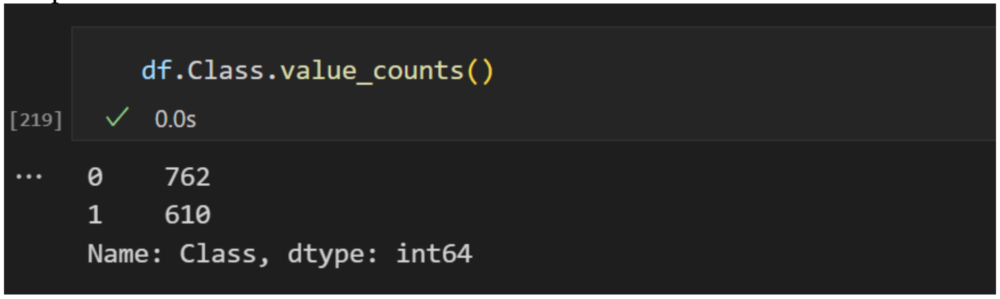
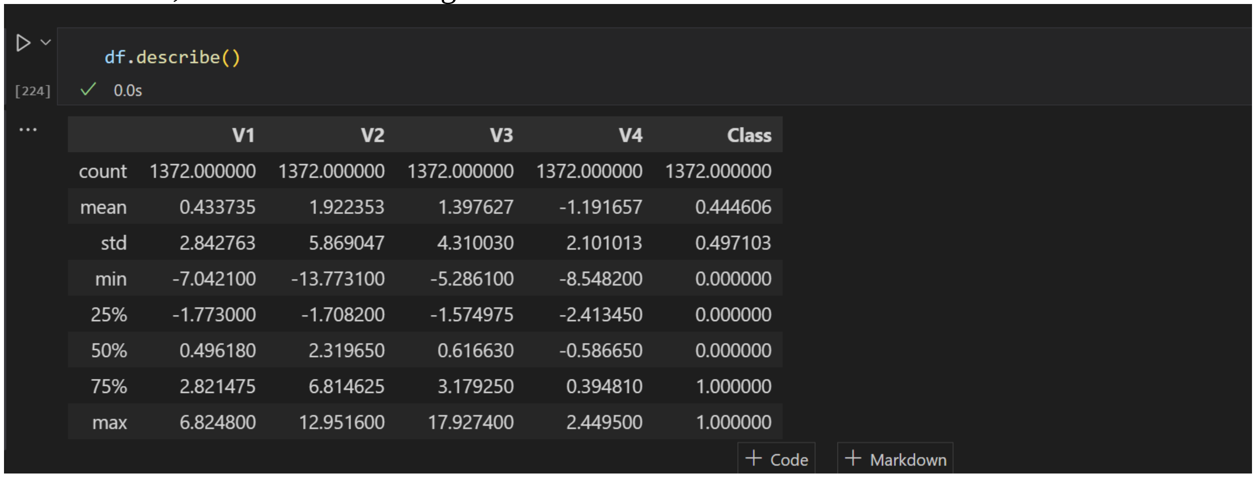
a) The full data set consisted of 5 columns, fours columns are the parameter that was used to tell the class of bank notes. We assumed (from the two values presented in classes are only ones and zeros) that if the class is one, the bank note is real and if not, the bank note is fake.

b) Exploring the data to see what values are presented in the dataset by using this method in the picture below.



```
df.Class.value_counts()
[219]  ✓  0.0s

...  0    762
     1    610
     Name: Class, dtype: int64
```

