Lukas Jigberg | 10 hours

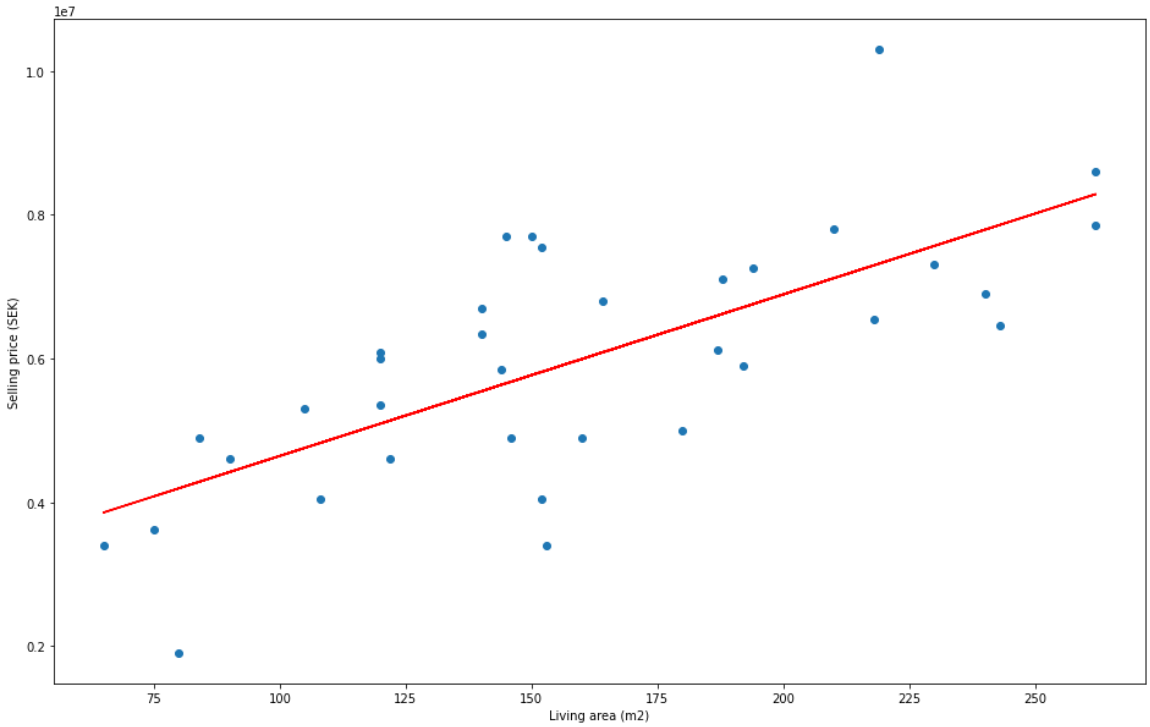
1. Hemnet Data

Regression models & Data

In [90]:

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

hd = pd.read_csv('dataSets/hemnetLandvetter.csv')
LA_KEY = 'Size'
SP_KEY = 'Selling price'

y = pd.DataFrame(hd[SP_KEY])
x = pd.DataFrame(hd[LA_KEY])

# Make the graphs bigger
plt.rcParams['figure.figsize'] = (16,10)

# Create the model
model = LinearRegression().fit(x, y)

# Fit the line
xfit = x.values
yfit = model.predict(xfit)

# Plot the graph
plt.xlabel("Living area (m2)")
plt.ylabel("Selling price (SEK)")
plt.scatter(x, y)
plt.plot(xfit, yfit, color='red')
plt.show()

print()
# model.coef for slope and model.intercept for the interception
print("Slope:",(model.coef_),"Intercept:",(model.intercept_))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X
does not have valid feature names, but LinearRegression was fitted with fe
ature names
  "X does not have valid feature names, but"
```




