HW 1

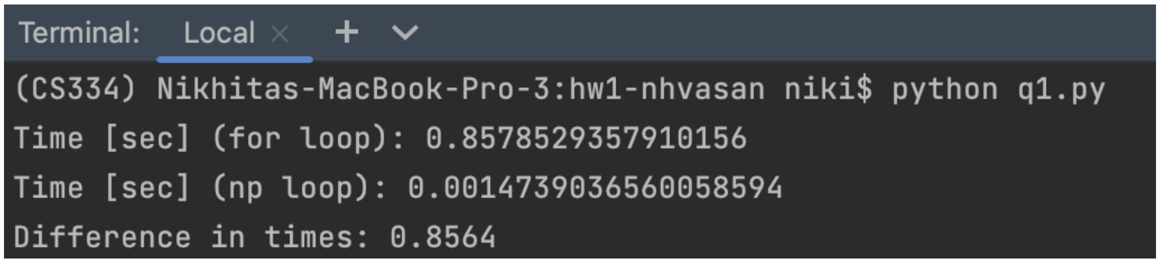
Q1: Numerical Programming

a. See q1.py in hw1-nhvasan.zip

b. See q1.py in hw1-nhvasan.zip

c. See q1.py in hw1-nhvasan.zip

d. The vectorized approach was faster than the first approach 0.8564 seconds faster than the for loop approach. See q1.py for further details.



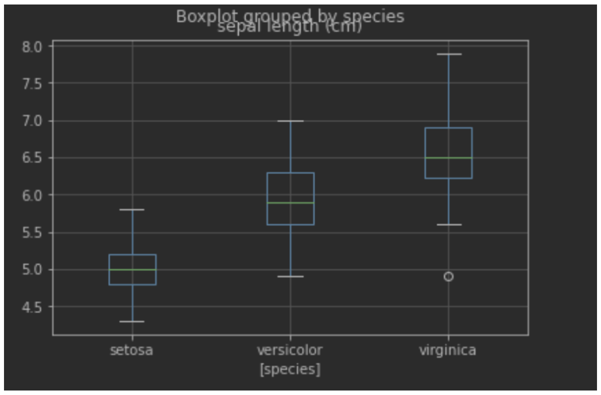
```
(CS334) Nikhitas-MacBook-Pro-3:hw1-nhvasan niki$ python q1.py
Time [sec] (for loop): 0.8578529357910156
Time [sec] (np loop): 0.0014739036560058594
Difference in times: 0.8564
```

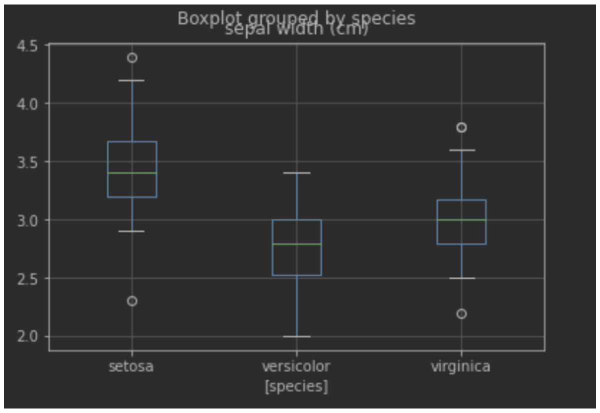
Q2: Visualization Exploration

a. See q2.py in hw1-nhvasan.zip

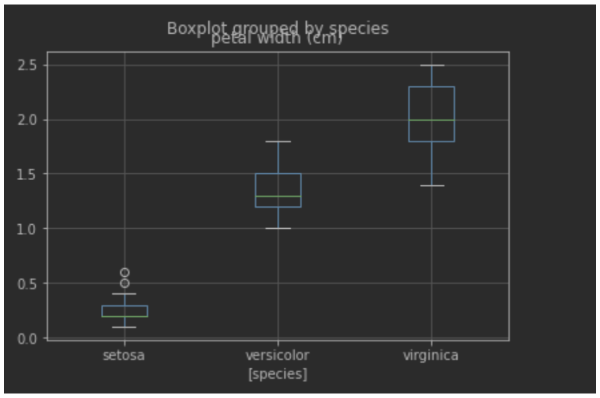
b. **Plots:** Distribution of Features by Species