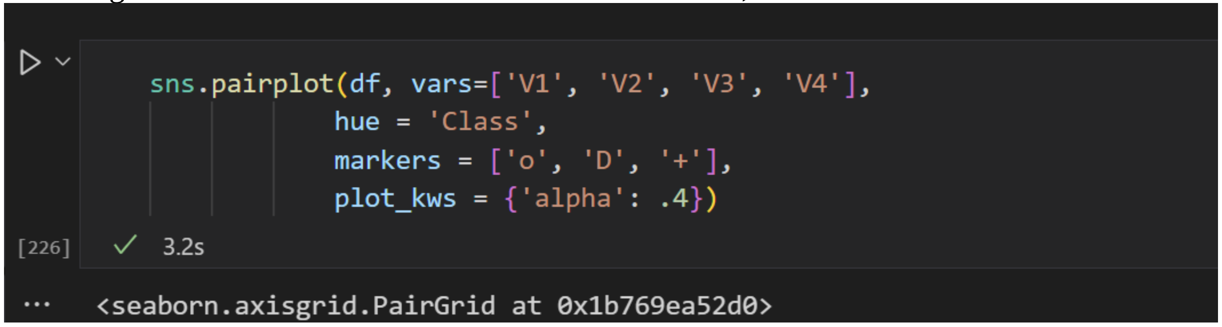
As mentioned above, the value in the Class column contains only 0 and 1 which was used to interpret that the value predicted should never be greater than 1. For the other columns in the dataset, we use the following method.



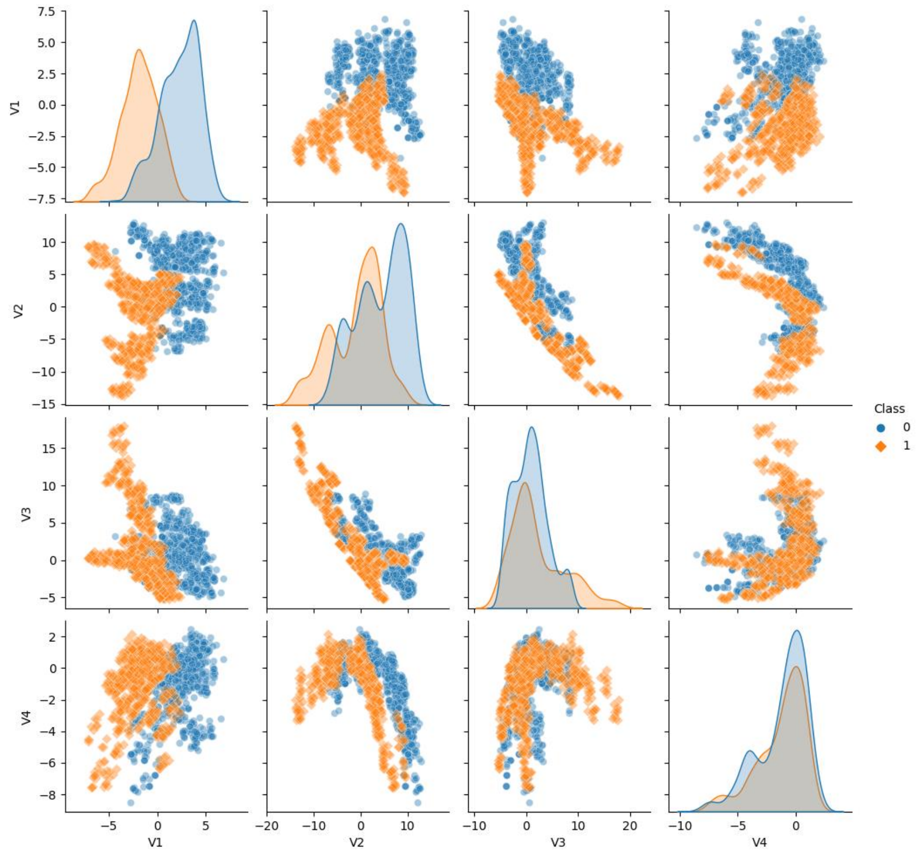
```
df.describe()
[224]  ✓  0.0s
```

	V1	V2	V3	V4	Class
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

The method above describes the overview of the data. This will tell us the useful information like the mean, standard deviation and the data distribution percentage including min and the max value. To visualize the dataset, we use the method below.

