ABOUT THE CODE

In the forthcoming Annex, we are going to provide a detailed description of the code developed for the recommendation analysis.

DATA REVISION

In the first place it is necessary to upload both sets of information, understand the diverse variables that are present and what is intended to be found.

```
In [12]: from IPython.core.display import display, HTML

display(HTML("<style>.container { width:70% !important; }</style>")) # Increase cell width
display(HTML("<style>.rendered_html { font-size: 12px; }</style>")) # Increase font size

import warnings
warnings.filterwarnings('ignore')

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

import pandas as pd
import numpy as np
import sklearn
```

```
In [13]: training_set=pd.read_csv('training_set.csv')
test_set=pd.read_csv('test_set.csv')
```

It's necessary to review the number of columns and rows of both sets.

```
In [15]: #Review the columns and rows of both sets

print('Training_set =',training_set.shape)
print('Test_set =',test_set.shape)

Training_set = (1340, 21)
Test_set = (660, 20)
```

By printing the head of both sets, we can appreciate the "test set" has one less column compared to the "training set". In fact, the column missing is "price range", which is the column we need to predict in the "test set".

In addition, we can see the that the values within the data sets do not possess the same size, so it's necessary to do a series of transformations.

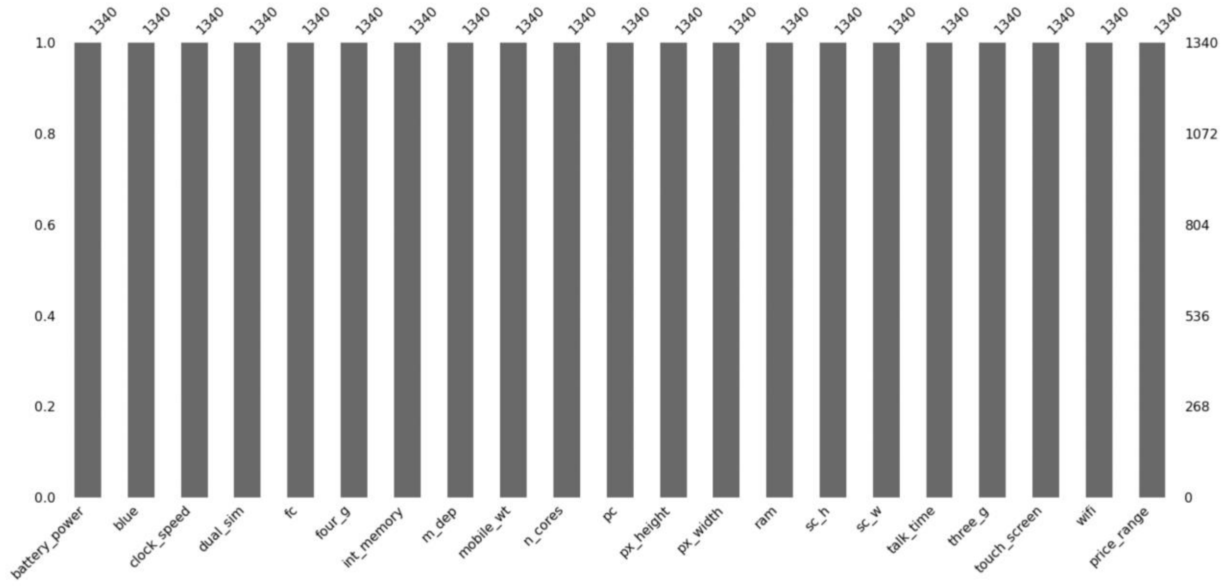
Next, it's important to review the type of data in the "training set" we are going to work and check if there are null values that can affect the preliminary analysis.

```
training_set.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1340 entries, 0 to 1339
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  1340 non-null   int64
 1   blue           1340 non-null   int64
 2   clock_speed    1340 non-null   float64
 3   dual_sim       1340 non-null   int64
 4   fc             1340 non-null   int64
 5   four_g         1340 non-null   int64
 6   int_memory     1340 non-null   int64
 7   m_dep          1340 non-null   float64
 8   mobile_wt      1340 non-null   int64
 9   n_cores        1340 non-null   int64
 10  pc             1340 non-null   int64
 11  px_height      1340 non-null   int64
 12  px_width       1340 non-null   int64
 13  ram            1340 non-null   int64
 14  sc_h           1340 non-null   int64
 15  sc_w           1340 non-null   int64
 16  talk_time      1340 non-null   int64
 17  three_g        1340 non-null   int64
 18  touch_screen   1340 non-null   int64
 19  wifi           1340 non-null   int64
 20  price_range    1340 non-null   int64
dtypes: float64(2), int64(19)
memory usage: 220.0 KB
```

```
training_set.isnull().sum()

Out[19]: battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

Finally, we can clearly visualize all the columns are complete and do not present any missing data. Furthermore, we can appreciate there is similarity between the types of data so it can be analyzed.

