Dataset

Statistical Test

Question 1.1
Which statistical test did you use to analyze the NYC subway data? Did you use a one-tail or a two-tail P value? What is the null hypothesis? What is your p-critical value?

The distribution of `ENTRIESn_hourly` isn't normal if we look at the entirety of it, see the answer to the next question for my rational. In this example I will compare the distribution of `ENTRIESn_hourly` for rainy and non-rainy days where the overall distribution is considered nonparametric. The null hypothesis, H_0 is that the two samples come from the same population. While the alternative hypothesis, H_A is that the sample sets come from two different populations. For this test I will use a p-critical value, p, of 0.05:

$$p = 0.05 \tag{1}$$

While the Central Limit Theorem on a data set of this size is potentially a reasonable choice I have chosen two nonparametric continuous tests to test H_0, the two-sample Kolmogorov-Smirnov two sided test [0] and the Mann-Whitney U one sided test [0].

Question 1.2
Why is this statistical test applicable to the dataset? In particular, consider the assumptions that the test is making about the distribution of ridership in the two samples.

It is often useful to determine how well a sample conforms to a normal distribution to determine the appropriate statistical tests to be applied to it. One may argue that due to the large sample size more than

42,000 lines of raw data in this case the central limit theorem (CLT)) is applicatable to the distribution of means however we often care about other characteristics of the sample set that cannot be determined using the assumption of normality. As an initial visual test for normality a histogram can be constructed, where the variable of interest is `ENTRIESn_hourly`, please see **??**

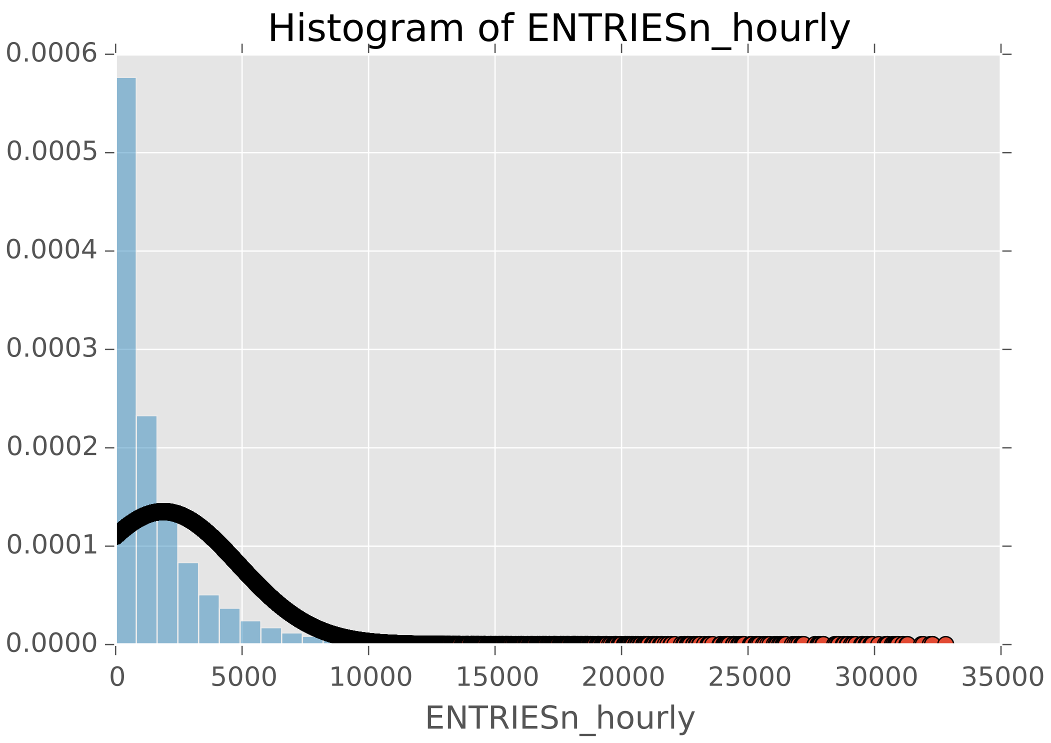

Figure 1: The histogram a simple and visual test for normality. In this case it indicates that the sample is certainly not normal. Above the histogram is normalized comparing it to a normal distribution with the same mean and standard deviation. The following were used to generate this plot [0] and [0].

The histogram indicates that the data is certainly not normal, and it is heavily skewed to the left. The lack of normality can also be seen by performing a statistical normality tests. For this I've used both the Shapiro-Wilk [0] and D'Agostino-Pearson omnibus [0] tests both indicating that the distribution is in no way normal:

```python
import scipy
p_shapiro = scipy.stats.shapiro(df.ENTRIESn_hourly);
p_dandp = scipy.stats.mstats.normaltest(df.ENTRIESn_hourly);
print(p_shapiro,p_dandp)
(0.5943876504898071, 0.0) (36050.081245042486, 0.0)
```

The second value in each case 0.0 represent the approximate p-value, i.e. 0. As these are goodness of fit tests the notion of one vs two tails isn't something to consider, though the D'Agostino-Pearson is two-sided Chi-square test. Even though the distribution of the total sample isn't normal we could use the CLT to make some educated gueses in regards to comparing the similarity of our two samples, however if one looks at

?? the jarring differences between the approximate normal and the actual distribution becomes very clear. While normal tests are incredibly robust I will use the nonparametric tests as specified in Question **??** for this analysis.

Due to the nature of the questions surrounding the analysis, not in the hypothesis testing but in further questions the one-sided Kolmogorov-Smirnov test is of greater overall use than the Mann-Whitney U test because it allows us to make stronger claims of the nature of the two distributions, that one is skewed to the left or right of each other if the overall shape of the distributions are similar.

Question 1.3
What results did you get from this statistical test? These should include the following numerical values: p-values, as well as the means for each of the two samples under test.

The results of the two nonparametric tests were as follows:

```
rain = df[df.rain == 0]['ENTRIESn_hourly']
norain = df[df.rain == 1]['ENTRIESn_hourly']
p_KS = 'The p-value using the one sided Kolmogorov-Smirnov test is:'
p_MW = 'The p-value using the one sided Mann-Whitney test is:'
mean_rainy = 'The mean ENTRIESn_hourly for rainy days is:'
mean_norain = 'The mean ENTRIESn_hourly for non-rain days is:'
print(p_KS, scipy.stats.ks_2samp(rainy,norain)[1], \
      '\n' + p_MW, scipy.stats.mannwhitneyu(rainy,norain)[1], \
      '\n' + mean_rainy, rainy.mean(), \
      '\n' + mean_norain, norain.mean())
```

```
The p-value using the one sided Kolmogorov-Smirnov test is: 1.42662617869e-05
The p-value using the one sided Mann-Whitney test is: 2.74106957124e-06
The mean ENTRIESn_hourly for rainy days is: 2028.19603547
The mean ENTRIESn_hourly for non-rain days is: 1845.53943866
```

As a two sided test is being performed, that is a test of whether the data comes from the same distribution, and not that one is greater or less than another the Mann-Whitney, a one sided test, should be multiplied by two to get the proper two sided p-value.

