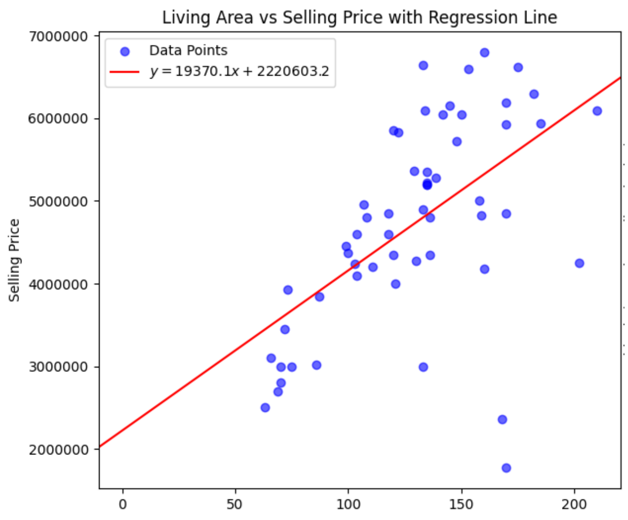
Assignment 2
Mihael Zlatev

Task 1

Living areas vs selling prices with a linear regression line



Plotting the data

1. Loading the "villas_metadata.csv" file.
2. Cleaning any row which contain **N/A** value in it.
3. Training a Linear Regression model using the **"sklearn.linear_model"** library
4. Getting the slope and intercept from the model.
5. Plotting the data points and regression line using the slope and intercept values.

```
pl1.scatter(data[LIVING_AREA], data[SELLING_PRICE], color='blue',
label='Data Points', alpha=0.6)
pl1.axline(xy1=(0, intercept), slope=slope, color='red', label=f'$y =
{slope:.1f}x {intercept:+.1f}$')
```

Slope and Intercept values

```
slope = model.coef_[0]
intercept = model.intercept_
```

slope = 19370.1
intercept = 2220603.2

Using the model to predict the selling prices of houses which have living area 100 m2, 150 m2 and 200 m2

To predict the prices the code uses the **model.predict** method.

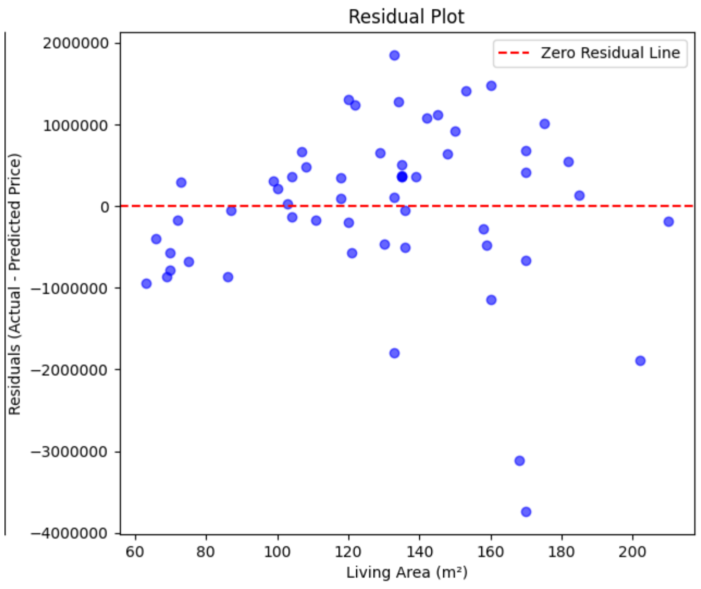
```
living_areas = pd.DataFrame({LIVING_AREA: LIVING_AREA_PREDICTIONS})
predicted_prices = model.predict(living_areas)
```

Predicted prices for:

- **100 m² = 4157617**
- **150 m² = 5126124**
- **200 m2 = 6094630**

Residual plot

1. **Predicting selling prices with the model**
2. **Calculating residuals (residuals = actual selling prices – predicted prices)**



There are only two outliers at the bottom of the plot, but most of the residuals are randomly scattered around the horizontal line at zero, it suggests that the model fits the data well.