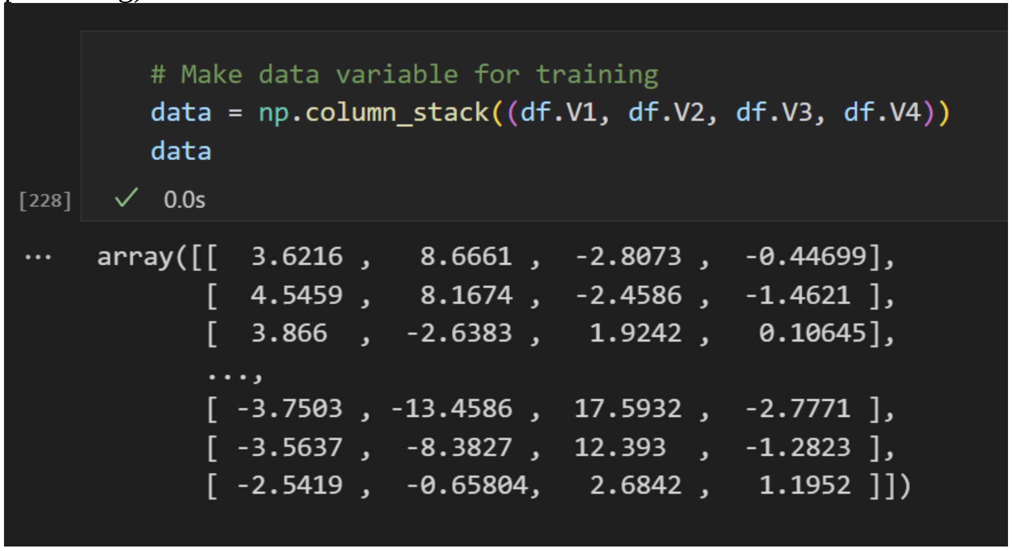


```
sns.pairplot(df, vars=['V1', 'V2', 'V3', 'V4'],
             hue = 'Class',
             markers = ['o', 'D', '+'],
             plot_kws = {'alpha': .4})
[226]  ✓  3.2s

...  <seaborn.axisgrid.PairGrid at 0x1b769ea52d0>
```



This is a visual representation of the data. The cluster scatter that is orange are belong to class 1 and the data cluster scatter that is blue are belong to class 2. This explains that there are two main clusters of data.

c) When training the model by using K-Means clustering, a data readable and computable by Python was needed. The method below was used to make a Numpy column stack for all the data columns except for the last column. (Which is the one that we will be predicting).

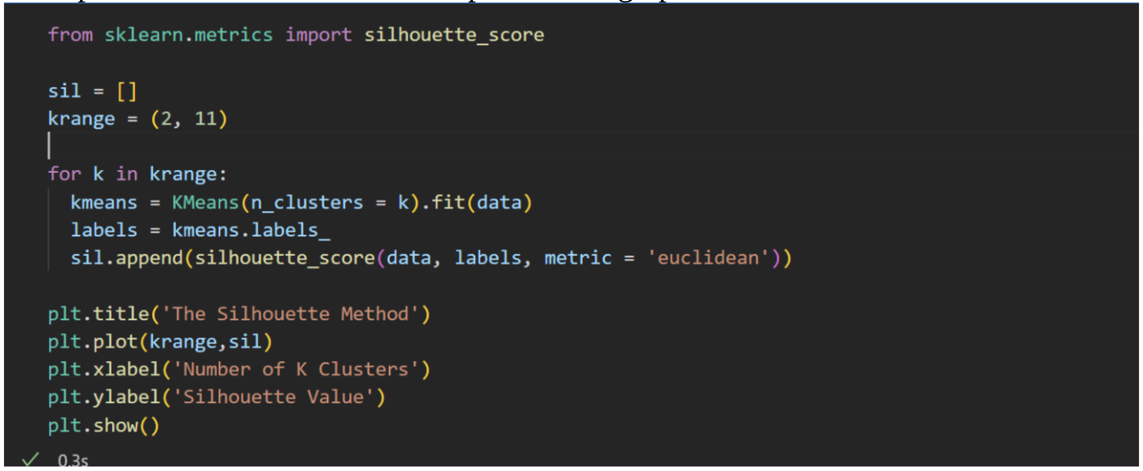


```
# Make data variable for training
data = np.column_stack((df.V1, df.V2, df.V3, df.V4))
data
```

```
[228]  ✓  0.0s
```