```
In [20]: import missingno as msno
import matplotlib.pyplot as plt
msno.bar(training_set)
plt.show()
```



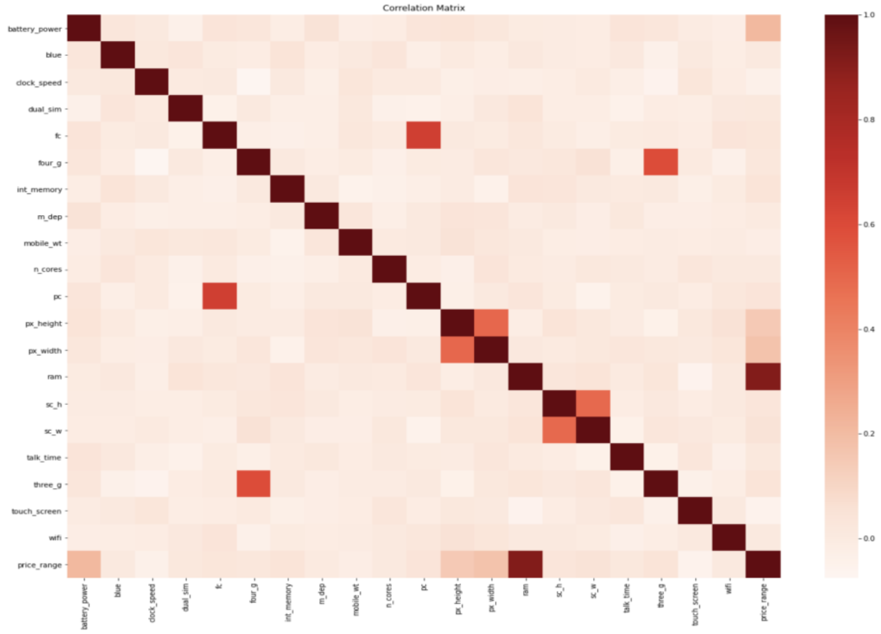
DATA ANALYSIS AND TRANSFORMATION

CORRELATION ANALYSIS

When variables are found to be related, we often want to know how close the relationship is. The primary objective is to measure the strength or degree of linear association between these variables.

In this case, we can identify not all the variables are important to predict the “price range”. Afterwards, to better understand their relationship, we will analyze them in order.

```
In [22]: import seaborn as sns
         import matplotlib.pyplot as plt

         corr_analysis=training_set.corr()
         fig = plt.figure(figsize=(21,16))
         r = sns.heatmap(corr_analysis, cmap='Reds')
         r.set_title("Correlation Matrix")

Out[22]: Text(0.5, 1.0, 'Correlation Matrix')
```

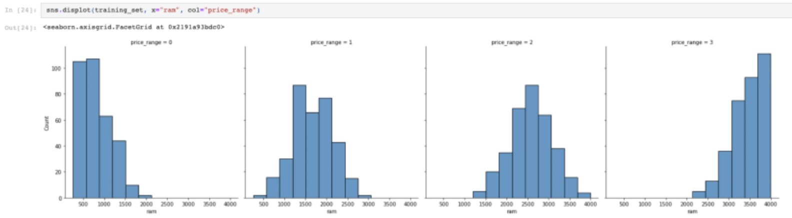



```
In [23]: corr_analysis.sort_values(by=["price_range"],ascending=False).iloc[0].sort_values(ascending=False)