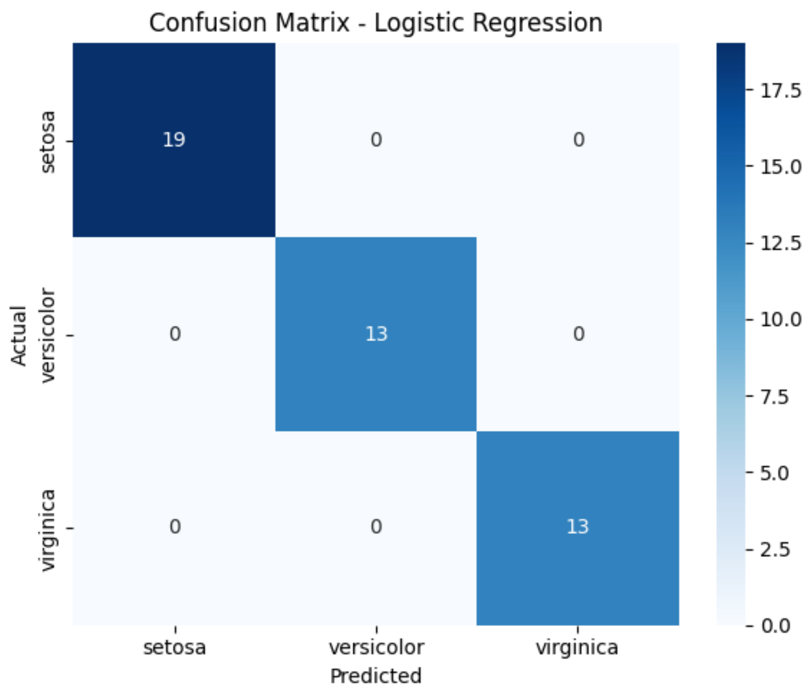
Task 2

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

- Logistic Regression is a type of supervised learning algorithm that models the relationship between input features and the target variable by fitting a logistic function to the data.
- To get a good model we split the iris data on a **training** and a **test** set.
  The test data is not used during training. After the model has learned from the training data, we use the test data to evaluate how well the model is able to make predictions

on new data. This simulates how the model would perform when it encounters new data.

- Once the training process is done the model does predictions on the test data. The confusion matrix below gives us information on the model accuracy.



It's a 3x3 table, where each row corresponds to an actual class, and each column corresponds to a predicted class.

- Diagonal values are the correct predcitions
- Non-Diagonal are missclassifications

In this example every classification is correct. The model classified 19 Setosa flowers, 13 Versicolor and 13 Virginica flowers.

Using a confusion matrix to evaluate the use of a k-nearest neighbors (KNN) to classify the iris data set

The process of training the K-nearest neighbors is similar to the logistic regression. The model is training on training data and then is making predictions based on the testing data. The idea of the KNN is to classify data point based on its **k closest** data points in the dataset.

KNN has two weighting strategies

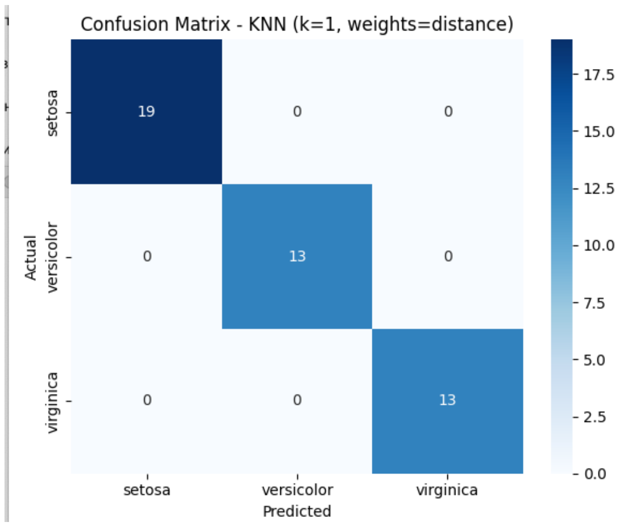
- **Uniform weights**: All neighbors contribute equally. It works well all points are distributed evenly.
- **Distance-based weights**: Closer neighbors have more influence than distant ones. It works well when closer points are more important for classification.