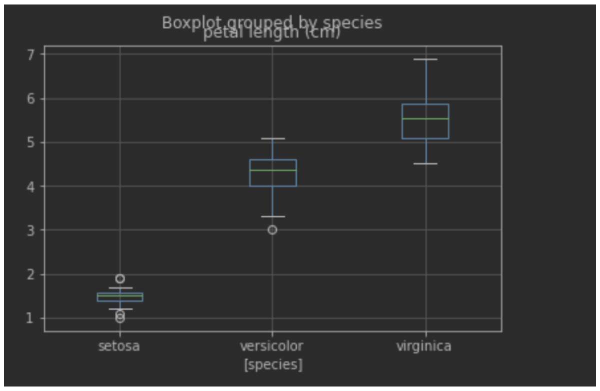
Sepal Length (cm) by Species **Sepal Width (cm) by Species**





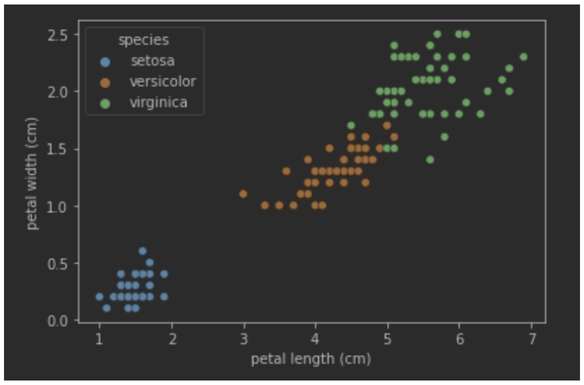
Petal Length (cm) by Species **Petal Width (cm) by Species**

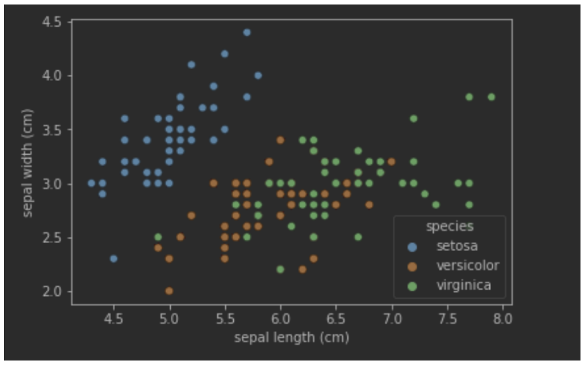




c. **Plots:** Petal and Sepal Distribution by Feature

Petal Width and Length **Sepal Width and Length**





d. Here is a set of “rules” that could classify species type:

- If the sepal length is small (~5.0cm), the sepal width is large (~3.4cm), but the petal length and width are small (~1.5cm and 0.25cm respectively), then the flower is an I**ris Setosa**.
- If the sepal length is medium (~5.8cm), the sepal width the small (~2.8cm), but the petal length and width are medium-sized (~4.5cm and ~1.25cm respectively), then the flower is an **Iris Versicolor**.

- If the sepal length is large (~6.5cm), the sepal width is medium-sized (~3.0cm), and the petal length and width are also large (~5.5cm and ~2.0cm respectively), then the flower is an **Iris Virginica.**

