**CS5590 APS - Python Programming**

**Lab 2.**

Fall 2018

UMKC

Kim-Ndor Djimadoumngar

1. **Introduction**

This is a report by Kim-Ndor Djimadoumngar for Assignment 2 of the Special Topic in Applied Programming Learning (APL) series: CS5590-0001 python and Deep Learning. My student ID is 6; my Lab ID is 7. The course is taught by Dr. Yugyung Lee and instructed by Saria Goudarzvand.

1. **Objectives**

The purposes of this lab are triple:

* To make categorical plots seaborn library or matplotlib
* To use the algorithms such as Naïve Bayes, Support Vector Machine with linear, rbf, and poly kernel to make predictive and classification models and report their accuracies
* To apply the techniques such as lemmatization, bigram, bigram-frequency, concatenation, and summarization on the text.

1. **Approaches/Methods**

Pycharm Community Edition 2018 was used to create Python in order to implement the exercises. Notepad was used to create txt files.

1. **Results/Discussion: Input/output**

**Question 1**

Pick any dataset from the dataset sheet in the class sheet

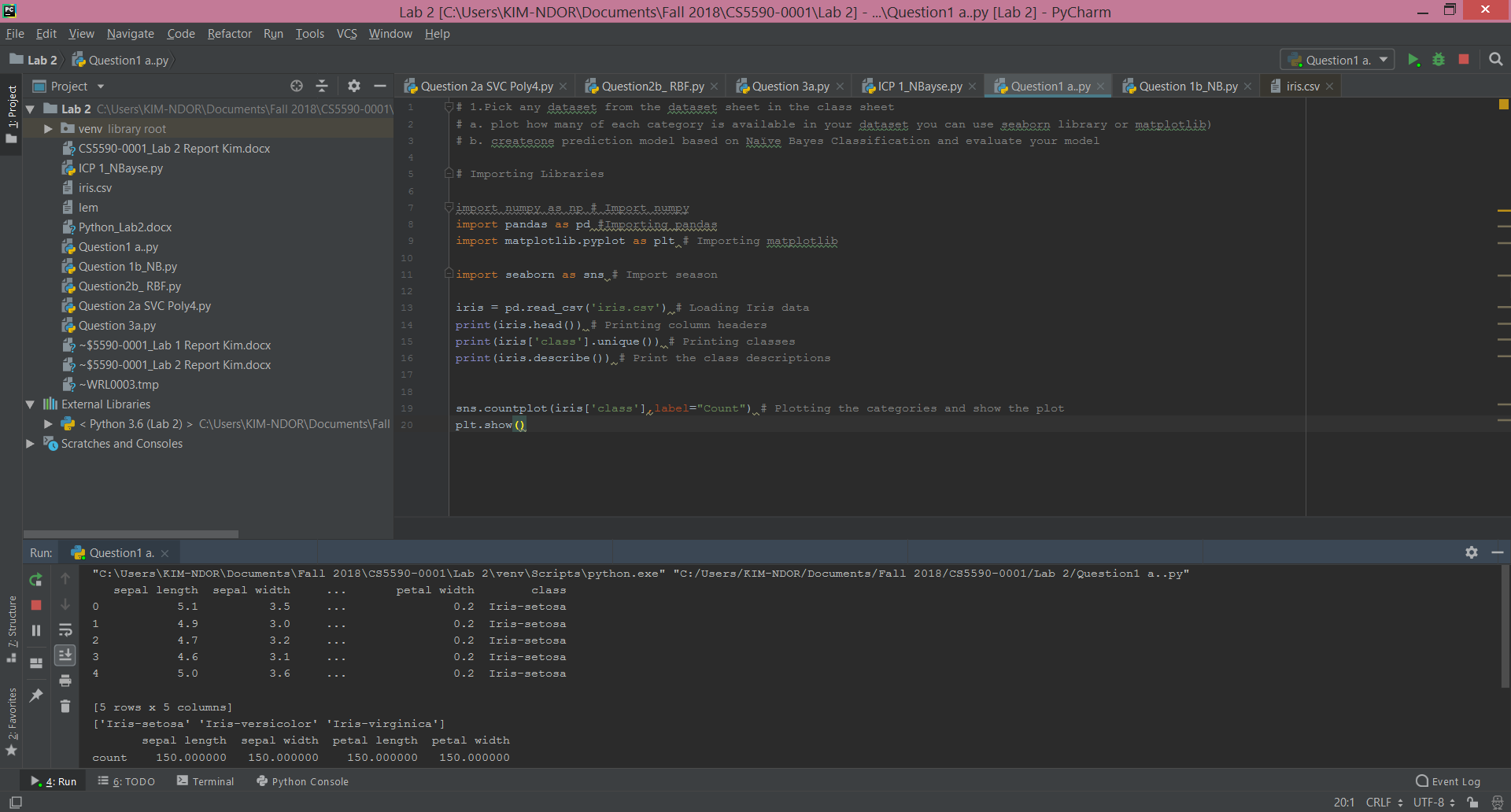
1. plot how many of each category is available in your dataset (you can use seaborn library or matplotlib).

I use Iris dataset, pandas, seaborn, and matplotlib libraries to run this exercise. The codes are shown below.