Out[23]: price_range      1.000000
         ram              0.915584
         battery_power    0.213641
         px_width         0.172941
         px_height        0.151711
         sc_w             0.048215
         int_memory       0.044948
         three_g          0.043548
         pc               0.041643
         sc_h             0.034198
         fc               0.033815
         four_g           0.022698
         talk_time        0.020428
         dual_sim         0.019353
         blue             0.012093
         n_cores          0.009519
         wifi             0.005374
         m_dep            0.004608
         mobile_wt       -0.018811
         clock_speed     -0.036722
         touch_screen    -0.050248
         Name: price_range, dtype: float64
```

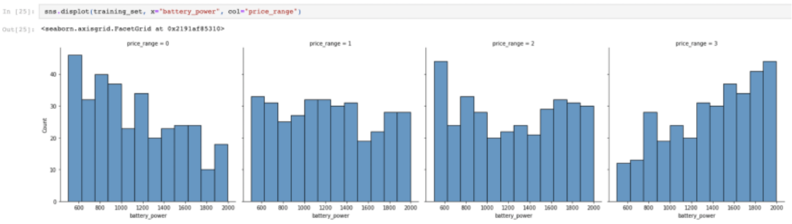
VISUALIZATION ANALYSIS

Regarding the correlation analysis, we can 'dive in' to visualize the top 5 variables associated to "price range". The graphs show there is a strong relationship between the variable "price range" and "ram". Also, "battery power" and "px_width" have impact on the target variable.

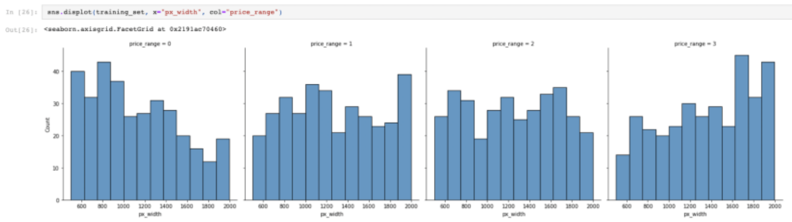
Ram



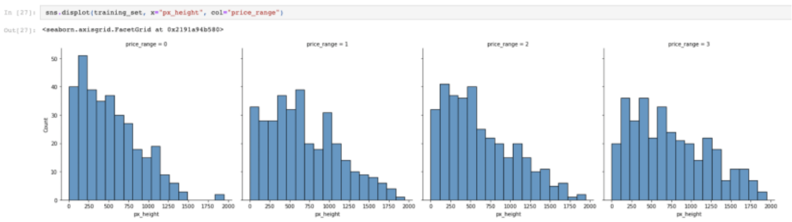
Battery power



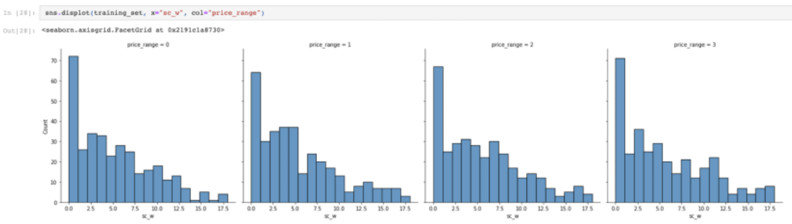
Px_width



Px_height



Sc_w



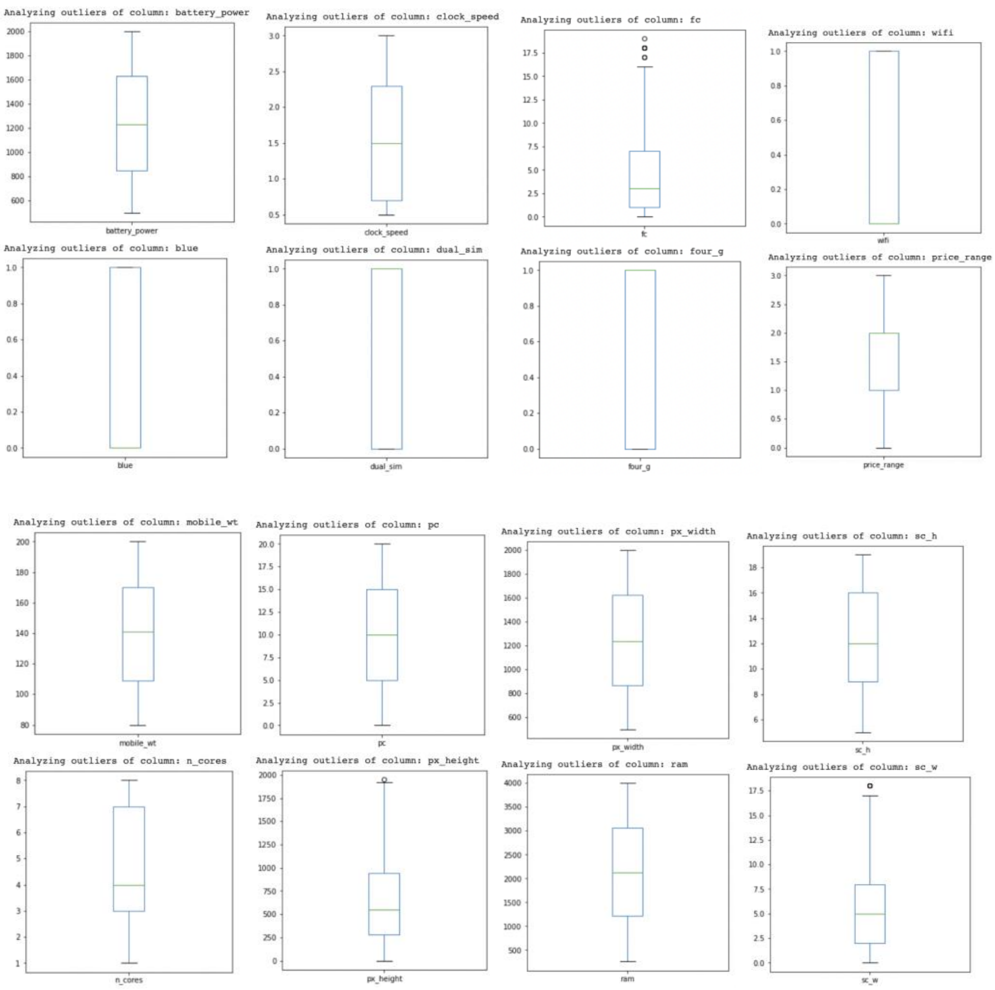
OUTLIER REMOVAL

Machine learning algorithms are sensitive to the range and distribution of attribute values. Data outliers can spoil and mislead the training process resulting in less accurate and ultimately poor results.

For this reason, we will analyze if there are outliers in our features.