Meaning of different k-values

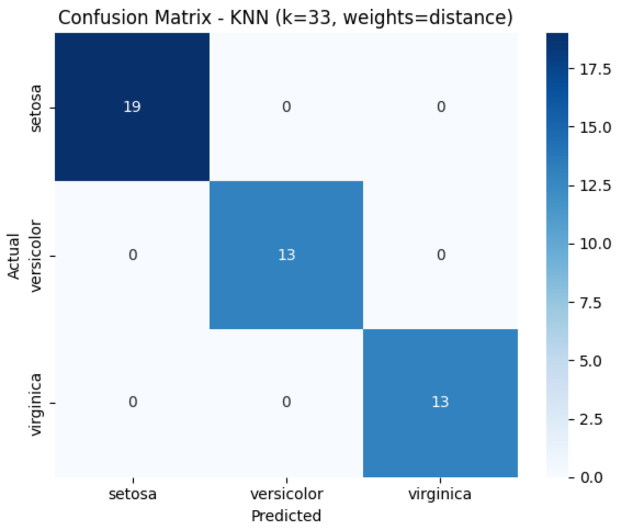
- **Small k** - The model assigns the class based on the closest neighbor, but this technique is sensible to noice. If the closes neighbor is an outlier it might lead to a misclassification
- **Big k** – The model reduces noise sensitivity. However, if k is very large (E.g >20) it might not correctly differentiate classes.