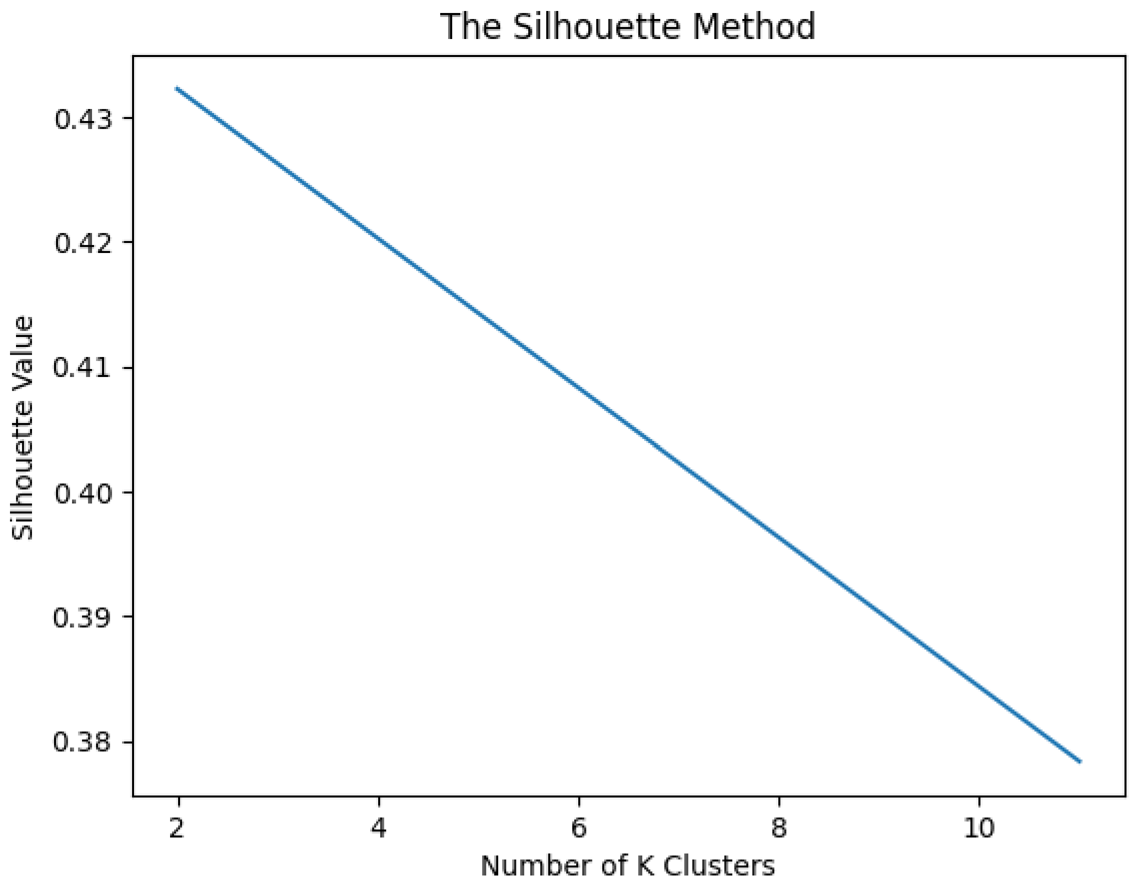
```
...  array([[  3.6216 ,   8.6661 ,  -2.8073 ,  -0.44699],
            [  4.5459 ,   8.1674 ,  -2.4586 ,  -1.4621 ],
            [  3.866  ,  -2.6383 ,   1.9242 ,   0.10645],
            ...,
            [ -3.7503 , -13.4586 ,  17.5932 ,  -2.7771 ],
            [ -3.5637 ,  -8.3827 ,  12.393  ,  -1.2823 ],
            [ -2.5419 ,  -0.65804,   2.6842 ,   1.1952 ]])
```

Both for the model evaluation step, the silhouette score was used to find the value of K. The optimal K value is found in the peak of the graph which is K = 2.