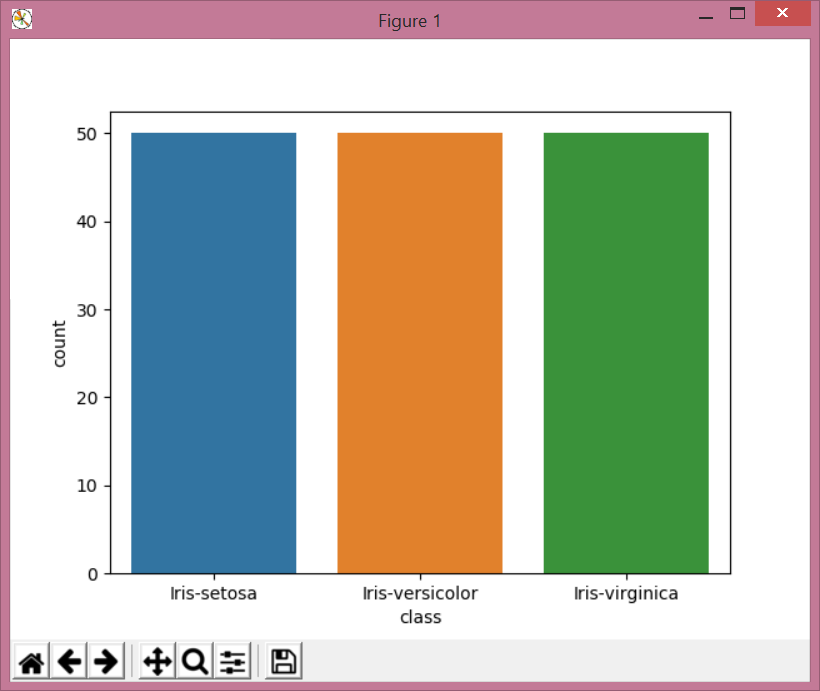
# 1.Pick any dataset from the dataset sheet in the class sheet  
# a. plot how many of each category is available in your dataset you can use seaborn library or matplotlib)  
# b. createone prediction model based on Naïve Bayes Classification and evaluate your model  
  
# Importing Libraries  
import numpy as np # Import numpy  
import pandas as pd #Importing pandas  
import matplotlib.pyplot as plt # Importing matplotlib  
  
import seaborn as sns # Import season  
  
iris = pd.read\_csv('iris.csv') # Loading Iris data  
print(iris.head()) # Printing column headers  
print(iris['class'].unique()) # Printing classes  
print(iris.describe()) # Print the class descriptions  
sns.countplot(iris['class'],label="Count") # Plotting the categories and show the plot  
plt.show()

**Results**

The top section of the result is shown in the console.

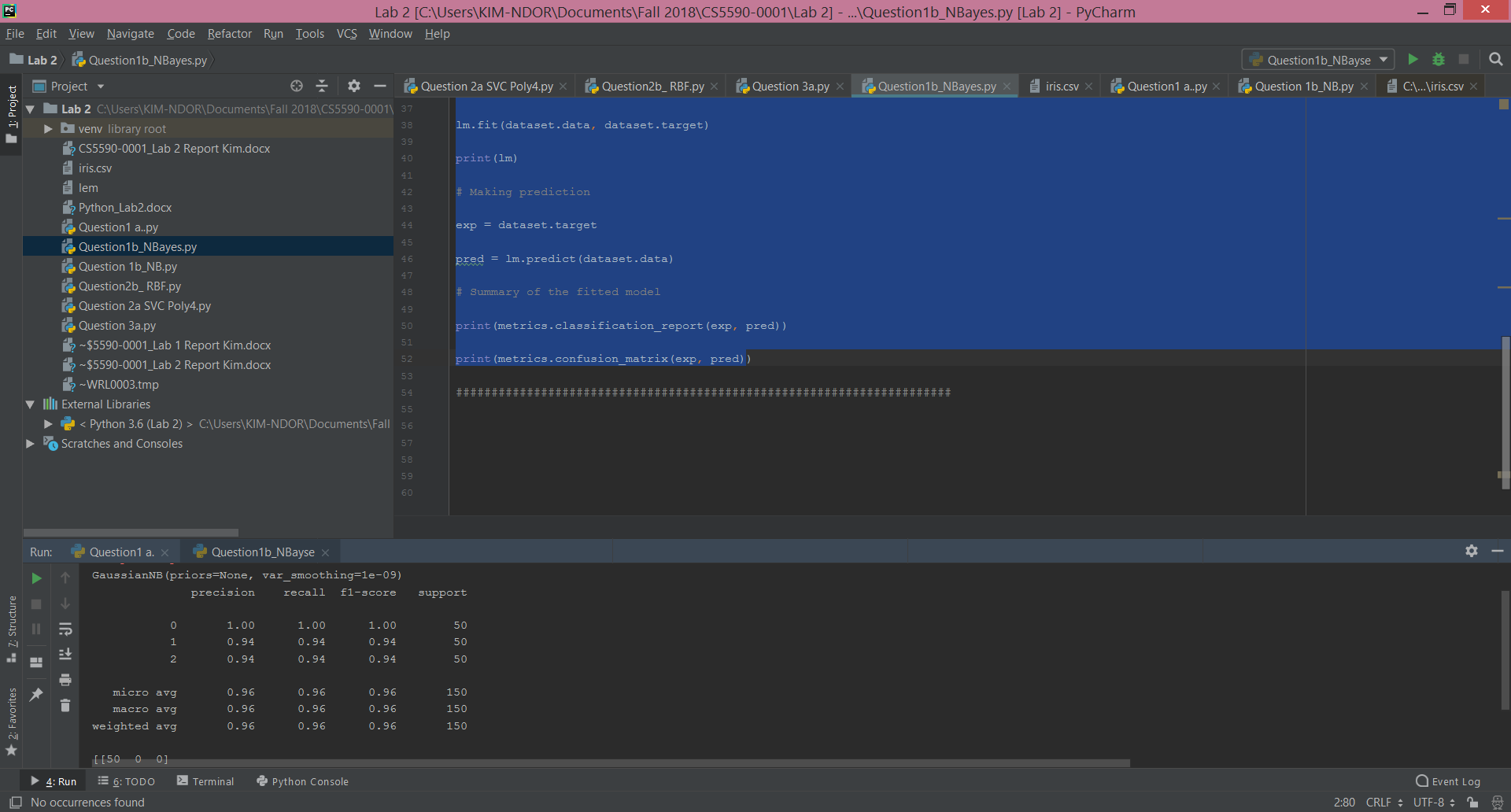


The categorical plot of the Iris dataset is below:



1. create one prediction model based on Naïve Bayes Classification and evaluate your model
2. #############################################################  
   # #Naive Bayse (Gaussian)  
   #  
   from sklearn import datasets  
     
   from sklearn import metrics  
     
   from sklearn.naive\_bayes import GaussianNB  
     
   #Loading iris data  
     
   dataset = datasets.load\_iris()  
     
     
   # Fitting NB Model to the data  
     
   lm = GaussianNB()  
     
   lm.fit(dataset.data, dataset.target)  
     
   print(lm)  
     
   # Making prediction  
     
   exp = dataset.target  
     
   pred = lm.predict(dataset.data)  
     
   # Summary of the fitted model  
     
   print(metrics.classification\_report(exp, pred))  
     
   print(metrics.confusion\_matrix(exp, pred)

**Results**



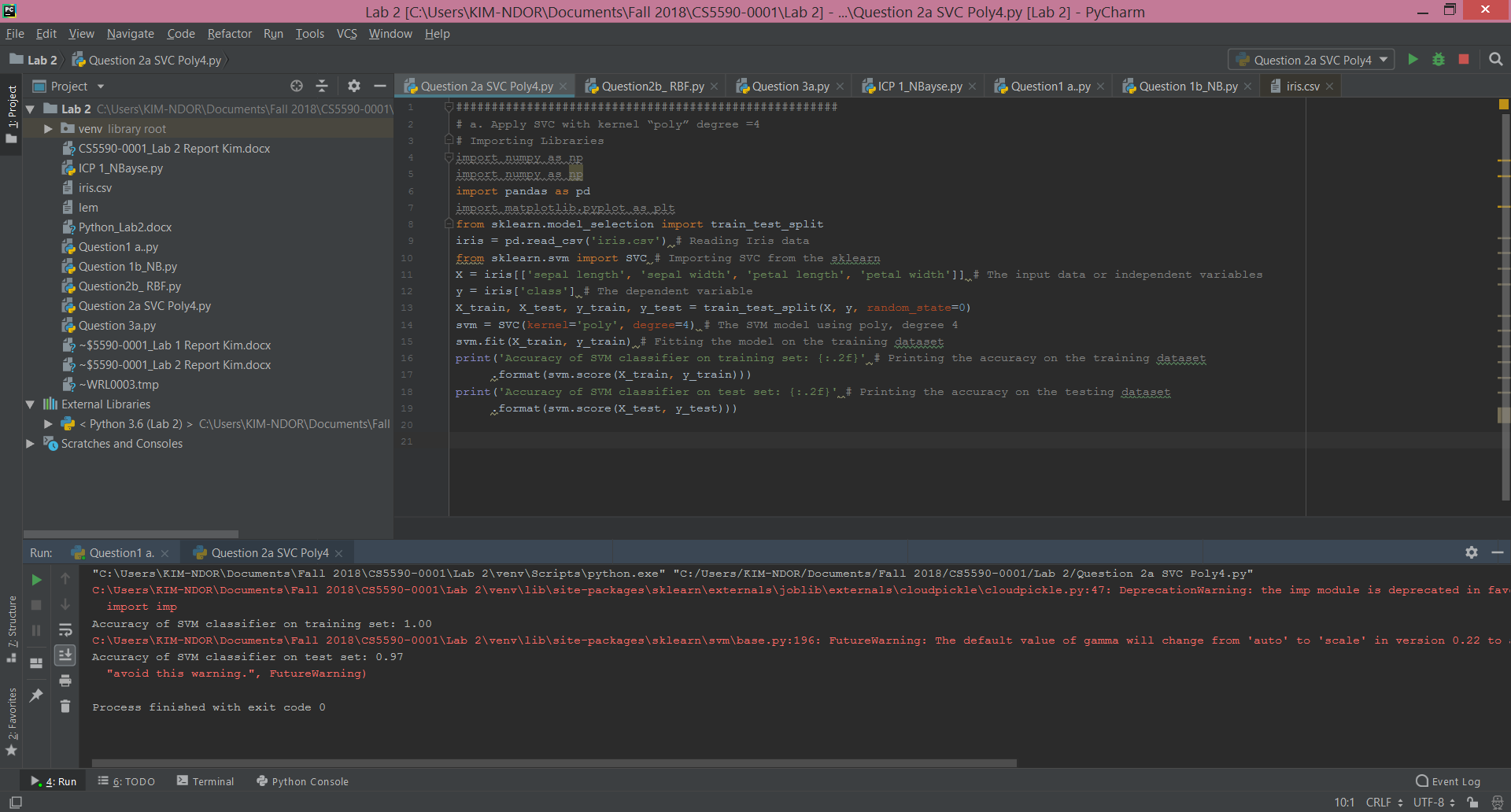
**Question 2**

**a. Apply SVC with kernel “poly” degree =4**

######################################################  
# a. Apply SVC with kernel “poly” degree =4  
# Importing Libraries  
import numpy as np  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
iris = pd.read\_csv('iris.csv') # Reading Iris data  
from sklearn.svm import SVC # Importing SVC from the sklearn  
X = iris[['sepal length', 'sepal width', 'petal length', 'petal width']] # The input data or independent variables  
y = iris['class'] # The dependent variable  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)  
svm = SVC(kernel='poly', degree=4) # The SVM model using poly, degree 4  
svm.fit(X\_train, y\_train) # Fitting the model on the training dataset  
print('Accuracy of SVM classifier on training set: {:.2f}' # Printing the accuracy on the training dataset  
 .format(svm.score(X\_train, y\_train)))  
print('Accuracy of SVM classifier on test set: {:.2f}' # Printing the accuracy on the testing dataset  
 .format(svm.score(X\_test, y\_test)))

**Results**

The screenshot below show the output result of the above codes.

