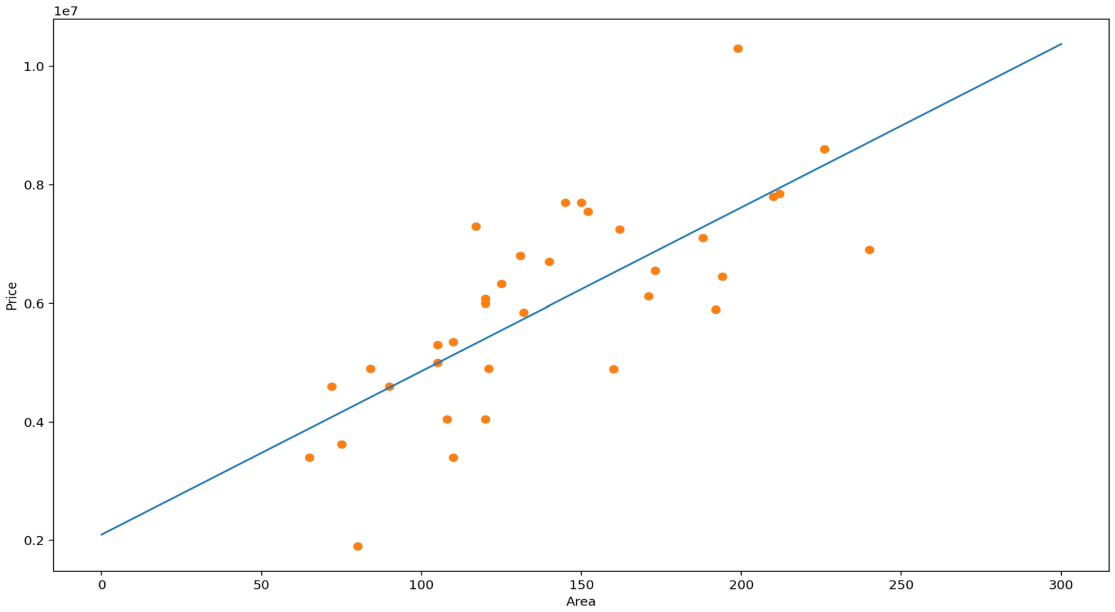
Malte Carlstedt - 8h spent
Johan Östling - 8h spent

1.
Here is our scatter plot with an added regression line.



(i)
The intercept value and slope value is given by the following code. Where x = house area and y = price of house.

```
model = LinearRegression().fit(x, y)
print(model.intercept_)
```

The intercept value is: 2099110.90750558

```
print(model.coef_)
```

The slope value is: 27595.45641054

(ii)
To predict the cost of a house of 100m2 we use the following code.

```
print(model.predict([[100]]))
```

The predicted value for houses of sizes 100m2,150m2, 200m2 are:
100m2 -> 4858656.54855992 kr
150m2 -> 6238429.36908709 kr
200m2 -> 7618202.18961425 kr

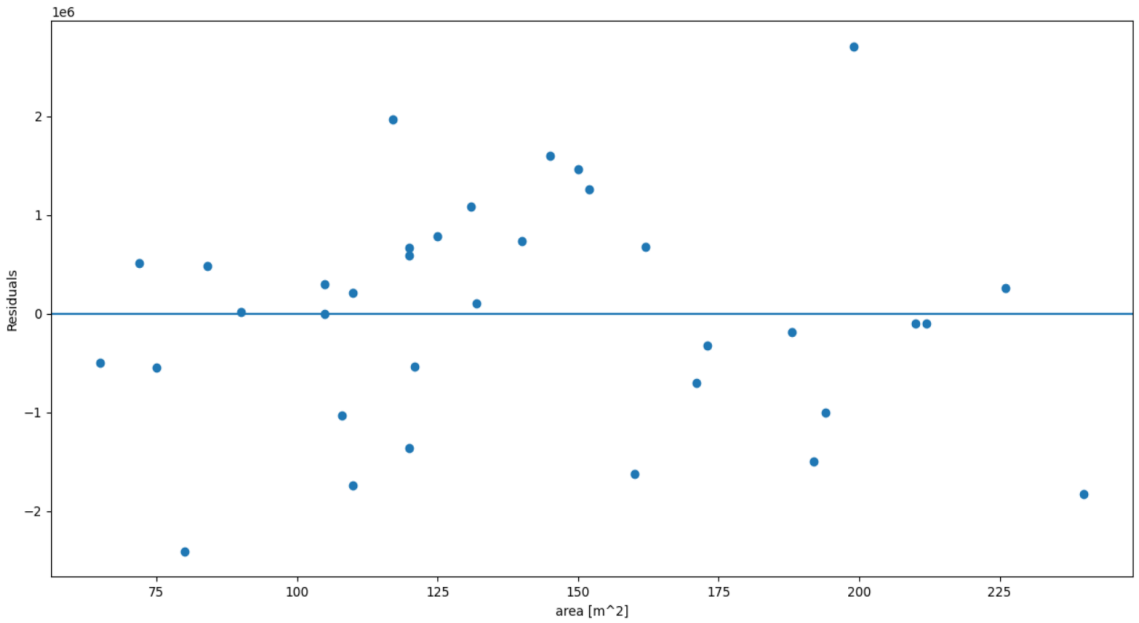
(iii) Residual Plot:

