DAT405 Assignment 2 – Group 53

Venkata Sai Dinesh Uddagiri - (10 hrs)
Mudhumitha Venkatesan (10 hrs)

November 15, 2022

Problem 1

a. Find a linear regression model that relates the living area to the selling price. If you did any data cleaning step(s), describe and explain why you did that.

```
import pandas as pnd
#Import matplotlib to plot the linear regression model
import matplotlib.pyplot as plt
#Import LinearRegression model
from sklearn.linear_model import LinearRegression
#Reading csv file in to data frame
livingArea_sellingPrice_raw=pnd.read_csv("data_assignment2.csv")
#Checking for any neagtive or zero values in Living_area and Selling_Price coloumns
livingArea_sellingPrice=(livingArea_sellingPrice_raw[(livingArea_sellingPrice_raw['
    Living_area'] > 0)&(livingArea_sellingPrice_raw['Selling_price'] >= 0)]).dropna(
    subset=['Living_area','Selling_price'])
x_living_area=livingArea_sellingPrice['Living_area'].values.reshape(-1,1)
y_selling_price=livingArea_sellingPrice['Selling_price'].values.reshape(-1,1)
#Generating a linear regression model
model=LinearRegression().fit(x_living_area,y_selling_price)
#Plot the Scatter plot between Living_area and Selling_price
plt.scatter(x_living_area,y_selling_price)
#Plot Linear regression line
plt.plot(x_living_area,model.predict(x_living_area),c='r',label='Regression line')
#Declare labels and titles for the plot
plt.title("living area vs selling price", fontsize = 18)
plt.xlabel("Living area (m^2)", fontsize = 18)
plt.ylabel("Selling price (millions)", fontsize = 18)
plt.legend()
plt.show()
```

Listing 1: Python code – Linear regression model.

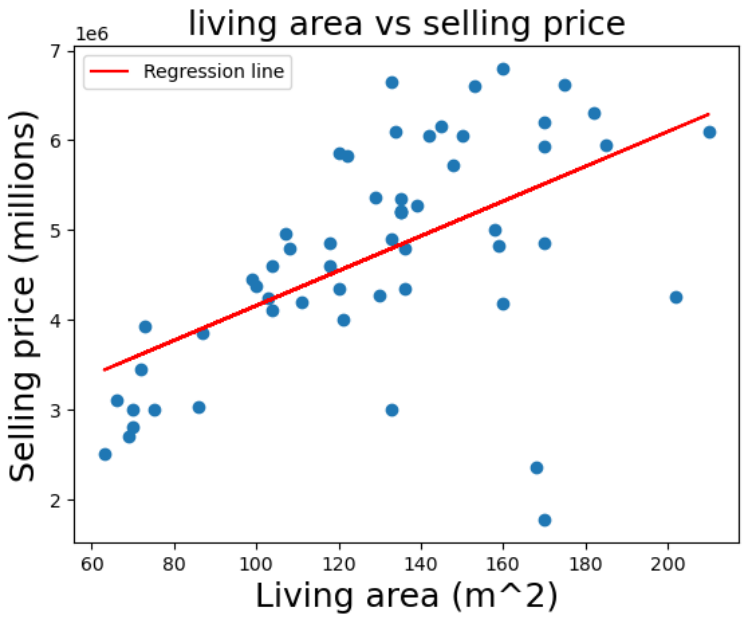


Figure 1: Plot of linear regression model that relates the living area to the selling price

By observing the info of raw data frame, we can see that entries in Living_area, Selling_price are equal to ID's column. This observation allows us to conclude that the Living area and Selling price columns do not contain any null values. However, there's a potential that the data could contain negative or zero numbers. So, we have added the check in the code.

b. What are the values of the slope and intercept of the regression line?

```
#Slope of Linear regression
print("Slope of Linear regression line: ", model.coef_ )
#Intercept of Linear regression
print("Intercept of Linear regression line: ", model.intercept_)
```

Listing 2: Python code – Linear regression model slope and intercept.

The values of slope and intercept of the regression line is 19370.13854733 and 2220603.24335587.

c. Use this model to predict the selling prices of houses which have living area 100 m2, 150 m2 and 200 m2

```
import numpy as np
living_areas=np.array([100, 150, 200]).reshape(-1,1)
# Selling price predection for given living areas
predicted_selling_price=model.predict(living_areas)
print("Predicted selling price of house with living area 100 m2 is", float(
    predicted_selling_price[0]))
print("Predicted selling price of house with living area 150 m2 is", float(
    predicted_selling_price[1]))
print("Predicted selling price of house with living area 200 m2 is", float(
    predicted_selling_price[2]))
```

Listing 3: Python code – Linear regression model- Predict the selling prices of houses.

The Predicted selling prices of house with living areas 100 m2, 150 m2, 200 m2 is 4157617.0980890268 SEK, 5126124.025455605 SEK, 6094630.952822184 SEK respectively.

d. Draw a residual plot.

```
#Creation of the Residual plot for Linear regression model
sns.residplot(x = x_live_area , y = y_sell_price, data = livingArea_sellingPrice_clean
    )
plt.xlabel("Living area (m^2)", fontsize = 18)
plt.ylabel("Residual errors", fontsize = 18)
plt.show()
```

Listing 4: Python code – Linear regression model- Residual plot.

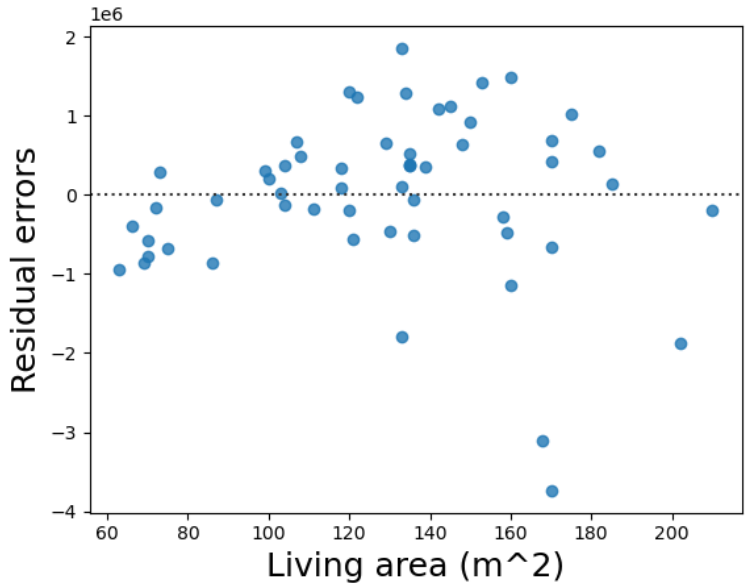


Figure 2: Residual plot of linear regression model

e. Discuss the results, and how the model could be improved.

When the living area was small, there were numerous residual points below the zero line, and when the living area was medium, there were many residual points above the zero line. Additionally, we may see some residual locations that are outliers and have a high living space but a low selling price. We are unsure if the linear regression model is the best model in light of the aforementioned points.

However, we are unsure if we can classify them as outliers or not because the selling price of the home also depends on a number of other variables, such the age of the structure, the region in which it is located, the amenities nearby, etc.

We need to eliminate the outliers without taking the case into account in order to improve the linear regression model in one method.Additionally, taking into account the possibility of including extra

features that one could anticipate having an impact on the selling prices may help the model perform better. One of the instances is taking age into account when setting the selling price.