Question 1.4
What is the significance and interpretation of these results?

Both tests indicate with p-values less than one in a ten-thousand that H_0 should be rejected in favor of H_A using the criteria in **??** of $p = 0.05$, and the Mann-Whitney due to its sidedness shows us that we can expect the rainy distribution to be shifted to the left and have a larger mean if the distributions are roughly the same. Calculating the means independently confirmed that the rainy days had a significantly larger ridership than non-rainy days at that, $2,028.19603547$ to $1,845.53943866$ on average.

Linear Regression

Question 2.1
What approach did you use to compute the coefficients theta and produce prediction for `ENTRIESn_hourly` in your regression model:

- OLS using Statsmodels or Scikit Learn
- Gradient descent using Scikit Learn
- Or something different?

The method I have used in this example is Gradient of descent using the Sckikit Learn implementation [0]. It is far more robust than OLS and I'm not familiar enough with other methods yet to implement them properly.

Question 2.2
What features (input variables) did you use in your model? Did you use any dummy variables as part of your features?

The feature list is as follows,

```
feature_list = [
                'hour',
                'day_week',
                'weekday',
                'latitude',
                'longitude',
                'fog',
                'precipi',
                'pressurei',
                'rain',
                'tempi',
                'wspdi',
                'meanprecipi',
                'meanpressurei',
                'meantempi',
                'meanwspdi',
                'weather_lat',
                'weather_lon'
                ]
features = dataframe[feature_list]
dummy_units = pandas.get_dummies(dataframe['UNIT'], prefix='unit')
features = features.join(dummy_units)
dummy_units = pandas.get_dummies(dataframe['conds'], prefix='conds')
features = features.join(dummy_units)
```

That is the feature list is as follows, 'day_week, fog, hour, latitude, longitude, meantempi, meanwspdi, precipi rain, tempi, wspdi', with the addition of the two dummy variables 'UNIT', and 'conds'.

Question 2.3
Why did you select these features in your model? We are looking for specific reasons that lead you to believe that the selected features will contribute to the predictive power of your model.

- Your reasons might be based on intuition. For example, response for fog might be: “I decided to use fog because I thought that when it is very foggy outside people might decide to use the subway more often.”

- Your reasons might also be based on data exploration and experimentation, for example: “I used feature X because as soon as I included it in my model, it drastically improved my R2 value.”

Initially the entire decision was based off of the value of R^2 value produced as compared to a more sparse feature set’s R^2 value. If R^2 was greater upon adding the feature I’d keep the feature, and if it was lower I’d remove it. I changed to partially automating it by running through all of the possible subsets that could be made and selecting the largest R^2 value. Then the feature list obtained like its parent, and so on until a maximum value was found or the number of features fell to seven, a semi-random cut off point. I’m not certain if I gained all that much but it was a good exercise. Here is the code, I’ve used mostly the exercise code removing the comments and adding a few functions for the automation,

```python
import numpy as np
import pandas as pd
import itertools
from sklearn.linear_model import SGDRegressor

df = pd.read_csv(data_file)

def normalize_features(features):
    '''
    Returns the means and standard deviations of the given features, along with
    a normalized feature matrix.
    '''
    means = np.mean(features, axis=0)
    std_devs = np.std(features, axis=0)
    normalized_features = (features - means) / std_devs
    return means, std_devs, normalized_features
```

```python
def recover_params(means, std_devs, norm_intercept, norm_params):
    '''
    Recovers the weights for a linear model given parameters that were fitted
    using normalized features. Takes the means and standard deviations of the
    original features, along with the intercept and parameters computed using
    the normalized features, and returns the intercept and parameters that
    correspond to the original features.
    '''
    intercept = norm_intercept - np.sum(means * norm_params / std_devs)
    params = norm_params / std_devs
    return intercept, params

def linear_regression(features, values):
    """
    Perform linear regression given a data set with an arbitrary number of
    features.
    """

    clf = SGDRegressor(alpha=0.028,n_iter=10)
    clf.fit(features, values)
    intercept = clf.intercept_[0]
    params = clf.coef_

    return intercept, params
```

```python
def predictions(df, feature_list):
    print(feature_list)
    features = df[feature_list]
    dummy_units = pd.get_dummies(df['UNIT'], prefix='unit')
    features = features.join(dummy_units)
    dummy_units = pd.get_dummies(df['conds'], prefix='conds')
    features = features.join(dummy_units)

    # Values
    values = df['ENTRIESn_hourly']

    # Get numpy arrays
    features_array = features.values
    values_array = values.values

    means, std_devs, normalized_features_array = \
        normalize_features(features_array)

    # Perform gradient descent
    norm_intercept, norm_params = \
        linear_regression(normalized_features_array, values_array)

    intercept, params = \
        recover_params(means, std_devs, norm_intercept, norm_params)

    predictions = intercept + np.dot(features_array, params)

    return predictions
```

```python
def compute_r_squared(data, predictions):
    ave = np.mean(data)
    SStot = np.dot(data - ave,data - ave)
    SSres = np.dot(data - predictions, data - predictions)
    r_squared = 1 - SSres/SStot

    return r_squared
```

and here are the functions for iterating through the sets of features to find R^2:

```python
def r_square_max(parent_feature_list):
    combs = []
    ind = len(parent_feature_list)

    for i in range(ind - 2,ind - 1):
        for x in itertools.combinations(parent_feature_list, i):
            #print(x)
            combs.append(list(x))
    print(len(combs))

    r_squared = pd.DataFrame(columns= ['Features', 'r_squared'])
    ' '.join(parent_feature_list)

    j = 0
    for i in combs:
        r_squared.loc[j] = [
            ','.join(i),
            compute_r_squared(df.ENTRIESn_hourly, predictions(df,i))
                               ]
        j += 1
    return r_squared

'''
define new parent_feature_list
r_squared is a list containing feature lists with an associated R^2 value
rsqd_max_val is the previous max value for R^2
min_len is the minimum number of feature items we are willing to allow
'''
def r_max_iter(r_squared, rsqd_max_val, min_len):
    r_sorted = r_squared.sort_values('r_squared', ascending=False)
    r_sorted = r_sorted.reset_index(drop=True)
    if r_sorted.iloc[0,1] > rsqd_max_val:
        parent_feature_list = r_sorted.iloc[0,0].split(sep=',')
        if len(r_sorted.iloc[0,0].split(sep=',')) > min_len:
            return [ parent_feature_list, r_sorted.iloc[0,1] ]
        else:
            print("Feature list is smaller than", min_len)
            return [ parent_feature_list, rsqd_max_val ]
```