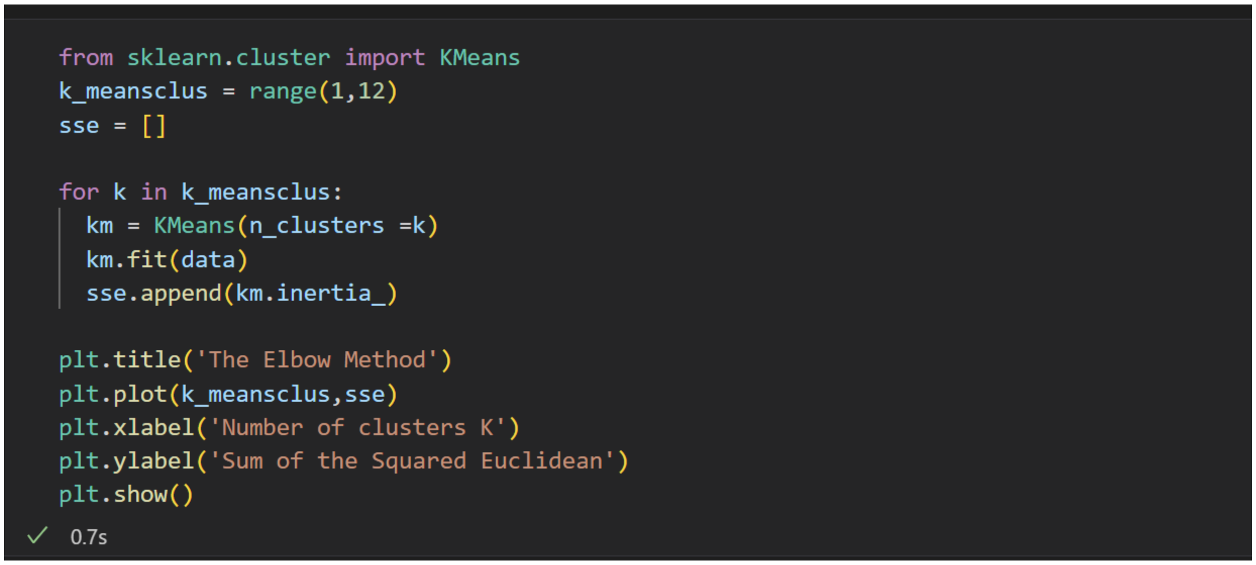


```
from sklearn.metrics import silhouette_score

sil = []
krange = (2, 11)

for k in krange:
  kmeans = KMeans(n_clusters = k).fit(data)
  labels = kmeans.labels_
  sil.append(silhouette_score(data, labels, metric = 'euclidean'))

plt.title('The Silhouette Method')
plt.plot(krange,sil)
plt.xlabel('Number of K Clusters')
plt.ylabel('Silhouette Value')
plt.show()
```

