Introduction to Data Science and AI – Report – Assignment 2

Big Data Technologies, Sofia University "St. Kliment Ohridski"

Stefan Dimitrov Velev, 0MI3400521

*In this assignment I had to work with two data sets and **Python** so as to solve problems related to regression and classification.*

The packages I have used in this assignment are: ***Pandas*** (for reading and storing data), ***NumPy*** (for making statistics), ***Seaborn*** and ***Matplotlib*** (for visualisation), ***Scikit-learn*** (machine learning library that supports supervised and unsupervised learning). The specific modules that I have used can be found in the following code snippet which is compulsory in the situations when external classes and functions are required:

```
 # Import required Python libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

Task 1: Regression

In that task, the dataset I have to work with was downloaded from www.hemnet.se. It contains information about selling prices of villas in Landvetter for the period 10/2019 – 10/2020. I have to answer the following questions:

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

The first step is to load the dataset using *Pandas* `read_csv(…)` function:

```
# Read the given CSV file
df = pd.read_csv('./data/data_assignment2.csv', delimiter=',')
```

The next step is to leave only the columns which are applicable to the problem. In this case, they are the *Living Area*, *Selling Price* and *ID* (not a must but it is a good practice to have an identifier in addition to the default index of the data).

```
# Remove the unnecessary columns in the data frame
df = df[['ID', 'Living_area', 'Selling_price']]
```

A following step is to remove all the rows with missing values. In that case, there are not any in the dataset. However, this step should always be considered whenever starting to work with unfamiliar data. It is not compulsory to remove these rows but the consideration process is a necessity.

```
# Remove rows with missing values (if there are any)
df = df.dropna()
```

To gain an impression of the data, another important step is to plot it. In the regression problem, it is explicitly said that I have to relate the living area to the selling price – in other words, only two numerical characteristics. Having that in mind, I decided to draw a scatter plot:

```python
# Draw scatter plot of the data
plt.figure(figsize=(11, 6))
plt.scatter(df['Living_area'], df['Selling_price'], color='green')
plt.xticks(range(60, 220, 10))
plt.xlabel('Living area (square metres)')
plt.ylabel('Selling price (million Swedish kronor)')

plt.title('Relationship Between Living Area and Selling Price for Villas in Landvetter 10/2019 -
10/2020')
plt.show()
```

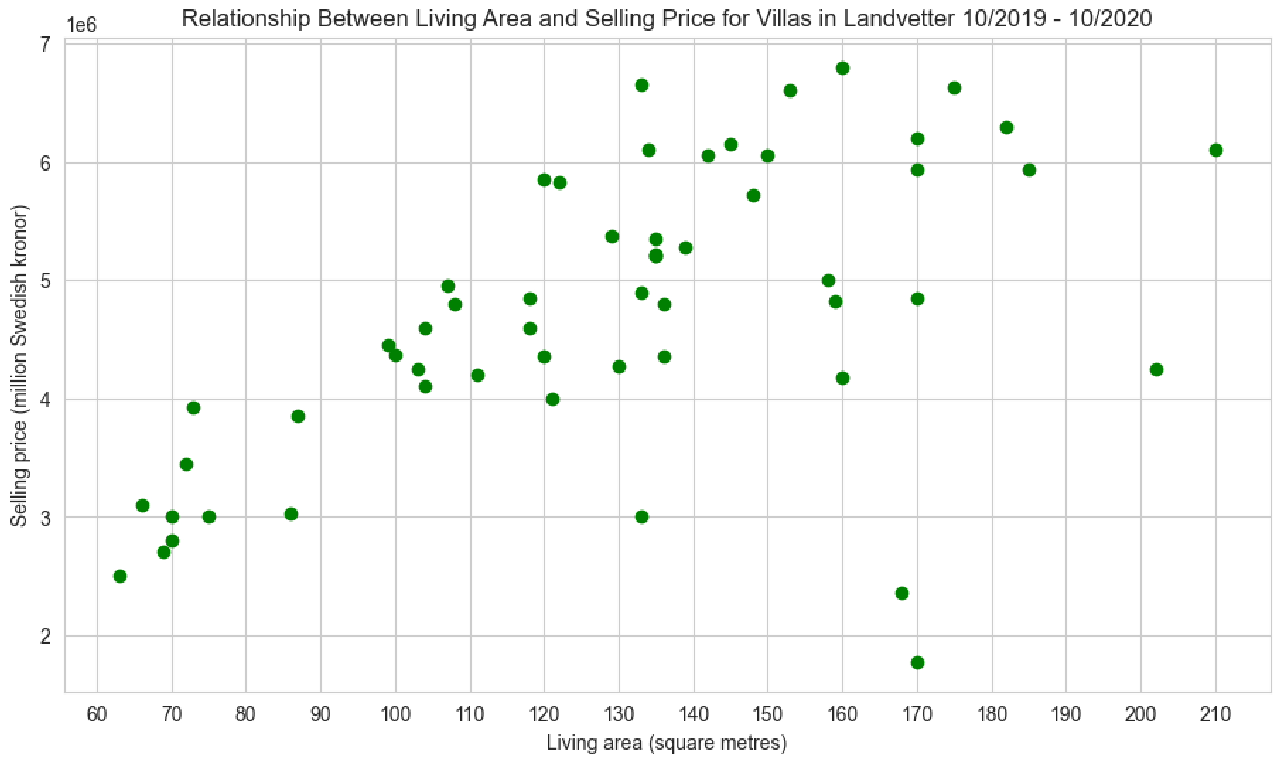

***Figure 1:** Relationship Between Living Area and Selling Price for Villas in Landvetter 10/2019 - 10/2020 (scatter plot)*

Something perturbing can be noticed in the data. There are two villas with relatively large living areas but low selling prices for houses of that kind compared to the others. In my point of view, I can call them outliers since this is an exceptional case in the data. It is not a trend. For that reason, the future linear regression model would not benefit since that would deteriorate the whole picture. That is why I decided to perform a data cleaning process.

