Assignment Group 2
Felix Dunér (10 h)
Sara Hillström (10 h)

DAT 405 – Lab 2

1.

```
#reading in the data from Hemnet, which we saved in a CSV (col 1 = living area, col 2 = selling price)
df = pd.read_csv('Landvetter_selling_price_living_area.csv')

#creating arrays of the columns. x = living area, y = selling price
x = df.iloc[:, 0].values.reshape(-1, 1)
y = df.iloc[:, 1].values.reshape(-1, 1)

#generating regression model
model = LinearRegression().fit(x,y)

#creating the regression line
xfit = np.linspace(50,220,1000)
yfit = model.predict(xfit[:, np.newaxis])
```

Code for loading data, reshaping it into arrays for x and y and creating the linear regression.

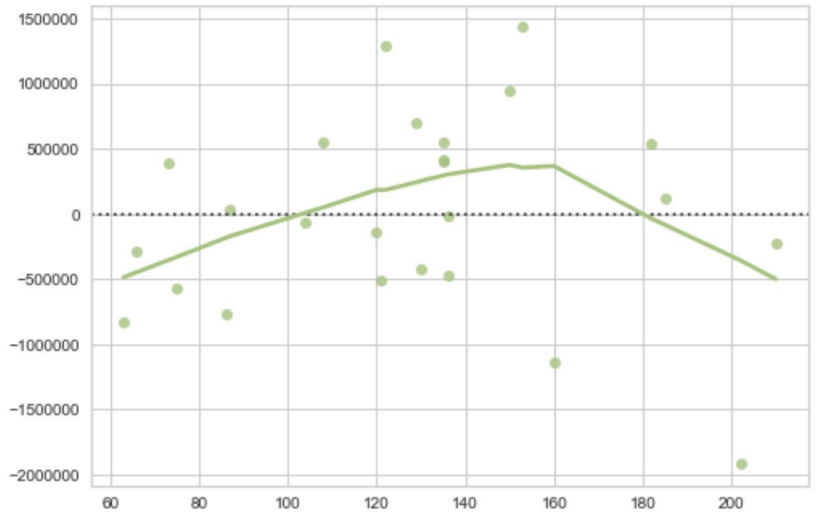
a) Slope = 20 434 (SEK/m²), intercept = 2 039 968 (SEK)

b)

100 m²	150 m²	200 m²
4 083 390 SEK	5 105 101 SEK	6 126 811 SEK

c) *Code for making predictions and comparing them to real values.*

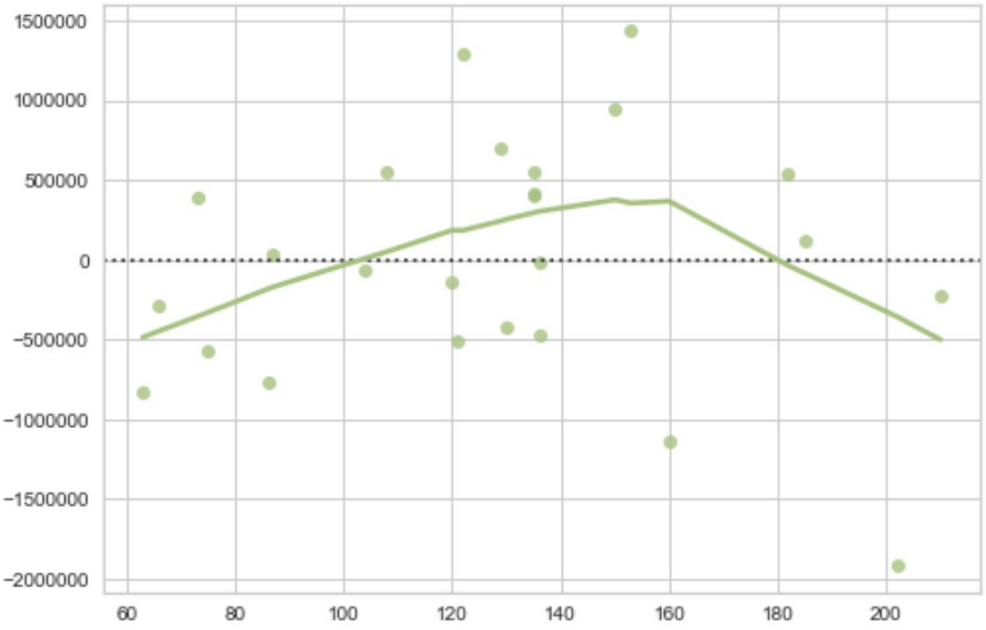
```
#Printing correlation score R2 for model (abs(1)=high linear correlation)
ypred = model.predict(x)
print(f'The correlation score for our model is {r2_score(y,ypred)}')
```