To scatter the plot with residuals we use the following line of code:

```
plt.scatter(x,y-model.predict(x))
```

As we can tell by the code residuals are defined by as predictedY - f(real x). To get the vertical line we simply add it to the plot by the following line of code:

```
plt.axhline(y=0)
```



(iv)

From the residual plot above we can see how the regression line, which is our predictor of house prices, differs from the actual house prices. It differs because we are trying to fit a linear line onto points which we plotted from only one factor. Maybe if we added in factors such as the area of the house's plot, condition of the house, and operating cost, then our predictions could be better. Also, we intuitively know that, in general, bigger houses are more expensive. It is reasonable to assume a connection and therefore try to fit a linear line to predict prices, but we do not know if the connection is linear, perhaps it is logarithmic or exponential. Therefore it makes sense that we should get points in the residual plot that are not on the line y=0. Although, that would be the best case.

2.
This line used the logistic regression model to predict the right value for the test set.

```
predictions = logisticRegr.predict(x_test)
```

The method metrics.confusion_matrix() computes a confusion matrix by taking in the predicted values and the correct values as parameters

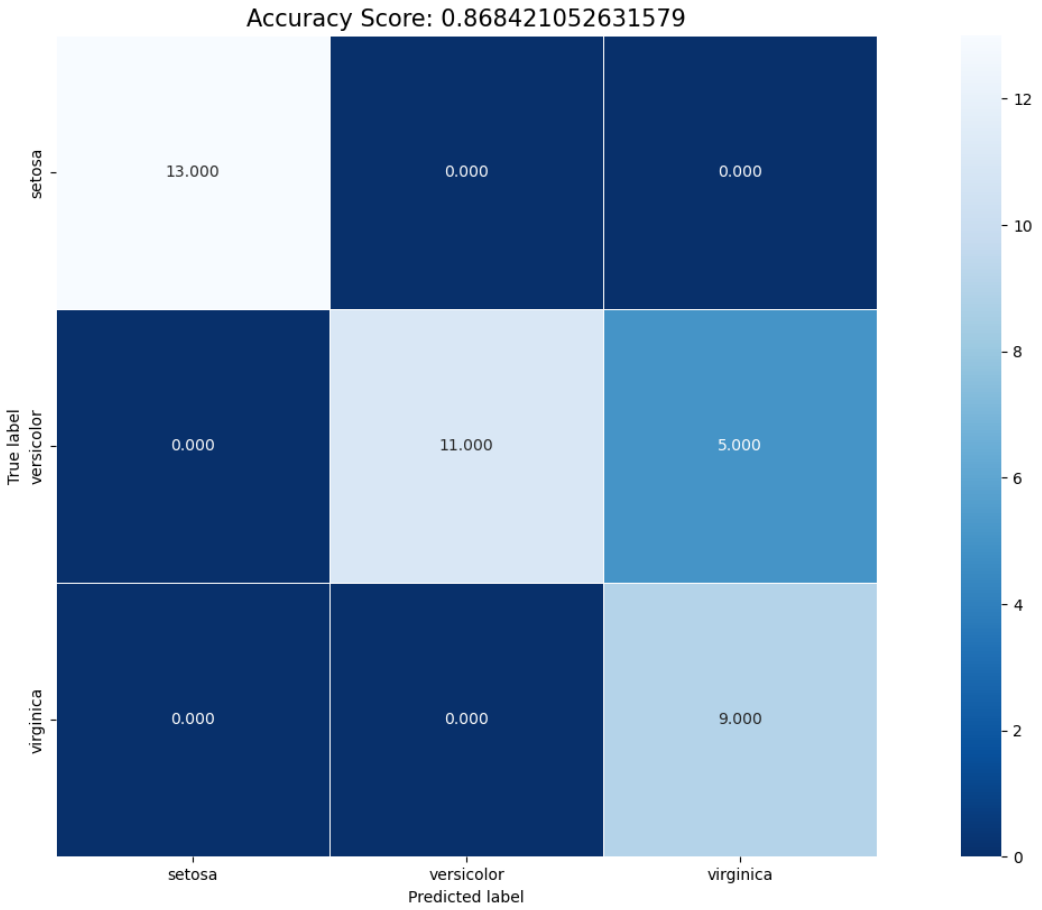
```
cm = metrics.confusion_matrix(y_test, predictions)
```