✓ 0.3s

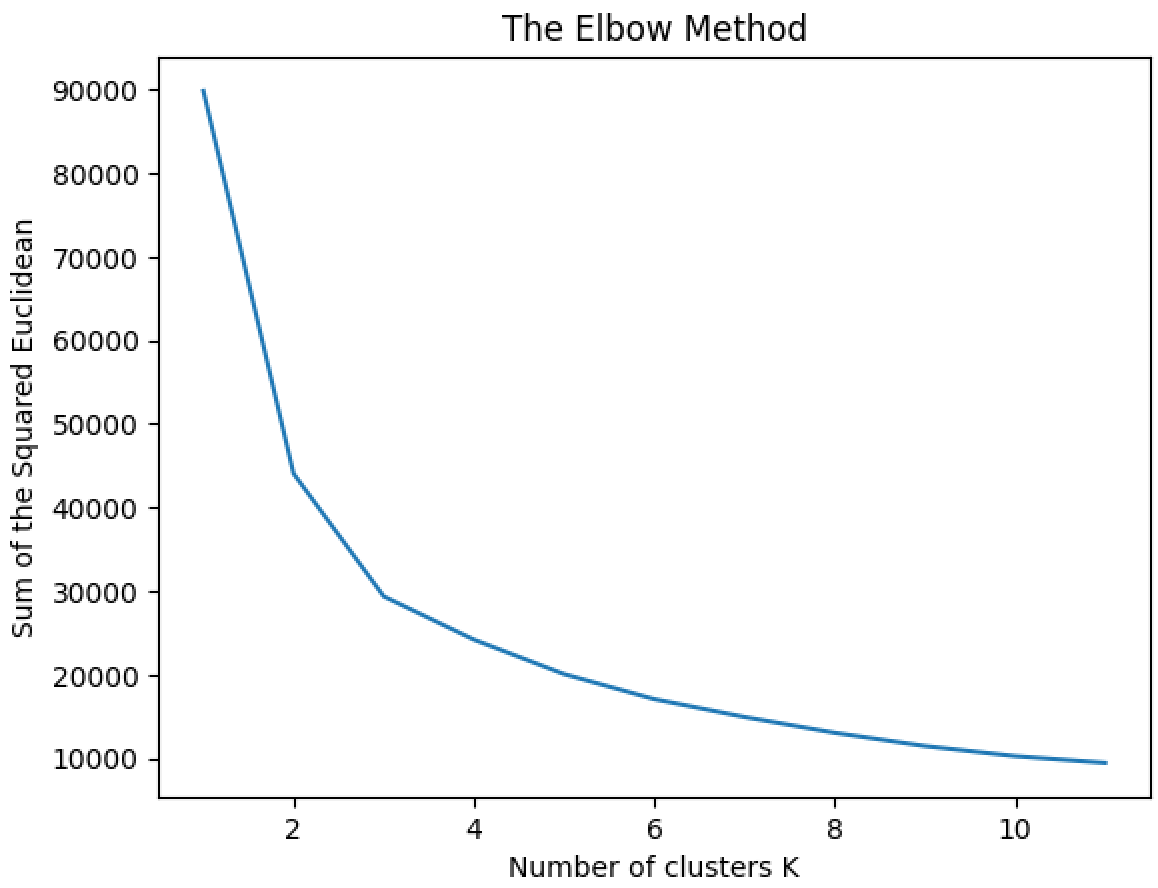


The elbow method was also used to find the optimal value of K. The value chosen for K is viewed at the "elbow junction of the graph".