```
In [30]: def remove_outlier(training_set_in, col_name):
             q1 = training_set_in[col_name].quantile(0.25)
             q3 = training_set_in[col_name].quantile(0.75)
             iqr = q3-q1 #Interquartile range
             fence_low  = q1-1.5*iqr
             fence_high = q3+1.5*iqr
             training_set_out = training_set_in.loc[(training_set_in[col_name] > fence_low) & (training_set_in[col_name] < fence_high)]
             print("{} outliers removed".format(len(training_set_in)-len(training_set_out)))
             return training_set_out
```

```
In [31]: for (columnName, _) in training_set.iteritems():
            if (training_set[columnName].dtype in ['float64','int64']):
             print("Analyzing outliers of column: {}".format(columnName))
             plt.figure(figsize=(5,5))
             training_set.boxplot([columnName], grid=False, fontsize=10)
             plt.show()
```



Subsequently, we can determine the features “fc”, “px_weight” and “sc_w” are strongly affected by the presence of outliers. For instance, we proceed to remove them from the dataset. Hence, the number of rows reduced from 1340 to 1321 in the dataset.

```
In [32]: remove_outlier(training_set,'fc')
         remove_outlier(training_set,'px_height')
         remove_outlier(training_set,'sc_w')

29 outliers removed
1 outliers removed
19 outliers removed
```

Out[32]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
0	999	0	2.9	1	11	1	64	0.2	199	4	...	1397	1616	2593	14	11	16	1	1	0	2
1	1265	0	0.6	1	4	1	49	0.5	90	1	...	275	687	1518	16	11	8	1	0	1	1
2	1000	1	0.5	0	12	0	63	0.7	179	8	...	1537	1761	3744	11	1	7	0	0	1	3
3	1874	1	2.7	1	10	0	8	0.9	166	7	...	964	1233	1246	8	1	2	0	0	1	1
4	1658	1	1.4	1	0	1	38	0.3	159	5	...	51	739	2609	12	11	19	1	0	0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1335	1975	1	1.9	1	2	0	31	0.9	151	1	...	775	1607	3022	13	5	19	0	0	1	3
1336	589	1	0.5	0	1	1	59	0.7	146	8	...	759	1858	362	16	10	6	1	1	1	0
1337	1829	1	0.5	0	0	1	15	0.4	160	5	...	729	1267	2080	16	11	12	1	0	1	2
1338	1927	0	0.9	1	3	0	11	0.4	190	8	...	491	1506	2916	16	11	18	0	1	1	3
1339	635	1	0.6	1	1	1	50	0.3	97	5	...	193	989	2107	13	12	12	1	0	0	1

1321 rows × 21 columns

STANDARDIZATION AND NORMALIZATION

As values within the worked dataset still present considerable size differences, they need to be standardized for obtaining the best comparable results.

```
In [33]: from sklearn.preprocessing import StandardScaler

         X = training_set.iloc[:,:20].values
         training_set.iloc[:,:20] = StandardScaler().fit_transform(X)