This method plots the confusion matrix and gives it different attributes, such as colouring each square in the confusion matrix in different shades of blue depending on the frequency.

```
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap =
'Blues_r')
```

In order to show to confusion matrix:

```
plt.show()
```



We chose not to normalize this confusion matrix because when the size of the test values is this small we can still see the ratio very clearly. And it is nice to be able to see exactly how many of the test values that were predicted incorrectly.

If the regression model could perfectly predict which sample belongs to the correct flower, then the confusion matrix would only show values on the diagonal from the upper left corner to the down right corner.

On the x-axis we have the predicted labels, which is what our logistic regression model predicted each test value to be. For example the model predicted that 13 of the test values that were ‘setosa’ was ‘setosa’. So it predicted correctly. Our model also predicted all the ‘versicolor’ correctly. But in the case of the prediction of ‘virginica’ it mistaked 5 out of 14 virginica flowers to be ‘versicolor’.

To evaluate the accuracy of the model an accuracy score is presented by dividing every correct prediction with every prediction. In this case we got 33/38, which tells us that our model predicted 87% of the test values correctly. Therefore we can assume that our model has a 87% chance to predict the correct flower if we add new values. 87% accuracy is very good. Since the majority of the flowers were predicted accurately.

3.
This code creates our model for different values of K.

```
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(x_train, y_train)
```

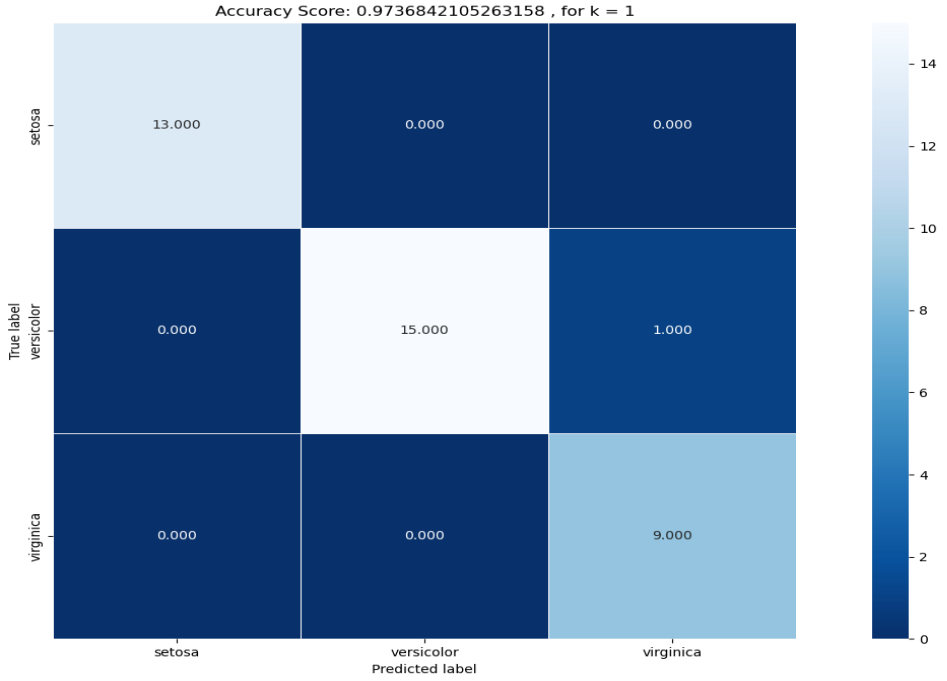
For increasing values of k we get the following accuracy score:

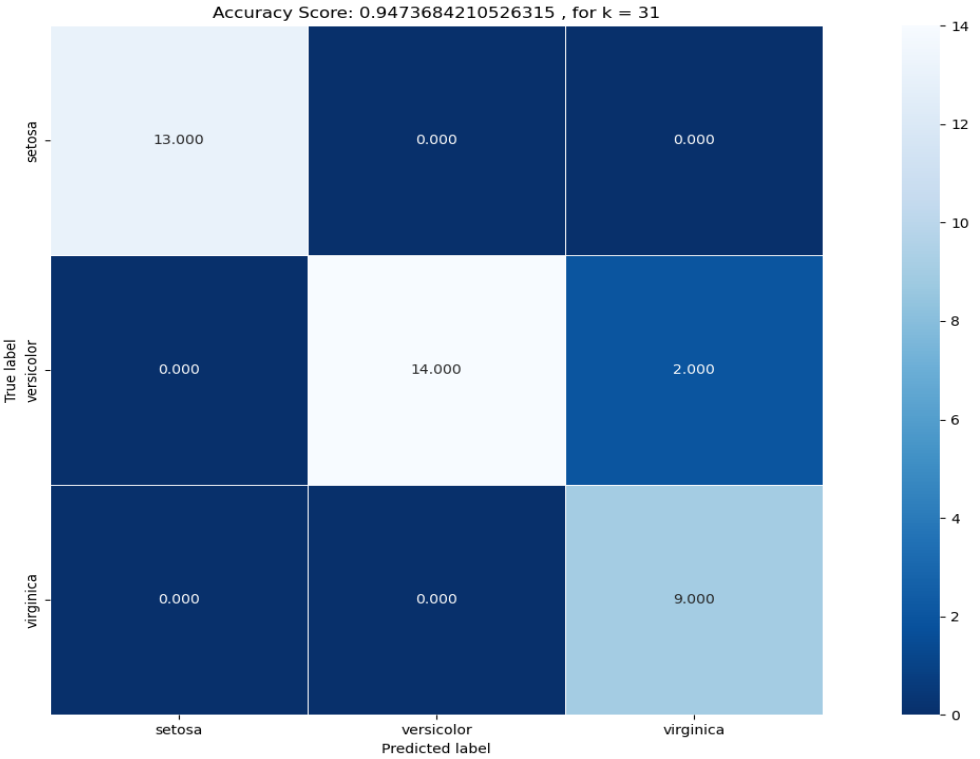
k=1 -> 0.97
k=11 -> 0.97
k=21 -> 0.97
k=31 -> 0.95
k=41 -> 0.89
k=51 -> 0.89
k=61 -> 0.89
…
k=100 -> 0.61
k=112 -> 0.24