KNN model classifying the Iris dataset

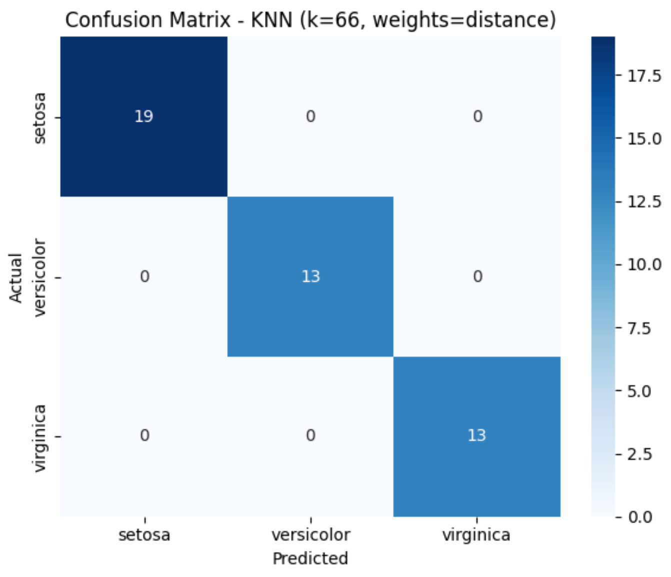
1. K = 1, Weight = distance



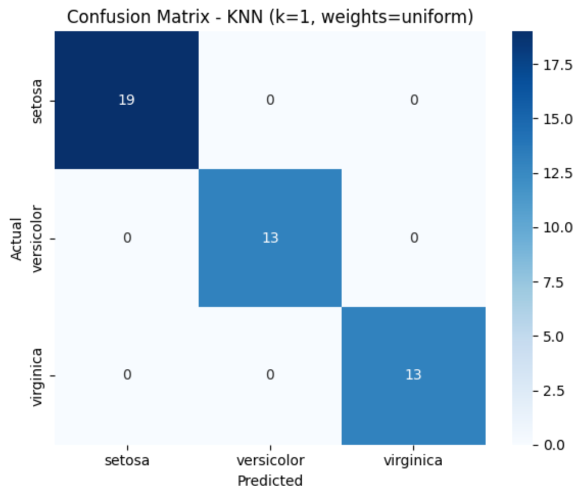
2. K = 33, Weight = distance



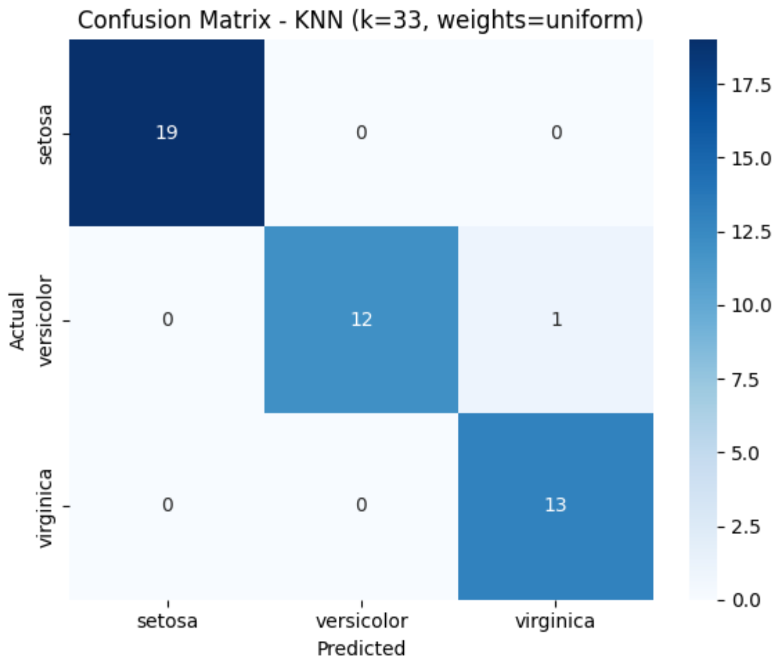
3. K = 66, Weight = distance



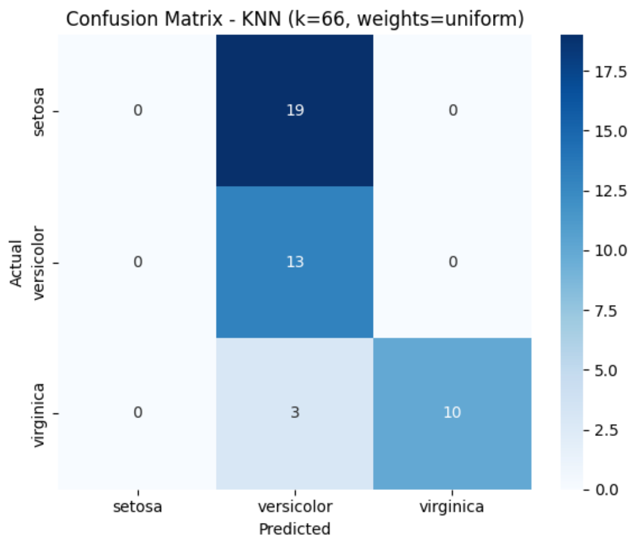
4. K = 1, Weight = uniform



5. K = 33, Weight = uniform



6. K = 66, Weight = uniform



- From the confusion matrices we can see that for **K=1 both weighting strategies** result in the same classifications. The model correctly classified 19 Setosa flowers, 13 Versicolor and 13 Virginica flowers.
- **For K=33, weight = distance** – the model works again correctly
- **For K = 66, weight = distance** – the model works again correctly. Even though, K is very large it seems that the data points are distributed in such way that neighbors close to the query point have more influence. Because of the **distance weighting strategy** the model continues to work correctly.
- **For K=33, weight = uniform** – the model **missclassifies 1 Versicolor flower.** Because the model works correctly for K=66 and distance weight, it means that for this spesific Iris dataset changing the K values doesn't matter so much. However, the weight matters a lot.
- **For K=66, weight = uniform** – the model **missclassifis 3 Versicolor flowers** showing that for very large K the model starts to underfit more data and has more errors.

Logistic Regression vs KNN

Logistic Regression:

- Assumes a **linear decision boundary**. Works well for linearly separable data.
- Faster for large datasets, as it requires training only once.
- May struggle with non-linear decision boundaries.

KNN:

- Does not assume a specific decision boundary. Handles **non-linear boundaries** better.
- Slower for large datasets, as predictions involve searching through the entire dataset.
- Sensitive to the choice of **K** and weighting strategy.

For the specific Iris dataset from the confusion matrices above we see that for **small K-values both uniform and distance weights** the model correctly classifies the data. The is also true for the Logistic Regression. The only difference is when the KNN model uses Large K-values.
In conclusion for the Iris is a small dataset and both models have similar performances. They both correctly classify the different data points (if the K-values are small for the KNN model.)