         training_set.head(10)
```

Out[33]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
0	-0.547855	-0.983715	1.663242	0.953327	1.561955	0.943391	1.727073	-1.067229	1.669421	-0.243341	...	1.716917	0.857252	0.429293	0.429921	1.234901	0.914257	0.560112	0.997019	-0.992565	2
1	0.051190	-0.983715	-1.129823	0.953327	-0.056955	0.943391	0.910231	-0.024385	-1.417145	-1.542453	...	-0.821703	-1.289582	-0.565343	0.902710	1.234901	-0.546374	0.560112	-1.002990	1.007491	1
2	-0.545603	1.016555	-1.251260	-1.048958	1.793228	-1.060006	1.672617	0.670845	1.103079	1.488808	...	2.033679	1.192334	1.494248	-0.279263	-1.059881	-0.728953	-1.785357	-1.002990	1.007491	3
3	1.422688	1.016555	1.420367	0.953327	1.330682	-1.060006	-1.322472	1.366074	0.734956	1.055770	...	0.737218	-0.027826	-0.817009	-0.988447	-1.059881	-1.641848	-1.785357	-1.002990	1.007491	1
4	0.936246	1.016555	-0.158322	0.953327	-0.982047	0.943391	0.311213	-0.719615	0.536736	0.189696	...	-1.328522	-1.169415	0.444097	-0.042869	1.234901	1.461994	0.560112	-1.002990	-0.992565	2
5	-1.282022	1.016555	0.448866	-1.048958	2.024501	0.943391	0.147844	-0.372000	0.055345	0.622733	...	1.035879	0.750950	1.306423	1.375499	2.611771	-1.276690	0.560112	-1.002990	1.007491	3
6	-0.259592	1.016555	1.663242	0.953327	0.174318	0.943391	1.345880	1.018459	0.650005	-1.542453	...	0.655765	-0.453033	-0.074964	-0.279263	-0.141968	1.096836	0.560112	0.997019	1.007491	1
7	-1.414893	1.016555	-1.008385	-1.048958	-0.056955	0.943391	1.019143	-1.414844	1.074761	1.055770	...	0.845822	-0.473831	0.670782	-0.515658	0.316988	0.183941	0.560112	-1.002990	1.007491	2
8	-0.608660	1.016555	-1.251260	0.953327	0.174318	-1.060006	-1.050191	-0.372000	1.499518	0.189696	...	1.705604	1.270905	-0.475594	-0.279263	0.087510	0.366520	-1.785357	0.997019	-0.992565	1
9	1.303330	-0.983715	0.205991	-1.048958	-0.056955	0.943391	-1.213560	1.018459	-0.029606	1.488808	...	-0.581869	-0.524671	1.009421	0.193526	0.546466	1.279415	0.560112	-1.002990	1.007491	3

10 rows × 21 columns

DIMENSIONALITY REDUCTION

Principal Component Analysis (PCA) is an unsupervised, non-parametric statistical technique primarily used for dimensionality reduction and overfitting in machine learning.

By applying PCA technique, we will be able to reduce the number of features concluding 18 features are the best fit for the models.

In [34]:
```
from sklearn.decomposition import PCA
pca_training_set = PCA()
principalComponents_training = pca_training_set.fit_transform(training_set.iloc[:,:20])
pd.DataFrame(principalComponents_training).describe()
```

Out[34]:

	0	1	2	3	4	5	6	7	8	9	10
count	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03
mean	6.081371e-17	-6.735905e-17	2.469003e-17	1.159935e-18	-4.018345e-17	1.019085e-17	9.105486e-17	4.656309e-17	6.876755e-18	-7.456722e-17	-2.038171e-17
std	1.296845e+00	1.283578e+00	1.242454e+00	1.208673e+00	1.061892e+00	1.052835e+00	1.030633e+00	1.023815e+00	1.014736e+00	9.952063e-01	9.903676e-01
min	-3.045134e+00	-3.352886e+00	-2.776483e+00	-3.231645e+00	-2.970132e+00	-3.396152e+00	-2.778896e+00	-3.201704e+00	-2.997853e+00	-2.935422e+00	-2.789918e+00
25%	-9.685690e-01	-9.042427e-01	-9.276412e-01	-8.414655e-01	-8.056734e-01	-7.290749e-01	-7.468155e-01	-7.391833e-01	-6.803767e-01	-6.737964e-01	-6.575399e-01
50%	-1.016138e-01	1.940555e-01	-1.364093e-01	-7.561266e-02	-3.098470e-02	-1.824109e-02	-6.569876e-03	-7.144255e-03	-1.453197e-02	-2.838573e-02	1.215584e-02
75%	8.942536e-01	9.418819e-01	8.650407e-01	8.215989e-01	7.887887e-01	7.112120e-01	7.632921e-01	7.275226e-01	7.096520e-01	7.002471e-01	6.633258e-01
max	4.032277e+00	3.421993e+00	4.010323e+00	3.380228e+00	3.024986e+00	3.573208e+00	3.160965e+00	3.451022e+00	3.006941e+00	3.229263e+00	3.487488e+00

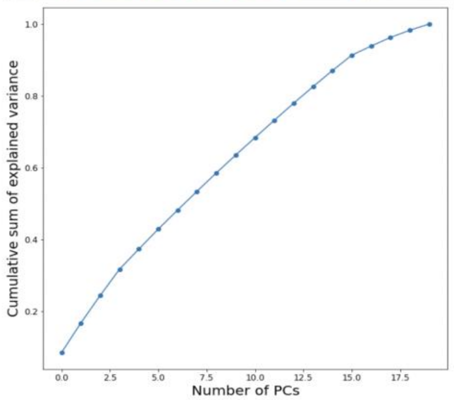
In [35]:
```
pc_set = pd.DataFrame(principalComponents_training, columns=training_set.iloc[:,:20].columns)
```

In [36]:
```
pc_set.describe()
```

Out[36]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc
count	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03	1.340000e+03
mean	6.081371e-17	-6.735905e-17	2.469003e-17	1.159935e-18	-4.018345e-17	1.019085e-17	9.105486e-17	4.656309e-17	6.876755e-18	-7.456722e-17	-2.038171e-17
std	1.296845e+00	1.283578e+00	1.242454e+00	1.208673e+00	1.061892e+00	1.052835e+00	1.030633e+00	1.023815e+00	1.014736e+00	9.952063e-01	9.903676e-01
min	-3.045134e+00	-3.352886e+00	-2.776483e+00	-3.231645e+00	-2.970132e+00	-3.396152e+00	-2.778896e+00	-3.201704e+00	-2.997853e+00	-2.935422e+00	-2.789918e+00
25%	-9.685690e-01	-9.042427e-01	-9.276412e-01	-8.414655e-01	-8.056734e-01	-7.290749e-01	-7.468155e-01	-7.391833e-01	-6.803767e-01	-6.737964e-01	-6.575399e-01
50%	-1.016138e-01	1.940555e-01	-1.364093e-01	-7.561266e-02	-3.098470e-02	-1.824109e-02	-6.569876e-03	-7.144255e-03	-1.453197e-02	-2.838573e-02	1.215584e-02
75%	8.942536e-01	9.418819e-01	8.650407e-01	8.215989e-01	7.887887e-01	7.112120e-01	7.632921e-01	7.275226e-01	7.096520e-01	7.002471e-01	6.633258e-01
max	4.032277e+00	3.421993e+00	4.010323e+00	3.380228e+00	3.024986e+00	3.573208e+00	3.160965e+00	3.451022e+00	3.006941e+00	3.229263e+00	3.487488e+00

In [37]:
```
plt.figure(figsize=(10,10))
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.plot(np.cumsum(pca_training_set.explained_variance_ratio_), '-o')
plt.xlabel('Number of PCs', fontsize=20)
plt.ylabel('Cumulative sum of explained variance' , fontsize=20);
```



In [38]:
```
training_pca = pd.concat([pc_set.iloc[:, 0:18], training_set.price_range], axis=1)
training_pca.columns = ["PC1","PC2","PC3","PC4","PC5","PC6","PC7","PC8","PC9","PC10","PC11","PC12","PC13","PC14","PC15","PC16","PC17","PC18",'price_range']
training_pca.head(5)
```

Out[38]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14	PC15	PC16	PC17	PC18	price_range
0	0.906253	2.079915	1.641113	0.460412	0.644617	0.422867	1.076047	0.789148	1.578362	-0.121912	-2.756401	-0.919886	-0.894586	-0.500315	0.523049	1.623318	-0.176391	-0.925471	2
1	-1.547714	1.077670	-1.238617	1.408699	0.802131	-0.852440	-1.941628	-1.237083	0.621782	-1.003907	0.934401	0.144286	0.177967	0.589480	0.463309	0.611324	-0.204316	-0.449514	1
2	2.794871	-0.381064	2.536904	-0.437750	1.215471	-1.672321	0.789476	-1.068554	-1.069741	2.514471	0.230687	-0.956332	0.647786	-0.118108	-1.048283	0.363871	0.588912	-0.343532	3
3	2.536930	-1.534650	0.920938	-0.748020	0.935107	0.384741	0.608301	1.696147	0.356471	0.700086	2.801067	1.112819	0.381106	0.018925	0.636325	0.006478	0.288922	-0.597643	1
4	-2.207914	0.048159	-1.741837	0.669120	-0.325676	-1.190075	1.488317	0.968220	-0.770308	-0.696772	-0.341327	0.372597	-0.388007	1.232627	0.317745	0.634711	-0.784835	-0.554337	2

DATA MODELLING

Once the data is transformed, training models can be applied to obtain the best predictor.

SUPPORT VECTOR MACHINES (SVM)

SVM will try to find the decision boundary that separates the classes by creating the largest possible margin between the decision boundary and the classes. It can be used both for classification and regression, however, its strongly recommended for classification analysis.

For instance, we used it to predict the "price range", sustained with an **accuracy of 0.940.**