```python
# Find the two outliers with relatively large living areas but low selling prices for a villa
min_price_outliers = df.nsmallest(n= 2, columns = 'Selling_price')
print(min_price_outliers)
# Remove the outliers from (as these are special cases which will change the regression line if left)
df = df.drop(min_price_outliers.index)
```

In that situation, I am ready to start building the linear regression model. All the models which I have to use can be found in the *Scikit-learn* library. What is common among them is that they are trained with the `fit` method and the results for given values are returned from the `predict` method. I trained the model with the cleaned dataset and plotted the results on the scatter plot above.

```
# Construct a linear regression model and visualise the regression line
model = LinearRegression()
model.fit(df['Living_area'].values[:, np.newaxis], df['Selling_price'])
xfit = np.array([58, 212])
yfit = model.predict(xfit[:, np.newaxis])

plt.figure(figsize=(11, 6))
plt.scatter(df['Living_area'], df['Selling_price'], color='green')
plt.plot(xfit, yfit, color='red')

plt.xticks(range(60, 220, 10))
plt.xlabel('Living area (square metres)')
plt.ylabel('Selling price (million Swedish kronor)')

plt.title('Regression Line of Living Area vs Selling Price for Villas in Landvetter 10/2019 - 10/2020')
plt.show()
```

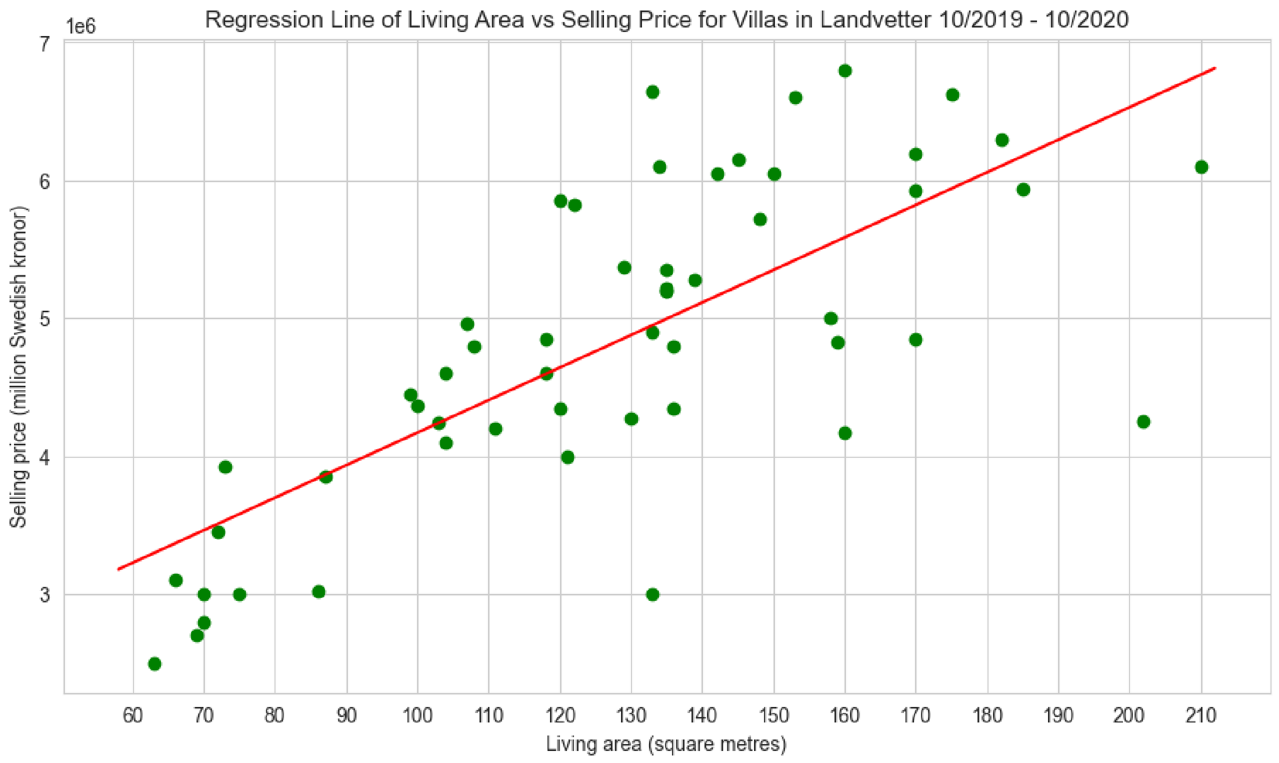

Figure 2: *Regression Line of Living Area vs Selling Price for Villas in Landvetter 10/2019 - 10/2020*

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

The slope (gradient) k and the intercept m can be taken directly from the already constructed linear regression model with specific class properties.

```
# Find the values of the slope and the intercept with the help of the constructed model
print('The value of the slope (gradient) is:', model.coef_[0])
print('The value of the intercept is:', model.intercept_)
```

As a result, I got that the slope k is 23 597.79 and the intercept m is 1 809 821.22. This mean that the equation of the regression line is: $f(x) = k.x + m = 23\,597.79\, x + 1\,809\,821.22$.

c. Use this model to predict the selling prices of houses which have living area 100 m^2, 150 m^2 and 200 m^2.

Given the living area, we can predict the selling price with the built model. This can happen with the `predict` method executed on each separate value.