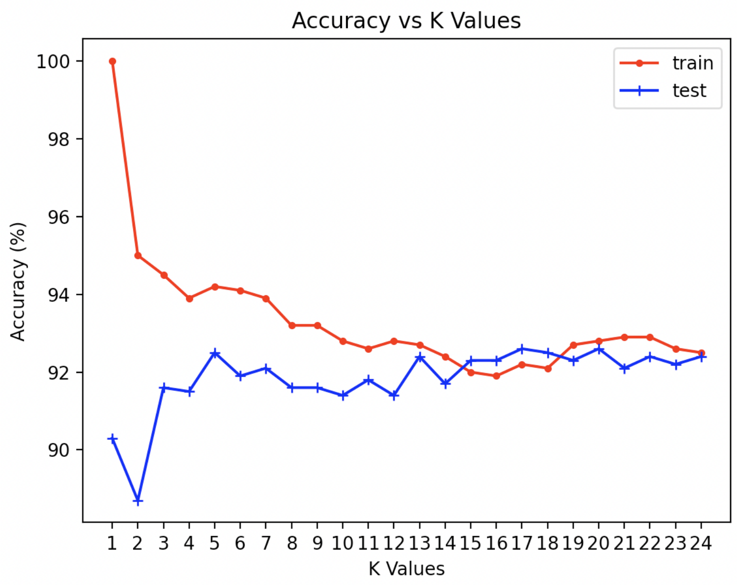
Q3: KNN Implementation

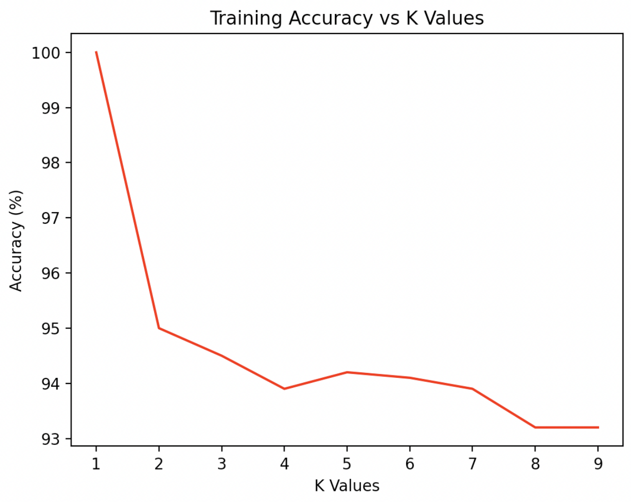
a. See knn.py in hw1-nhvasan.zip

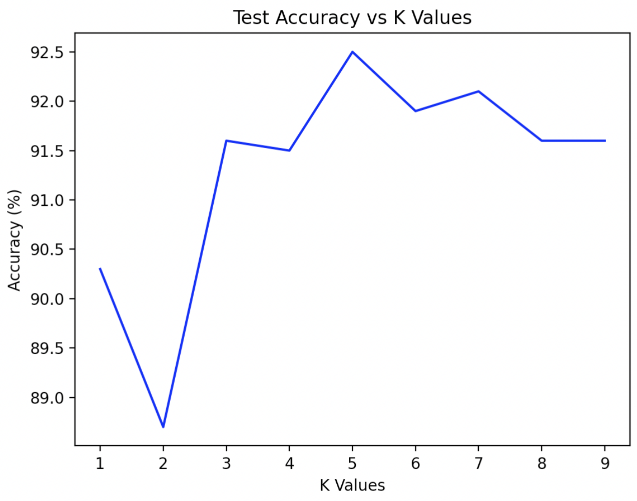
b. See knn.py in hw1-nhvasan.zip

c. See knn.py in hw1-nhvasan.zip

d. **Plots**







The first plot shows a comparison of the two side by side with k values on the range [1-25]. We can see that after around k=20, the test and training accuracies start to converge, which is why I picked the range [1-25] for this plot. The second and third plots zoom in on the test and training accuracy individually on a smaller range, 1-10.

We can see that the training accuracy decreases as the number of neighbors considered in the algorithm increases. This makes sense, because predictions are made by averaging across neighbors. If the algorithm is being tested on the same set is was trained on, k=1 should have 100% accuracy while k=10 should have the lowest accuracy (second plot).

The test accuracy has more fluctuations depending on different k-values [1-10]. We can see that k=5 appears to yield the highest overall test accuracy while k=2 yields the lowest test accuracy for this dataset (third plot).

e.

The computation complexity of my predict function is $O(nd)$ + $O(nlogn)$.

First, I loop over each row in the test data. For each observation, I calculate the euclidean distance between the new observation and each point in the training array. Since each distance calucation is $O(d)$, the complexity of this step is $O(nd)$.

Then, I sort the distances in ascending order using `np.argsort`, which is an $O(nlogn)$ function and keep the k nearest neighbors of the given test observation. This is an $O(1)$ function.