Then I just ran each iteration manually, first creating the master feature list,

```python
pattern = 'E*Sn|E*Sn_hourly|Unnamed|datetime|TIMEn|DATEn|UNIT|conds|station'
feature_list = df.columns[df.columns.str.contains(pattern) == False].tolist()
```

where I'm adding by hand the two dummy variables 'UNIT' and 'conds' and then,

```
min_len = 7
rs_l1 = r_square_max(feature_list)
val = r_max_iter(rs_l1,.4,7)
fl2 = val[0]
rs_max = val[1]
```

and then again, using the R^2_{max} value from the previous iteration:

```
min_len = 7
rs_l2 = r_square_max(fl2)
val = r_max_iter(rs_l2,rs_max,7)
fl2 = val[0]
rs_max = val[1]
```

It appears that using almost all of the features, except for 'station' which as a dummy variable I had to test it manually, appears to produce the highest R^2 values. The last caveat is that I didn't use the 'datetime' variable although I suspect converting it to epoch time, i.e. seconds, would have provided an easy way of utilizing it in the interpolation.

Question 2.4

What are the parameters (also known as "coefficients" or "weights") of the non-dummy features in your linear regression model?

As the dummy variables are added at the end the parameters for the non-dummy variables can be obtained by determining the number of parameters without the dummy variable, and then extracting that number of parameters, starting with the indices at 1 as 0 contains the 'y-intercept' or θ_0 value which isn't should remain 1. In this case it's 17 which ammounts to the following list of 17 features:

```
[
  'hour',
  'day_week',
  'weekday',
  'latitude',
  'longitude',
  'fog',
  'precipi',
  'pressurei',
  'rain',
  'tempi',
  'wspdi',
  'meanprecipi',
  'meanpressurei',
  'meantempi',
  'meanwspdi',
  'weather_lat',
  'weather_lon'
]
```

The values for $\{\theta\}$ are:

```
[   43.15215935  1095.13586939  2191.2348662  -1867.70463212  -114.92320615
 -2764.85499069  -441.17345176  -107.81100064    39.01877468    12.56298462
  8871.15668632  -126.41631315   -55.66694997   -25.83967506  1337.26604487
 -2320.43792845 -1339.93169656]
```

Question 2.5
What is your model's R^2 (coefficients of determination) value?

I couldn't get a better value than,

$$R^2 = 0.49025647407159878 \ \alpha = 0.015 \tag{2}$$

at 3000 iterations. This is, however, not much better than a much less expensive 100 iterations

$$R^2 = 0.48342788955310945 \ \alpha = 0.026 \tag{3}$$

Question 2.6
What does this R^2 value mean for the goodness of fit for your regression model? Do you think this linear model to predict ridership is appropriate for this dataset, given this R^2 value?

The R^2 value of 0.49 isn't a particularly high value although due to the complexity and the human element it isn't a really low value. Using a pragmatic definition of R^2 it means that the model and features being used explains nearly but not quite half of the observed assumed dependent phenomenon, that is the values of `ENTRIESn_hourly`.

The value of R^2 will never be able to determine the adequacy of the model [0]. By this it is meant that R^2 is a good but incomplete estimator. By itself it can indicate that a model under investigation is likely good or bad but a high R^2 value can still produce many inaccurate predictions, and a low R^2 value can still be extremely useful in making predictions.

So loosely The value of $R^2 \approx 0.5$ means it is adequate in predicting trends but not completely precise enough for making decisions based off of precise insight. That is given that we know the current state of all of the features we could not use this to determine the value for `ENTRIESn_hourly`, however for general trends we likely would be able to do a reasonable job depending, again on the needs. Secondly the prediction capabilities of the model are only as good within the parameters already observed. That is we don't have justification to extend the predictive model out to parameters never recorded, higher winds than expected, etc..

Visualization

Please include two visualizations that show the relationships between two or more variables in the NYC subway data. Remember to add appropriate titles and axes labels to your plots. Also, please add a short description below each figure commenting on the key insights depicted in the figure.

Question 3.1

- One visualization should contain two histograms: one of `ENTRIESn_hourly` for rainy days and one of `ENTRIESn_hourly` for non-rainy days.

- If you decide to use to two separate plots for the two histograms, please ensure that the x-axis limits for both of the plots are identical. It is much easier to compare the two in that case.
- For the histograms, you should have intervals representing the volume of ridership (value of `ENTRIESn_hourly`) on the x-axis and the frequency of occurrence on the y-axis. For example, each interval (along the x-axis), the height of the bar for this interval will represent the number of records (rows in our data) that have `ENTRIESn_hourly` that falls in this interval.
- Remember to increase the number of bins in the histogram (by having larger number of bars). The default bin width is not sufficient to capture the variability in the two samples.