```
living_area_list = [100, 150, 200]
for current_living_area in living_area_list:
    print(f'The predicted selling price of a villa with a living area of {current_living_area} m2 is
{model.predict(np.array([[current_living_area]]))[0]:.2f} Swedish kronor.')
```

As a result, I get:

The predicted selling price of a villa with a living area of 100 m2 is 4169600.69 Swedish kronor.

The predicted selling price of a villa with a living area of 150 m2 is 5349490.43 Swedish kronor.

The predicted selling price of a villa with a living area of 200 m2 is 6529380.17 Swedish kronor.

d. Draw a residual plot.

The strength of a linear trend can be expressed by the correlation. It is the ratio between the covariance of two variables and the product of their standard deviations. In other words, it is a normalized measurement of the covariance, such that the result always has a value between -1 and 1. Before moving on to the residuals, I decided to find that coefficient of correlation in my dataset. I did that with the help of the *Numpy* package.

```
# Find correlation (quantifies the strength of a linear trend) using Numpy
y_pred = model.predict(df['Living_area'].values[:, np.newaxis])
correlation_matrix = np.corrcoef(df['Selling_price'], y_pred)
print(f'The correlation R between the actual selling prices and the predicted selling prices with the
model is {correlation_matrix[0, 1]:.2f}.')
```

As a result, I discovered that the coefficient of correlation for the linear regression model is 0.73. It means quite a good linear trend in the data. Now, let's move on to the residuals. We calculate them by finding the difference between the observed (actual) and the expected (based on the model fit) values. I stored them in *Pandas Series* that replaces the role of the predicted values by the model when drawing the residual plot.

```
# Calculate the residuals by finding the difference between observed and expected (based on the fit)
residuals = df['Selling_price'] - y_pred

# Draw a residual plot
plt.figure(figsize=(11, 6))
plt.scatter(df['Living_area'], residuals, color='green')
plt.axhline(0, color='black', linestyle='--')

plt.xticks(range(60, 220, 10))
plt.xlabel('Living area (square metres)')
plt.ylabel('Residuals (data - fit)')

plt.title('Residual Plot of Living Area vs Selling Price for Villas in Landvetter 10/2019 - 10/2020')
plt.show()
```

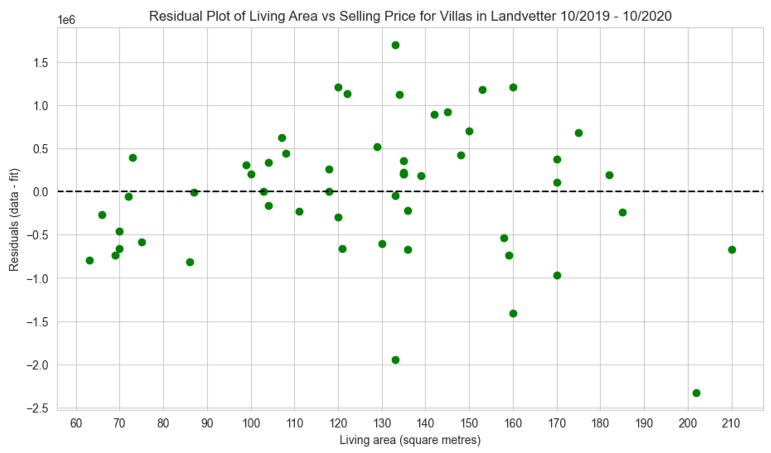

Figure 3: *Residual Plot of Living Area vs Selling Price for Villas in Landvetter 10/2019 - 10/2020*

Task 2: Classification

In that task, the dataset I have to work with is the classic Iris data set (R. A. Fisher, 1936 – "The use of multiple measurements in taxonomic problems"). It includes three iris species (Iris setosa, Iris versicolor, Iris virginica) with 50 samples of each described with some properties about each flower (petal length, petal width, sepal length, sepal width). I have to answer the following questions:

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

The first step was to load and explore the dataset that is included in the *Scikit-learn* library. In order to make visualisations in 2D of the results, I decided to focus only on the first two features of the data – the sepal length and the sepal width. This means, of course, that the constructed models wouldn't be so powerful as if they had the full knowledge of the four features. In *question c.* I will assess the models using the full dataset and we will be able to see the difference. For now, let's load and use the first two features of the data:

```
# Load Iris data set
iris = load_iris()
x_iris = iris.data[:, :2]
y_iris = iris.target
print('Feature names:', iris.feature_names)
print('Target names:', iris.target_names)
```

In order to evaluate the use of different models with different hyperparameters, I had to divide the dataset into training data and test data. This is a classical evaluation strategy which is included as well in the *Scikit-learn* library:

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x_iris, y_iris, test_size=0.2)
```

Now I can move on to the logistic regression model which is a classifier to be found in the *LogisticRegression* module of *Scikit-learn*. The phase of training and predicting is analogous to the linear regression model:

```
# Construct a logistic regression model
model_iris_logistic = LogisticRegression(max_iter=200)
model_iris_logistic.fit(X_train, y_train)
y_pred = model_iris_logistic.predict(X_test)
```

For the construction of the confusion matrix, I used the *Scikit-learn* function with the same name. I visualised the results with the *Seaborn* package with the help of the `heatmap` function.

```
# Construct a confusion matrix to evaluate the use of the logistic regression model
conf_matrix_logistic_regression = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix_logistic_regression, annot=True, fmt='d', cmap='Blues',
xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted Values')
plt.ylabel('True Values')
plt.title('Confusion Matrix on Iris Dataset with Logistic Regression')
plt.show()
```