Problem 2

a. Use a confusion matrix to evaluate the use of logistic regression to classify the iris data set.

```
#To get access to a iris dataset
from sklearn.datasets import load_iris
#Import train_test_split function
from sklearn.model_selection import train_test_split
#Import LogisticRegression model
from sklearn.linear_model import LogisticRegression
#Import Confusion matrix
from sklearn.metrics import confusion_matrix
#Import matplotlib to plot the confusion matrix
import matplotlib.pyplot as plt
#Import seaborn package for adding additional graphics to plot
import seaborn as sns
#Loading iris dataset
iris_dataSet=load_iris()
#Splitting data in to test and train sets
x_train, x_test, y_train, y_test = train_test_split(iris_dataSet.data, iris_dataSet.
    target, test_size=0.3, random_state=4)
#Create instance of model
iris_logisticRegr = LogisticRegression(multi_class='ovr', solver='liblinear')
#Train the Logistic Regression
iris_logisticRegr.fit(x_train, y_train)
#Predict the values for x_test set
test_predictions = iris_logisticRegr.predict(x_test)
#Evaluate the accuracy score of Logistic Regression
score = iris_logisticRegr.score(x_test, y_test)
#Create the confusion matrices betweeen original value and pridected valuee of x_test
iris_logistic_regression_cm = confusion_matrix(y_test, test_predictions)
#size of the figure
plt.figure(figsize=(6,6))
#Obtaining confusion matrix using sns
sns.heatmap(iris_logistic_regression_cm, annot=True, fmt=".3f", linewidths=3, square =
     True, xticklabels=iris_dataSet.target_names, yticklabels=iris_dataSet.target_names
    , cmap = 'OrRd');
# Declaring labels and title for confusion matrix
plt.ylabel('Actual label\n');
plt.xlabel('\n Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title("Iris Logistic Regression Confusion Matrix \n\n"+all_sample_title, size =
    15);
plt.show()
```

Listing 5: Python code – Logistic regression to classify the iris data set.

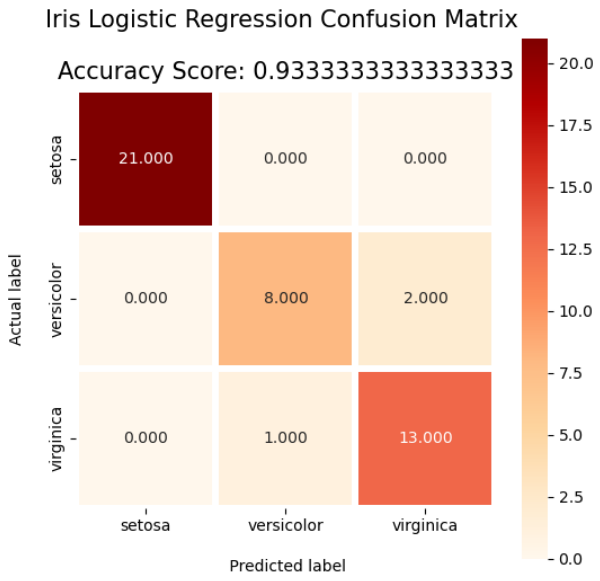


Figure 3: Residual plot of linear regression model

In above confusion matrix with random_state=4 for splitting the train and test data, setosa has predicted with 100 percent accuracy, but versicolor and virginica is not accurately detected. Total accuracy of model with random_state=4 is 0.933333333.

b. Use k-nearest neighbours to classify the iris data set with some different values for k, and with uniform and distance-based weights. What will happen when k grows larger for the different cases? Why?

```
#To get access to a iris dataset
from sklearn.datasets import load_iris
#Import train_test_split function
from sklearn.model_selection import train_test_split
#Import KNeighborsClassifier model
from sklearn.neighbors import KNeighborsClassifier
#Import Confusion matrix
from sklearn.metrics import confusion_matrix
#Import matplotlib to plot the confusion matrix
import matplotlib.pyplot as plt
#Import seaborn package for adding additional graphics to plot
import seaborn as sns
#Loading iris dataset
iris_dataSet=load_iris()
test_value=0.3
#Splitting data in to test and train sets
x_train, x_test, y_train, y_test = train_test_split(iris_dataSet.data, iris_dataSet.
    target, test_size=test_value, random_state=4)
#Calculating maximum K value - length of data set excluding test data set
max_kValue=int(iris_dataSet.data.shape[0] - iris_dataSet.data.shape[0]*test_value)
#List of weights of k nearest neighbour
weights = ['uniform' , 'distance']
#List intializations to append accuracy score for different k values for different
    weights
distance_accuracy_list=[]
```

```
unifrom_accuracy_list=[]
for weight in weights:
    for k in range(1,max_kValue+1):
        #Create instance of k nearest neighbor model
        knn = KNeighborsClassifier(n_neighbors=k, weights=weight)
        #Train k nearest neighbor model
        knn.fit(x_train, y_train)
        #Predict the values for x_test set using k nearest neighbor model
        test_predictions = knn.predict(x_test)
        if weight=='uniform':
            #Evaluate the accuracy score for each value of k and adding data to list
            unifrom_accuracy_list.append(metrics.accuracy_score(y_test,
    test_predictions))
        else:
            #Evaluate the accuracy score for each value of k and ading data to list
            distance_accuracy_list.append(metrics.accuracy_score(y_test,
    test_predictions))
#Create Plot of k-nearest neighbours with uniform weights
plt.plot(list(range(1,max_kValue+1)),unifrom_accuracy_list, color = 'red')
#Create Plot of k-nearest neighbours with distance weights
plt.plot(list(range(1,max_kValue+1)),distance_accuracy_list, color = 'blue')
# Declaring labels and title for plot
plt.legend(weights, loc = "lower left")
plt.xlabel('K value')
plt.ylabel('accuracy_score')
plt.title('Accuracy score analysis with uniform and distance-based weights for
    diffrent K value\n')
plt.show()
```

Listing 6: Python code – Accuracy score analysis with uniform and distance-based weights for diffrent K value.

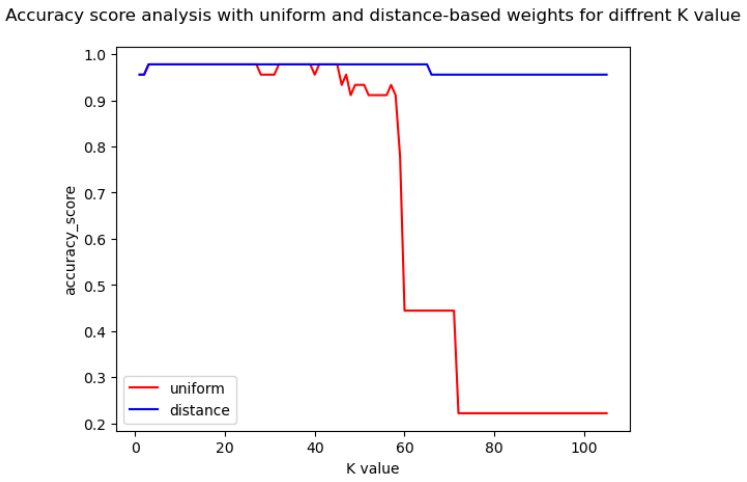


Figure 4: Accuracy score analysis plot with uniform and distance-based weights for diffrent K value

By looking at the graph, accuracy score analysis with uniform and distance-based weights for various K values—differ. The accuracy score of a uniform k-nearest neighbors is good for low k values but declines at high K values. While the distance k-nearest neighbors model's accuracy score does not significantly alter when different k values are used. Therefore, it makes sense that allocating weights depending on distance (giving the highest weight to points with the shortest Euclidean distance and lowest weight to point with largest Euclidean distance) would increase the model's accuracy.