```
Slope: [[22477.80691977]] Intercept: [2394868.38645922]
```

Prediceted house prices for living areas 100, 150 200 m2

In [91]:

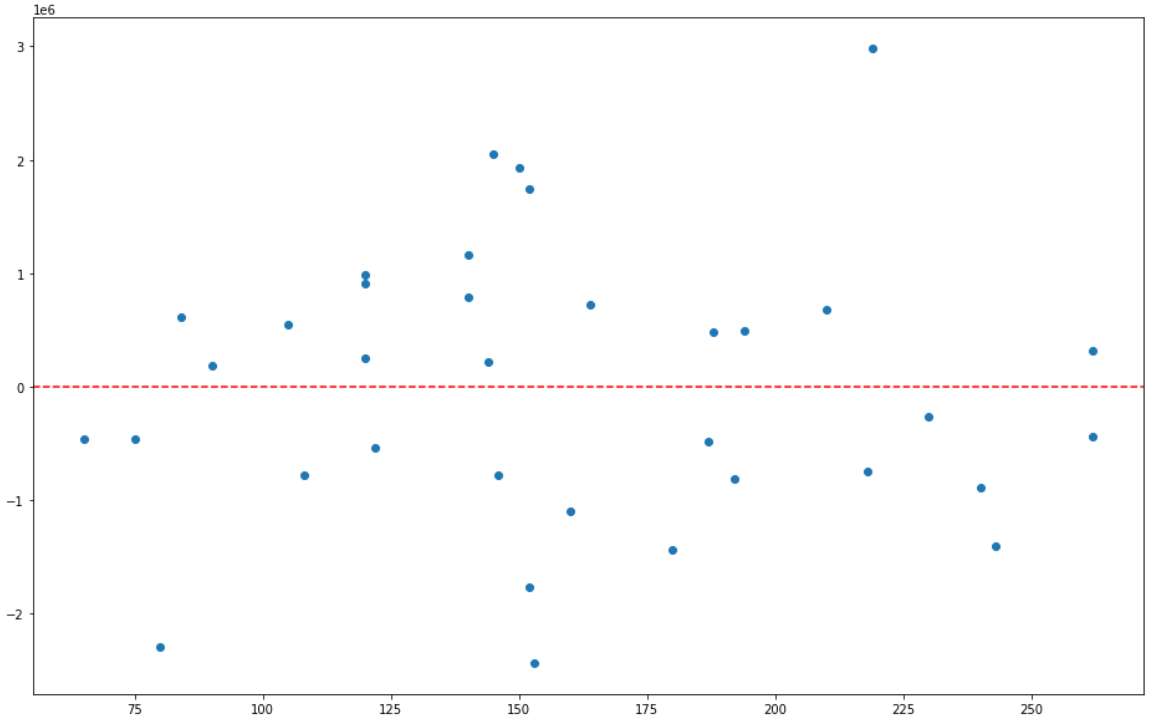
```
predict_values = np.array([100,150,200]).reshape(-1,1)
print(model.predict(predict_values))
```

```
[[4642649.07843633]
 [5766539.42442488]
 [6890429.77041344]]
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X
does not have valid feature names, but LinearRegression was fitted with fe
ature names
  "X does not have valid feature names, but"
```

In [92]:

```
residuals = y - yfit
plt.scatter(x, residuals)
plt.axhline(y=0, color='r', linestyle="dashed")
plt.show()
```



Discussion

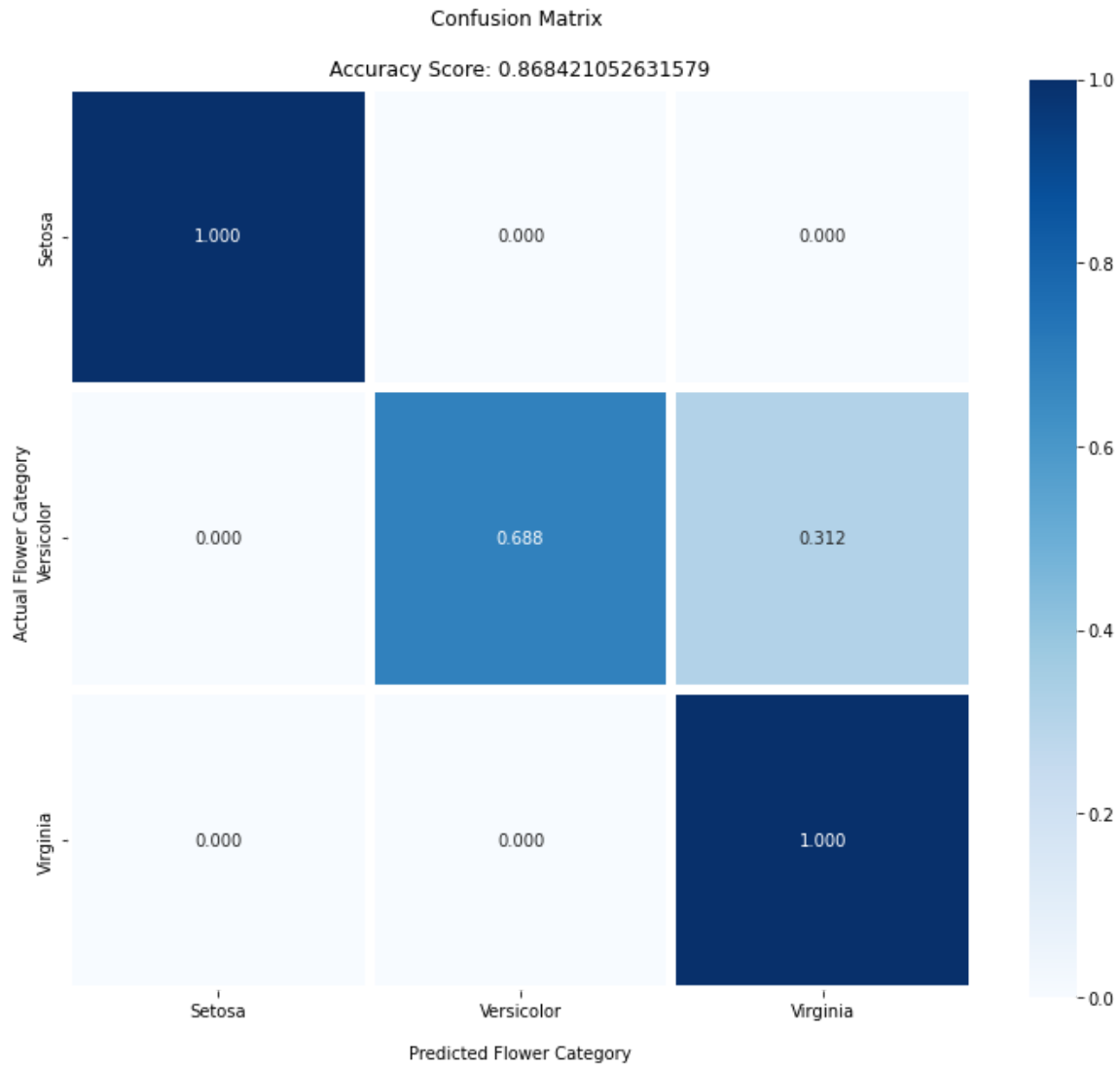
Is the data we show reliable? \ Looking at the two graphs there are some outlining dots that we might need to investigate as a safety percussion. We of course see them on both graphs, but they are easier to read on the second, residual. We look at the dots that are far away from our regression line and see if there is a factor to why they cost more/less than the others in the close area. For this exercise there is no need to go in depths on what caused that particular price and we can instead just double check the data/advert on hemnet. The spread that we get around the regression line is plausible and is predicted to be flexible. The final price of a house is product of many factors. Size (not just square meters), shape, area, beauty etc. So looking at only square meters will not tell the whole story. But since square meters creates a rather nice regression line to price is it safe to presume that square meters may be one of / or the biggest factor when establishing house prices.