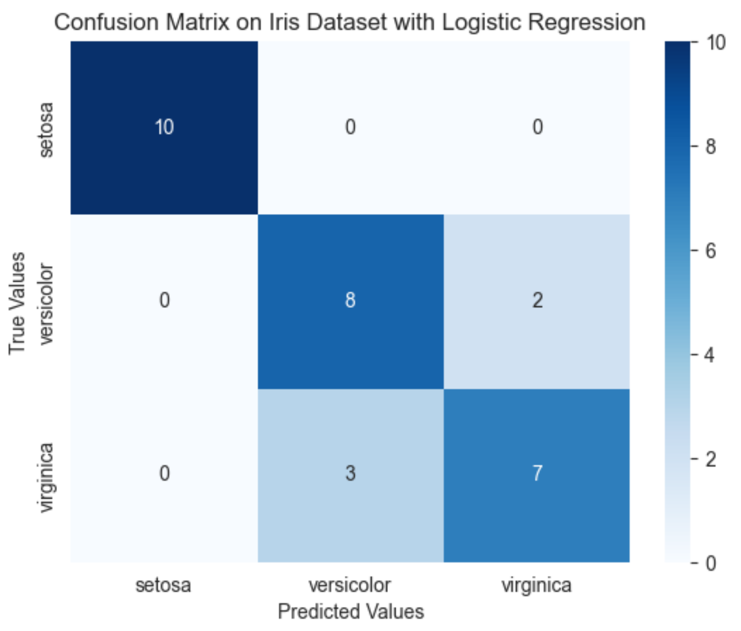

***Figure 4:** Confusion Matrix on for the Iris Dataset with Logistic Regression*

The accuracy which we can calculate from the confusion matrix is equal to the number of correct predictions divided by the total number of predictions made. This means:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{25}{30} \approx 0.83$$

We can also find it with the accuracy_score function of the *Scikit-learn* machine learning library.

```
# Find the accuracy of the logistic regression model
accuracy_logistic_regression = accuracy_score(y_test, y_pred)
print(f'The accuracy of the model on the dataset is {accuracy_logistic_regression:.2f}.')
```

The next step that would help me compare the models was to make a proper plot of the results of the confusion matrix. I used a scatter plot for the two features of the flowers. The predicted values for the separate species from the logistic regression model are presented with colours filling the background of the plot. The actual values can be seen from the colour of the circle which represents the flower. The results are for the test set which is 20% of the whole dataset, i.e. 30 species.

```
h = 0.02
x_min, x_max = X_train[:, 0].min() - 2, X_train[:, 0].max() + 2
y_min, y_max = X_train[:, 1].min() - 2, X_train[:, 1].max() + 2
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

Z = model_iris_logistic.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

cmap_light = ListedColormap(["lightblue", "lightgreen", "lightcoral"])
cmap_bold = ListedColormap(["blue", "green", "red"])

plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.3)

for i, color in enumerate(cmap_bold.colors):
    idx = y_test == i
    plt.scatter(X_test[idx, 0], X_test[idx, 1], c=color, edgecolor="k", label=iris.target_names[i])

plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('Classification on the Iris Dataset with Logistic Regression')
plt.legend()
plt.show()
```

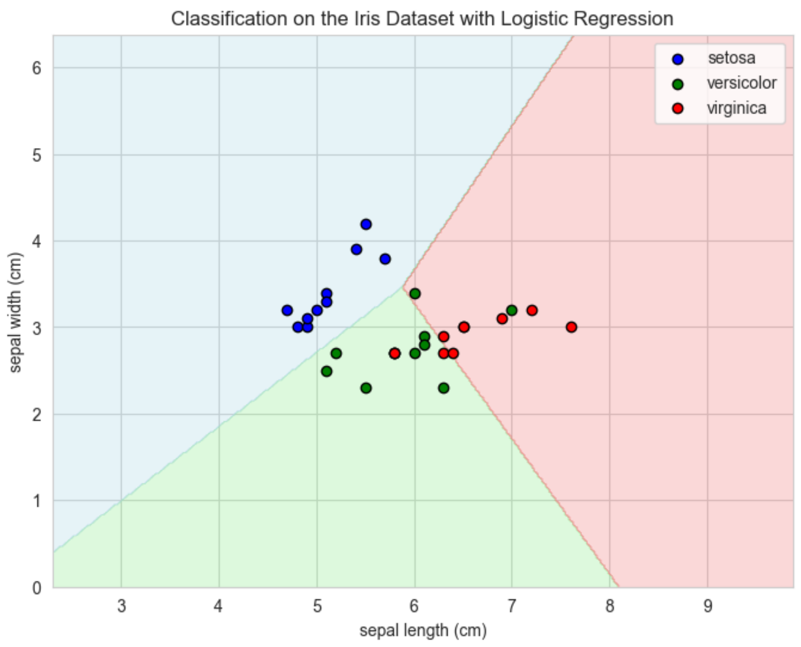

Figure 5: *Classification of the Iris Dataset with Logistic Regression*

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?

We are required to conduct different comparisons of the K-Nearest Neighbours classifier using different number of neighbours k and different weights – uniform or distance. Let's first focus on plotting different visualisations and after that we will discuss whether the results are what the theory says.

Since I had to perform a lot of experiments, I defined a function for constructing the model and visualising the plot. The arguments which the function needed from are the number of neighbours k, the weights (distance or uniform) and a flag parameter that I used for information whether a plot is required or not and on what data to test the model. The function returns the accuracy as we already defined it.

```
def knn_classifier(neighbours, weights, plot = True):
    model_iris_knn = KNeighborsClassifier(n_neighbors=neighbours, weights=weights)

    if plot == False:
        model_iris_knn.fit(X_train, y_train)
        y_pred = model_iris_knn.predict(x_iris)
        accuracy_knn = accuracy_score(y_iris, y_pred)
        return accuracy_knn

    model_iris_knn.fit(X_train, y_train)
    y_pred = model_iris_knn.predict(X_test)
    accuracy_knn = accuracy_score(y_test, y_pred)
    print(f'The accuracy of the KNN model with k = {neighbours}, w = "{weights}" on the Iris dataset is
{accuracy_knn:.2f}')

    h = 0.02
    x_min, x_max = X_train[:, 0].min() - 2, X_train[:, 0].max() + 2
    y_min, y_max = X_train[:, 1].min() - 2, X_train[:, 1].max() + 2
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    Z = model_iris_knn.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    cmap_light = ListedColormap(["lightblue", "lightgreen", "lightcoral"])
    cmap_bold = ListedColormap(["blue", "green", "red"])

    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.3)
    for i, color in enumerate(cmap_bold.colors):
        idx = y_test == i
        plt.scatter(X_test[idx, 0], X_test[idx, 1], c=color, edgecolor="k", label=iris.target_names[i])

    plt.xlabel(iris.feature_names[0])
    plt.ylabel(iris.feature_names[1])
    plt.title(f'Classification on the Iris Dataset with KNN k = {neighbours}, weights = {weights}')

    plt.legend()
    plt.show()

    return accuracy_knn
```