c. Compare the classification models for the iris data set that are generated by k-nearest neighbours (for the different settings from question 3) and by logistic regression. Calculate confusion matrices for these models and discuss the performance of the various models.

```
#To get access to a iris dataset
from sklearn.datasets import load_iris
#Import train_test_split function
from sklearn.model_selection import train_test_split
#Import KNeighborsClassifier model
from sklearn.neighbors import KNeighborsClassifier
#Import Confusion matrix
from sklearn.metrics import confusion_matrix
#Import matplotlib to plot the confusion matrix
import matplotlib.pyplot as plt
#Import seaborn package for adding additional graphics to plot
import seaborn as sns
#Loading iris dataset
iris_dataSet=load_iris()
#Splitting data in to test and train sets
x_train, x_test, y_train, y_test = train_test_split(iris_dataSet.data, iris_dataSet.
    target, test_size=0.3, random_state=4)
#K values list to obtain confusion matrices
kValue=[3 , 35, 65, 100]
#List of weights of k nearest neighbour
weights = ['distance','uniform']
#List intializations to append test predections for different k values for different
    weights
distance_test_predictions=[]
uniform_test_predictions=[]
for weight in weights:
    for i in range(len(kValue)):
        #Create instance of k nearest neighbor model
        knn = KNeighborsClassifier(n_neighbors=kValue[i], weights=weight)
         #Train k nearest neighbor model
        knn.fit(x_train, y_train)
        #Predict the values for x_test set using k nearest neighbor model
        test_predictions = knn.predict(x_test)
        if weight=='uniform':
            #Evaluate the predicted value of x_test using k nearest neighbor uniform
    weight model
            uniform_test_predictions.append(test_predictions)
        else:
            #Evaluate the predicted value of x_test using k nearest neighbor distance
    weight model
            distance_test_predictions.append(test_predictions)
for i in range(len(kValue)):
    #plot confusion matrices for different k values
    fig, ((ax1, ax2)) = plt.subplots(1, 2,figsize=(10,10))
    cm_uniform = metrics.confusion_matrix(y_test, uniform_test_predictions[i])
    #Obtaining plot of confusion matrix for k nearest neighbor uniform weight model
    using sns
    sns.heatmap(cm_uniform, ax = ax1, annot=True, fmt=".3f", linewidths=.5, square =
    True, xticklabels=iris_dataSet.target_names, yticklabels=iris_dataSet.target_names,
     cmap = 'OrRd')
    cm_distance = metrics.confusion_matrix(y_test, distance_test_predictions[i])
    #Obtaining plot of confusion matrix for k nearest neighbor uniform distance model
    using sns
    sns.heatmap(cm_distance, ax = ax2, annot=True, fmt=".3f", linewidths=.5, square =
    True, xticklabels=iris_dataSet.target_names, yticklabels=iris_dataSet.target_names,
     cmap = 'OrRd')
    # Declaring labels and title for each subplot
```

```
    ax1.set_xlabel('Predicted label')
    ax1.set_ylabel('Actual label')
    ax1.set_title('Uniform weight k='+ str(kValue[i])+"\n" +'Accuracy Score ='+str(
    metrics.accuracy_score(y_test, uniform_test_predictions[i])), size=10)
    ax2.set_xlabel('Predicted label')
    ax2.set_ylabel('Actual label')
    ax2.set_title('Distance weight k='+ str(kValue[i])+"\n" +'Accuracy Score ='+str(
    metrics.accuracy_score(y_test, distance_test_predictions[i])), size=10)
    plt.show()
```

Listing 7: Python code – To Obtain confusion matrices for k-nearest neighbours uniform and distance-based weights.