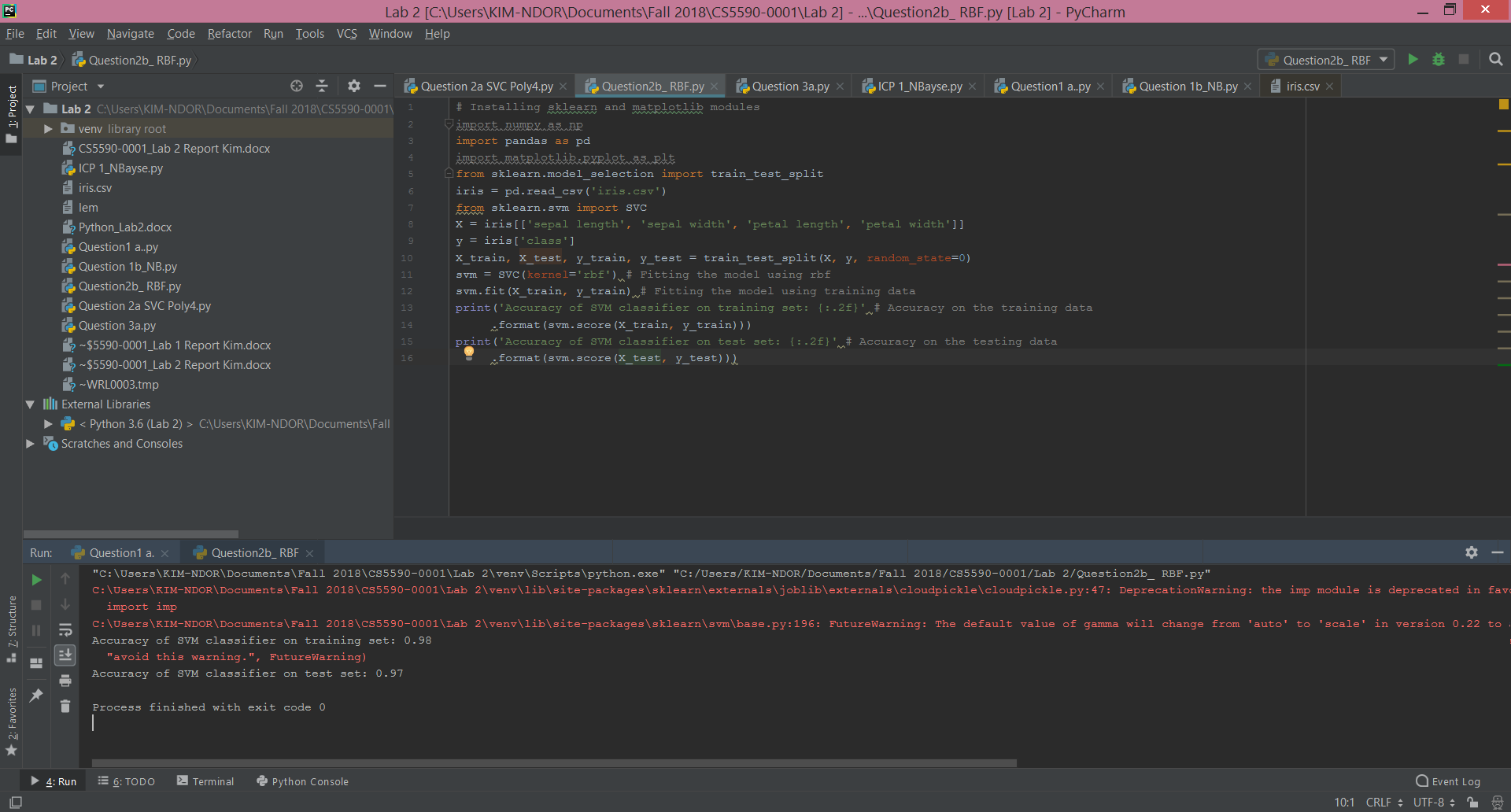


The accuracy on the training set is 100% while the testing accuracy is 97%.

b.**Apply SVC with “rbf” kernel**

1. # Installing sklearn and matplotlib modules  
   import numpy as np  
   import pandas as pd  
   import matplotlib.pyplot as plt  
   from sklearn.model\_selection import train\_test\_split  
   iris = pd.read\_csv('iris.csv')  
   from sklearn.svm import SVC  
   X = iris[['sepal length', 'sepal width', 'petal length', 'petal width']]  
   y = iris['class']  
   X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)  
   svm = SVC(kernel='rbf') # Fitting the model using rbf  
   svm.fit(X\_train, y\_train) # Fitting the model using training data  
   print('Accuracy of SVM classifier on training set: {:.2f}' # Accuracy on the training data  
    .format(svm.score(X\_train, y\_train)))  
   print('Accuracy of SVM classifier on test set: {:.2f}' # Accuracy on the testing data  
    .format(svm.score(X\_test, y\_test)))

**Results**



The accuracy on the training set is 98% while the testing accuracy is 97%.

**c. change gamma and C parameters in the model to see how the result may change**

The accuracy is a little bit higher on training than on testing data.

