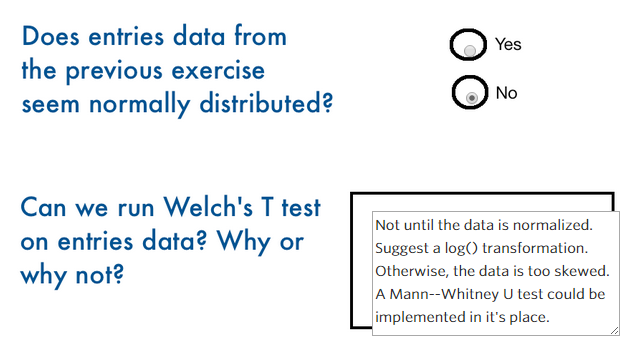
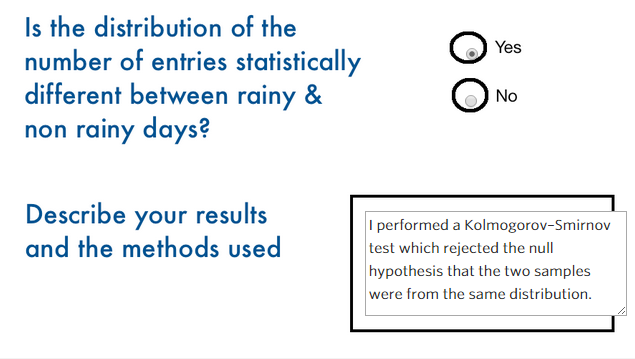
1. Problem Set 3.1
2. import numpy as np
3. import pandas
4. import matplotlib.pyplot as plt
6. def entries\_histogram(turnstile\_weather):

9. plt.figure()
10. turnstile\_weather[turnstile\_weather['rain'] == 1]['ENTRIESn\_hourly'].hist() # your code here to plot a historgram **for** hourly entries when it is raining
11. turnstile\_weather[turnstile\_weather['rain'] == 0]['ENTRIESn\_hourly'].hist() # your code here to plot a historgram **for** hourly entries when it is not raining
12. **return** plt

15. Problem Set 3.2
16. 
18. Problem Set 3.3
20. import numpy as np
21. import scipy
22. import scipy.stats
23. import pandas
25. def mann\_whitney\_plus\_means(turnstile\_weather):
27. ### YOUR CODE HERE ###
28. with\_rain\_mean = np.mean(turnstile\_weather[turnstile\_weather['rain'] == 1]['ENTRIESn\_hourly'])
29. without\_rain\_mean = np.mean(turnstile\_weather[turnstile\_weather['rain'] == 0]['ENTRIESn\_hourly'])
30. U, p = scipy.stats.mannwhitneyu(turnstile\_weather[turnstile\_weather['rain'] == 1]['ENTRIESn\_hourly'],
31. turnstile\_weather[turnstile\_weather['rain'] == 0]['ENTRIESn\_hourly'])