Code Task 1

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

VILLAS_CSV = 'villas_metadata.csv'
LIVING_AREA = 'Living_area'
SELLING_PRICE = 'Selling_price'
LIVING_AREA_PREDICTIONS = [100, 150, 200]

def init_data():
    data = pd.read_csv(VILLAS_CSV)
    data = data[[LIVING_AREA, SELLING_PRICE]].dropna()
    living_areas = data[[LIVING_AREA]]
    selling_prices = data[SELLING_PRICE]
    return data, living_areas, selling_prices

def plot_data(X, y, model):
    slope = model.coef_[0]
    intercept = model.intercept_

    living_areas = pd.DataFrame({LIVING_AREA: LIVING_AREA_PREDICTIONS})
    predicted_prices = model.predict(living_areas)
    print(f"Predicted prices for 100 m², 150 m², and 200 m²: {predicted_prices}")

    y_pred = model.predict(X)
    residuals = y - y_pred

    _, (pl1, pl2) = plt.subplots(1, 2, figsize=(15, 6))

    pl1.scatter(data[LIVING_AREA], data[SELLING_PRICE], color='blue', label='Data 
Points', alpha=0.6)
    pl1.axline(xy1=(0, intercept), slope=slope, color='red', label=f'$y = 
{slope:.1f}x {intercept:+.1f}$')
    pl1.set_xlabel('Living Area (m²)')
    pl1.set_ylabel('Selling Price')
```

```python
    pl1.set_title('Living Area vs Selling Price with Regression Line')
    pl1.ticklabel_format(axis='y', style='plain')
    pl1.legend()

    # Plot the second graph: Residual Plot
    pl2.scatter(X, residuals, color='blue', alpha=0.6)
    pl2.axhline(y=0, color='red', linestyle='--', label='Zero Residual Line')
    pl2.set_xlabel('Living Area (m²)')
    pl2.set_ylabel('Residuals (Actual - Predicted Price)')
    pl2.set_title('Residual Plot')
    pl2.ticklabel_format(axis='y', style='plain')
    pl2.legend()

    plt.show()


if __name__ == '__main__':
    data, X, y = init_data()
    model = LinearRegression()
    model.fit(X, y)
    plot_data(X, y, model)
```

Code Task 2

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

UNIFORM = 'uniform'
DISTANCE = 'distance'
K_VALUES = [1, 33, 66]

def load_training_and_test_data(iris):
    X = iris.data
    y = iris.target
    return train_test_split(X, y, test_size=0.3, random_state=42)

def train_logistic_regression_model(X_train, y_train):
    log_reg = LogisticRegression(max_iter=200)
    log_reg.fit(X_train, y_train)
    return log_reg


def train_knn_model(X_train, y_train, k, weight_type):
    knn = KNeighborsClassifier(n_neighbors=k, weights=weight_type)
    knn.fit(X_train, y_train)
    return knn


def plot_confusion_matrix(confusion_matrix, name_of_the_plot, iris):
    plt.figure(figsize=(6, 5))
    sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=iris.target_names, yticklabels=iris.target_names)
    plt.title(name_of_the_plot)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.tight_layout()


def plot_regression_classification(X_train, y_train, X_test, y_test,iris):
    log_reg_model = train_logistic_regression_model(X_train, y_train)
    y_predictions_log_reg = log_reg_model.predict(X_test)
    confusion_matrix_log_reg = confusion_matrix(y_test, y_predictions_log_reg)

    plot_confusion_matrix(confusion_matrix_log_reg, 'Confusion Matrix - Logistic
Regression', iris)


def plot_knn_classifications(X_train, y_train, X_test, y_test, k_values,
weight_type, iris):
    for k in k_values:
        knn_model = train_knn_model(X_train, y_train, k, weight_type)
        y_predictions_knn = knn_model.predict(X_test)
        confusion_matrix_knn = confusion_matrix(y_test, y_predictions_knn)

        plot_confusion_matrix(confusion_matrix_knn, f'Confusion Matrix - KNN
(k={k}, weights={weight_type})', iris)


if __name__ == '__main__':
    iris = load_iris()
    X_train, X_test, y_train, y_test = load_training_and_test_data(iris)
    plot_regression_classification(X_train, y_train, X_test, y_test, iris)

    plot_knn_classifications(X_train, y_train, X_test, y_test,K_VALUES, UNIFORM,
iris)
    plot_knn_classifications(X_train, y_train, X_test, y_test,K_VALUES, DISTANCE,
iris)

    plt.show()
```