The first types of experiments which I have performed are related to drawing plots of the first 15 neighbours ($k = 1, \dots, 15$) once for distance-based weights and after that the plots with uniform-based weights. I did that with the help of separate loops after which I displayed the returned accuracy scores. I stored them in a list which then I converted into *Pandas Series*.

```
accuracy_list_knn_distance = []
for i in range(1, 16):
    accuracy_list_knn_distance.append(knn_classifier(i, 'distance'))
print(accuracy_list_knn_distance)
accuracy_series_knn_distance = pd.Series(accuracy_list_knn_distance)
```

[0.7666666666666667, 0.7666666666666667, 0.8, 0.8, 0.8333333333333334, 0.8333333333333334, 0.8333333333333334, 0.8333333333333334, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8333333333333334]

```
accuracy_list_knn_uniform = []
for i in range(1, 16):
    accuracy_list_knn_uniform.append(knn_classifier(i, 'uniform'))
print(accuracy_list_knn_uniform)
accuracy_series_knn_uniform = pd.Series(accuracy_list_knn_uniform)
```

[0.7666666666666667, 0.8, 0.7666666666666667, 0.8, 0.8, 0.8, 0.7666666666666667, 0.7333333333333333, 0.7333333333333333, 0.6333333333333333, 0.7, 0.6666666666666666, 0.7333333333333333, 0.7, 0.7]

Some of the visualisations can be seen below:

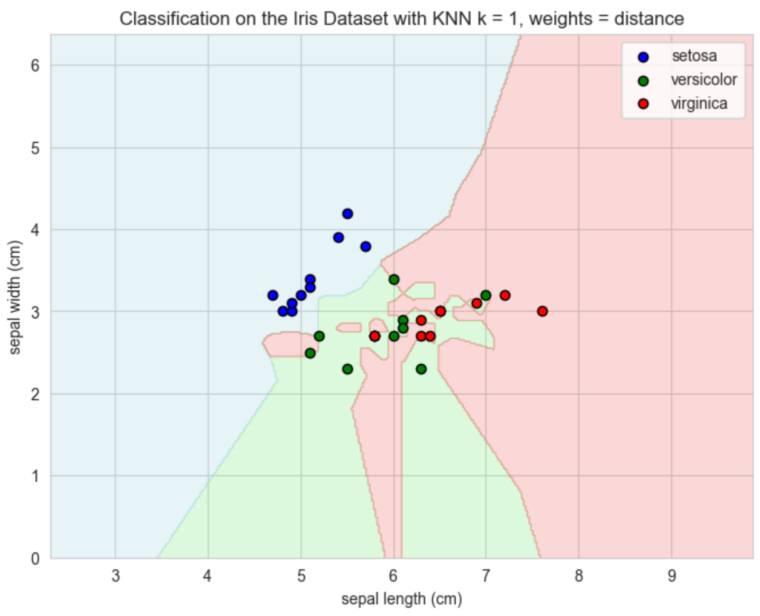

Figure 6: *Classification on the Iris Dataset with KNN k = 1, weights = distance*

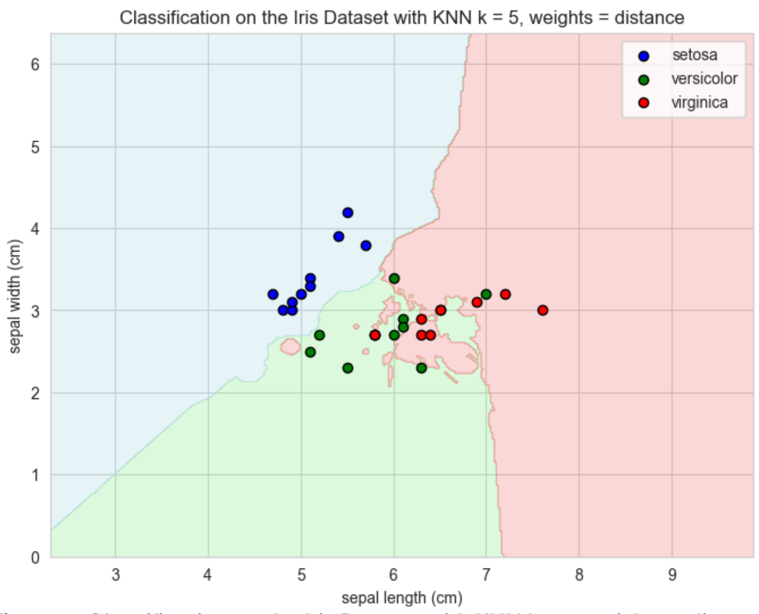

Figure 7: *Classification on the Iris Dataset with KNN k = 5, weights = distance*