I would not recommend using the graphs to predict current or future house prices in the area. As we just discussed there is a huge flexibility in house prices due to its many factors, so guessing/predicting on the spot would not be accurate. If we would instead predict in a range, determining that the price would be between x million and y million, would be a lot better, but not accurate enough and not that helpful to the seller and buyer. A ballpark figure yes, but we don’t need graph and data to be inaccurate. I suggest that there is to many factors to houses to estimate a clear price and suggest using another method to predict prices efficiently.
To improve the data may it be good to consider removing the big outliers or restructuring the data from the beginning so that they aren’t apart of it. They affect the graphs much and don’t fit into the “normal” ‘housing price to square meter‘ for that area. They are extremes and using them to predict the “normal” will lead to inaccuracy. But the data needs to be examined before removing it. It may be the case that the outlaying house prices are the actuall norm and that the other house prices are the outliers. Removing data the instant it seems out of place will give more accuracy to the data we use now at the moment, but may make future data assessment inaccurate or straight up false.

2. Confusion Matrix

In [93]:

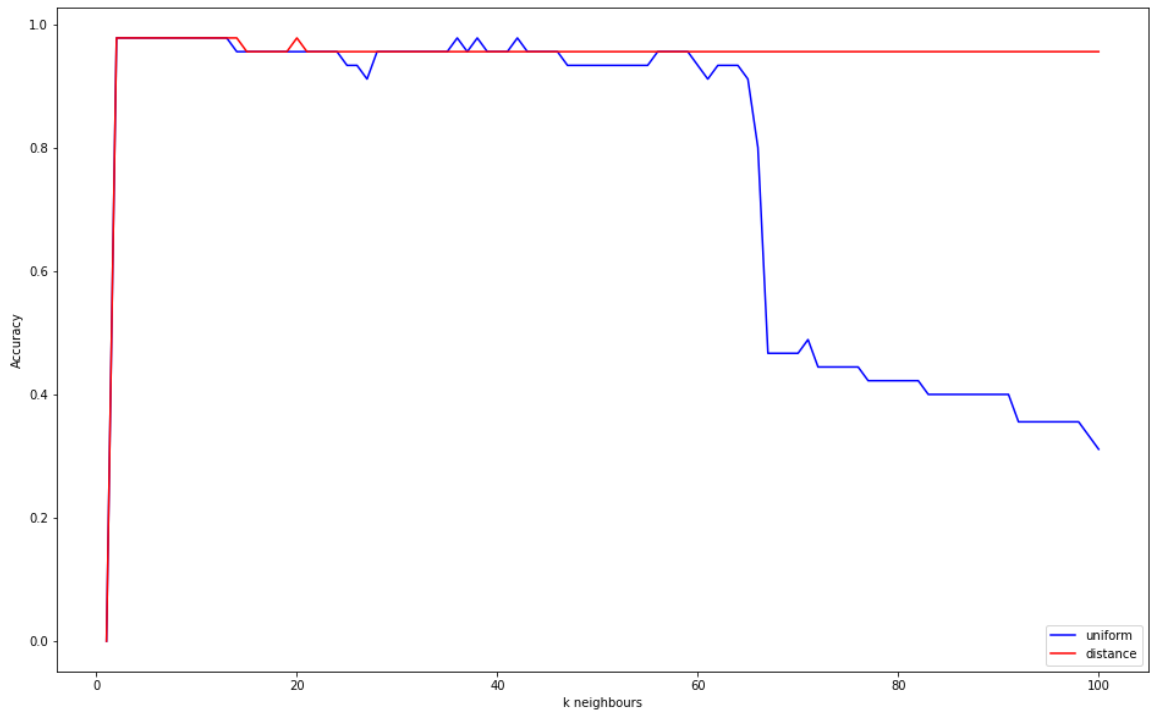
```
from sklearn.datasets  import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
import seaborn as sns