Assignment Group 2
Felix Dunér (10 h)
Sara Hillström (10 h)

d)

Looking at the regression line plot above, the model looks okay although we have a few outliers (especially relative to the total amount of data points) that could indicate that the model is not the best fit. To get a better idea if the model is good or not, we also calculated the model correlation (R) that quantifies the strength of a linear trend. We got the value 0.54, which when looking in relation to a perfect R-value (=abs(1)) indicated that we're not seeing a clear linear trend.

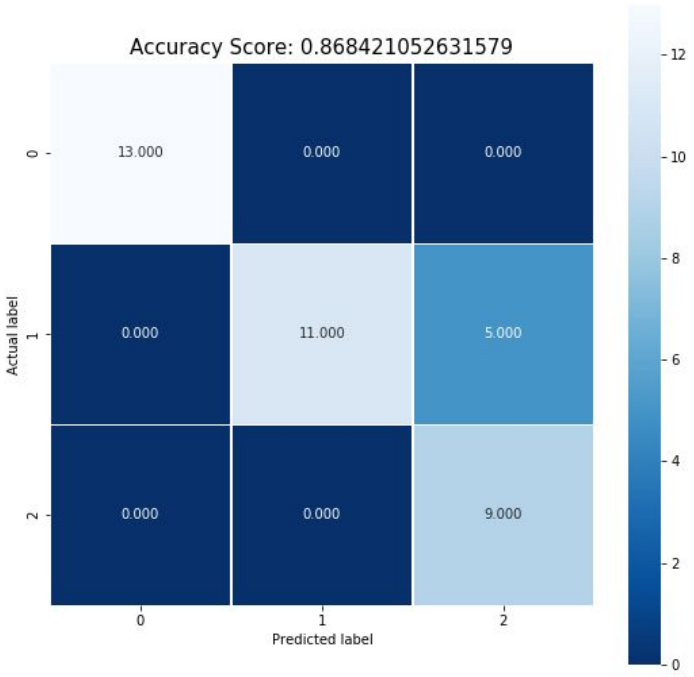


When plotting the residual plot that can be seen above, we did see that the curvature of the data could indicate that a more advanced, non-linear model would be better for the data. But something that we thought contradicts this is the overall random pattern

that the data points in the residual plot form, and to be able to make a definite conclusion on whether the model is good enough or not, we ideally would like to improve the model in some aspects. For instance, it would have been good to have more parameters than living area and selling price. Some parameters to include could be postal code, year the property was built and garden area, since these parameters most likely will have an impact on final selling price. Another way to get a better result could be to have more data points, which would be possible if we included a greater time interval than the past 6 months and/or a bigger geographical area.

If the linear regression model still were to have a lot of outliers, and the residual model still were to show a curvature with less outliers, we would suggest improving the model by using a more advanced type of regression than the current linear one.

2.



Maximal accuracy equals 1.00.

```
#split data into training and test data
x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.25, random_state=0)

#creating the logistic regression, using one vs. rest multiclass classification
logisticRegr = LogisticRegression(multi_class='ovr', solver='liblinear')

#training model on train data
logisticRegr.fit(x_train, y_train)

# Making predictions on entire test data
predictions = logisticRegr.predict(x_test)

# calculate accuracy of model
score_lr = logisticRegr.score(x_test, y_test)
```

Code for splitting data, creating and training logistic regression model.
See 'Assignment 2, Task 1-4.ipynb' for code behind confusion matrix.