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

37. Problem Set 3.4
38. 
40. Problem Set 3.5
42. import numpy as np
43. import pandas
44. from ggplot import \*

47. def normalize\_features(df):
48. mu = df.mean()
49. sigma = df.std()
51. **if** (sigma == 0).any():
52. raise Exception("One or more features had the same value for all samples, and thus could " + \
53. "not be normalized. Please do not include features with only a single value " + \
54. "in your model.")
55. df\_normalized = (df - df.mean()) / df.std()
57. **return** df\_normalized, mu, sigma
59. def compute\_cost(features, values, theta):
61. # your code here
62. cost = np.square(np.dot(features, theta) - values).sum()
64. **return** cost
66. def gradient\_descent(features, values, theta, alpha, num\_iterations):
67. """
68. Perform gradient descent given a data set with an arbitrary number of features.
70. This can be the same gradient descent code as in the lesson #3 exercises,
71. but feel free to implement your own.
72. """
74. m = len(values) \* 1.0
75. cost\_history = []
77. **for** i in range(num\_iterations):
78. # your code here
79. update = compute\_cost(features, values, theta)
80. cost\_history.append(update)
81. theta = theta + (1/m) \* alpha \* np.dot((values - np.dot(features, theta)), features)
82. **return** theta, pandas.Series(cost\_history)
84. def predictions(dataframe):
85. # Select Features (try different features!)
86. features = dataframe[['Hour', 'maxdewpti', 'maxtempi', 'mindewpti']]
88. # Add UNIT to features using dummy variables
89. dummy\_units = pandas.get\_dummies(dataframe['UNIT'], prefix='unit')
90. features = features.join(dummy\_units)
92. # Values
93. values = dataframe['ENTRIESn\_hourly']
94. m = len(values)
96. features, mu, sigma = normalize\_features(features)
97. features['ones'] = np.ones(m) # Add a column of 1s (y intercept)
99. # Convert features and values to numpy arrays
100. features\_array = np.array(features)
101. values\_array = np.array(values)
103. # Set values for alpha, number of iterations.
104. alpha = 0.3 # please feel free to change **this** value
105. num\_iterations = 100 # please feel free to change **this** value
107. # Initialize theta, perform gradient descent
108. theta\_gradient\_descent = np.zeros(len(features.columns))
109. theta\_gradient\_descent, cost\_history = gradient\_descent(features\_array,
110. values\_array,
111. theta\_gradient\_descent,
112. alpha,
113. num\_iterations)
115. plot = None
116. # -------------------------------------------------
117. # Uncomment the next line to see your cost history
118. # -------------------------------------------------
119. plot = plot\_cost\_history(alpha, cost\_history)
120. #
121. # Please note, there is a possibility that plotting
122. # this in addition to your calculation will exceed
123. # the 30 second limit on the compute servers.
125. predictions = np.dot(features\_array, theta\_gradient\_descent)
126. **return** predictions, plot

129. def plot\_cost\_history(alpha, cost\_history):
130. """This function is **for** viewing the plot of your cost history.
131. You can run it by uncommenting **this**
133. plot\_cost\_history(alpha, cost\_history)
135. call in predictions.
137. If you want to run **this** locally, you should print the **return** value
138. from **this** function.
139. """
140. cost\_df = pandas.DataFrame({
141. 'Cost\_History': cost\_history,
142. 'Iteration': range(len(cost\_history))
143. })
144. **return** ggplot(cost\_df, aes('Iteration', 'Cost\_History')) + \
145. geom\_point() + ggtitle('Cost History for alpha = %.3f' % alpha )


149. Problem Set 3.6
151. import numpy as np
152. import scipy
153. import matplotlib.pyplot as plt
155. def plot\_residuals(turnstile\_weather, predictions):
157. plt.figure()
158. (turnstile\_weather['ENTRIESn\_hourly'] - predictions).hist()
159. **return** plt

162. Problem Set 3.7
164. import numpy as np
165. import scipy
166. import matplotlib.pyplot as plt
167. import sys
168. def compute\_r\_squared(data, predictions):
169. r\_squared = 1 - ((np.square(data - predictions).sum())/(np.square(data - np.mean(data)).sum()))
171. **return** r\_squared
172. Problem Set 3.8
174. import numpy as np
175. import pandas
176. import scipy
177. import statsmodels.api as sm
178. import datetime
180. def predictions(weather\_turnstile):
181. #
182. # Your implementation goes here. Feel free to write additional
183. # helper functions
184. #
185. df = weather\_turnstile
186. dummyunit = pandas.get\_dummies(df['UNIT'])
187. dummyhour = pandas.get\_dummies(df['Hour'])
188. features = dummyunit.join([dummyhour])
190. entries = df['ENTRIESn\_hourly']
191. prediction = sm.OLS(entries, features).fit().predict(features)
193. **return** prediction