xi = iris.data
yi = iris.target

# Change the graph size
plt.rcParams['figure.figsize'] = (12,10)

# Splits the data into training and tests. The test size is 25% of the total.
x_train, x_test, y_train, y_test = train_test_split(xi, yi, test_size=0.25, random_stat
e=0)
model = LogisticRegression(solver='liblinear', multi_class='ovr').fit(x_train, y_train)

# Get the models accuracy score and predict the test data.
predictions = model.predict(x_test)
score = model.score(x_test,y_test)

# Creates a confusion matrix to plot
cm = confusion_matrix(y_test, predictions, normalize='true')


# Plotting
ax = sns.heatmap(cm, annot = True, fmt=".3f", linewidths=5, square=True, cmap='Blues' )

ax.set_title('Confusion Matrix \n\nAccuracy Score: {0}'.format(score))
ax.set_xlabel('\nPredicted Flower Category')
ax.set_ylabel('Actual Flower Category ');

ax.xaxis.set_ticklabels(['Setosa','Versicolor', 'Virginia'])
ax.yaxis.set_ticklabels(['Setosa','Versicolor', 'Virginia'])

plt.show()

# Return the graph size
plt.rcParams['figure.figsize'] = (16,10)
```


Confusion Matrix
Accuracy Score: 0.868421052631579
Setosa
Versicolor
Virginia
Actual Flower Category
1.000
0.000
0.000
0.000
0.688
0.312
0.000
0.000
1.000
Setosa
Versicolor
Virginia
Predicted Flower Category
1.0
0.8
0.6
0.4
0.2
0.0

3. K

In [94]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

maxK = 100
accuracy = np.zeros(shape=[2,maxK])
x_train, x_test, y_train, y_test = train_test_split(xi, yi, test_size=0.3)

# Runs the test confusion test (uniform) maxK times, stores the accuracy results.
for k in range(1,maxK):
    knn = KNeighborsClassifier(n_neighbors= k,weights='uniform')
    knn.fit(x_train,y_train)
    y_pred = knn.predict(x_test)
    accuracy[0,k] = metrics.accuracy_score(y_test,y_pred)

# Runs the test confusion test (distance) maxK times, stores the accuracy results.
for k in range(1,maxK):
    knn = KNeighborsClassifier(n_neighbors= k,weights='distance')
    knn.fit(x_train,y_train)
    y_pred = knn.predict(x_test)
    accuracy[1,k] = metrics.accuracy_score(y_test,y_pred)

# Plotting
xAxisData = np.linspace(1,maxK,maxK)
plt.plot(xAxisData,accuracy[0,:],label='uniform', c='blue')
plt.plot(xAxisData,accuracy[1,:],label='distance', c='red')
plt.legend()
plt.xlabel('k neighbours')
plt.ylabel('Accuracy')
plt.show()
```



Note that the plot will change each time you run it.

Running it a few times makes it apparent that: \ Distance keeps its good result over time. \ Uniform drops of quickly after around 60 neighbours. \ Using distance always gives better results.