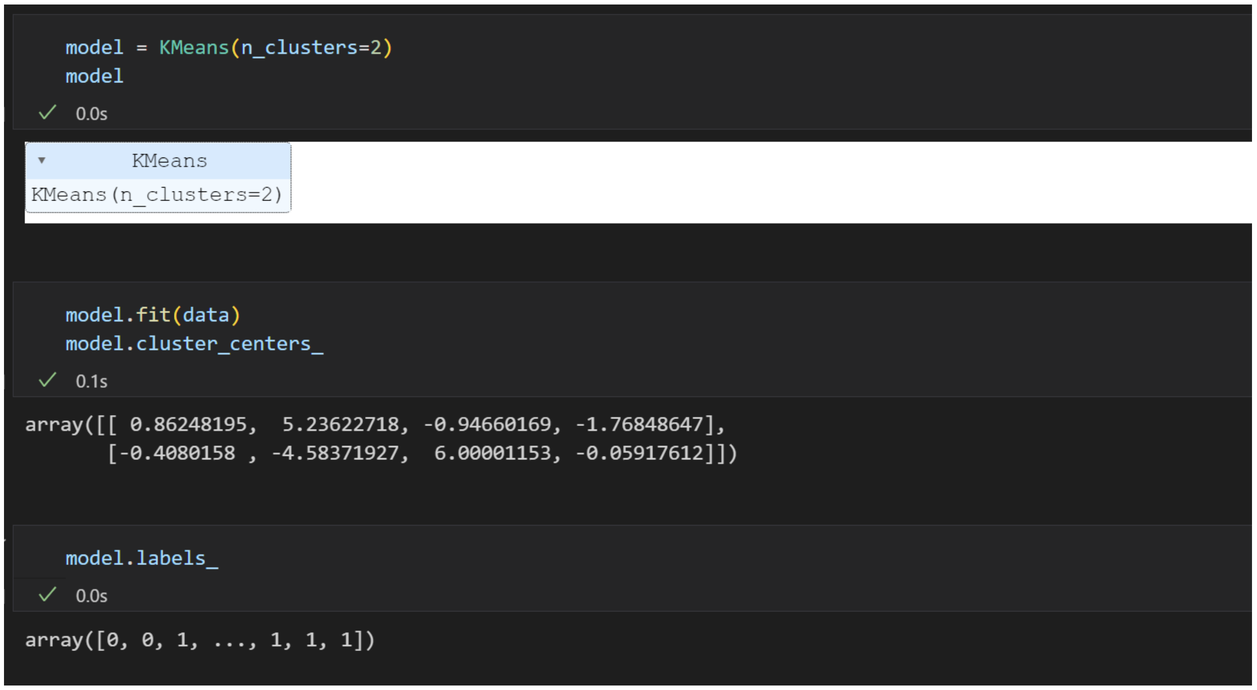


```
from sklearn.cluster import KMeans
k_meansclus = range(1,12)
sse = []

for k in k_meansclus:
    km = KMeans(n_clusters =k)
    km.fit(data)
    sse.append(km.inertia_)

plt.title('The Elbow Method')
plt.plot(k_meansclus,sse)
plt.xlabel('Number of clusters K')
plt.ylabel('Sum of the Squared Euclidean')
plt.show()
```

✓ 0.7s



After finding the optimal K-Value, it is time to use the K-Value estimated to train our model and compute out the value by using these functions



```
model = KMeans(n_clusters=2)
model
```
✓ 0.0s

```
▾ KMeans
KMeans(n_clusters=2)
```

```
model.fit(data)
model.cluster_centers_
```
✓ 0.1s

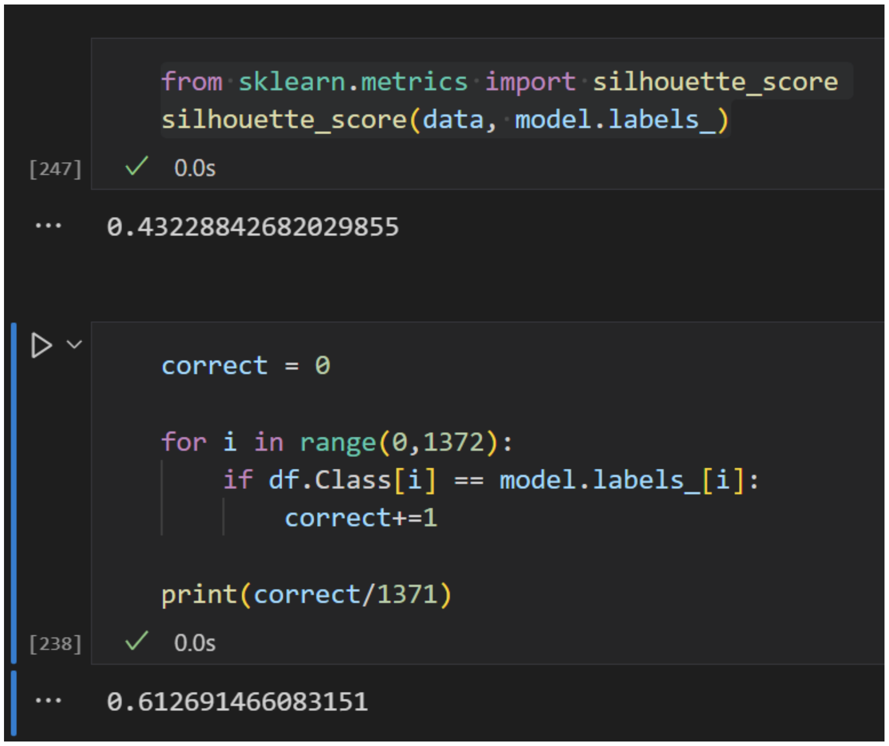
```
array([[ 0.86248195,  5.23622718, -0.94660169, -1.76848647],
       [-0.4080158 , -4.58371927,  6.00001153, -0.05917612]])
```

```
model.labels_
```
✓ 0.0s

```
array([0, 0, 1, ..., 1, 1, 1])
```