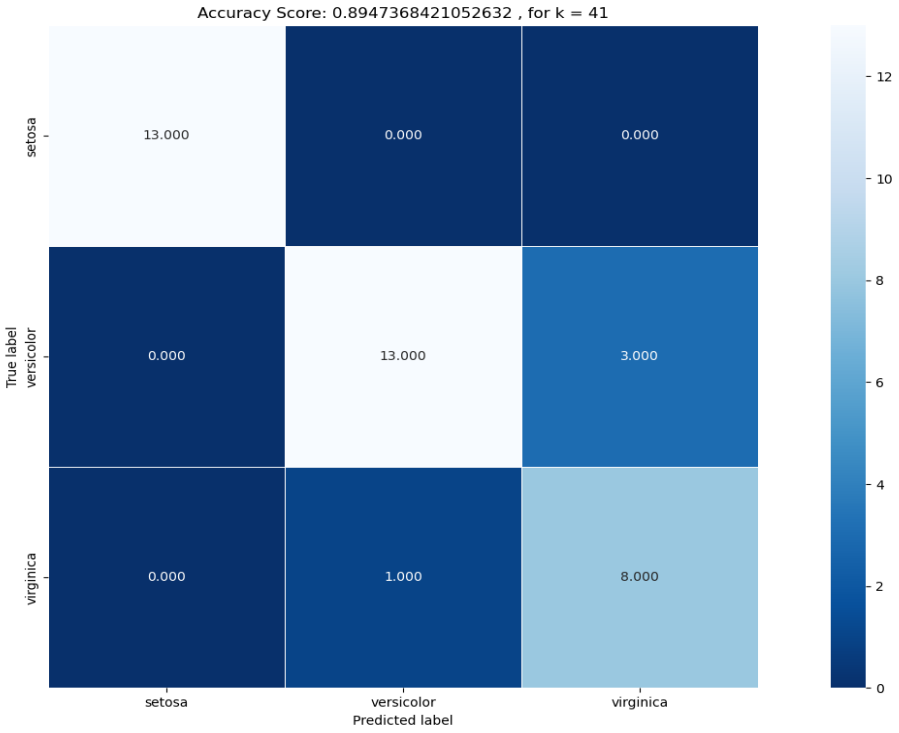
In our case with the iris data set we see that the most accurate our prediction can be is 0.97 by using the KNN-method.

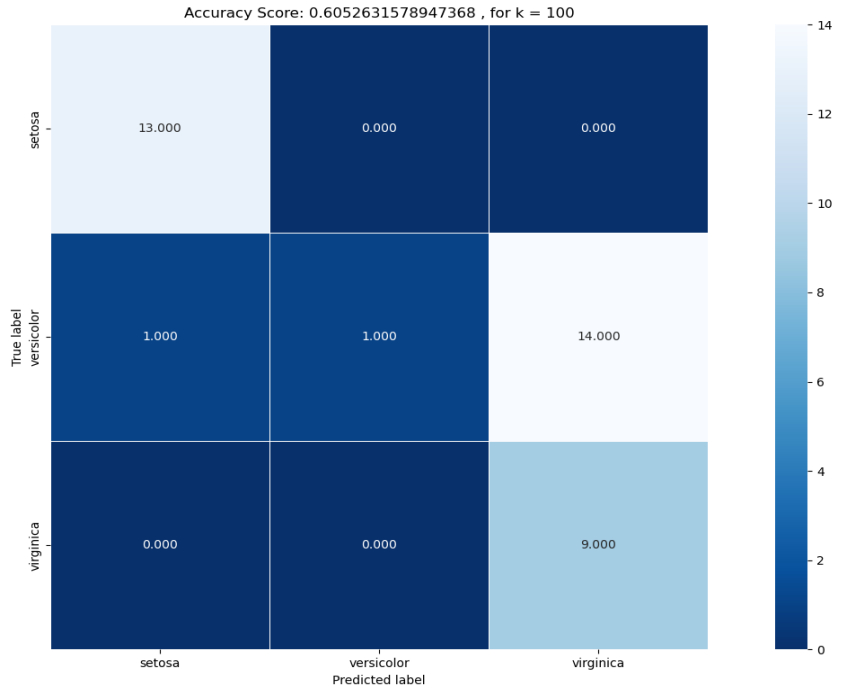
As our K values grow the accuracy score proceeds to get lower until we've reached its max at 112. Which are all the neighbors in the plot. The accuracy eventually starts shrinking when we start to get "overfitting" in our predictions or when k is too high. This occurs when our model learns our datasets too well and therefore negatively impacts a new random dataset. If our model is too good at knowing our dataset it no longer can generalize the model. In this case k=1 had the same accuracy as k = 21. However, most of the time this is not the case since a small k number of neighbors results in an "underfitting". This is the case when we have too few neighbors and we get high bias and low variance, making the model not fitting or being able to generalize. We want something in between the overfitting and underfitting case.