3. Depending on the *weight* parameter we can see very different behaviour from the classification algorithm when *k* grows. Firstly, using *uniform* weights and starting with k=1 we receive an accuracy of 97,37%, an accuracy which stands for each k ≤ 23. Using a k-value of 24 or higher the accuracy generally decreases, rather slowly at first and more dramatically when k nears the total number of samples. Note that the accuracy does not continuously decrease when k grows, which the span of k's from 27 to 35 illustrates this. Here we can see that when all points in the spanning neighbourhood are weighted uniformly the accuracy fluctuates, falling to around 89% when k=29 only to reach peak accuracy again when k=35. This kind of behaviour is not recorded when the points are weighted by the inverse of their distance (which is the case when the weights parameter is set to 'distance').

Value of k	Weights = uniform	Weights = distance
27	94.74	97.37
28	92.11	97.37
29	89.47	97.37
30	94.74	97.37
31	94.74	97.37
32	97.37	97.37
33	92.11	97.37
34	94.74	97.37
35	92.11	97.37

How accuracy (% of correct classifications) changes.

The reason behind this fluctuation in accuracy is that when each point has an equal effect on the classification result, no matter how far away it is from the point which is to be classified, including one more point might have a significant impact on the result. When instead taking into account the distance from the point to be classified to the points included in the so-called neighbourhood limited by our k-value, and their effect on the outcome thereafter we get a completely different result. No matter which k we choose to use (with reservation for not trying *all* possible values of k...) we still receive what seems to be peak accuracy for this particular model and dataset (≈ 97%). The impact of including new, far-away points which with uniform weights might've had a real effect on the classification outcome is now mitigated by taking their distance into account. Although what seems to be a more accurate model is obtained through weighting distance and taking all points in account, applying this

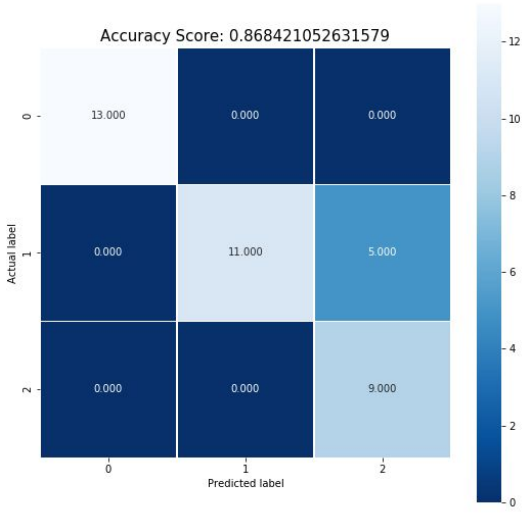
strategy to large-scale data might lead to extremely demanding computations.

Value of k	Weights = uniform	Weights = distance
1	97.37	97.37
11	97.37	97.37
21	97.37	97.37
31	94.74	97.37
41	89.47	97.37
51	89.47	97.37
61	89.47	97.37
71	65.79	97.37
81	63.16	97.37
91	60.53	97.37
101	57.89	97.37
111	23.68	97.37

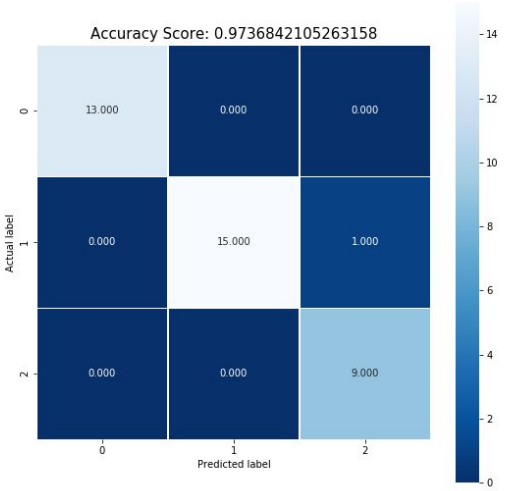
How accuracy (% of correct classifications) changes.