```
In [40]: from sklearn import svm
         from sklearn.svm import SVC

In [41]: clf = svm.SVC(kernel='linear') # Linear Kernel
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)

In [42]: from sklearn import metrics
         print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

         Accuracy: 0.9402985074626866
```

ADDING GRIDSEARCH-CV

This is a function that helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. Consequently, it enables us to select the best hyperparameters among the listed hyperparameters.

Hence, we can see how the **accuracy improved to 0.9841**, best accuracy so far.

```
In [43]: from sklearn.model_selection import GridSearchCV

         C=[1,0.1,0.25,0.5,2,0.75]
         kernel=["linear","rbf"]
         gamma=["auto",0.01,0.001,0.0001,1]
         decision_function_shape=["ovo","ovr"]

In [44]: svm_model=SVC(C=2,decision_function_shape="ovo",gamma="auto",kernel="linear",random_state=1)

In [45]: svm_model.fit(X_train,y_train)

Out[45]: SVC(C=2, decision_function_shape='ovo', gamma='auto', kernel='linear',
             random_state=1)

In [46]: print("train_accuracy:",svm_model.score(X_train,y_train))
         print("test_accuracy: ", svm_model.score(X_test,y_test))

         train_accuracy: 0.9841417910447762
         test_accuracy:  0.9514925373134329
```

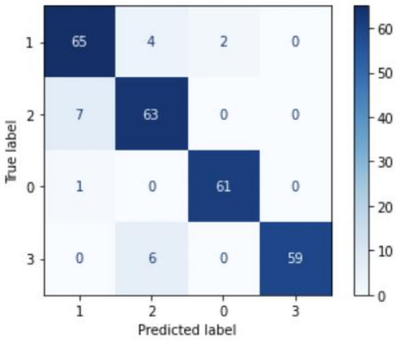
LINEAR DISCRIMINANT ANALYSIS (LDA)

LDA is most used as a dimensionality reduction technique. The goal is to project a dataset onto a lower-dimensional space with good class-separability to avoid overfitting.

It's very similar to the PCA technique (previously mentioned) but in addition to finding the component axes that maximize the variance of our data, we are **additionally** interested in the axes that maximize the separation between classes.

The model provided an **accuracy of 0.9254**, good performance but still worse than SVM.