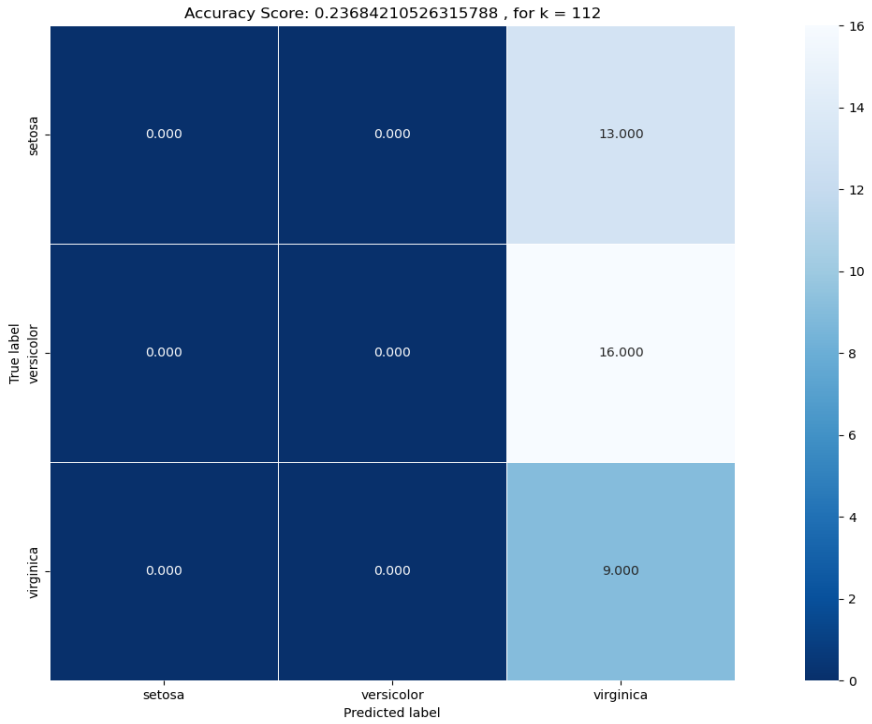
4. Comparing the classification models from k-nearest neighbours.


Accuracy Score: 0.9736842105263158 , for k = 1
setosa
versicolor
virginica
True label
13.000
0.000
0.000
0.000
15.000
1.000
0.000
0.000
9.000
setosa
versicolor
virginica
Predicted label
14
12
10
8
6
4
2
0


Accuracy Score: 0.9473684210526315 , for k = 31
setosa
versicolor
virginica
True label
13.000
0.000
0.000
0.000
14.000
2.000
0.000
0.000
9.000
setosa
versicolor
virginica
Predicted label
14
12
10
8
6
4
2
0


Accuracy Score: 0.8947368421052632 , for k = 41
setosa
versicolor
virginica
True label
13.000
0.000
0.000
0.000
13.000
3.000
0.000
1.000
8.000
setosa
versicolor
virginica
Predicted label
12
10
8
6
4
2
0





Above is the confusion matrices for our KNN-model with different values of K. We chose to plot for five different K-values, one K-value for every time the model's accuracy score decreases. It seems that for all models the versicolor flower was the most difficult to predict correctly. These matrices also give us the information that the accuracy of the model decreases as K grows bigger, as we concluded from task 3.

If we compare our KNN-models with the logistic regression model we can see that for appropriate values of K, the KNN-model is better at predicting correct values. But for poorly chosen K-values, the logistic regression model was better at predicting correct values.

This conclusion seems reasonable because the KNN method is not trying to fit an already decided shape of a curve onto data points, instead the KNN method adapts a curve depending on how many different labels are near it. If you just approximate an appropriate value for K, the KNN method adapts a very good curve.

5. It is important to use separate test sets to be able to see if our model has actually learnt how to predict new data sets. By separating our dataset into a test set and a training set. We train the module on the training set (25% of our entire dataset in our case) and then test it on our test set (the remaining 75%). By doing this we can confirm that the model has learnt something since the test set does not include the same values as the test set. If we were to use the same set in training and testing. We would not be able to confirm that the model has learnt something but instead might have just memorised the data. To explain this we would like to use training a monkey as a metaphor.

If we were to give a monkey a puzzle and train the monkey which puzzle piece connects with another we are training the monkey to solve a puzzle. If we were to give the monkey the same puzzle but to solve it on it's own, we do not know if the monkey has learned how to solve a puzzle or if it has memorised how we trained the monkey to solve it. If we instead give the monkey a different puzzle and the monkey still solves the puzzle, we can conclude that the monkey has learnt how to solve a puzzle.