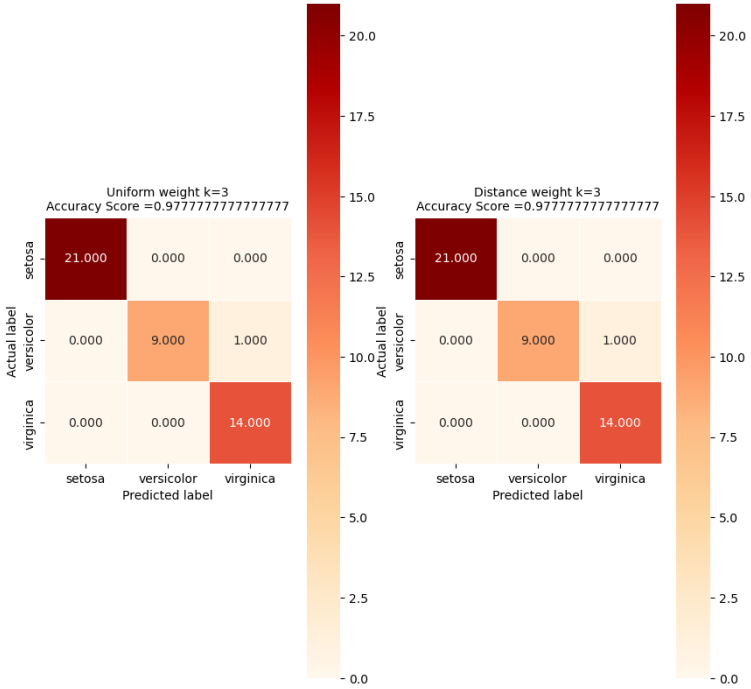


Figure 5: Knn confusion matrices with K=3

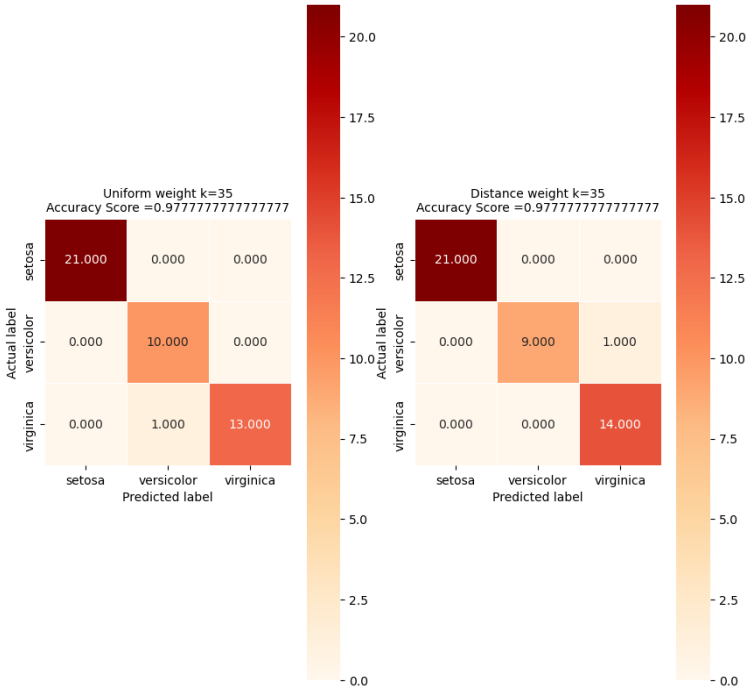


Figure 6: Knn confusion matrices with K=35

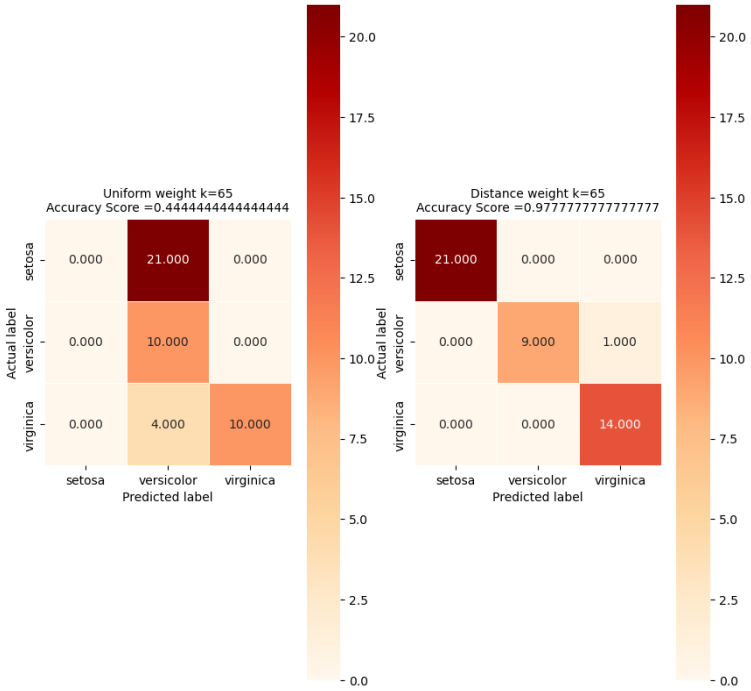


Figure 7: Knn confusion matrices with K=65

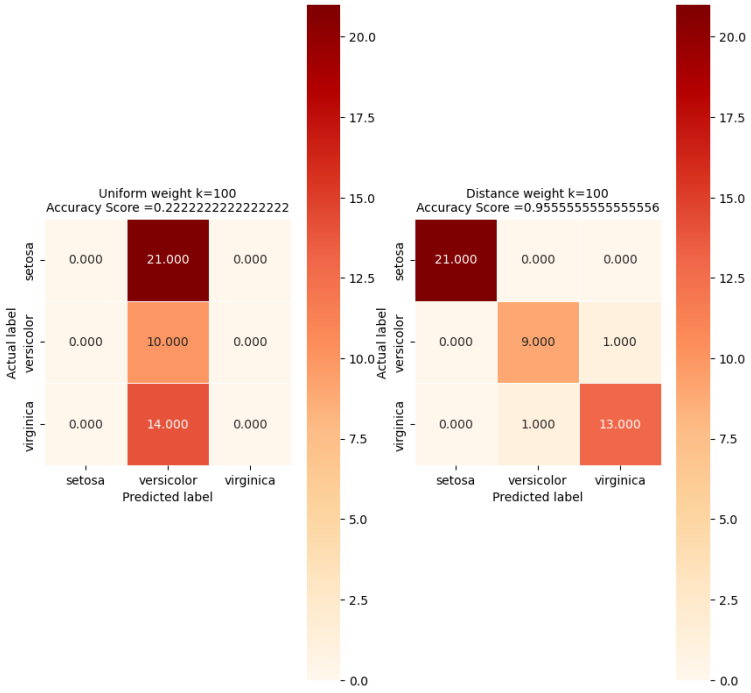


Figure 8: Knn confusion matrices with K=100

After looking at the confusion matrices for the k-nearest neighbor method using uniform, distance-based, and logistic regression. k-nearest neighbours with distance-based weights, given the results with high accuracy.Whereas accuracy score drops with increasing k values for k-nearest neighbors with uniform weights, as seen in plot of 2, (b). However, k-nearest neighbors with the lowest k values in both the uniform and distance-based weights have produced findings that are more precise than logistic regression model. The random state=4 is used to separate the train and test data in the logistic and k-nearest neighbor models. In order to compare accuracy with different test data we have modified the random state value. We found that the k-nearest neighbors with distance-based weights provided results that were more accurate than the logistic and k-nearest neighbors with uniform weights for any random state value. So, based on this, we conclude that the optimum model for classifying the iris data set is k-nearest neighbors with distance-based weights.

Problem 3

3. Explain why it is important to use a separate test (and sometimes validation) set.

The test data are used to provide data against which the model can be tested, which is essential for evaluating the model. Since the model is making predictions based on data it already possesses, using the same data for training and testing eliminates the need for testing. We may verify the model's effectiveness by test data sets. Considering that the trained model has no knowledge of the test set. This allows us to monitor the overfitting/underfitting problems and reduce them by adjusting the train data set.

assignment2

November 15, 2022

0.0.1 DAT405 Assignment 2 – Group 53

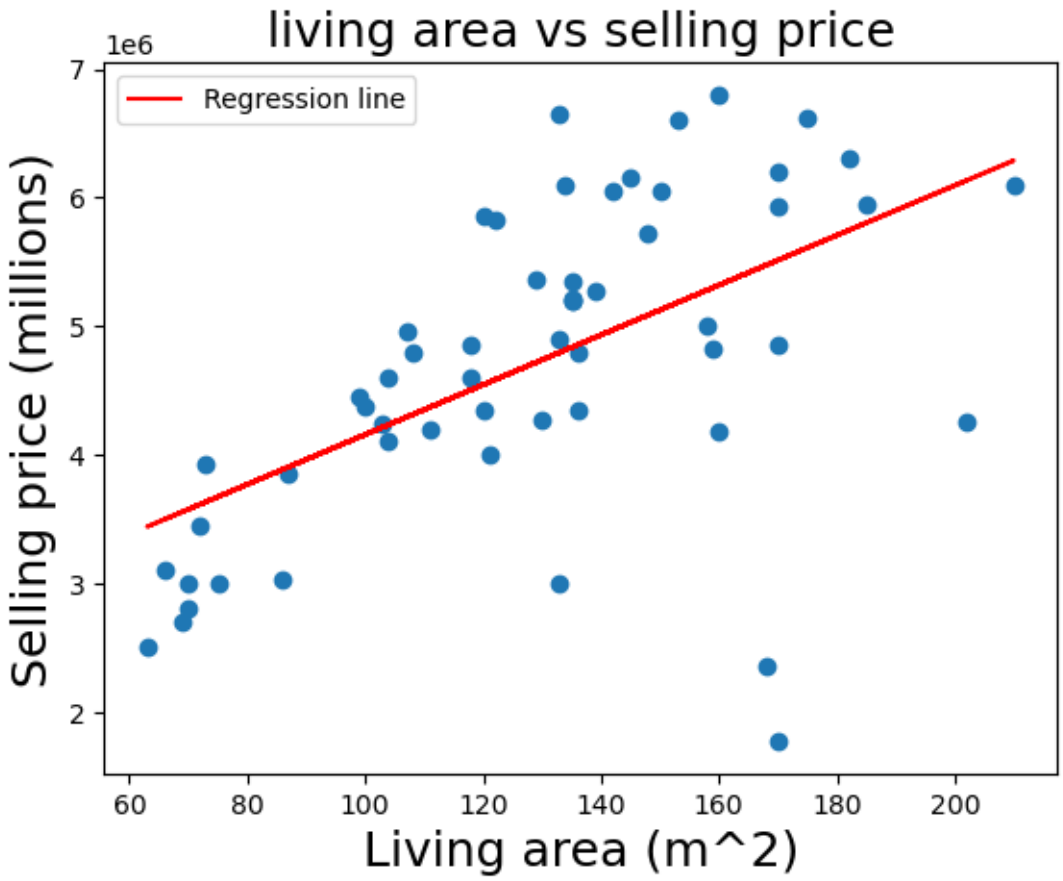
0.0.2 Venkata Sai Dinesh Uddagiri - (10 hrs)

0.0.3 Mudhumitha Venkatesan (10 hrs)

1 Problem 1

a. Find a linear regression model that relates the living area to the selling price. If you did any data cleaning step(s), describe and explain why you did that.

[1]:
```
import pandas as pnd
#Import matplotlib to plot the linear regression model
import matplotlib.pyplot as plt
#Import LinearRegression model
from sklearn.linear_model import LinearRegression
#Reading csv file in to data frame
livingArea_sellingPrice_raw=pnd.read_csv("data_assignment2.csv")
#Checking for any neagtive or zero values in Living_area and Selling_Price␣
 ↪coloumns
livingArea_sellingPrice=(livingArea_sellingPrice_raw[(livingArea_sellingPrice_raw['Living_area
 ↪> 0)&(livingArea_sellingPrice_raw['Selling_price'] >= 0)]).
 ↪dropna(subset=['Living_area','Selling_price'])
x_living_area=livingArea_sellingPrice['Living_area'].values.reshape(-1,1)
y_selling_price=livingArea_sellingPrice['Selling_price'].values.reshape(-1,1)
#Generating a linear regression model
model=LinearRegression().fit(x_living_area,y_selling_price)
#Plot the Scatter plot between Living_area and Selling_price
plt.scatter(x_living_area,y_selling_price)
#Plot Linear regression line
plt.plot(x_living_area,model.predict(x_living_area),c='r',label='Regression␣
 ↪line')
#Declare labels and titles for the plot
plt.title("living area vs selling price", fontsize = 18)
plt.xlabel("Living area (m^2)", fontsize = 18)
plt.ylabel("Selling price (millions)", fontsize = 18)
plt.legend()
plt.show()
```



[2]:
```
livingArea_sellingPrice_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56 entries, 0 to 55
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   ID             56 non-null     int64
 1   Living_area    56 non-null     int64
 2   Rooms          54 non-null     float64
 3   Land_size      55 non-null     float64
 4   Biarea         32 non-null     float64
 5   Age            56 non-null     int64
 6   Selling_price  56 non-null     int64
dtypes: float64(3), int64(4)
memory usage: 3.2 KB
```

b. What are the values of the slope and intercept of the regression line?

[3]:
```
#Slope of Linear regression
print("Slope of Linear regression line: ", model.coef_ )
#Intercept of Linear regression
print("Intercept of Linear regression line: ", model.intercept_)
```

```
Slope of Linear regression line:  [[19370.13854733]]
Intercept of Linear regression line:  [2220603.24335587]
```