4. Using both *logistic regression* and *k-nearest neighbour* (with different k-values and weighting) we obtained different results regarding classification accuracy. Comparing these two models, we selected a few different k-values when using uniform weights for the k-NN models to illustrate how its accuracy decreases when the k-value increases. Weighting the distance of the points, we decided to use k-values of 11 and 111 to highlight the implications this has on the accuracy of the model despite a high k-value. Below are the confusion matrices for the logistic regression and k-NN models respectively. As we can see the optimized k-NN classifier performed quite clearly better than the one using logistic regression, with only one wrongful classification versus five, resulting in an accuracy of more than 97 % using k-NN with the linear regression model at barely 87 %. The logistic regression model outperforms its k-NN equivalent only when the k-NN models are using uniform weights combined with a k-value somewhere around 60-70 and upwards. A discussion around how and why the k-NN model behaves the way it does can be found in the previous question.

Assignment Group 2
Felix Dunér (10 h)
Sara Hillström (10 h)

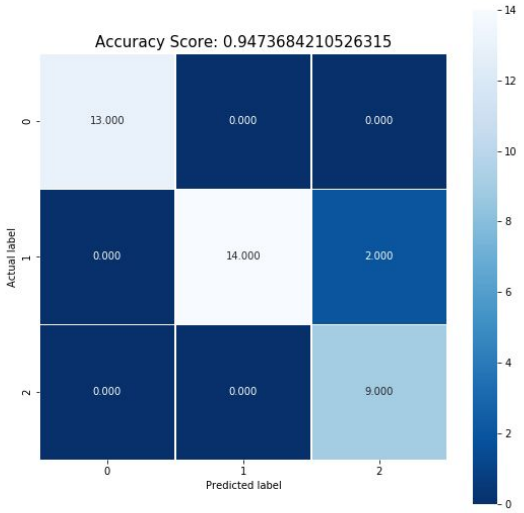


Confusion matrix for the logistic regression model where maximal accuracy equals 1.00.

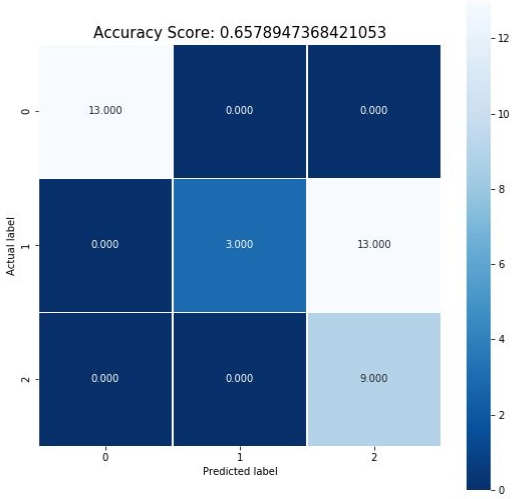


Confusion matrix for the k-NN model with k = 11, weights = uniform.

Assignment Group 2
Felix Dunér (10 h)
Sara Hillström (10 h)

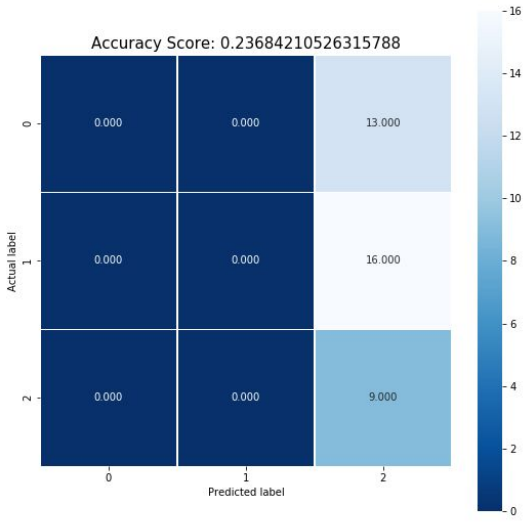


Confusion matrix for the k-NN model with k = 31, weights = uniform.