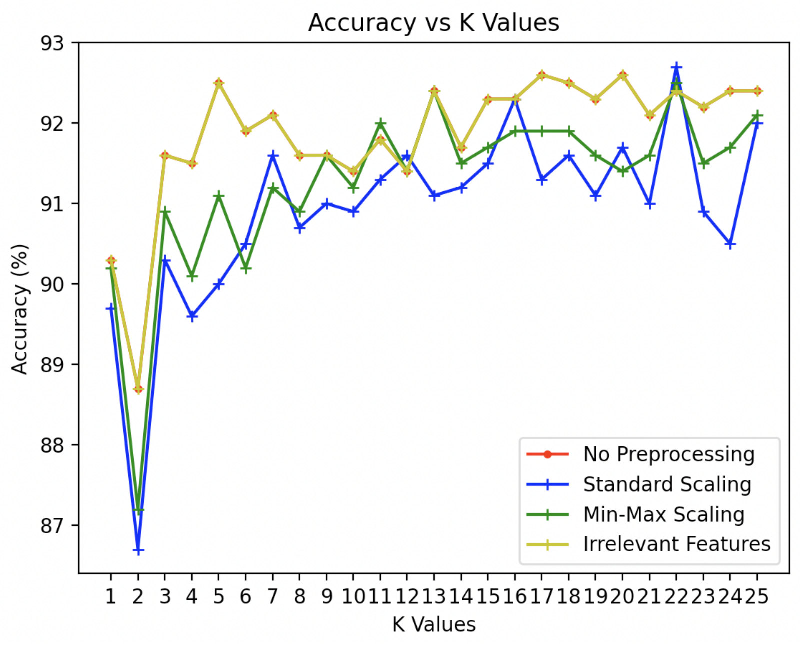
Q4: KNN Performance

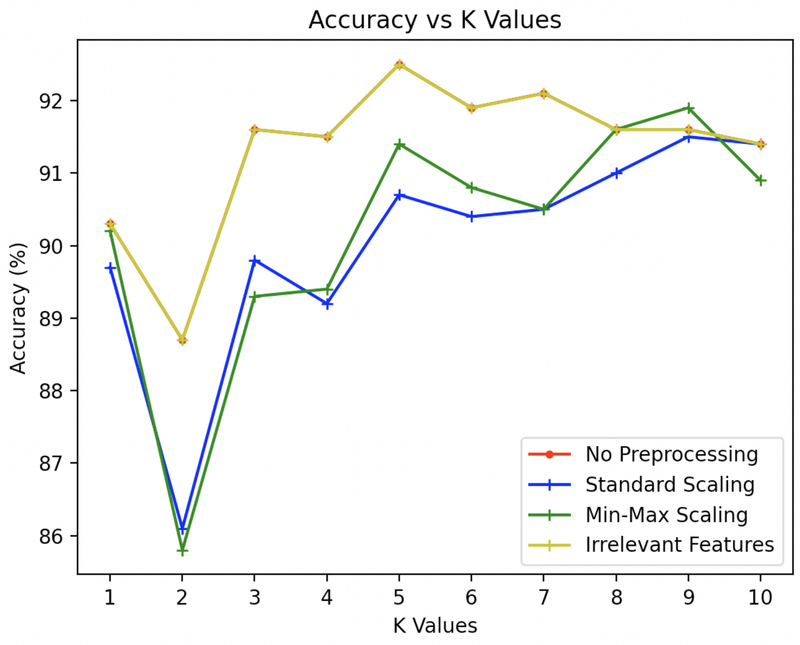
a. See q4.py in hw1-nhvasan.zip

b. See q4.py in hw1-nhvasan.zip

c. See q4.py in hw1-nhvasan.zip

d. **Plots**





These plots demonstrate the test accuracy of my KNN implementation using three different forms of preprocessing (no preprocessing, standard scaling and min-max scaling) and irrelevant features.

We can see that the model is insensitive towards noise in this dataset, as the test accuracy for the irrelevant features follows that of the no preprocessing model exactly. In terms of the two pre-processing methods tested, we see that they result in relatively similar accuracies over the different k values.

Based off of the top plot where k is in the range 1-25, we can see that the Min-Max scaling overall yields a higher test accuracy. However, from k = 1 to k = 4, the two methods yield very similar results, as we can see highlighted in the bottom plot.