```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 numpy.random as npr
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

# 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
# array) with the number of observations in the 1st quartile (0-25%), 2nd
# quartile (25-50%), 3rd quartile (50-75%) and 4th quartile (75-100%) for a
# specified observation column.
# The name of the indicator and observation variable should be given to the
# function as strings.
# Be sure that your keys are exactly as specified above and that the values have
# the data type specified above.
def exercise1(df,ind_col,obs_col):
    # percentage = the number of exact patients / the number of all patients = mean
    per = round(df[ind_col].mean()*100,1)
    group = pd.qcut(df[obs_col],4)
    quart = pd.value_counts(group).values
    df_dict = {'Percentage' : per, 'Quartiles': list(quart) }
    return df_dict

# Suggested test
exercise1(eb,'ebola','prob')
# This should return
# {'Percentage': 10.4, 'Quartiles': [128, 125, 125, 121]}
# or:
# {'False Neg': 0.07, 'False Pos': 0.126}
# in this exact form. If it does not your function is not correct.
# Note that due to finite precision, your function may output the number
# correctly rounded to 3 decimal places, with a string of zeros followed by a
# non-zero digit, e.g. 0.07000000000000007 for the false negative rate. This is
# still a correct answer.


# In classification problems one must define a cutoff, a value for which
# observations above that value are classified as positive results (and assigned
# the value 1), and those less than or equal to the value are classified as
# negative results (and assigned the value 0). If we had a perfect classifier,
# the predicted values (0s or 1s) would match the indicator variable. A false
# positive is defined as the classifier predicting a positive result (1), when
# the actual result was negative (0). A false negative is the opposite, the
# classifier predicts a negative result (0), when the true result was positive (1).

# Q2 Write a function that takes a DataFrame, two strings specifying the names of
```

```
# indicator column and the observation column, and a cutoff value, and
# returns the rate of false positive and rate of the false negative as a dict.
# The keys of the dict should be 'False Pos' and 'False Neg' and the values
# must be rounded to 3 decimal places.
def exercise2(df,ind_col,obs_col,cutoff):
    temp = list()
    for i in df[obs_col]:
        if i > cutoff:
            temp.append(1)
        else:
            temp.append(0)
    # use confusion_matrix to get the numbers of fp and fn
    cm = confusion_matrix(df[ind_col], temp)
    # create a array of fp and np then calculate the percentage of them,according to
    result = cm.sum(axis=1) - np.diag(cm)
    result = result/len(temp)
    Q2_dict = {'False Pos':round(result[0],3),'False Neg':round(result[1],3) }
    return Q2_dict
# Suggested test
exercise2(eb,'ebola','prob',0.15)
# This should return
# {'False Pos': 0.126, 'False Neg': 0.07}
# in this exact form. If it does not your function is not correct.

# We do not know a priori what the best cutoff value is, as such, we should
# loop over a range of cutoffs to decide on the best one.

# Q3 Write a function that takes the same inputs as Q2, but cutoff will now be
# replaced with cutoff_list (an array of different cutoff values to test). The
# function should run the classification for each value in cutoff_list and
# determine which is the best cutoff value. We will define the best classifier
# as having the lowest false results (false positives plus false negatives). The
# function should return a dict with the keys 'Cutoff value', 'False Pos' and
# 'False Neg', and the values of the false positive rate and false negative rate
# should be roundedto 3 decimal places
def exercise3(df,ind_col,obs_col,cutoff_list):
    # create result_store to store fp and fn
    result_store = list()
    # create sum_store to store the result of fp plus fn
    sum_store = list()
    for j in range(len(cutoff_list)):
        temp = list()
        for i in df[obs_col]:
            if i > cutoff_list[j]:
                temp.append(1)
            else:
                temp.append(0)
        # use confusion_matrix to get the numbers of fp and fn
        cm = confusion_matrix(df[ind_col], temp)
        result = cm.sum(axis=1) - np.diag(cm)
        result_store.append(result/len(temp))
        sum_store.append(sum(result))
    # find the index of the minimum sum of fp and fn
    min_false = sum_store.index(min(sum_store))
    Q3_dict = {'Cutoff value': cutoff_list[min_false],'False Pos':round(result_store[
    return Q3_dict


# Suggested test
exercise3(eb,'ebola','prob',np.arange(0,1,0.01))
# This should return
# {'Cutoff value': 0.21, 'False Pos': 0.0, 'False Neg': 0.102}
# in this exact form. If it does not your function is not correct.

# A related concept to the choice of cut-offs is the Receiver Operator Characteristic
```

```
# (ROC) curve. The ROC curve aims to quantify how well a classifier beats a random
# classifier for any level of probability cut-off. An introduction can be found here:
# http://en.wikipedia.org/wiki/Receiver_operating_characteristic
# The idea is to plot the false positive rate against the true positive rate for
# every possible cut-off

# Q4 Write a function which calculates the ROC curve. The function should have
# three arguments, the DataFrame df, the name of the indicator variable ind_col
# and the name of the observation variable obs_col
# Your ROC curve function should perform the following steps
# 1) Find the unique values in the observation column
# 2) Use each of these unique values as a cutoff value and
# a) Classify all the obs as either positive or negative based on the current cutoff
# b) Calculate the number of true positives (tp), true negatives (tn), false positive
# Note 1: a tp is when the classification value and actual value are both 1, a tn is
# Note 2: tp, fn, etc must all be vectors/Series of the same length as the vector/Ser
# Note 4: be careful when you're at the maximum cutoff value that you can still calcu
# 3) Create the true positive rate (tpr) as tp/(tp+fn)
# 4) Create the false positive rate (fpr) as fp/(fp+tn)
# 5) Create a DataFrame, indexed by the cutoff values (unique values of
# observation column), with columns 'True_Pos' and 'False_Pos', containing tpr
# and fpr, respectively.
# 6) Return the DataFrame sorted by index (lowest to highest)
def exercise4(df,ind_col,obs_col):
    cutoff_list = np.unique(df[obs_col])
    tpr_list = list()
    fpr_list = list()
    for j in range(len(cutoff_list)):
        temp = list()
        for i in df[obs_col]:
            if i > cutoff_list[j]:
                temp.append(1)
            else:
                temp.append(0)
        # use confusion_matrix to get the numbers of tn,fp,fn,tp.
        tn, fp, fn, tp = confusion_matrix(df[ind_col], temp).ravel()
        tpr = tp/(tp+fn)
        fpr = fp/(fp+tn)
        tpr_list.append(round(tpr,3))
        fpr_list.append(round(fpr,3))
    # create a dataframe and set the indexes as cutoff_list
    df1 = pd.DataFrame(index = cutoff_list)
    df1['True_Pos'] = tpr_list
    df1['False_Pos'] = fpr_list
    df1 = df1.sort_index()
    return df1
# Suggested test
Q4_ans = exercise4(eb,'ebola','prob')
Q4_ans.plot(x='False_Pos',y='True_Pos')
# 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
```