```
In [47]: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
         from sklearn.metrics import plot_confusion_matrix, accuracy_score

         lda_clf = LinearDiscriminantAnalysis().fit(X_train,y_train)
         print("Accuracy of the PCA-based LDA classifier = {0:.4f}".format(accuracy_score(lda_clf.predict(X_test), y_test)))
         plot_confusion_matrix(lda_clf, X_test, y_test, y_test.unique(), cmap=plt.cm.Blues);

Accuracy of the PCA-based LDA classifier = 0.9254
```



RANDOM FOREST

Random forest is an implementation of decision trees in which we use *bagging*. It consists of training several predictors (trees) using a subsample of the total training data in each. Finally, once we have the predictions, we average them (regression) or use the majority vote (classification).

The model provided an **accuracy of 0.6305**, being the least accurate so far.

```
In [48]: from sklearn.ensemble import RandomForestClassifier
         clf=RandomForestClassifier(n_estimators=100)
         clf.fit(X_train,y_train)
         y_pred=clf.predict(X_test)

In [49]: from sklearn import metrics
         print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

Accuracy: 0.6305970149253731
```

In this case it was the least efficient of the models, having the lowest rate of result.

PREDICTIONS

Once we tested all the models, we can conclude the **best model is SVM** (improved with GridSearchCV function) with an **accuracy of 0.9841**.

Thus, the real price range is compared with the predicted price range from the test training set. As we can see, “y_true” and “y_pred” are very similar, validating the high accuracy of the model.

```
In [50]: y_pred=svm_model.predict(X_test)

In [51]: svm_test=X_test

In [52]: svm_test["y_true"]=y_test
         svm_test["y_pred"]=y_pred

In [53]: svm_test.head(25)