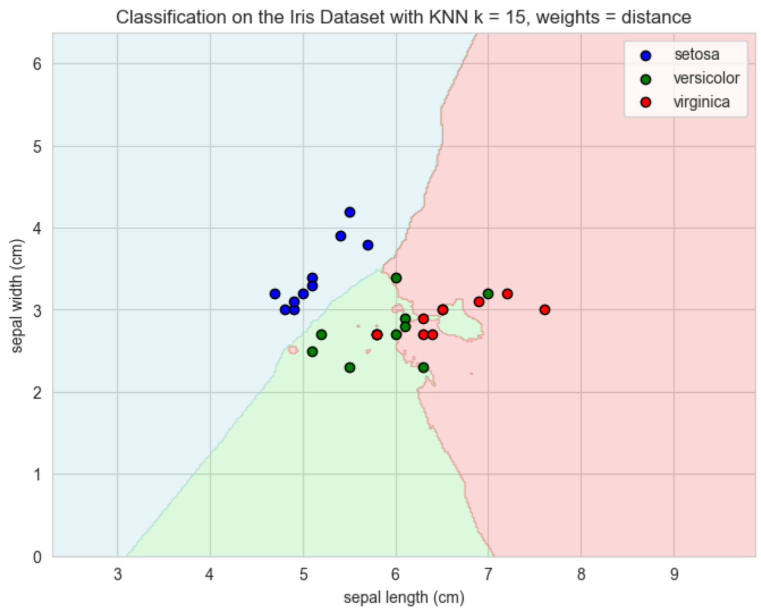

Figure 8: *Classification on the Iris Dataset with KNN k = 15, weights = distance*

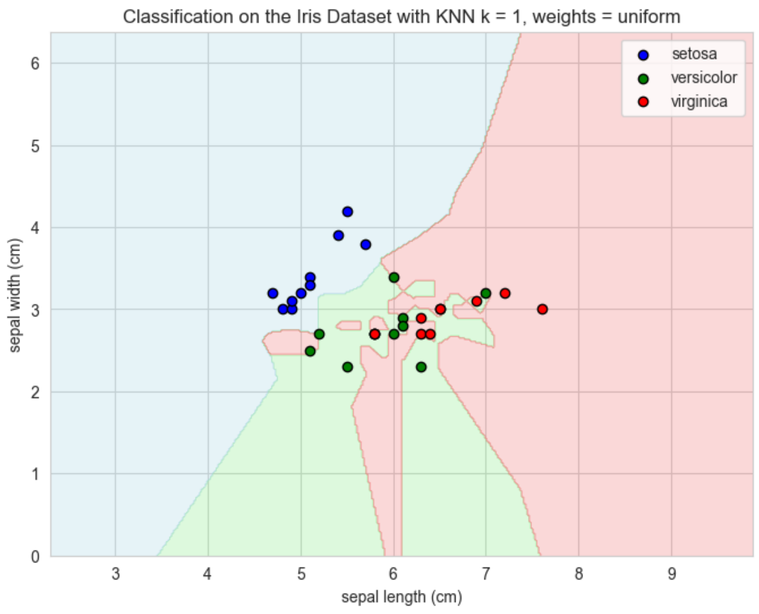

Figure 9: *Classification on the Iris Dataset with KNN k = 1, weights = uniform*

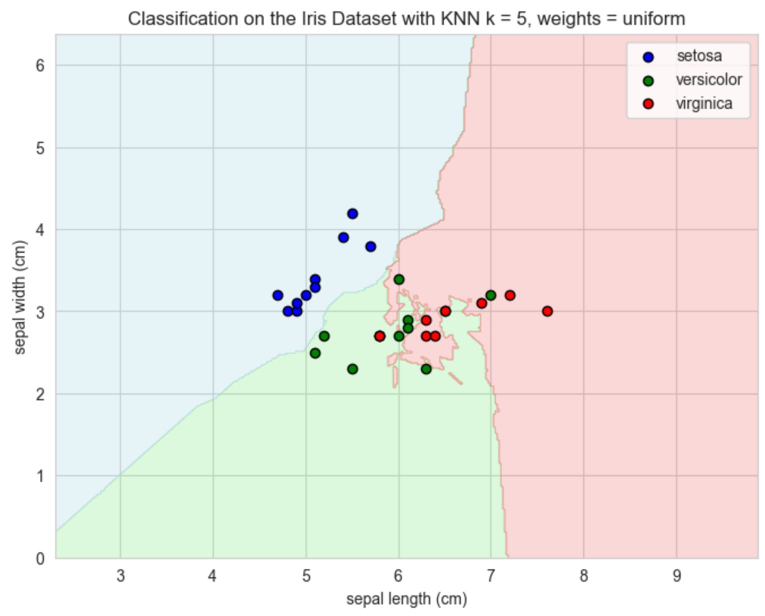

Figure 10: *Classification on the Iris Dataset with KNN k = 5, weights = uniform*

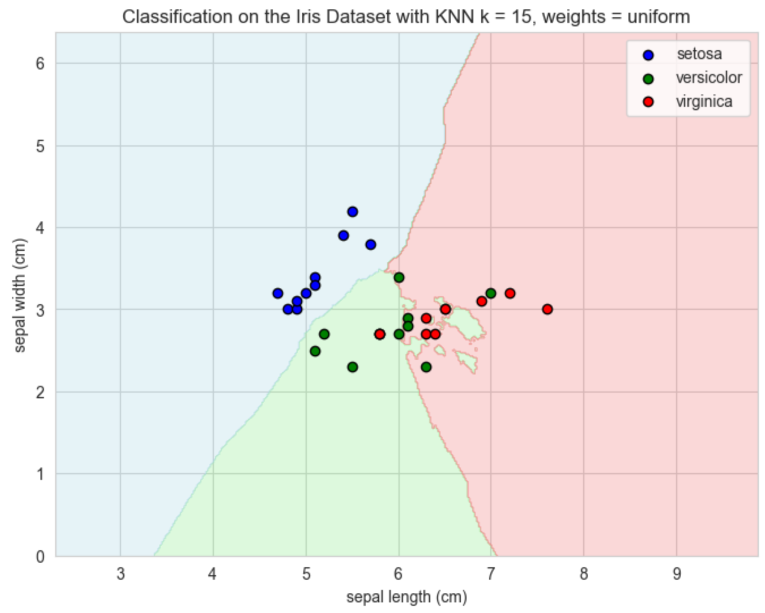

Figure 11: *Classification on the Iris Dataset with KNN k = 15, weights = uniform*