**d. Report the accuracy of the model on both models separately and with which parameters you got better result.**

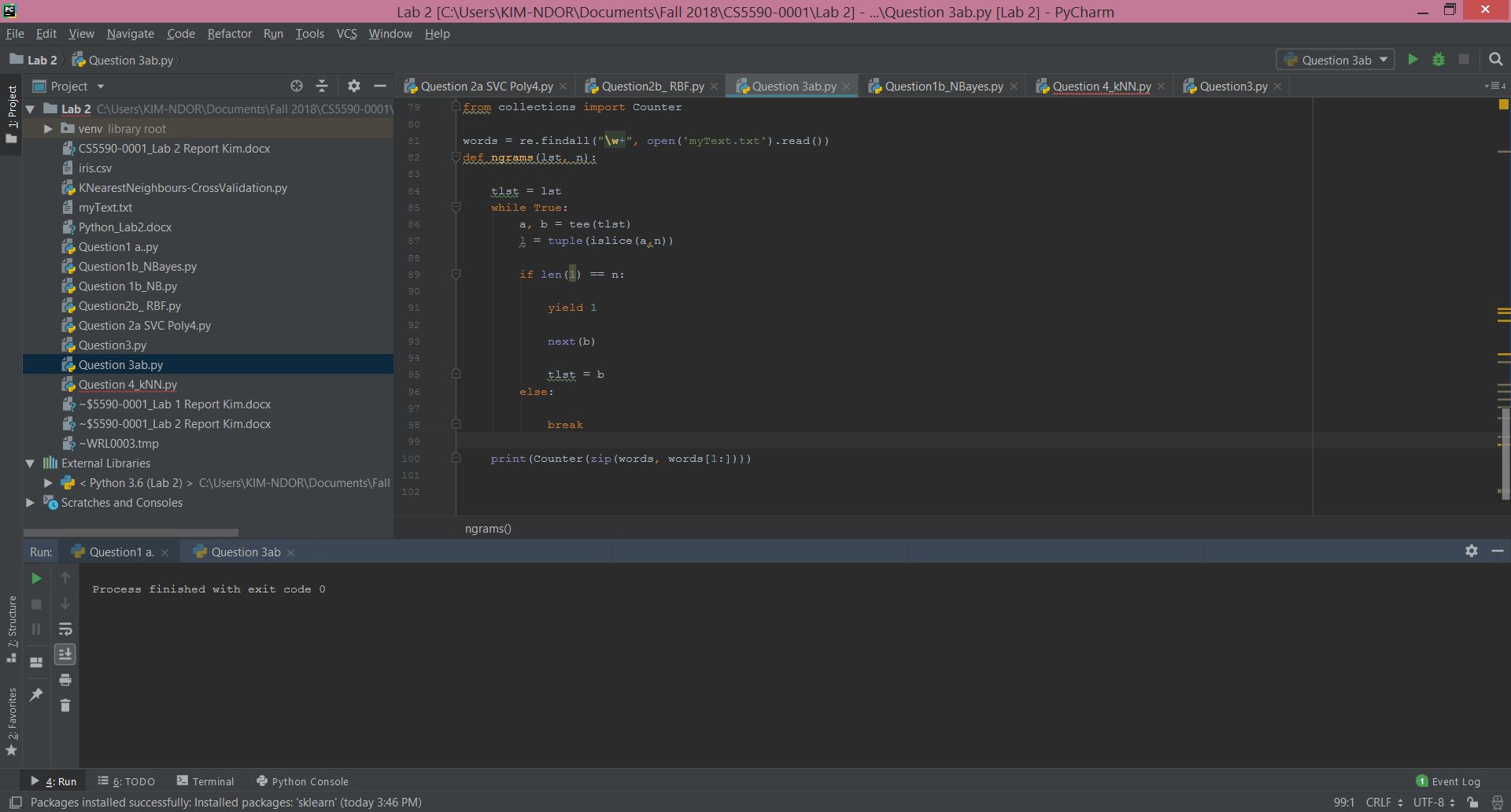
The model using poly, degree 4 performs better than the model with rbf

**Question 3**

**Question 3.a & b**

##################################################################################################  
  
#Get the frequency  
  
from itertools import tee, islice  
  
import re  
  
from collections import Counter  
  
words = re.findall("\w+", open('myText.txt').read())  
def ngrams(lst, n):  
  
 tlst = lst  
 while True:  
 a, b = tee(tlst)  
 l = tuple(islice(a,n))  
  
 if len(1) == n:  
  
 yield 1  
  
 next(b)  
  
 tlst = b  
 else:  
  
 break  
  
 print(Counter(zip(words, words[1:])))

