```python
# Assessed exercises 7
# Look at cuts and creating ROC curves

from pandas import Series, DataFrame
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as npr

# This week we will use the ebola_test dataset to test the code. It contains the
# results for a new medical test for detecting Ebola that is much faster, though
# less accurate, than that currently available. The new test has been applied to
# a number of patients who are known (from the old test) to have the disease or
# not. The first column of the data set (called 'prob') is the probability under
# the new test that the patient has Ebola. The second column ('ebola') is the
# result from the older test which definitively says whether the patient has ebola

# (ebola = 1) or not (ebola=0). Download the dataset, load it in and have a look
# at the first few entries to see what it looks like.
eb = pd.read_csv('ebola_test.csv')


# Q1 Write a function that returns a dict with some information about the
# DataFrame df. The keys of the dict should be 'Percentage' and 'Quartiles'. The
# value for 'Percentage' should be a single number (not a list with a number in
# it), specifying the percentage of entries with positive results for the given
# criteria, i.e. indicator variable is 1. This value should be rounded to 1
# decimal place. The value for 'Quartiles' should be a list (not a Series or
```

```
28 # array) with the number of observations in the 1st quartile (0-25%), 2nd
29 # quartile (25-50%), 3rd quartile (50-75%) and 4th quartile (75-100%) for a
30 # specified observation column.
31 # The name of the indicator and observation variable should be given to the
32 # function as strings.
33 # Be sure that your keys are exactly as specified above and that the values have
34 # # the data type specified above.
35 def exercise1(df, ind_col, obs_col):
36     index_ind_col = df.columns.get_loc(ind_col)
37     index_obs_col = df.columns.get_loc(obs_col)
38     Percentage = round((sum(df.iloc[:, index_ind_col]) / len(df)) * 100, 1)
39     Quartiles = [sum(1 for i in range(len(df)) if
40                      np.quantile(df.iloc[:, index_obs_col], 0) <= df.iloc[i,
   index_obs_col] <= np.quantile(
41                          df.iloc[:, index_obs_col], 0.25))]
42     Quartiles.append(sum(1 for i in range(len(df)) if
43                          np.quantile(df.iloc[:, index_obs_col], 0.25) < df.iloc[i
   , index_obs_col] <= np.quantile(
44                              df.iloc[:, index_obs_col], 0.5)))
45     Quartiles.append(sum(1 for i in range(len(df)) if
46                          np.quantile(df.iloc[:, index_obs_col], 0.5) < df.iloc[i,
   index_obs_col] <= np.quantile(
47                              df.iloc[:, index_obs_col], 0.75)))
48     Quartiles.append(sum(1 for i in range(len(df)) if
49                          np.quantile(df.iloc[:, index_obs_col], 0.75) < df.iloc[i
   , index_obs_col] <= np.quantile(
50                              df.iloc[:, index_obs_col], 1)))
51     return {'Percentage': Percentage, 'Quartiles': Quartiles}
```

```
52
53
54 # Suggested test
55 exercise1(eb, 'ebola', 'prob')
56
57
58 # This should return
59 # {'Percentage': 10.4, 'Quartiles': [128, 125, 125, 121]}
60 # in this exact form. If it does not your function is not correct.
61
62
63 # In classification problems one must define a cutoff, a value for which
64 # observations above that value are classified as positive results (and assigned
65 # the value 1), and those less than or equal to the value are classified as
66 # negative results (and assigned the value 0). If we had a perfect classifier,
67 # the predicted values (0s or 1s) would match the indicator variable. A false
68 # positive is defined as the classifier predicting a positive result (1), when
69 # the actual result was negative (0). A false negative is the opposite, the
70 # classifier predicts a negative result (0), when the true result was positive (1
   ).
71
72 # Q2 Write a function that takes a DataFrame, two strings specifying the names of
73 # indicator column and the observation column, and a cutoff value, and
74 # returns the rate of false positive and rate of the false negative as a dict.
75 # The keys of the dict should be 'False Pos' and 'False Neg' and the values
76 # must be rounded to 3 decimal places.
77 def exercise2(df, ind_col, obs_col, cutoff):
78     index_ind_col = df.columns.get_loc(ind_col)
```

```
 79      index_obs_col = df.columns.get_loc(obs_col)
 80      df['prediction'] = (df.iloc[:, index_obs_col] > cutoff).astype(int)
 81      false_pos = 0
 82      false_neg = 0
 83      for i in range(len(df)):
 84          if df.loc[i, 'prediction'] == 1 and df.iloc[i, index_ind_col] == 0:
 85              false_pos += 1
 86          if df.loc[i, 'prediction'] == 0 and df.iloc[i, index_ind_col] == 1:
 87              false_neg += 1
 88      return {'False Pos': round(false_pos / len(df), 3), 'False Neg': round(
    false_neg / len(df), 3)}
 89
 90
 91 # Suggested test
 92 exercise2(eb, 'ebola', 'prob', 0.15)
 93
 94
 95 # This should return
 96 # {'False Pos': 0.126, 'False Neg': 0.07}
 97 # in this exact form. If it does not your function is not correct.
 98
 99 # We do not know a priori what the best cutoff value is, as such, we should
100 # loop over a range of cutoffs to decide on the best one.