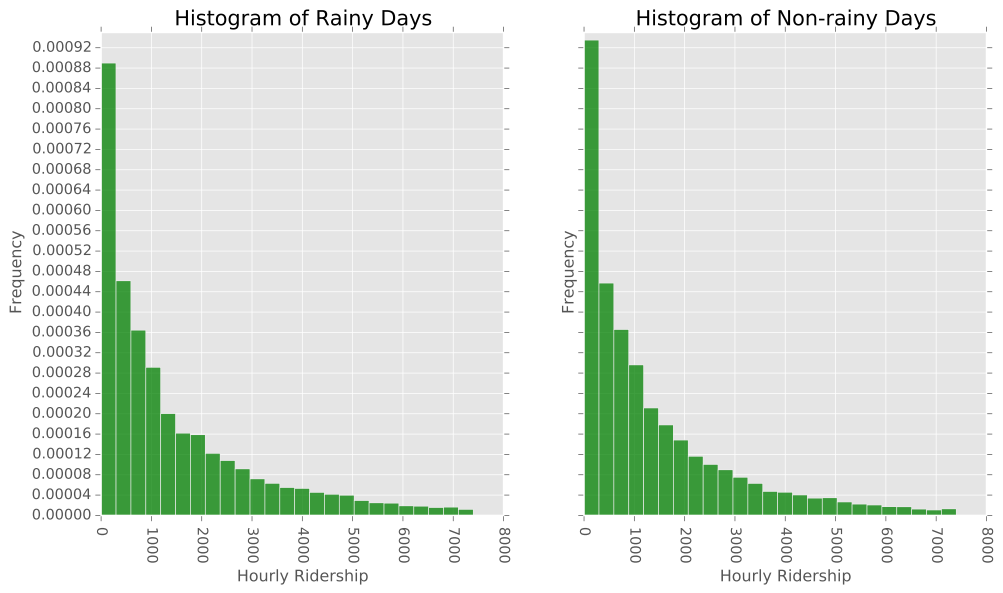

Figure 2: The above histograms have been normed which highlights the similarity between the two sets of data. That is rainy and non-rainy days are very similarly distributed, although there are considerably fewer rainy days so looking at the raw data hisograms this is much less obvious. There were some outliers on both graphs. To show the general trend of the graphs I chose to limit the points used to the 95% cutoff, using the larger of the two cutoff points which in this case was for non-rainy days at roughly $7,400$ as the x-max value, instead of the maximum value of both sets which is more than four times that value, roughly $33,000$ and $32,000$ for rainy and non-rainy respectfully.

Question 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. Some suggestions are:

- Ridership by time-of-day

- Ridership by day-of-week

Please look to **??** for the second visualization for this section. I used the in built PANDAS visualization functions, which in turn is utilizing Matplotlib, see [0] which goes into detail how quite aesthetically pleasing visualizations can be constructed using a GGplot like environment implemented in Matplotlib, as well as providing a reference for creating a stacked bar graph from a data frame. I used the following other references to construct this visualization [0], [0], [0], [0]

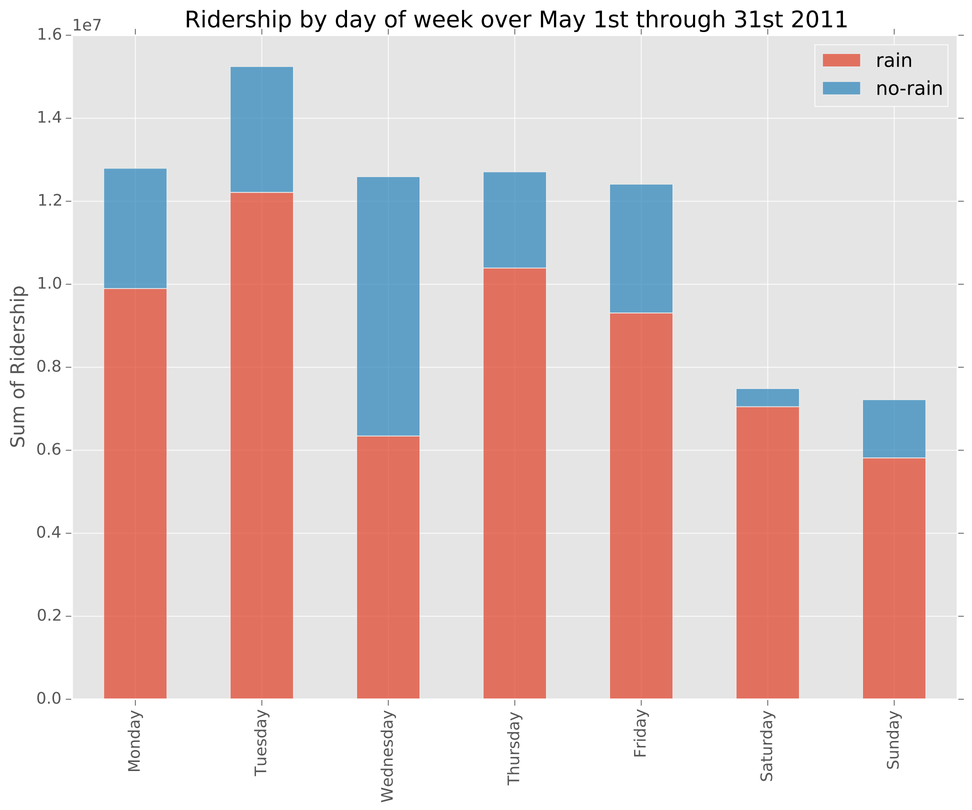

Figure 3: The above bar graph shows the sum of the ridership divided into the days of week, and each day's values has been partitioned into being either rainy or non-rainy days. There is a drastic dip in subway usage during the weekends, more rain fell on Wednesday by far then any other day, and there are significantly more passengers that ride the subway on Tuesday than any other day of the week.

Conclusion

Please address the following questions in detail. Your answers should be 1-2 paragraphs long.

Question 4.1
From your analysis and interpretation of the data, do more people ride the NYC subway when it is raining or when it is not raining?

The question in a sense needs to be refined. The average `ENTRIESn_hourly` is one method for measuring if more riders ride when it rains. If this is the metric used for answering this question then indeed the average, $\bar{x}_{rain} = 2028.19603547$, is greater than the average hourly ridership when it's not raining, $\bar{x}_{norain} = 1845.53943866$. This holds true even if the largest, 2% on the right hand side, outliers are removed, with a mean of 1701.10986905 and 1551.04237393 for the remaining rainy and non-rainy data. Not just the means alone provide support for this analysis, but indeed the one-sided Kolmogorov-Smirnov analyzed in **??** combined with the distributions having a very similar form as shown in the descriptive histogram in **??**. As the distributions are similarly shaped but the rainy days has a greater mean, as well as a small but positive value for the Kolmogorov-Smirnov D value which means the distributions are potentially from a similar distribution implies that using the average `ENTRIESn_hourly` as the metric it must be concluded that more people ride the NYC subway when it's raining.

Alternatively a metric that is equally reasonable to use to answer this question would be the overall amount of ridership for rainy and non-rainy days. As there are far fewer rainy days in May and the two distributions despite the slightly higher mean for rainy days are in actuality very close, please see the **??** for confirmation of this assertion, there are simply more riders during non-rainy days, as during rainy, $19,440,259$ and $61,020,916$ for rainy and non-rainy days respectfully, or more than three times the amount of riders for non-rainy days as rainy. Please see **??** as confirmation to this, note how the red part of the bar the non-rainy portion is except for Wednesday much greater than the rainy, or blue section. So that using this metric more people ride the subway on non-rainy days.