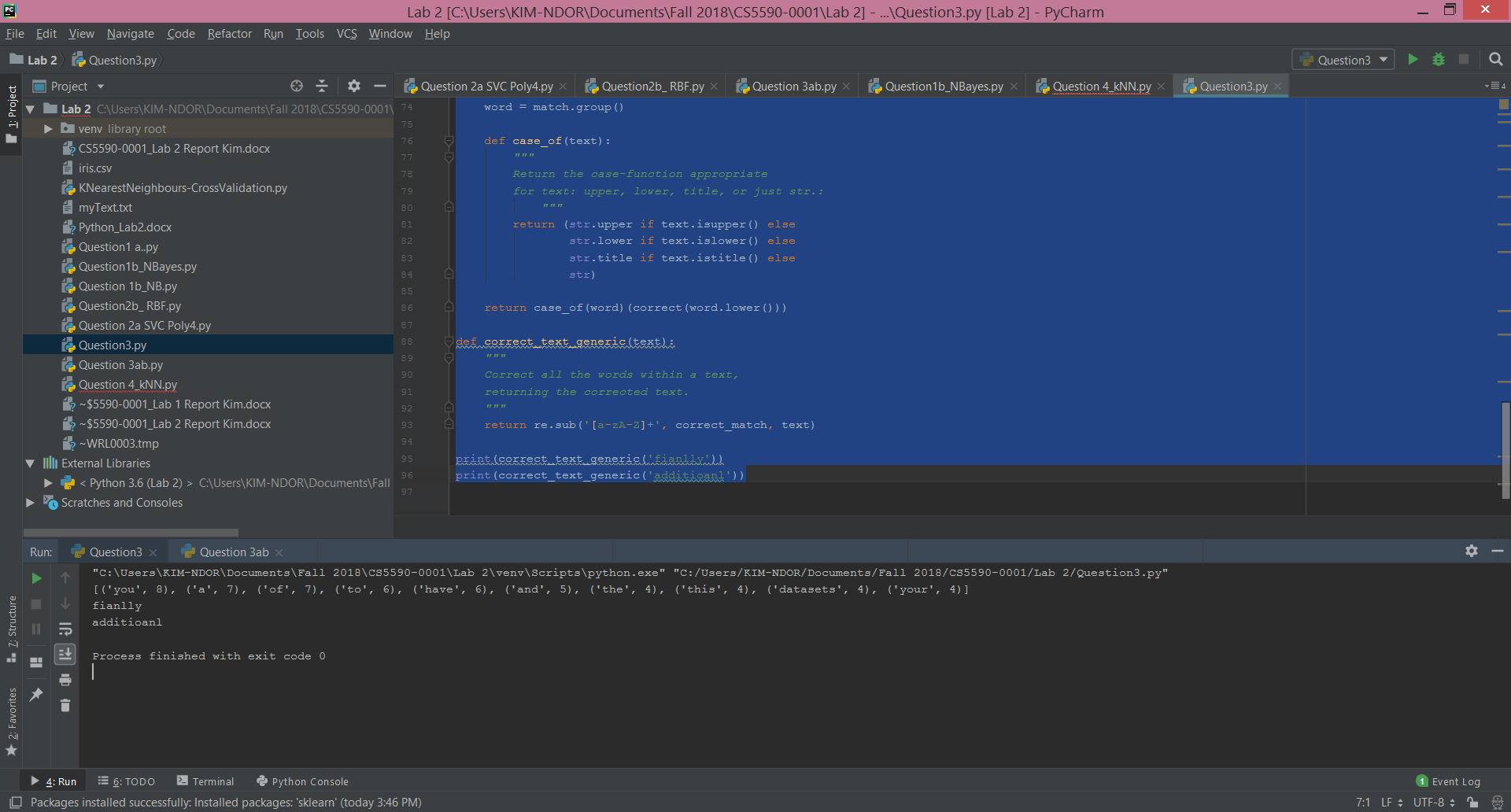
**Results**



**Question 3.c\_i**

import re, collections  
  
def tokens(text):  
 *"""  
 Get all words from the corpus  
 """* return re.findall('[a-z]+', text.lower())  
  
WORDS = tokens(open('myText.txt').read())  
WORD\_COUNTS = collections.Counter(WORDS)  
  
# top 10 words in corpus  
print(WORD\_COUNTS.most\_common(10))  
  
def known(words):  
 *"""  
 Return the subset of words that are actually   
 in our WORD\_COUNTS dictionary.  
 """* return {w for w in words if w in WORD\_COUNTS}  
  
def edits0(word):  
 *"""  
 Return all strings that are zero edits away   
 from the input word (i.e., the word itself).  
 """* return {word}  
  
def edits1(word):  
 *"""  
 Return all strings that are one edit away   
 from the input word.  
 """* alphabet = 'abcdefghijklmnopqrstuvwxyz'  
  
 def splits(word):  
 *"""  
 Return a list of all possible (first, rest) pairs   
 that the input word is made of.  
 """* return [(word[:i], word[i:])  
 for i in range(len(word) + 1)]  
  
 pairs = splits(word)  
 deletes = [a + b[1:] for (a, b) in pairs if b]  
 transposes = [a + b[1] + b[0] + b[2:] for (a, b) in pairs if len(b) > 1]  
 replaces = [a + c + b[1:] for (a, b) in pairs for c in alphabet if b]  
 inserts = [a + c + b for (a, b) in pairs for c in alphabet]  
 return set(deletes + transposes + replaces + inserts)  
  
def edits2(word):  
 *"""Return all strings that are two edits away   
 from the input word.  
 """* return {e2 for e1 in edits1(word) for e2 in edits1(e1)}  
  
def correct(word):  
 *"""  
 Get the best correct spelling for the input word  
 """* # Priority is for edit distance 0, then 1, then 2  
 # else defaults to the input word itself.  
 candidates = (known(edits0(word)) or  
 known(edits1(word)) or  
 known(edits2(word)) or  
 [word])  
 return max(candidates, key=WORD\_COUNTS.get)  
  
def correct\_match(match):  
 *"""  
 Spell-correct word in match,   
 and preserve proper upper/lower/title case.  
 """* word = match.group()  
  
 def case\_of(text):  
 *"""  
 Return the case-function appropriate   
 for text: upper, lower, title, or just str.:  
 """* return (str.upper if text.isupper() else  
 str.lower if text.islower() else  
 str.title if text.istitle() else  
 str)  
  
 return case\_of(word)(correct(word.lower()))  
  
def correct\_text\_generic(text):  
 *"""  
 Correct all the words within a text,   
 returning the corrected text.  
 """* return re.sub('[a-zA-Z]+', correct\_match, text)  
  
print(correct\_text\_generic('fianlly'))  
print(correct\_text\_generic('additioanl'))