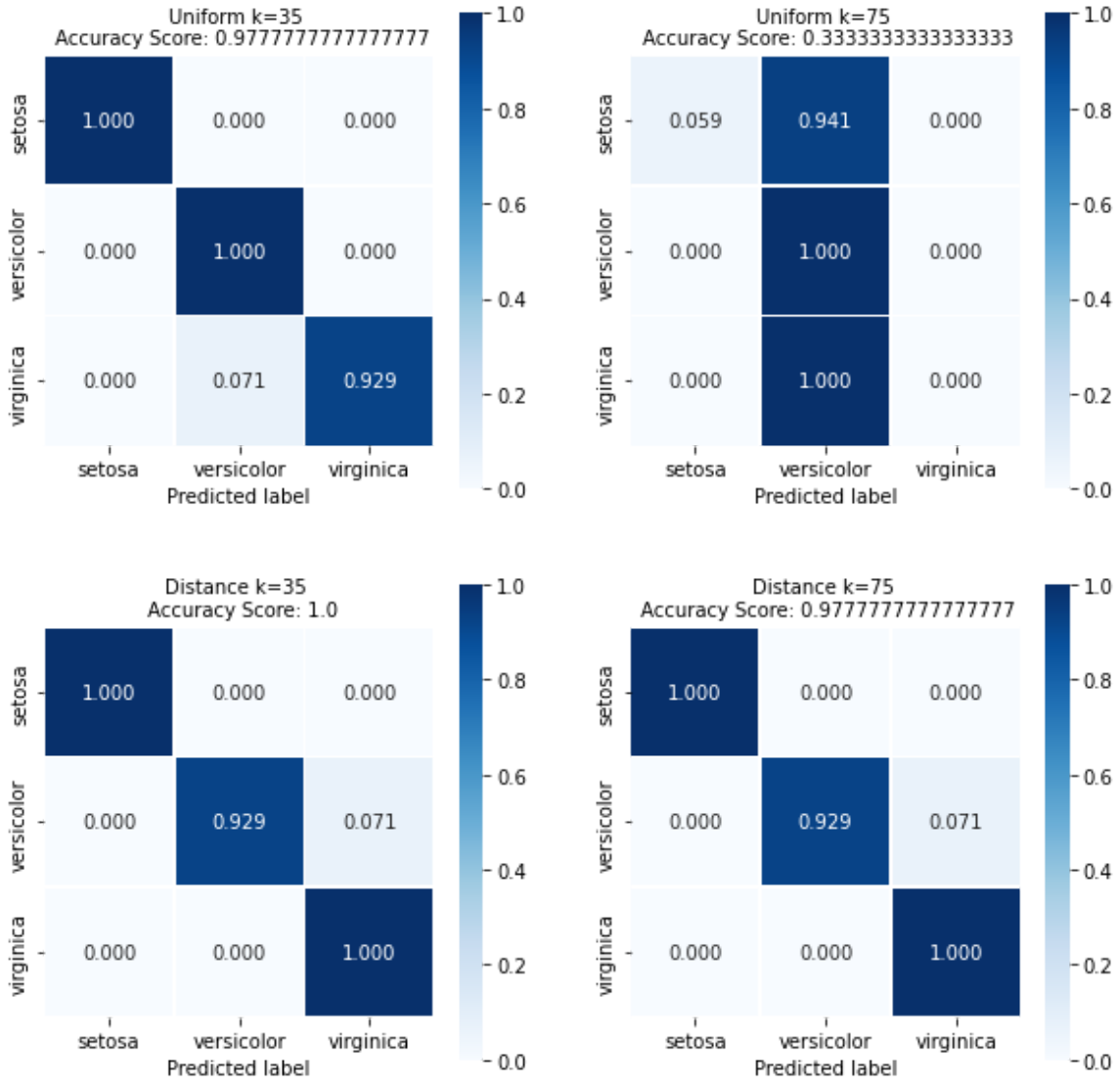
Uniform decreasing isn't that weird when we consider that if we place a new dot and check almost all other dots to see which there are more of ‘close to it’, distance will not matter anymore. The most impactful factor will instead be the number of species dots. Placing the species Setosa close to its brethren will not matter if there is only 10 of them and we are checking for 50 species. It will become the same as the species with the most already placed dots.

4. Compare Models

To create the confusion matrixes must we first choose specific k values to compare with. We have already created a graph that shows how accurate the different K’s are so it maybe look a bit redundant creating matrices. But two good k values would be 35 and 75. One that is standard and one that is more on the deep end. \ The plan is to create 4 matrices, 1 for each unique pair of K and distance/uniform. We then compare those 4 and the logistical regression we created for our first matrix.