Question 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.

From the two metrics above it really matters what question is actually being asked. If hourly ridership, that is the rate of ridership, is what we are defining as more people, which for many problems is a good definition then more people ride when it's rainy. If on the other hand we care about the total number of riders longitudinally over the month of May then by far, by over three times the riders, non-rainy days clearly is greater than rainy days.

If nothing else what can be concluded from the linear regression is that we don't know enough yet to make very accurate predictions. As the highest R^2 value obtained was 0.49, see **??**, this implies that the best predictions for `ENTRIESn_hourly` using a linear model is globally useful but not a particularly strong statistic on its own. That being the case what is of interest to answering this question is that the θ_{rain} weight isn't particularly high, in fact it's in terms of absolute value the penultimate weight, at 39.0, where the maximum value was 8871.15668632 for Wind speed which seems to be a rather odd variable to have such a profound effect on `ENTRIESn_hourly`, note that normalizing the θ values which for this comparison is a more appropriate comparison doesn't change things very much. As an indication of how relatively little rain appears to play a role in `ENTRIESn_hourly` we can look to **??**. Note that while Wednesday has more rain by a factor of more than two than any other day, and for many days many times over this amount, the overall `ENTRIESn_hourly` for Wednesday is almost the same as for Monday, Thursday, and Friday, all of which also have varying amounts of rain, although not as apparent as Wednesday.

Reflection

Please address the following questions in detail. Your answers should be 1-2 paragraphs long.

Question 5.1
Please discuss potential shortcomings of the methods of your analysis, including:

- Dataset
- Analysis, such as the linear regression model or statistical test.

As discussed above the R^2 value isn't very high to make general predictions surrounding NYC ridership. Either there are more variables that need to be taken into account or the real NYC subway ridership must be modeled using nonlinear regression, which is beyond the scope of this project. One must keep in mind that the data analyzed was just one month, May, out of one year, 2011. In order to make accurate predictions either about the rest of the year or about following years simply more data from different months and years would be necessary. The Kolmogrov-Smirnov test is useful in describing the differences between rainy and non-rainy days however the D value was small ≈ 0.02 which is why another nonparametric test was run with it the Mann-Whitney U test, because otherwise the H_0 shouldn't not be discarded, see [0].

Part II

Code and Notes to Problem Sets 2,3 and 4

What follows are my solutions to problem sets 2,3, and 4 of [0].

Problem Set 2: Wrangling Subway Data

Problem 2.1
Number of Rainy Days

The SQL query I used was,

```sql
SELECT COUNT(rain) FROM weather_data WHERE rain > 0.0
```

Problem 2.2
Temp on Foggy and Nonfoggy Days1 - Number of Rainy

With the help of [0] and [0] the following query provides the desired result:

```sql
SELECT fog, MAX(maxtempi) FROM weather_data GROUP BY fog
```

Problem 2.3
Mean Temp on Weekends1 - Number of Rainy Days

This one was extremely difficult as it required piecing together the hint about using the `CAST()` function in references to data in more than one column as well as using the proper syntax to convert the `date` column to the day of the week to allow filtering on Saturday and Sunday. Another challenge was realizing that `pandasql` isn't using SQL per se but and in this particular case it matters because `STRFTIME()` isn't something one will use in `MySQL`, but for this purpose it appears like the only function to call. [0] So continuing the research I found that SQLite stores stores in ISO format as a string. [0], [0]

```
SELECT AVG(CAST(meantempi AS INTEGER)) FROM weather_data WHERE
    CAST(STRFTIME('%w',date) as integer) = 6 OR
    CAST(STRFTIME('%w',date) as integer) = 0;
```

Problem 2.4
Mean Temp on Rainy Days1 - Number of Rainy Days

```
SELECT AVG(CAST(mintempi AS INTEGER)) FROM weather_data WHERE
    CAST(rain AS INTEGER) == 1 and
    CAST(mintempi AS INTEGER) > 55;
```

Problem 2.5
Fixing Turnstile Data

So using the tutorial on utilizing CSV data files with Python [0] I used the following code to the solve this problem, note the field descriptions are available for the original data see [0]

```
import csv

def fix_turnstile_data(filenames):
    for name in filenames:
            f_in = open(name,'r')
            sfile = name.split('/') # sfile = split file
            sfile[-1] = "updated_" + sfile[-1]
            output = '/'.join(sfile)
            f_out = open(output, 'w')

            reader_in = csv.reader(f_in, delimiter=',')
            writer_out = csv.writer(f_out,delimiter=',')
            for line in reader_in:
                i, outline = 1, []
                while i * 5 + 2 <= len(line):
                    outline = line[0:3] + line[i * 5 - 2:i * 5 + 3]
                    writer_out.writerow(outline)
                    i+=1
            f_in.close()
            f_out.close()
```

Problem 2.6
Combining Turnstile Data

```python
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')
            for line in f_in:
                master_file.write( line )
        f_in.close()
        master_file.close()
```

I used the following Stackoverflow thread to ensure I wasn't missing something [0].

PROBLEM 2.7
Filtering Irregular Data

```python
import pandas as pd
import numpy as np

def filter_by_regular(filename):
    turnstile_data = pd.read_csv(filename)
    return turnstile_data[turnstile_data.DESCn == 'REGULAR']
```

Note what is really frustrating is that if you assume there's no header and that you're expected to make one the test will fail.

```python
import pandas as pd
import numpy as np

def filter_by_regular(filename):
    Names = ['C/A','UNIT','SCP','DATEn','TIMEn','DESCn','ENTRIESn','EXITSn']
    turnstile_data = pd.read_csv(filename, names=Names, header=None)
    print(turnstile_data.dtypes)
    return turnstile_data[turnstile_data.DESCn == 'REGULAR']
```

It fails in such a way that the imported data are all treated as 'objects' i.e. strings,

```
C/A         object
UNIT        object
SCP         object
DATEn       object
TIMEn       object
DESCn       object
ENTRIESn    object
EXITSn      object
dtype: object
```

and the output data, note which is the same and incorrect with the code that actually does work, because the entries for the last two columns for the output are always, that is no matter if the 'correct' answer is given or the inccorect is shown incorrectly with padded zeros, as example,

```
your output:
        C/A  UNIT       SCP      DATEn      TIMEn     DESCn    ENTRIESn                           EXITSn
```

```
0     A002  R051  02-00-00  05-01-11  00:00:00  REGULAR  003144312                 001088151
1     A002  R051  02-00-00  05-01-11  04:00:00  REGULAR  003144335               001088159
```

It makes sense if the header in the CSV files is taken into data frame that columns normally of `int` would be forced to be treated as an `object`, however the output, shown above doesn't indicate that the first row should be treated as a header, just the opposite in fact, and I have no explanation for all of the extra space in the last two columns.