**Results**



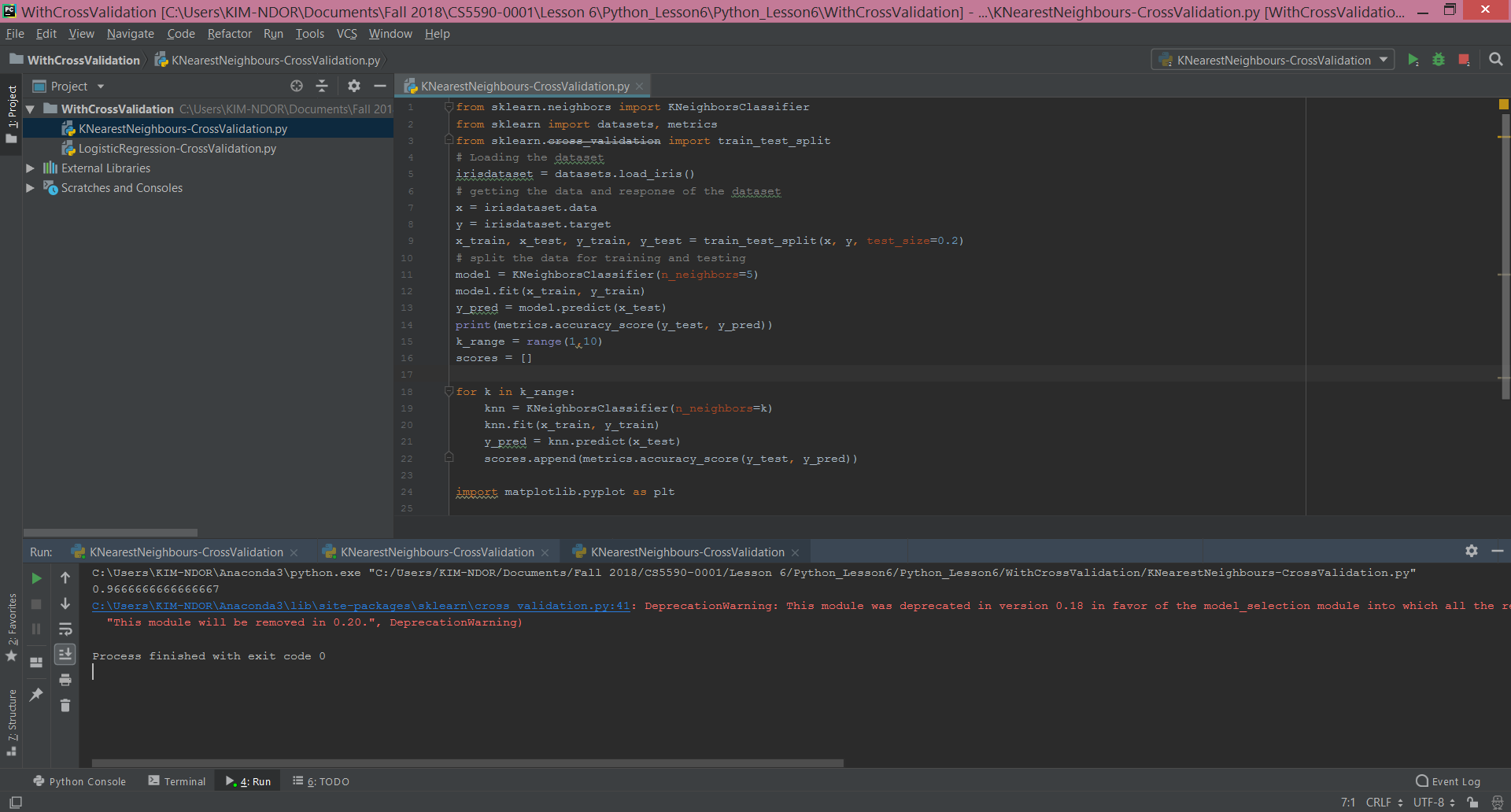
**Question 4**

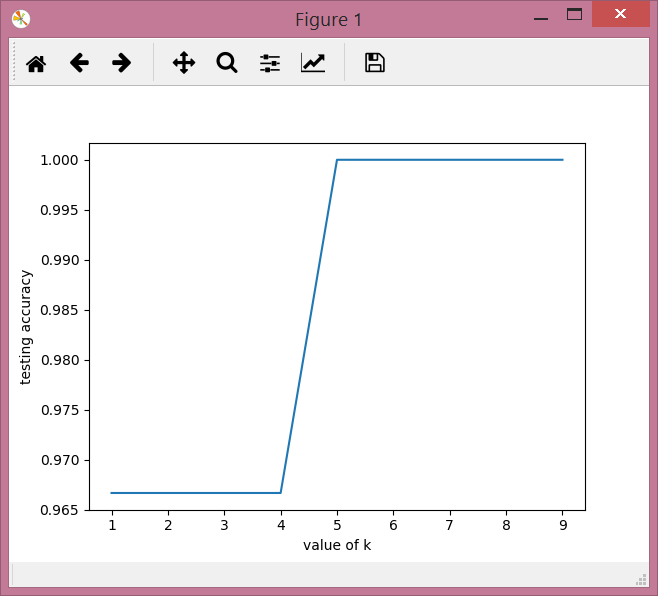
Report your views on the k nearest neighbor algorithm when we change the K how it will affect the accuracy. Provide a good justification for the changes of the accuracy when we change the amount of K. For example: compare the accuracy when K=1 and K is a big number like 50, why the accuracy will change.

from sklearn.neighbors import KNeighborsClassifier  
from sklearn import datasets, metrics  
from sklearn.cross\_validation import train\_test\_split  
#from sklearn import cross\_validation  
irisdataset = datasets.load\_iris()  
# getting the data and response of the dataset  
x = irisdataset.data  
y = irisdataset.target  
#x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2)  
#x\_train, x\_test, y\_train, y\_test =cross\_validation.train\_test\_split(x, y, test\_size=0.2)  
split the data for training and testing  
model = KNeighborsClassifier(n\_neighbors=5)  
model.fit(x\_train, y\_train)  
y\_pred = model.predict(x\_test)  
  
print(metrics.accuracy\_score(y\_test, y\_pred))  
k\_range = range(1, 20)  
scores = []  
  
for k in k\_range:  
 knn = KNeighborsClassifier(n\_neighbors=k)  
 knn.fit(x\_train, y\_train)  
 y\_pred = knn.predict(x\_test)  
 scores.append(metrics.accuracy\_score(y\_test, y\_pred))