In [95]:

```
x_train, x_test, y_train, y_test = train_test_split(xi, yi, test_size=0.3)

kU35 = KNeighborsClassifier(n_neighbors=35,weights='uniform')
kU75 = KNeighborsClassifier(n_neighbors=75,weights='uniform')
kD35 = KNeighborsClassifier(n_neighbors=35,weights='distance')
kD75 = KNeighborsClassifier(n_neighbors=75,weights='distance')


# 'Trains' all the classifiers
for classifier in [kU35, kU75, kD35, kD75]:
  classifier.fit(x_train, y_train)

# Tries to predict the test data
kU35_predict = kU35.predict(x_test)
kU75_predict = kU75.predict(x_test)
kD35_predict = kD35.predict(x_test)
kD75_predict = kD75.predict(x_test)

# Get the accuracy scores
kU35_score = kU35.score(x_test,y_test)
kU75_score = kU75.score(x_test,y_test)
kD35_score = kD35.score(x_test,y_test)
kD75_score = kD75.score(x_test,y_test)

# Create matrixes
kU35_cm = confusion_matrix(y_test, kU35_predict, normalize = 'true')
kU75_cm = confusion_matrix(y_test, kU75_predict, normalize = 'true')
kD75_cm = confusion_matrix(y_test, kD35_predict, normalize = 'true')
kD75_cm = confusion_matrix(y_test, kD75_predict, normalize = 'true')

# Plots graphs
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2,2, figsize=(10,10))
sns.heatmap(kU35_cm, ax = ax1, annot = True, fmt=".3f", linewidths=.5, square=True, cma
p='Blues' )
sns.heatmap(kU75_cm, ax = ax2, annot = True, fmt=".3f", linewidths=.5, square=True, cma
p='Blues' )
sns.heatmap(kD75_cm, ax = ax3, annot = True, fmt=".3f", linewidths=.5, square=True, cma
p='Blues' )
sns.heatmap(kD75_cm, ax = ax4, annot = True, fmt=".3f", linewidths=.5, square=True, cma
p='Blues' )

for i in [ax1, ax2, ax3, ax4]:
  i.set_xlabel('Predicted label')
  i.set_xticklabels(iris.target_names)
  i.set_yticklabels(iris.target_names)

ax1.set_title('Uniform k=35 \n Accuracy Score: {0}'.format(kU35_score), size=10)
ax2.set_title('Uniform k=75 \n Accuracy Score: {0}'.format(kU75_score), size=10)
ax3.set_title('Distance k=35 \n Accuracy Score: {0}'.format(kD35_score), size=10)
ax4.set_title('Distance k=75 \n Accuracy Score: {0}'.format(kD75_score), size=10)


plt.show()
```



The analytic score results are the same as for the accuracy graph. K Distance has kept its accuracy while Uniform has decreased. Running it over and over will usually result in a loss of around 30% accuracy for uniform. Proving its unreliability when handling larger K. \ Unlike the accuracy graph does the matrices provide some more information. We see that the species that was the most difficult to categorize was Virginica followed by Versicolor. This tells us that there was flower data that was hard to distinguish and could easily be mistaken for the other. \ Comparting our new knn matrices with the Logistic regression matrix from question 2 suggest that knn Distance is the clear winner in accuracy. Scoring around 97% almost always while Log Reg scores around 85%.

The importance of using seperate test sets

Testing with the same data as we trained the model will not help ss predict and minimize errors. If the model is well versed with one problem, then we shouldn’t keep testing that one. It isn’t useful for our cause to create a model that shouldn be able to handle every specific combination. That is why we want to subjugate our model to different problems so that it gets the chance to work out those flaws. We are able to see these problems when the model fails to catagorize something, showing us that we need to tweek it.\ This feeds into the idea that we need to use more then 1 test method to achieve/find the best result. For example, we saw in the accuracy graph and the matrices for K that there is a clearly ‘preferred’ specific k, a K that will give us the best result. Choosing one at random may be harmful to our data and testing might be the only way to find the best outcome.

Another thing to note is that whilst we do want to create a test that is different then the one we used training we don’t want to ‘over train’ our model. Simply running all data at once with all the may cause very varied result each time which are hard to tweak. Instead, we might want to take the test in turn and find part test that give us a desirable result and then sow together the perfect training session, rather than simply getting a result that has a hard time finding results due to how big and flexible the data can be. A good thumb rule is to be careful that the training is done properly and not subjugating the model to 'weird' or 'unfair' data that doesn't help it learn or might even prove to be harmful. Remove big outliers or be cautious about, for example, not including flowers that could be either versicolor or virginica according to experts.