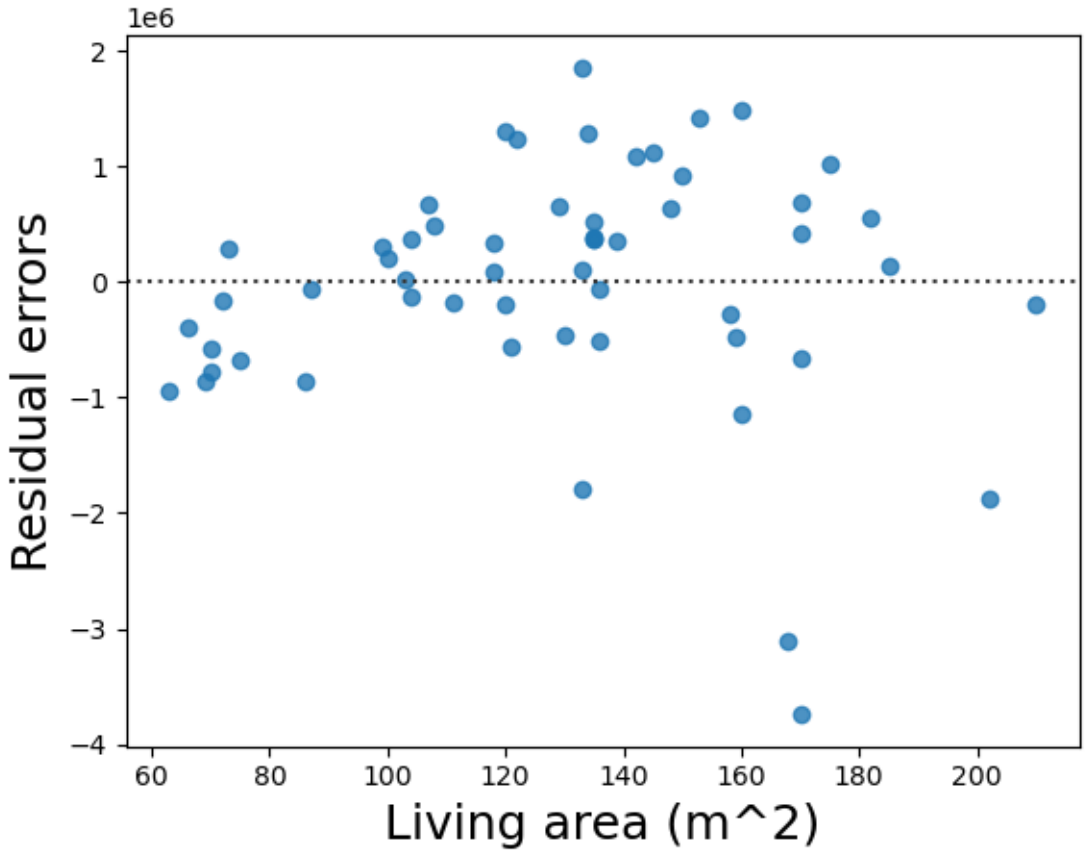
c. Use this model to predict the selling prices of houses which have living area 100 m2, 150 m2 and 200 m2.

[4]:
```
import numpy as np
living_areas=np.array([100, 150, 200]).reshape(-1,1)
# Selling price predection for given living areas
predicted_selling_price=model.predict(living_areas)
print("Predicted selling price of house with living area 100 m2 is",␣
 ↪float(predicted_selling_price[0]))
print("Predicted selling price of house with living area 150 m2 is",␣
 ↪float(predicted_selling_price[1]))
print("Predicted selling price of house with living area 200 m2 is",␣
 ↪float(predicted_selling_price[2]))
```

```
Predicted selling price of house with living area 100 m2 is 4157617.0980890268
Predicted selling price of house with living area 150 m2 is 5126124.025455605
Predicted selling price of house with living area 200 m2 is 6094630.952822184
```

d. Draw a residual plot.

[8]:
```
import seaborn as sns
#Creation of the Residual plot for Linear regression model
sns.residplot(x = x_living_area , y = y_selling_price, data =␣
 ↪livingArea_sellingPrice)
plt.xlabel("Living area (m^2)", fontsize = 18)
plt.ylabel("Residual errors", fontsize = 18)
plt.show()
```

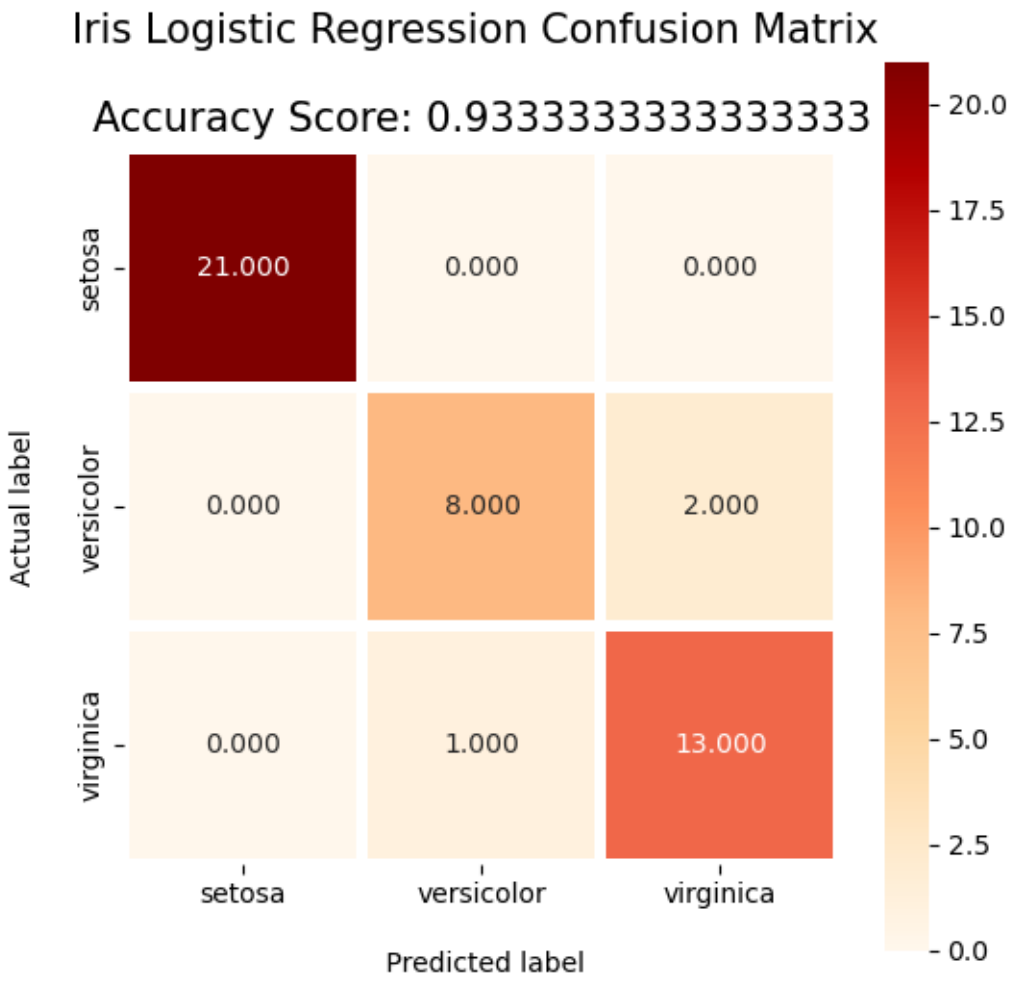


2 Problem 2

a. Use a confusion matrix to evaluate the use of logistic regression to classify the iris data set.

[9]:
```
#To get access to a iris dataset
from sklearn.datasets import load_iris
#Import train_test_split function
from sklearn.model_selection import train_test_split
#Import LogisticRegression model
from sklearn.linear_model import LogisticRegression
#Import Confusion matrix
from sklearn.metrics import confusion_matrix
#Import matplotlib to plot the confusion matrix
import matplotlib.pyplot as plt
#Import seaborn package for adding additional graphics to plot
import seaborn as sns
#Loading iris dataset
iris_dataSet=load_iris()
#Splitting data in to test and train sets
```

```
x_train, x_test, y_train, y_test = train_test_split(iris_dataSet.data,␣
 ↪iris_dataSet.target, test_size=0.3, random_state=4)
#Create instance of model
iris_logisticRegr = LogisticRegression(multi_class='ovr', solver='liblinear')
#Train the Logistic Regression
iris_logisticRegr.fit(x_train, y_train)
#Predict the values for x_test set
test_predictions = iris_logisticRegr.predict(x_test)
#Evaluate the accuracy score of Logistic Regression
score = iris_logisticRegr.score(x_test, y_test)
#Create the confusion matrices betweeen original value and pridected valuee of␣
 ↪x_test
iris_logistic_regression_cm = confusion_matrix(y_test, test_predictions)
#size of the figure
plt.figure(figsize=(6,6))
#Obtaining plot of confusion matrix using sns
sns.heatmap(iris_logistic_regression_cm, annot=True, fmt=".3f", linewidths=3,␣
 ↪square = True, xticklabels=iris_dataSet.target_names,␣
 ↪yticklabels=iris_dataSet.target_names, cmap = 'OrRd');
# Declaring labels and title for confusion matrix
plt.ylabel('Actual label\n');
plt.xlabel('\n Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title("Iris Logistic Regression Confusion Matrix \n\n"+all_sample_title,␣
 ↪size = 15);
plt.show()
```