101
102 # Q3 Write a function that takes the same inputs as Q2, but cutoff will now be
103 # replaced with cutoff_list (an array of different cutoff values to test). The
104 # function should run the classification for each value in cutoff_list and
105 # determine which is the best cutoff value. We will define the best classifier
```

```
106 # as having the lowest false results (false positives plus false negatives). The
107 # function should return a dict with the keys 'Cutoff value', 'False Pos' and
108 # 'False Neg', and the values of the false positive rate and false negative rate
109 # should be rounded to 3 decimal places
110 def exercise3(df, ind_col, obs_col, cutoff_list):
111     index_ind_col = df.columns.get_loc(ind_col)
112     index_obs_col = df.columns.get_loc(obs_col)
113     smallest = len(df)
114     best_cutoff = min(cutoff_list)
115     for cutoff in cutoff_list:
116         df['prediction'] = (df.iloc[:, index_obs_col] > cutoff).astype(int)
117         df['diff'] = np.abs(df['prediction'] - df[ind_col])
118         if sum(df['diff']) < smallest:
119             smallest = sum(df['diff'])
120             best_cutoff = cutoff
121     df['prediction'] = (df.iloc[:, index_obs_col] > best_cutoff).astype(int)
122     false_pos = 0
123     false_neg = 0
124     for i in range(len(df)):
125         if df.loc[i, 'prediction'] == 1 and df.iloc[i, index_ind_col] == 0:
126             false_pos += 1
127         if df.loc[i, 'prediction'] == 0 and df.iloc[i, index_ind_col] == 1:
128             false_neg += 1
129     return {'Cutoff value': best_cutoff, 'False Pos': round(false_pos / len(df),
    3),
130             'False Neg': round(false_neg / len(df), 3)}
131
132
```

```
133 # Suggested test
134 exercise3(eb, 'ebola', 'prob', np.arange(0, 1, 0.01))
135
136
137 # This should return
138 # {'Cutoff value': 0.21, 'False Pos': 0.0, 'False Neg': 0.102}
139 # in this exact form. If it does not your function is not correct.
140
141 # A related concept to the choice of cut-offs is the Receiver Operator
    Characteristic
142 # (ROC) curve. The ROC curve aims to quantify how well a classifier beats a
    random
143 # classifier for any level of probability cut-off. An introduction can be found
    here:
144 # http://en.wikipedia.org/wiki/Receiver_operating_characteristic
145 # The idea is to plot the false positive rate against the true positive rate for
146 # every possible cut-off
147
148 # Q4 Write a function which calculates the ROC curve. The function should have
149 # three arguments, the DataFrame df, the name of the indicator variable ind_col
150 # and the name of the observation variable obs_col
151 # Your ROC curve function should perform the following steps
152 # 1) Find the unique values in the observation column
153 # 2) Use each of these unique values as a cutoff value and
154 # a) Classify all the obs as either positive or negative based on the current
    cutoff value
155 # b) Calculate the number of true positives (tp), true negatives (tn), false
    positives (fp) and false negatives (fn)
```

```
156 # Note 1: a tp is when the classification value and actual value are both 1, a tn
     is when they're both 0
157 # Note 2: tp, fn, etc must all be vectors/Series of the same length as the vector
    /Series of cutoff values
158 # Note 4: be careful when you're at the maximum cutoff value that you can still
    calculate these values correctly
159 # 3) Create the true positive rate (tpr) as tp/(tp+fn)
160 # 4) Create the false positive rate (fpr) as fp/(fp+tn)
161 # 5) Create a DataFrame, indexed by the cutoff values (unique values of
162 # observation column), with columns 'True_Pos' and 'False_Pos', containing tpr
163 # and fpr, respectively.
164 # 6) Return the DataFrame sorted by index (lowest to highest)
165
166 def exercise4(df, ind_col, obs_col):
167     uni_obs = np.unique(df[obs_col])
168     d = {}
169     index = []
170     tpr = []
171     fpr = []
172     for cutoff in uni_obs:
173         df['prediction'] = (df.loc[:, obs_col] > cutoff).astype(int)
174         tp = 0
175         tn = 0
176         fp = 0
177         fn = 0
178         d[cutoff] = {}
179         for i in range(len(df)):
180             if df.loc[i, 'prediction'] == 1 and df.loc[i, ind_col] == 0:
```

```
                fp += 1
            if df.loc[i, 'prediction'] == 0 and df.loc[i, ind_col] == 1:
                fn += 1
            if df.loc[i, 'prediction'] == 0 and df.loc[i, ind_col] == 0:
                tn += 1
            if df.loc[i, 'prediction'] == 1 and df.loc[i, ind_col] == 1:
                tp += 1
        tpr_ = tp / (tp + fn)
        fpr_ = fp / (fp + tn)
        index.append(cutoff)
        tpr.append(tpr_)
        fpr.append(fpr_)
    df_roc = pd.DataFrame({'True_Pos': tpr, 'False_Pos': fpr}, index=index).
sort_index(axis=0)
    return df_roc


# Suggested test
Q4_ans = exercise4(eb, 'ebola', 'prob')
Q4_ans.plot(x='False_Pos', y='True_Pos')
plt.show()
# This should create a plot of the false positive rate vs true positive rate.
# When the false positive rate is ~0.5, the true positive rate is ~0.85

```