The only test that shows this to be the case is that `print(turnstile_data.dtypes)` shows up properly as,

```
C/A         object
UNIT        object
SCP         object
DATEn       object
TIMEn       object
DESCn       object
ENTRIESn     int64
EXITSn       int64
dtype: object
```

If the headers are assumed to be contained in the data set. This took me more than a day to figure out.

Problem 2.8
Get Hourly Entries

```python
import pandas

def get_hourly_entries(df):
    df['ENTRIESn_hourly'] = df.ENTRIESn - df.ENTRIESn.shift(1)
    df.ENTRIESn_hourly = df.ENTRIESn_hourly.fillna(1)
    return df
```

Problem 2.9
Get Hourly Exits

Change ENTRIES to EXITS and 1 to 0 and presto chango:

```python
import pandas

def get_hourly_exits(df):
    df['EXITSn_hourly'] = df.EXITSn - df.EXITSn.shift(1)
    df.EXITSn_hourly = df.EXITSn_hourly.fillna(0)
    return df
```

This seems much easier than the previous problems:

Problem 2.10
Time to HourGet Hourly Exits

```python
import pandas as pd

def time_to_hour(time):
    time = pd.to_datetime(time)
    hour = time.hour
    return hour
```

Datetimes are not trivial but this was an easy exercise showing some of the strengths of PANDAS, Numpy in regards to dealing with them.

Problem 2.11
Reformat Subway Dates

Using the datetime module the functions strftime() and strptime() are used to the solve this problem, [0]:

```python
import datetime as dt

def reformat_subway_dates(date):
    date_string = dt.datetime.strptime(date, '%m-%d-%y')
    date_formatted = dt.datetime.strftime(date_string, '%Y-%m-%d')
    return date_formatted
```

Problem Set 3: Analyzing Subway Data

Problem 3.1
Exploratory Data Analysis

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

def entries_histogram(turnstile_weather):
    plt.figure()
    turnstile_weather[turnstile_weather.rain == 1].ENTRIESn_hourly.
        plot(kind='hist', alpha=0.5)
    turnstile_weather[turnstile_weather.rain == 0].ENTRIESn_hourly.
        plot(kind='hist', alpha=0.5)
    return plt
```

Problem 3.2
Welch's t-Test

The data set isn't normally distributed and is tremendously skewed to the left. As such Welch's t-test wouldn't be a useful test for determining relevant statistical inforation from the data. While the sample size is greater than what Scipy can handle properly $N < 5000$ we can see that it's very unlikely, using the Shapiro-Wilk test, that the data was drawn from normal distributions:

```python
import scipy.stats as stats
print(stats.shapiro(turnstile_weather[turnstile_weather.rain 1].ENTRIESn_hourly))
print(stats.shapiro(turnstile_weather[turnstile_weather.rain 0].ENTRIESn_hourly))
(0.47159135341644287, 0.0)
(0.47661787271499634, 0.0)
```

Problem 3.3
Mann-Whitney U-Test

I used the following somewhat verbose code for this problem:

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

def mann_whitney_plus_means(turnstile_weather):
    with_rain = turnstile_weather[turnstile_weather.rain == 1].ENTRIESn_hourly
    with_rain_mean = with_rain.mean()
    without_rain = turnstile_weather[turnstile_weather.rain == 0].ENTRIESn_hourly
    without_rain_mean = without_rain.mean()

    stats = scipy.stats.mannwhitneyu(with_rain, without_rain)
    U = stats[0]
    p = stats[1]

    return with_rain_mean, without_rain_mean, U, p # leave this line for the grader
```

The results found were,

$$U, p = 1924409167.0, 0.024999912793489721 \tag{4}$$

where the averages for `ENTRIESn_hourly` were found to be

$$\bar{x}_{rainy} = 1105.4463767458733 \tag{5}$$
$$\bar{x}_{dry} = 1090.278780151855 \tag{6}$$

for rainy and non-rainy days respectively.

As $p < 0.05$ it is quite likely that the two distributions, although having fairly similar means are in fact from two distinct populations, that is they are statistically different.

Problem 3.4
Plotting Residuals

Using [0] to understand the ideas behind plotting residuals. From the errors generated it appears there are some severe outliers, meaning for some parts of the data the model did badly, even atrociously.

```python
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
```

Problem 3.5
Compute R^2

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

def compute_r_squared(data, predictions):
    ave = np.mean(data)
    SStot = np.dot(data - ave,data - ave)
    SSres = np.dot(data - predictions, data - predictions)
    r_squared = 1 - SSres/SStot
    #print(type(features))

    return r_squared
```

Instead of using `np.mean` or `np.sum` it was simply easier to use the dot product `np.dot` to produce the desired result, [0].

Problem 3.6
Gradient Descent (optional)

So using the official documentation on utilizing `sklearn.linear_model.SGDRegressor` [0]. This was not a brilliantly done job here is my code,

```
def linear_regression(features, values):
    """
    Perform linear regression given a data set with an arbitrary number of features.
    """

    clf = SGDRegressor(alpha=0.025,n_iter=20)
    clf.fit(features, values)
    intercept = clf.intercept_[0]
    params = clf.coef_

    return intercept, params
```

Problem Set 4: Visualizing Subway Data

Problem 4.1
Visualization I

I was having difficulty understanding how to use use multiple data frames and this is where I found a solution [0]. I also used the official documentation/examples to produce the graph below, [0] note there is much improvement I can see doing but it's something but I learned quite a lot doing it. I'm not impressed with the Python implementation of ggplot and can see that the original is a far more refined tool in R:

```
from pandas import *
from ggplot import *

def plot_weather_data(turnstile_weather):

    hour_mean = turnstile_weather.groupby('Hour', as_index=False).mean()
    hour_median = turnstile_weather.groupby('Hour', as_index=False).median()
    title = "Entries per hour from the 1st to the 30th of May 2011 "
    xlable = "Hour"
    ylable = "Entries per hour"

    plot = ggplot(aes(x='Hour', y='ENTRIESn_hourly'), data=turnstile_weather) + \
    geom_point(aes(x='Hour', y= 'ENTRIESn_hourly', size=100, alpha=0.1), color='steelblue') + \
    geom_point(color='yellow', data=hour_mean) + \
    geom_point(color='black', data=hour_median) + \
    scale_color_gradient2() + \
    xlab(xlable) + ylab(ylable) + ggtitle(title) + \
    ylim(0,4000) + xlim(-.3,24)

    return plot
```

Where I was unable to associate a legend but the output is as follows:

Problem 4.2
Visualization II

I decided to just do a variation on the same theme as the previous visualization. This time, however, I've taken the top 6 stations in respect to the total number of colors needed for representation and plotted their data with respect to the hours. What is of note is that it appears many stations are often times nearly unused for several hours

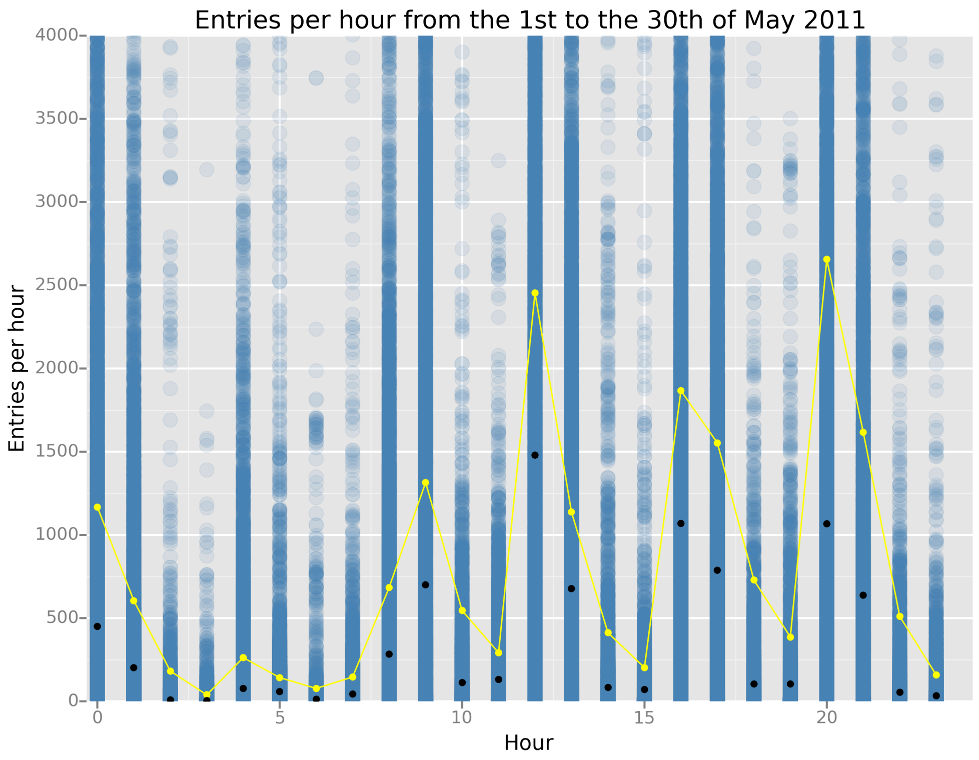

Figure 4: Each semi-transparent dot represents a point of data graphed for each hour in respect to the number of entries per hour rate. All outliers greater than 4, 000 were left out of the plot in order to provide a more meaningful view of the population. The average and median values are represented by the yellow and black dots respectively, and use all values including above the 4, 000 level. Note how even here most of the average values are much below 500 and that there are at least two distinct peaks at 12PM and 8PM.

```python
from pandas import *
from ggplot import *

def plot_weather_data(turnstile_weather):
    # Get sorted list of stations by the sum of ENTRIESn:
    by_unit = turnstile_weather.loc[:,['ENTRIESn_hourly' , 'UNIT']]
    by_unit = by_unit.groupby('UNIT').sum()
    by_unit = by_unit.sort_values('ENTRIESn_hourly',ascending = False)

    # Filter stations with at least 1/3 of the largest data point:
    by_unit_cut = by_unit[by_unit > by_unit.iloc[0]/3]
    # PANDAS is nice enough to keep the original df structure so drop NaN
    # values:
    by_unit_cut = by_unit_cut.dropna()

    # reduce total number of stations to 6,
    # too messy to see much of anything without minimizing this number
    if by_unit_cut.count()[0] > 6:
        by_unit_cut = by_unit_cut[0:6]

    # Use the generated list to filter out all of the data from the origial
    # data frame:
    by_unit_data = turnstile_weather.loc[turnstile_weather['UNIT']
    by_unit_data = by_unit_data.isin(by_unit_cut.index.tolist())]

    # Now generate a similar plot as above using a different colors for the 11
    # stations considered:

    by_unit_data = by_unit_data[['UNIT','ENTRIESn_hourly','Hour']]
    by_unit_data = by_unit_data.sort_values('Hour')
    by_unit_data = by_unit_data.groupby(['UNIT','Hour'], \
        as_index=False).sum()

    title = \
    "Entries per hour from the top 6 Stations during May 2011 1st through 30th"
    xlable = "Hour"
    ylable = "Entries per hour"

    plot = ggplot(by_unit_data, + \
        aes(x='Hour', y='ENTRIESn_hourly', color='UNIT')) + \
        geom_point(size=100) + \
        geom_point() + \
        xlab(xlable) + ylab(ylable) + ggtitle(title) + \
        xlim(-.3,24) + ylim(0,400000)