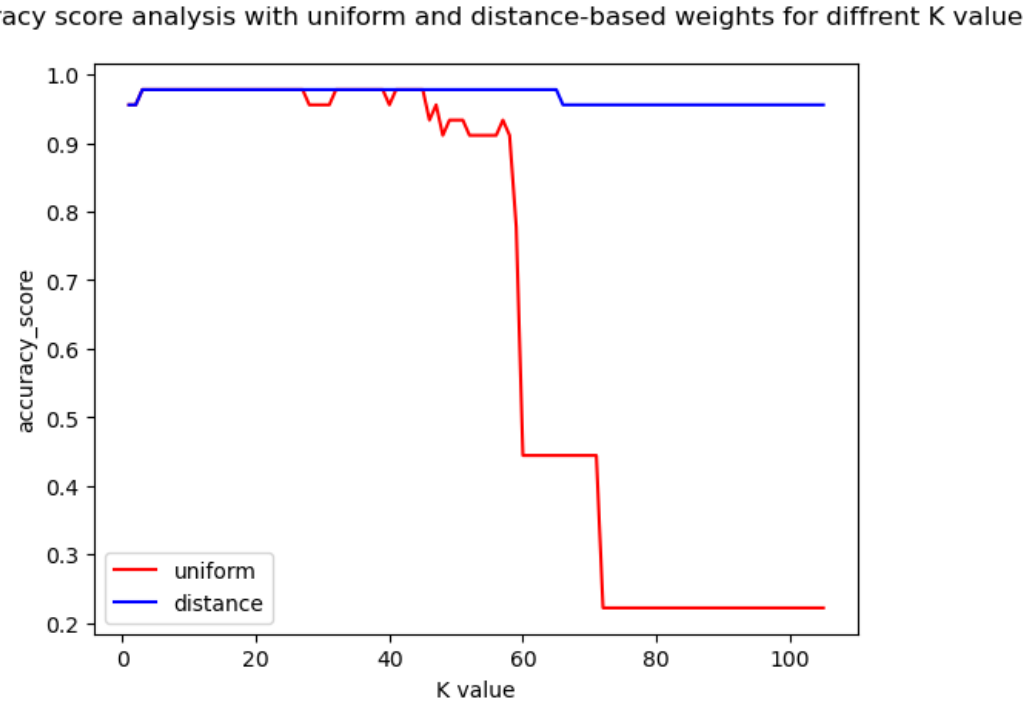
[10]:
```
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
```

b. Use k-nearest neighbours to classify the iris data set with some different values for k, and with uniform and distance-based weights. What will happen when k grows larger for the different cases? Why?

[12]:
```
#To get access to a iris dataset
from sklearn.datasets import load_iris
#Import train_test_split function
from sklearn.model_selection import train_test_split
#Import KNeighborsClassifier model
from sklearn.neighbors import KNeighborsClassifier
#Import Confusion matrix
from sklearn.metrics import confusion_matrix
```

```
#Import matplotlib to plot the confusion matrix
from sklearn import metrics
import matplotlib.pyplot as plt
#Import seaborn package for adding additional graphics to plot
import seaborn as sns
#Loading iris dataset
iris_dataSet=load_iris()
test_value=0.3
#Splitting data in to test and train sets
x_train, x_test, y_train, y_test = train_test_split(iris_dataSet.data,␣
 ↪iris_dataSet.target, test_size=test_value, random_state=4)
#Calculating maximum K value - length of data set excluding test data set
max_kValue=int(iris_dataSet.data.shape[0] - iris_dataSet.data.
 ↪shape[0]*test_value)
#List of weights of k nearest neighbour
weights = ['uniform' , 'distance']
#List intializations to append accuracy score for different k values for␣
 ↪different weights
distance_accuracy_list=[]
unifrom_accuracy_list=[]
for weight in weights:
    for k in range(1,max_kValue+1):
        #Create instance of k nearest neighbor model
        knn = KNeighborsClassifier(n_neighbors=k, weights=weight)
        #Train k nearest neighbor model
        knn.fit(x_train, y_train)
        #Predict the values for x_test set using k nearest neighbor model
        test_predictions = knn.predict(x_test)
        if weight=='uniform':
            #Evaluate the accuracy score for each value of k and adding data to␣
 ↪list
            unifrom_accuracy_list.append(metrics.accuracy_score(y_test,␣
 ↪test_predictions))
        else:
            #Evaluate the accuracy score for each value of k and ading data to␣
 ↪list
            distance_accuracy_list.append(metrics.accuracy_score(y_test,␣
 ↪test_predictions))
#Create Plot of k-nearest neighbours with uniform weights
plt.plot(list(range(1,max_kValue+1)),unifrom_accuracy_list, color = 'red')
#Create Plot of k-nearest neighbours with distance weights
plt.plot(list(range(1,max_kValue+1)),distance_accuracy_list, color = 'blue')
# Declaring labels and title for plot
plt.legend(weights, loc = "lower left")
plt.xlabel('K value')
plt.ylabel('accuracy_score')
```

```
plt.title('Accuracy score analysis with uniform and distance-based weights for␣
  ↪diffrent K value\n')
plt.show()
```