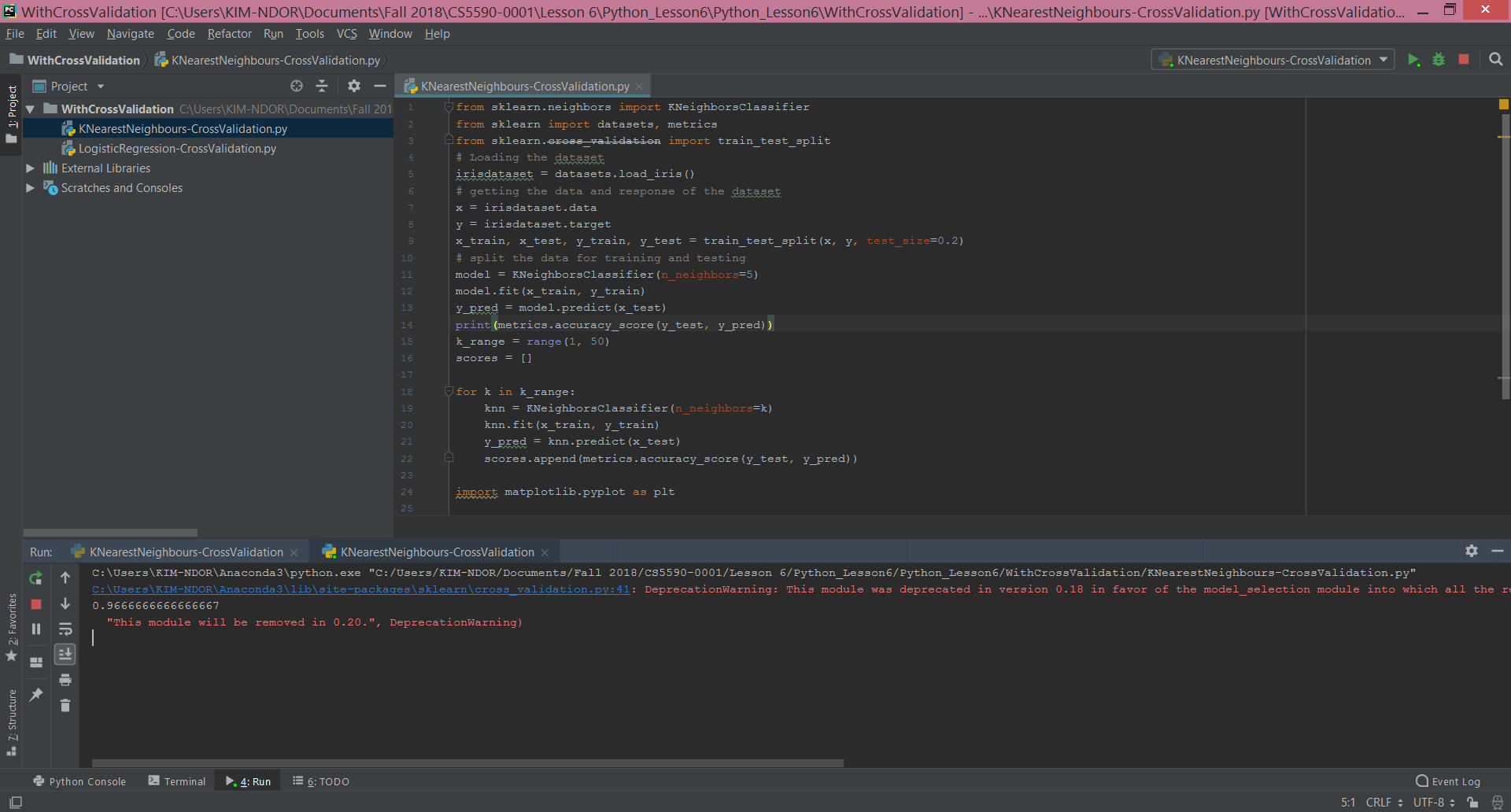
**Results**

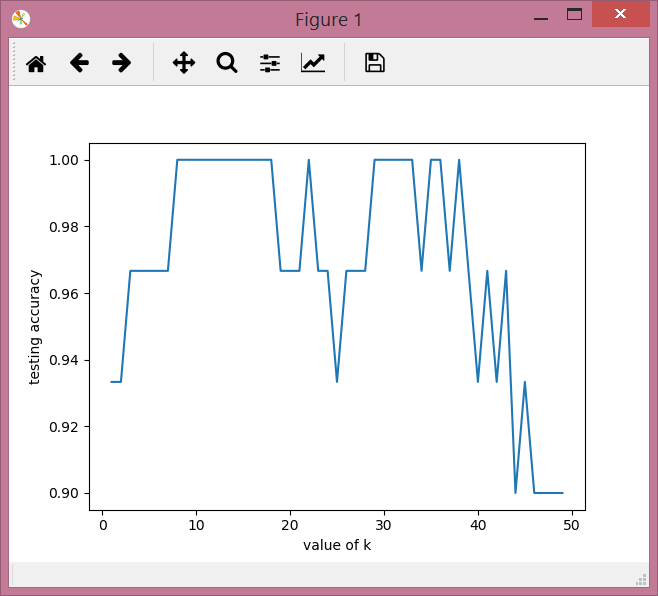
**K = 10**





**K = 50**





The accuracy is higher when k = 1 than k = 50. The accuracy increases from k = 1 to about k = 20. It oscillated from k = 20 to k = 40. Then decreases to k = 50 +.

**References**

Python Exercises, Practice, Solution at <http://www.w3resource.com/python-exercises/> (accessed on Oct. 10, 2018).

Learningpython.org at <https://www.learnpython.org/> (accessed on Oct. 9, 2018).

Medium at <https://medium.com/@sifium/machine-learning-types-of-classification-9497bd4f2e14> (accessed on Oct. 10, 2018).

Essentials of Machine Learning Algorithms (with Python and R Codes) at <https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/> (accessed on Oct. 10, 2018).