In the second types of experiments, I am doing the same but until the number of neighbours k reaches 60. As it gets high, I am training the model with the training data again but testing it on the whole dataset. In general, we have to avoid situations of testing the model with data it has already seen. However, it is admissible for that case with the KNN classifier as the training part consists of only storing the data. There is not a clear training phase here as in the other classifiers. That is why the accuracy won't be curved so much. The other reason is because we don't have much data since our test set is with 30 species only. In that case, the average value won't be precise enough. After we store the 60 accuracy scores for the distance-based weights and for the uniform-based weights, I draw a plot which compares the two types with the increasing of k. The results can be seen on the following image:

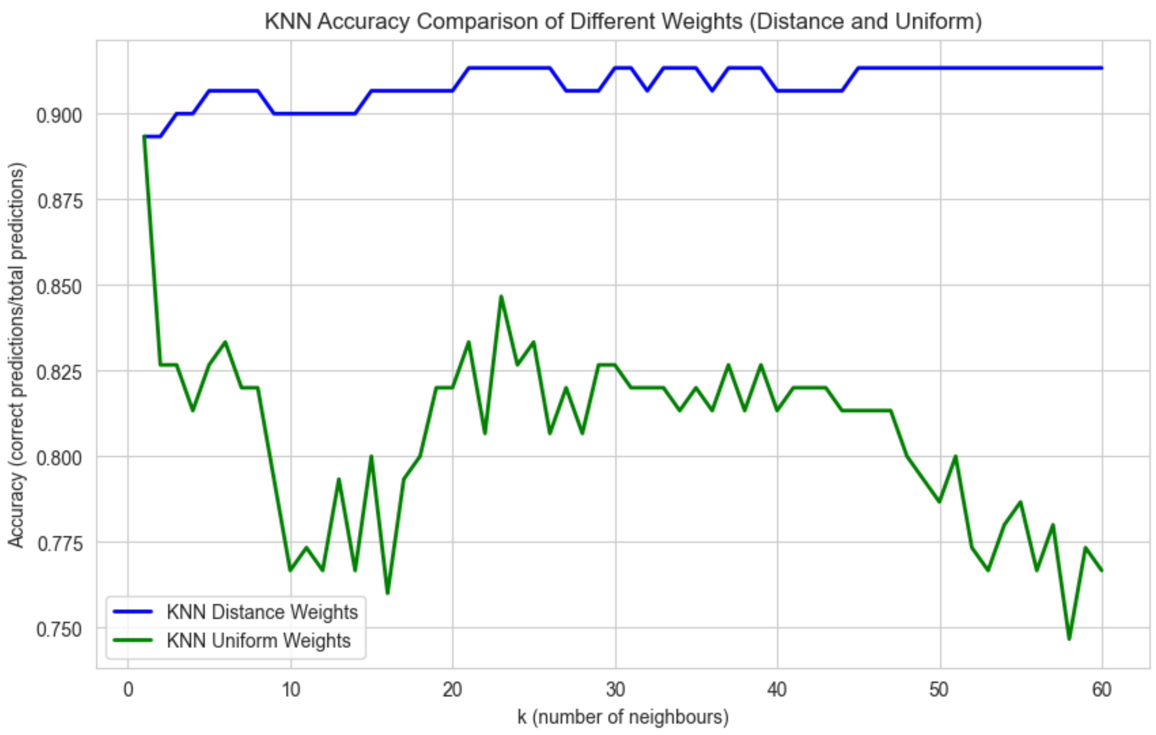

Figure 12: *KNN Accuracy Comparison of Different Weights (Distance and Uniform)*

What conclusions can we derive? The basic KNN algorithm works quite well in many cases but it treats all neighbours equally. On the other hand, weighted KNN introduces the concept of assigning different weights to neighbours based on their proximity to the query point which can lead to an improved performance. That is the case when we have distance-based weights. This means that the algorithm weights points by the inverse of their distance. In this case, closer neighbours of a query point will have a greater influence than neighbours which are further away. Therefore, the expected behaviour is that with the number of neighbours k increasing, the accuracy of the algorithm will not change significantly and there will be a moment from which the line of accuracy remains horizontal. That is because when we have many points, they get farther and farther away from the query point which means that their weights become insignificant and they do not reflect on the accuracy of the algorithm. This trend can be noticed in practice as well in *Figure 12*.

The default version of the K-Nearest Neighbours classifier is with uniform-based weights. In other words, all points in each neighbourhood are weighted equally. This means that with the number of neighbours k increasing, we will have more distant points to consider and since all points have equal weights, we will have to think of a closer point and a distant point having the same power. In that way, we are breaking the whole idea of the KNN algorithm and that's why the accuracy score is expected to decrease gradually from one moment on. That is what theory says and what we can actually observe in *Figure 12*.

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

In the previous questions I discovered the accuracy scores of the logistic regression and the KNN classifier. Let's concentrate on the results when using a single test data for evaluation. I estimated that the accuracy of the logistic regression model on the Iris dataset is 0.83. The same accuracy I obtained from the weighted KNN classifier (distance-based) for k = 5.

In the above scenarios, the results were based only on the first two features of the Iris data set – *sepal length* and *sepal width*. The reason for this choice was the possibility to visualise the results. Now, I would like to compare the models when using the full power of the dataset – the four features: *sepal length*, *sepal width*, *petal length*, *petal width*. For that reason, I trained the new models with the training data of the original dataset. I evaluated them in two ways – using the single accuracy score of the test data as well as with 5-fold cross-validation. In these scenarios, 2D visualisations are not possible but I can compare the models with the help of confusion matrices.