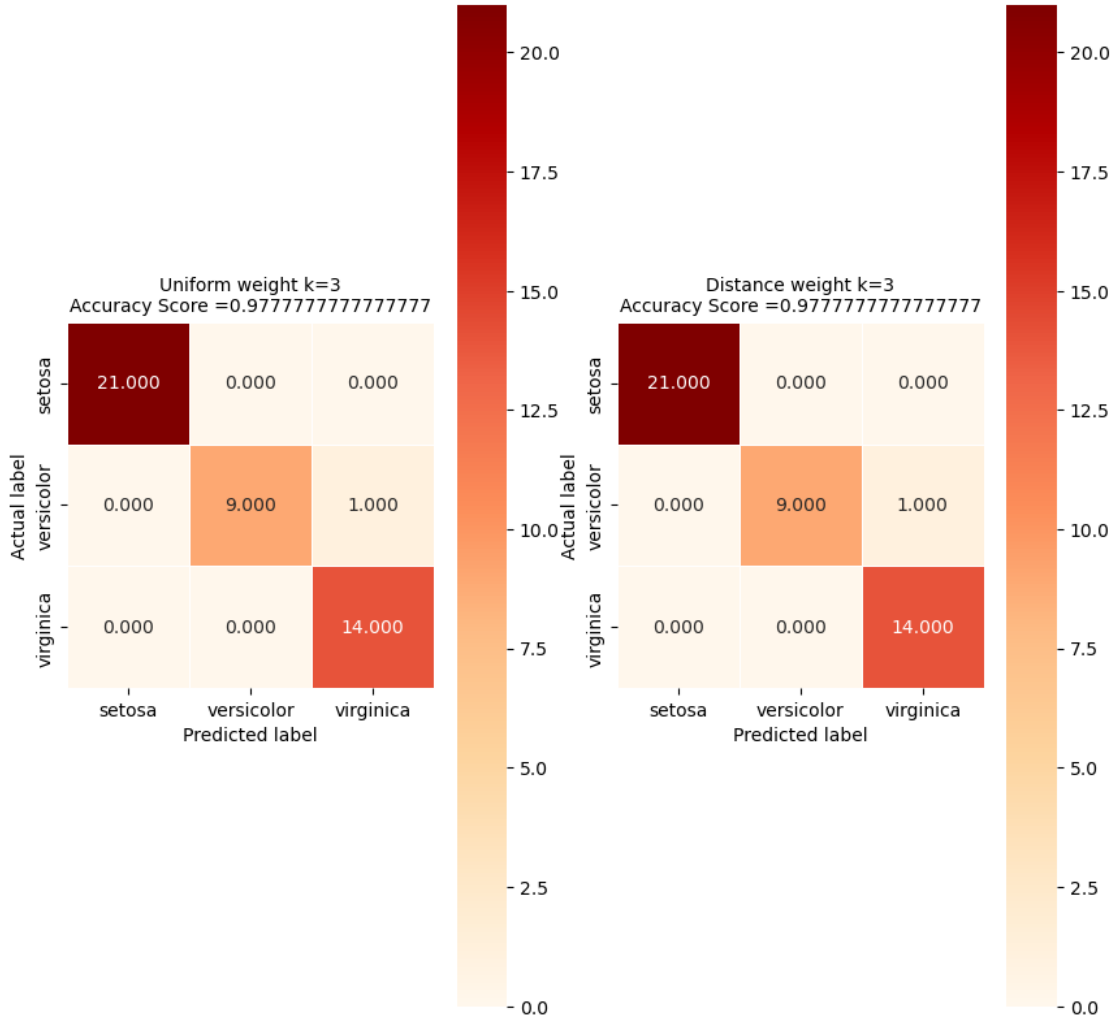


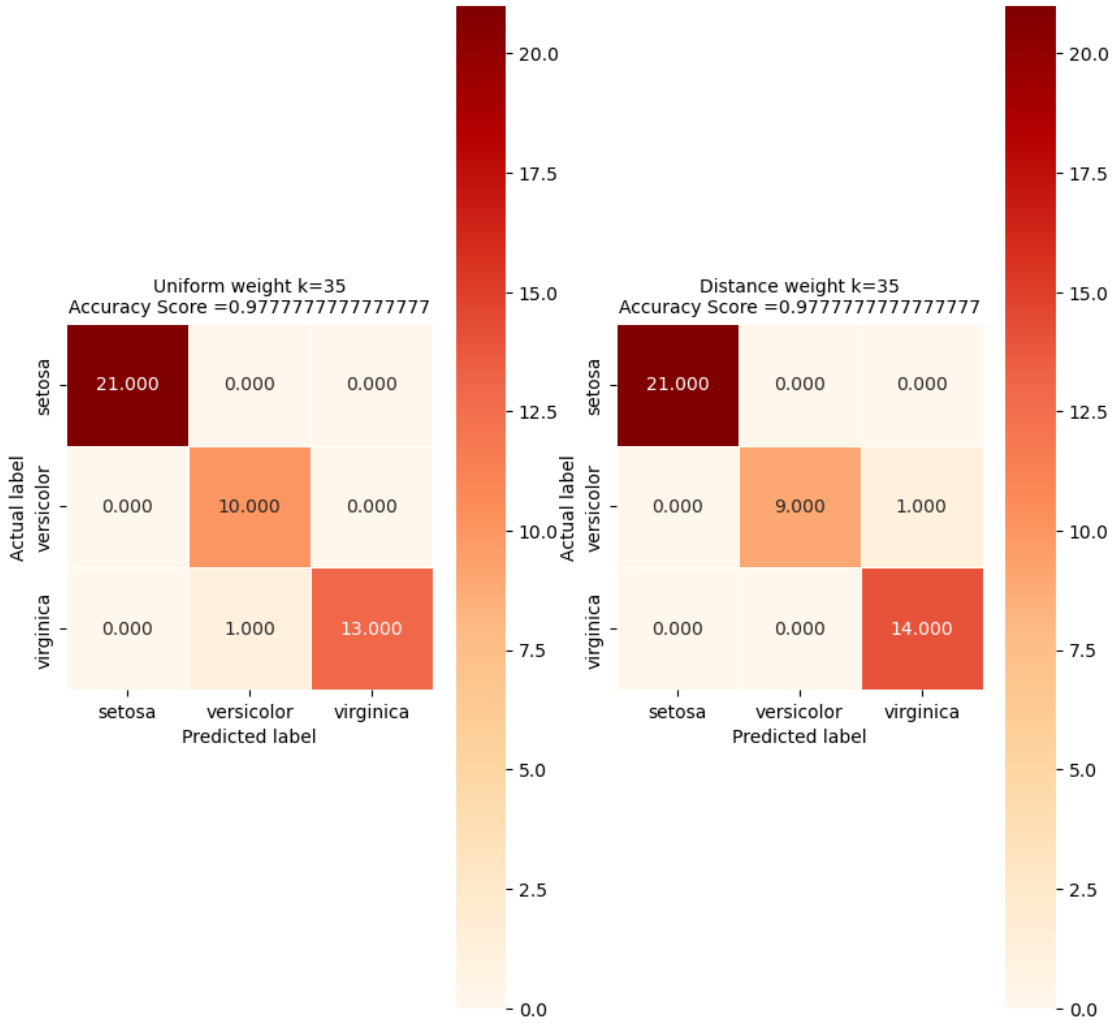
c. Compare the classification models for the iris data set that are generated by k-nearest neighbours (for the different settings from question 3) and by logistic regression. Calculate confusion matrices for these models and discuss the performance of the various models.

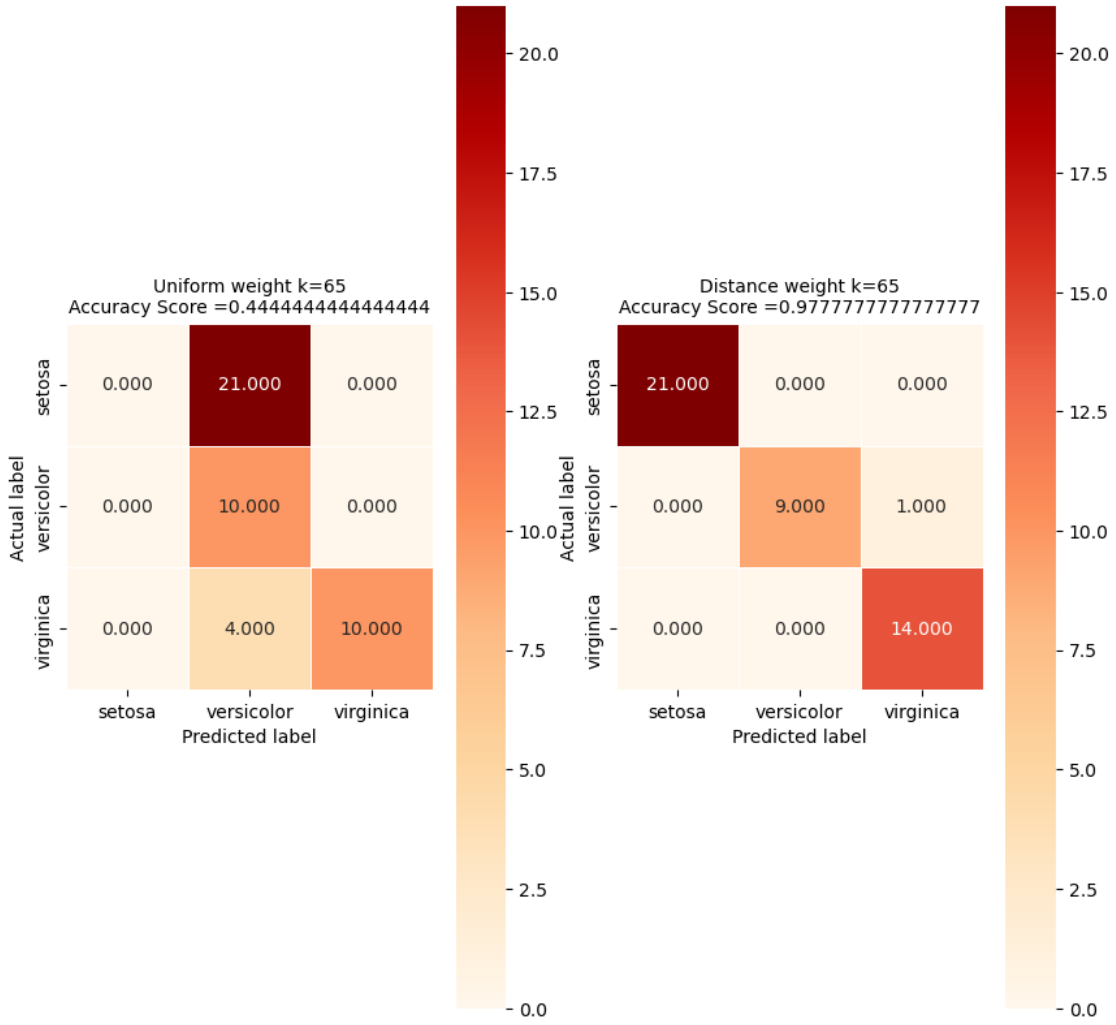
[13]:
```
#To get access to a iris dataset
from sklearn.datasets import load_iris
#Import train_test_split function
from sklearn.model_selection import train_test_split
#Import KNeighborsClassifier model
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
#Import Confusion matrix
from sklearn.metrics import confusion_matrix
#Import matplotlib to plot the confusion matrix
import matplotlib.pyplot as plt
#Import seaborn package for adding additional graphics to plot
import seaborn as sns
#Loading iris dataset
iris_dataSet=load_iris()
#Splitting data in to test and train sets
```