    return plot
```

Which produces a very colorful but barely acceptable graph:

PANDAS references

[0] PANDAS. Indexing and Selecting Data. 2015. URL: http://pandas.pydata.org/pandas-docs/stable/indexing.html.

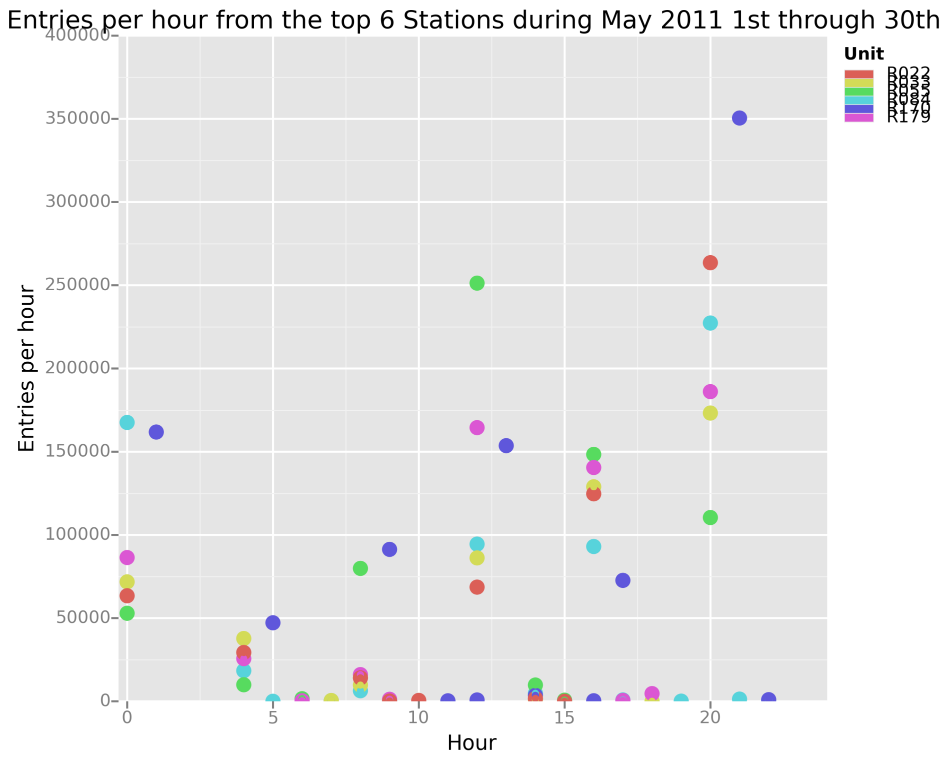

Figure 5: This is a graph per hour of the entries found for the top 6 stations. Note the low values indicate times when a given station isn't being used. This seems to imply there might be a possibility of increasing the efficiency of these stations.

[0] PANDAS. pandas.DataFrame. 2015. URL: http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html.

[0] PANDAS. Visualizations. Nov. 2015. URL: http://pandas.pydata.org/pandas-docs/stable/visualization.html.

[0] Inc. Stack Exchange. how do you filter pandas dataframes by multiple columns. Feb. 2014. URL: http://stackoverflow.com/questions/22086116/how-do-you-filter-pandas-dataframes-by-multiple-columns.

[0] Inc. Stack Exchange. save a pandas.Series histogram plot to file. Sept. 2013. URL: http://stackoverflow.com/questions/18992086/save-a-pandas-series-histogram-plot-to-file.

Python references

[0] Python Software Foundation. 8.1. datetime — Basic date and time types. Sept. 2015. URL: https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior.

[0] Inc. Stack Exchange. combine multiple text files into one text file using python. July 2014. URL: http://stackoverflow.com/questions/17749058/combine-multiple-text-files-into-one-text-file-using-python.

[0] Inc. Stack Exchange. Plot Normal distribution with Matplotlib. Nov. 2013. URL: http://stackoverflow.com/questions/20011494/plot-normal-distribution-with-matplotlib.

[0] Udacity. CSV ReaderWriter tutorial. Nov. 2015. URL: https://docs.google.com/document/d/1S4Gk42ZPBKAUZh7IPbqyzq_vk18oCvcniVcA4byBXCE/pub.

SQL references

[0] Oracle Corporation. MySQL 5.7 Reference Manual: 3.6.1 The Maximum Value for a Column. Nov. 2015. URL: https://dev.mysql.com/doc/refman/5.7/en/example-maximum-column.html.

[0] sqlite.org. Datatypes In SQLite Version 3. Nov. 2015. URL: https://www.sqlite.org/datatype3.html.

[0] sqlite.org. SQL As Understood By SQLite: Date And Time Functions. Nov. 2015. URL: https://www.sqlite.org/lang_datefunc.html.

[0] Inc. Stack Exchange. Selecting multiple max values using a single SQL statement. Nov. 2015. URL: http://stackoverflow.com/questions/18494829/selecting-multiple-max-values-using-a-single-sql-statement.

SciPy references

[0] SciPy developers. numpy.dot. 2015. URL: http://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html.

[0] SciPy developers. scipy.stats.ks_2samp. 2015. URL: http://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.stats.ks_2samp.html.

[0] SciPy developers. scipy.stats.mannwhitneyu. 2015. URL: http://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.stats.mannwhitneyu.html.

[0] SciPy developers. scipy.stats.mstats.normaltest. 2015. URL: http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.mstats.normaltest.html.

[0] SciPy developers. scipy.stats.shapiro. 2015. URL: http://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.stats.shapiro.html.

Udacity references

[0] Udacity. Analyzing the NYC Subway Dataset. Nov. 2015. URL: https://docs.google.com/document/d/16T3kirC0IxvtfxlZb7n5kOz5xFF_JTwrG31J2OZj8KM/pub.

[0] Udacity. Analyzing the NYC Subway Dataset Short Answer Rubric. Nov. 2015. URL: https://docs.google.com/document/d/1ZWdmlEgtRhreyN7AaiEfoYP70GqxOZqrajWtzIov8HM/pub.

[0] Udacity. Intro to Data Science: Learn What It Takes to Become a Data Scientist. Nov. 2015. URL: https://www.udacity.com/course/intro-to-data-science--ud359.

GGplot references

[0] Inc. Stack Exchange. Plotting one scatterplot with multiple dataframes with ggplot in python. Jan. 2014. URL: http://stackoverflow.com/questions/22377539/plotting-one-scatterplot-with-multiple-dataframes-with-ggplot-in-python.

[0] yhat. ggplot from ŷhat. 2014. URL: http://ggplot.yhathq.com/docs/index.html.

Machine Learning references

[0] David Cournapeau. sklearn.linear_model.SGDRegressor. 2015. URL: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html.

other references

[0] The Metropolitan Transportation Authority. Field Description. 2015. URL: http://web.mta.info/developers/resources/nyct/turnstile/ts_Field_Description.txt.

Statistics references

[0] Jim Frost. Regression Analysis: How Do I Interpret R-squared and Assess the Goodness-of-Fit? 2013. URL: http://blog.minitab.com/blog/adventures-in-statistics/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit.

[0] NIST/SEMATECH. 1.3.5.16. Kolmogorov-Smirnov Goodness-of-Fit Test. 2012. URL: http://www.itl.nist.gov/div898/handbook/eda/section3/eda35g.htm.

[0] NIST SEMATECH. 5.2.4. Are the model residuals well-behaved? 2015. URL: http://www.itl.nist.gov/div898/handbook/eda/section3/histogra.htm.