The logistic regression classifier becomes multivariate (4 features) and multiclass (3 classes). The results I obtained show that with a single train-test split we get an accuracy score of 0.93. The 5-fold cross-validation is preferred technique because it reduces the variability in the results as it uses the entire dataset for training and validation. It provides a better estimate of model performance on unseen data. That's why the accuracy score with that evaluation strategy is better – 0.97.

```
x_iris = iris.data
y_iris = iris.target

X_train, X_test, y_train, y_test = train_test_split(x_iris, y_iris, test_size=0.2)
model_iris_logistic = LogisticRegression(max_iter=200)
model_iris_logistic.fit(X_train, y_train)
y_pred = model_iris_logistic.predict(X_test)

conf_matrix_logistic_regression = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix_logistic_regression, annot=True, fmt='d', cmap='Blues',
xticklabels=iris.target_names,
            yticklabels=iris.target_names)
plt.xlabel('Predicted Values')
plt.ylabel('True Values')
plt.title('Confusion Matrix on Iris Dataset with Logistic Regression')
plt.show()
accuracy_logistic_regression = accuracy_score(y_test, y_pred)
print(f'The accuracy of the logistic regression model on the Iris dataset is
{accuracy_logistic_regression:.2f}')
```

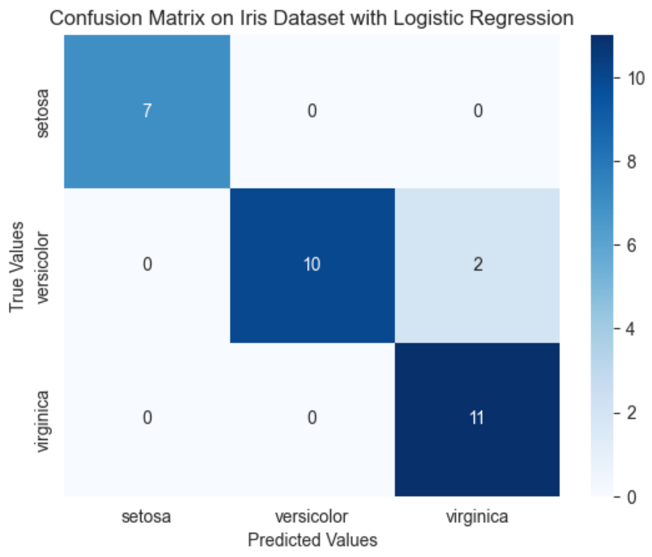

Figure 13: *Confusion Matrix on Iris Dataset with Logistic Regression*

```
cv_scores_logistic_regression = cross_val_score(model_iris_logistic, x_iris, y_iris, cv=5)
print(f"Average accuracy of linear regression model with 5-fold cross validation:
{np.mean(cv_scores_logistic_regression):.2f}")
```

Average accuracy of linear regression model with 5-fold cross validation: 0.97

The weighted KNN classifier also becomes multivariate (4 features) and multiclass (3 classes). The results I obtained with k = 4 and distance-based weights show an accuracy score of 0.97 only with a single train-test split. The more complex evaluation strategy, 5-fold cross-validation, shows the same result – an accuracy score of 0.97.

```
model_iris_knn = KNeighborsClassifier(n_neighbors=4, weights='distance')
model_iris_knn.fit(X_train, y_train)
y_pred = model_iris_knn.predict(X_test)

conf_matrix_knn = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix_knn, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_names,
yticklabels=iris.target_names)
plt.xlabel('Predicted Values')
plt.ylabel('True Values')
plt.title('Confusion Matrix on Iris Dataset with KNN (k = 4, weights = "distance")')
plt.show()

accuracy_knn = accuracy_score(y_test, y_pred)
print(f'The accuracy of the KNN model with k = 4, weights = "distance" on the Iris dataset is
{accuracy_knn:.2f}')
```

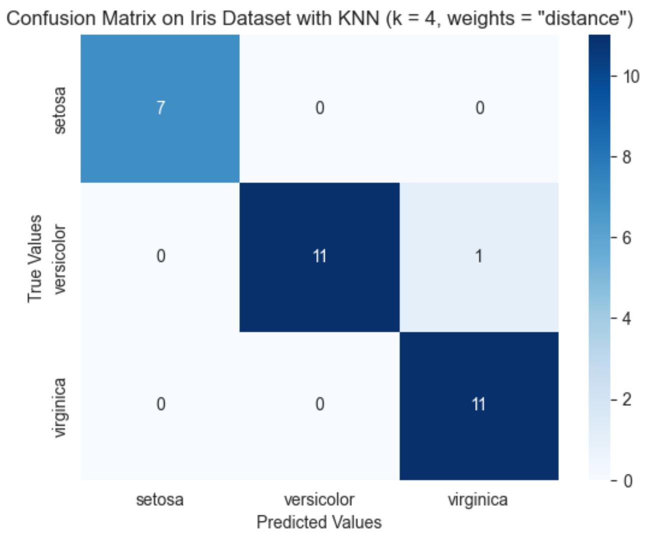

Figure 14: *Confusion Matrix on Iris Dataset with KNN (k = 4, weights = "distance")*

```
cv_scores_knn = cross_val_score(model_iris_knn, x_iris, y_iris, cv=5)
print(f"Average accuracy of KNN model with 5-fold cross validation: {np.mean(cv_scores_knn):.2f}")
```

Average accuracy of KNN model with 5-fold cross validation: 0.97

In conclusion, both of the classifiers have a better performance when they use the four features of the Iris dataset. The reasons are hidden in the more relations that are formed among the data. The only drawback is that the four coordinates for the features are not easy to be visualised. For the specific case, the KNN classifier tends to behave better when the evaluation strategy used is a single train-test split. However, both of the classifiers have equal accuracy scores of 0.97 with 5-fold cross-validation. I believe that is one of the best accuracy scores which can be obtained from the two classifiers on that specific dataset.