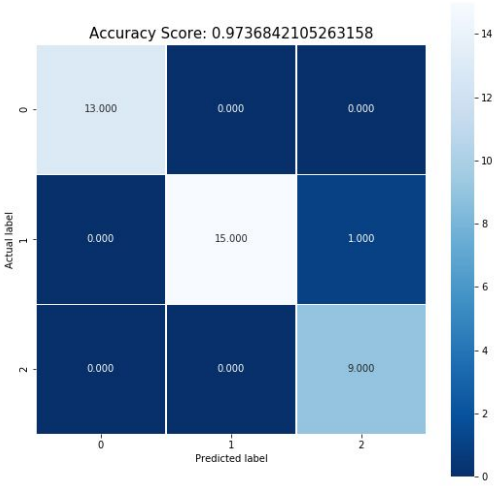


Confusion matrix for the k-NN model with k = 71, weights = uniform.

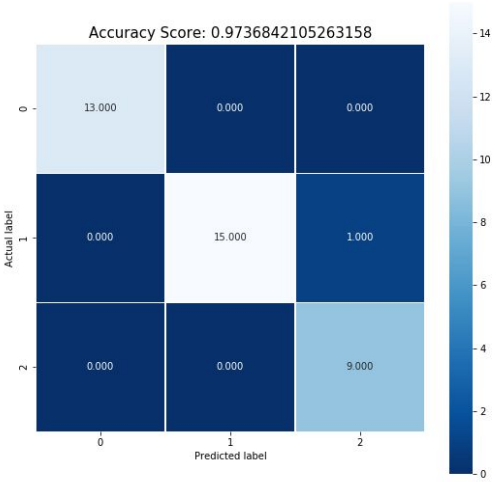
Assignment Group 2
Felix Dunér (10 h)
Sara Hillström (10 h)



Confusion matrix for the k-NN model with k = 111, weights = uniform.



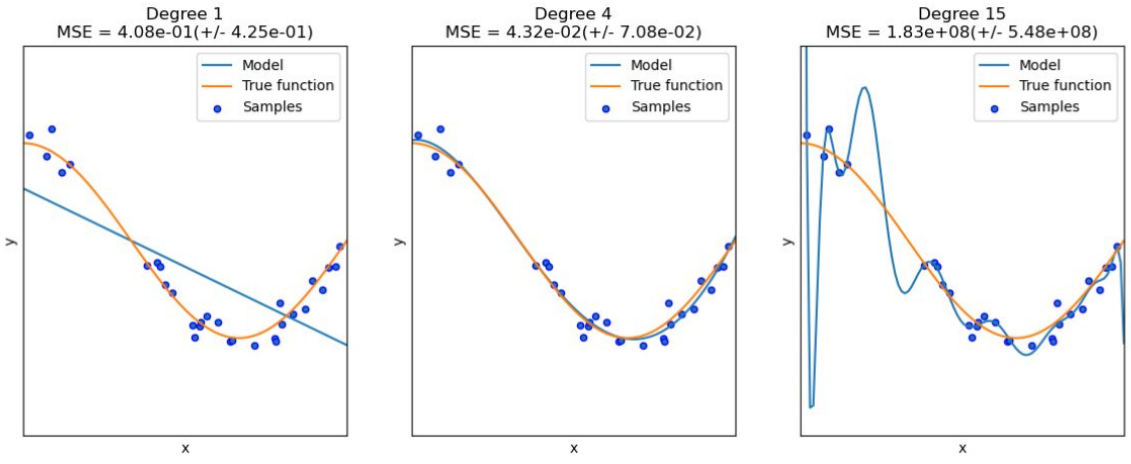
Confusion matrix for the k-NN model with k = 11, weights = distance.



Confusion matrix for the k-NN model with k = 111, weights = distance.

5. After using training sets to learn the data trend the model uses test sets to unbiasedly evaluate the final model fit, this in comparison to other models. Using validation sets on the other hand, provides an estimate of the model fit while still adjusting the model parameters and preparing data.

   It is important to use test sets to avoid the risk of over- and underfitting the model. By overfitting a model it's meant that the model describes error in the data rather than relationships between the variables. As can be seen in the furthest right graph in the screenshot below, overfitting causes the graph to consider data points (errors) that are off from the general data trend. To the left below we can see an underfit model, which doesn't consider many of the data variations and makes strong assumptions about the data.



Source: https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html