The model.labels_ is the array of computed values that was used to predict the value of "Class". The Numpy array (data) that was made ago was used to train the data and output the statement above (model.labels and model.cluster_centers_). The cluster centers are found inside the variable model.cluster_centers_.

d) When assessing the model, these two metrics are used. The silhouette score and the correctness value.



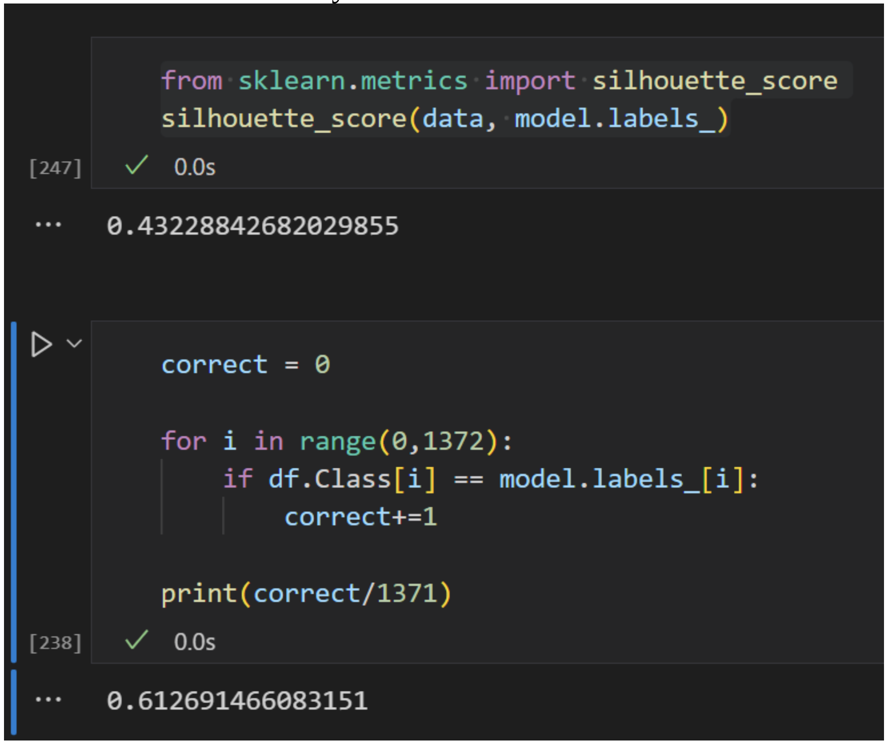
```
from sklearn.metrics import silhouette_score
silhouette_score(data, model.labels_)
```
[247] ✓ 0.0s

```
0.43228842682029855
```

```
correct = 0

for i in range(0,1372):
    if df.Class[i] == model.labels_[i]:
        correct+=1

print(correct/1371)
```
[238] ✓ 0.0s

```
0.612691466083151
```

The silhouette score of 0.43 is acceptable as its similar to the value shown in the graph above (The one used to find the K values). The accuracy score by looking at the similarities as the original data (2nd algorithm), shows that the data is 0.61 in accuracy. Which means the accuracy is 61%.



```
from sklearn.metrics import silhouette_score
silhouette_score(data, model.labels_)
```

[247] ✓ 0.0s

```
0.43228842682029855
```

```
correct = 0

for i in range(0,1372):
    if df.Class[i] == model.labels_[i]:
        correct+=1

print(correct/1371)
```

[238] ✓ 0.0s

```
0.612691466083151
```

e) In conclusion, the accuracy of the model is good enough. 61% in accuracy result isn't the best result but it showed that the model could be used to predict the values reasonably well within the margin of error.