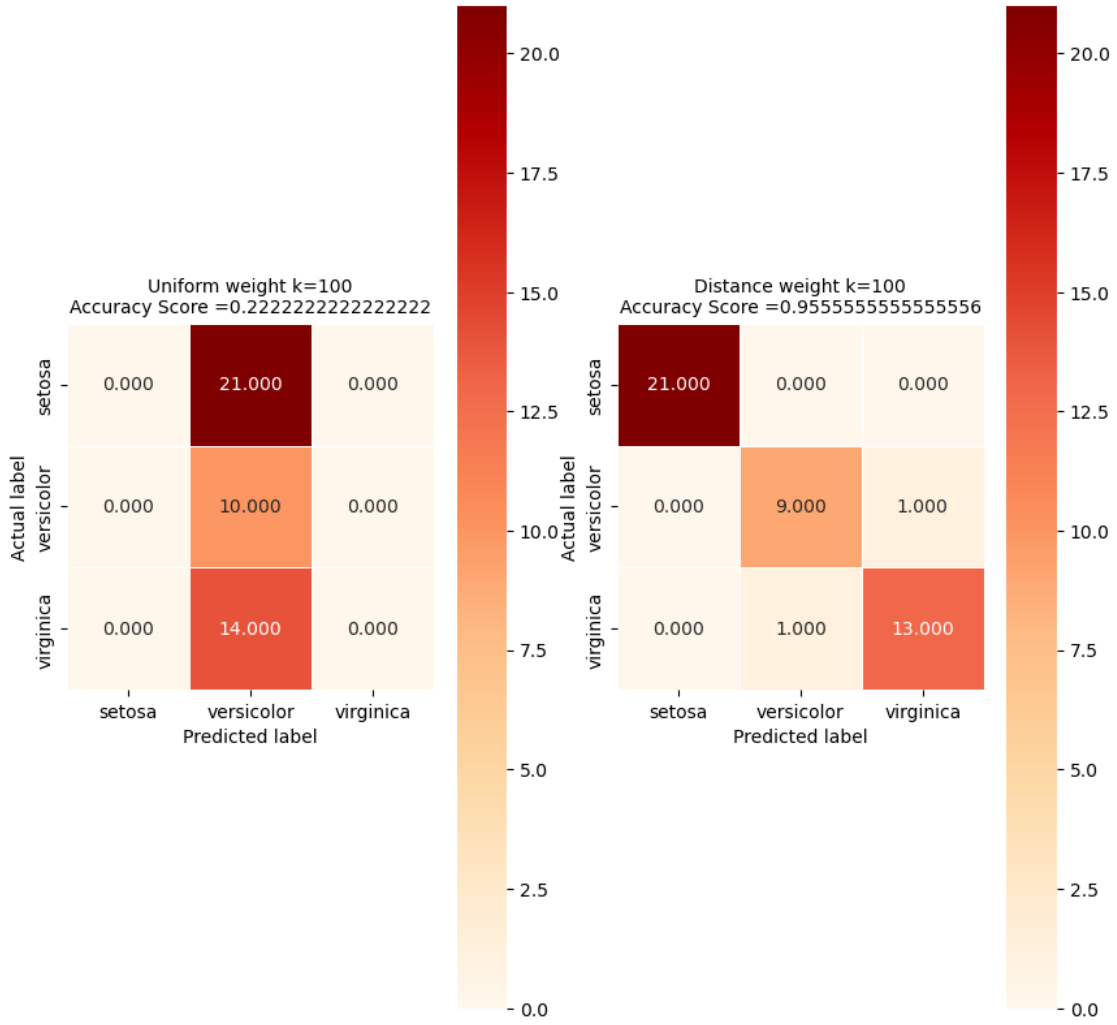
```
x_train, x_test, y_train, y_test = train_test_split(iris_dataSet.data,␣
 ↪iris_dataSet.target, test_size=0.3, random_state=4)
#K values list to obtain confusion matrices
kValue=[3 , 35, 65, 100]
#List of weights of k nearest neighbour
weights = ['distance','uniform']
#List intializations to append test predections for different k values for␣
 ↪different weights
distance_test_predictions=[]
uniform_test_predictions=[]
for weight in weights:
    for i in range(len(kValue)):
        #Create instance of k nearest neighbor model
        knn = KNeighborsClassifier(n_neighbors=kValue[i], weights=weight)
         #Train k nearest neighbor model
        knn.fit(x_train, y_train)
        #Predict the values for x_test set using k nearest neighbor model
        test_predictions = knn.predict(x_test)
        if weight=='uniform':
            #Evaluate the predicted value of x_test using k nearest neighbor␣
 ↪uniform weight model
            uniform_test_predictions.append(test_predictions)
        else:
            #Evaluate the predicted value of x_test using k nearest neighbor␣
 ↪distance weight model
            distance_test_predictions.append(test_predictions)
for i in range(len(kValue)):
    #plot confusion matrices for different k values
    fig, ((ax1, ax2)) = plt.subplots(1, 2,figsize=(10,10))
    cm_uniform = metrics.confusion_matrix(y_test, uniform_test_predictions[i])
    #Obtaining plot of confusion matrix for k nearest neighbor uniform weight␣
 ↪model using sns
    sns.heatmap(cm_uniform, ax = ax1, annot=True, fmt=".3f", linewidths=.5,␣
 ↪square = True, xticklabels=iris_dataSet.target_names,␣
 ↪yticklabels=iris_dataSet.target_names, cmap = 'OrRd')
    cm_distance = metrics.confusion_matrix(y_test, distance_test_predictions[i])
    #Obtaining plot of confusion matrix for k nearest neighbor uniform distance␣
 ↪model using sns
    sns.heatmap(cm_distance, ax = ax2, annot=True, fmt=".3f", linewidths=.5,␣
 ↪square = True, xticklabels=iris_dataSet.target_names,␣
 ↪yticklabels=iris_dataSet.target_names, cmap = 'OrRd')
    # Declaring labels and title for each subplot
    ax1.set_xlabel('Predicted label')
    ax1.set_ylabel('Actual label')
    ax1.set_title('Uniform weight k='+ str(kValue[i])+"\n" +'Accuracy Score␣
 ↪='+str(metrics.accuracy_score(y_test, uniform_test_predictions[i])), size=10)
```

```
   ax2.set_xlabel('Predicted label')
   ax2.set_ylabel('Actual label')
   ax2.set_title('Distance weight k='+ str(kValue[i])+"\n" +'Accuracy Score␣
↪='+str(metrics.accuracy_score(y_test, distance_test_predictions[i])),␣
↪size=10)
   plt.show()
```




Uniform weight k=35
Accuracy Score =0.9777777777777777
Actual label
setosa
versicolor
virginica
21.000
0.000
0.000
0.000
10.000
0.000
0.000
1.000
13.000
setosa
versicolor
virginica
Predicted label
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Distance weight k=35
Accuracy Score =0.9777777777777777
Actual label
setosa
versicolor
virginica
21.000
0.000
0.000
0.000
9.000
1.000
0.000
0.000
14.000
setosa
versicolor
virginica
Predicted label
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0


Uniform weight k=65
Accuracy Score =0.4444444444444444
Actual label
setosa
versicolor
virginica
0.000
21.000
0.000
0.000
10.000
0.000
0.000
4.000
10.000
setosa
versicolor
virginica
Predicted label
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Distance weight k=65
Accuracy Score =0.9777777777777777
Actual label
setosa
versicolor
virginica
21.000
0.000
0.000
0.000
9.000
1.000
0.000
0.000
14.000
setosa
versicolor
virginica
Predicted label
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0


Uniform weight k=100
Accuracy Score =0.2222222222222222
Actual label
setosa
versicolor
virginica
0.000
21.000
0.000
0.000
10.000
0.000
0.000
14.000
0.000
setosa
versicolor
virginica
Predicted label
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Distance weight k=100
Accuracy Score =0.9555555555555556
Actual label
setosa
versicolor
virginica
21.000
0.000
0.000
0.000
9.000
1.000
0.000
1.000
13.000
setosa
versicolor
virginica
Predicted label
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0