Out[53]:
```

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14	PC15	PC16	PC17	PC18	y_true	y_pred
773	-2.122998	0.608616	0.445852	1.002699	-0.725973	0.493993	1.680977	1.552063	0.431395	-1.251120	0.011261	2.136802	-0.223502	1.695538	0.997658	-0.020607	0.048337	-0.953925	1	1
1088	-0.558915	-1.360482	0.664680	-1.441653	1.573774	0.429547	-1.256944	0.422010	0.274123	-0.322089	-0.473348	0.293370	-1.129444	-0.038799	2.140324	-1.591074	-0.073025	0.361242	1	1
301	-1.166878	-2.483369	2.278715	0.173413	1.079847	0.501064	0.637255	-1.032279	1.155540	-1.630845	-0.954117	-0.964584	-0.393427	1.130605	-0.083870	-0.719900	-0.178571	-0.933926	1	1
383	0.795052	2.346857	-0.474006	1.585434	-0.347518	0.340918	0.858349	-1.005014	0.810515	-0.404238	0.570700	1.624505	0.400873	-2.226609	-0.419188	2.197298	-0.144983	-1.013565	2	2
318	1.727334	-1.762811	0.294581	-0.753569	0.542761	1.356071	-0.778589	-2.113056	-1.906206	-2.096129	-0.706393	-0.275405	-0.293296	-0.322999	0.314975	-0.568486	-0.254170	-0.907039	2	2
223	1.103324	0.117304	-0.947681	1.609554	-1.537290	0.830753	-0.727717	-1.680439	2.415664	-0.012860	0.074086	0.260542	-1.000752	0.481186	0.420129	0.735078	0.075545	-0.212408	0	0
358	-0.173114	-2.107822	1.746180	-0.618173	0.813298	0.115707	0.379837	-1.041362	0.033482	-1.757176	-0.032050	-0.998620	0.099498	1.061944	-1.028101	0.010247	-0.764309	-0.555055	2	2
361	-0.084869	-0.679035	-1.358609	0.038971	2.444003	0.828744	-0.228532	-0.726045	-1.899960	1.203996	-0.354410	-0.166111	-0.263451	0.487265	-0.291366	1.369146	-0.433702	-0.737145	1	1
820	-0.867713	-1.170935	1.356377	-3.159549	-1.406694	-0.728363	-1.570929	-0.512298	-0.471605	-0.416605	-0.445967	1.964011	0.558614	0.597730	-0.476336	-0.192824	-0.178468	-0.151880	1	1
591	-2.127996	0.270860	-0.258458	0.409793	-1.043947	0.904461	-1.925729	-0.590160	2.798839	0.051178	1.655338	0.434790	-0.347667	-0.367836	-0.335668	-0.417100	-0.051738	-0.645520	0	0
883	-1.678495	0.685039	-2.117166	0.680430	1.062364	-0.870587	-1.205807	-1.067997	0.370783	0.907275	-1.405063	0.907031	0.489028	-0.090225	-1.803718	0.035452	-0.216876	-0.106978	2	2
1101	0.545075	-0.713625	-1.797978	-0.289959	-0.346688	-1.837980	1.115559	-0.654958	-0.898459	-1.402117	-0.772193	-1.476861	0.056549	-0.925861	-0.332578	0.215374	-0.274860	-0.141980	3	3
87	-1.968142	0.774852	0.379413	0.184177	0.825746	0.789283	-1.864696	-1.538048	0.033016	0.396994	-0.783331	0.315746	0.259214	1.507329	-0.200561	-0.803698	0.599581	0.776712	1	1
7	-0.725114	0.720579	-0.129072	-0.313736	1.012628	-0.070053	1.051917	-1.309829	-1.009155	1.950688	-0.741904	-0.912873	0.077615	1.625359	-0.656118	-0.096032	-0.012923	-1.168680	2	2
730	-3.045134	1.027230	2.187419	0.210684	-0.153869	-0.734380	-1.633718	1.351433	-0.358748	-0.987526	-0.415787	-0.063516	0.925304	0.638801	0.668127	0.792831	-0.896318	-0.501571	1	1
528	-0.490278	-3.214857	-1.835070	0.219817	0.754932	0.750022	0.746751	-0.027441	-0.407059	-0.314898	-0.100371	-0.673832	1.540649	-0.709237	-0.723314	0.133542	0.811537	-0.217085	0	0
217	1.901501	-2.087065	-0.754043	-0.809745	-0.536658	-0.279264	2.225009	-0.004778	-0.631284	-1.660222	-0.593872	0.723059	-0.592032	-0.180519	-0.758987	0.577311	-0.242413	0.045702	3	3
562	0.365220	-1.992448	-0.415861	1.492226	-0.069082	0.036984	1.657904	0.465620	0.038907	-0.841723	-0.282135	-0.046887	0.210754	-0.356640	-0.438985	0.593650	1.907290	0.400888	2	2
495	-1.771009	0.056549	-0.638491	-0.386804	1.876704	0.228574	-0.650377	0.447751	-1.374848	-0.460462	-1.257086	-0.624304	-1.773627	1.447232	0.405363	-0.388926	1.055679	-0.634024	2	2
846	1.201823	0.726839	1.327703	-0.228486	0.733548	-2.722519	-1.074001	-1.002396	-0.235334	-0.457281	0.465568	-0.261222	-0.952526	-0.650460	0.681293	0.788042	1.139713	-0.076208	3	3
832	0.057732	0.142066	1.964958	-0.768066	-0.320254	-1.210746	0.946719	-0.051089	-1.777362	1.093838	0.738427	0.154112	-0.785575	1.333168	-0.904992	0.026333	0.093553	1.151971	3	3
1104	0.801615	1.567362	0.085515	-0.308331	-1.772008	0.937914	0.413099	1.134331	-0.076629	0.682828	1.303852	-0.544394	2.032634	-0.120003	-0.914991	0.431040	0.773103	0.647564	0	0
533	0.542829	1.351423	-1.362772	0.094700	-1.027914	0.461666	0.514474	-0.370004	-0.814957	-0.178370	-1.276555	0.885420	-1.165476	-0.820742	-1.316386	0.122377	0.388848	0.800750	3	3
874	0.198249	-1.208026	3.492514	0.584908	-1.828539	-0.494388	-0.119714	-0.758144	-0.012137	-1.222490	0.928410	-0.424433	-0.405907	-0.012432	0.127695	-2.037662	-0.313538	-1.115271	3	2
507	-0.170572	-2.171541	2.450323	-0.624168	-1.044546	1.080609	-1.047445	-0.922990	1.075440	0.463221	0.625287	0.428730	-0.452016	-0.356745	-0.158866	0.068494	0.278778	-0.660212	1	1

Once we selected a model and validated its accuracy, we will proceed to do the prediction with the test set. But first, we will need to do several adaptations and record the results.

```
In [55]: from sklearn.decomposition import PCA

         pca_test_set = PCA()

         principalComponents_test = pca_test_set.fit_transform(test_set.iloc[:,:20])

         pd.DataFrame(principalComponents_test).describe()

Out[55]:
```

	0	1	2	3	4	5	6	7	8	9	10
count	6.600000e+02	6.600000e+02	6.600000e+02	6.600000e+02	6.600000e+02	6.600000e+02	6.600000e+02	6.600000e+02	6.600000e+02	6.600000e+02	6.600000e+02
mean	-3.998418e-14	-7.165716e-14	-1.037392e-13	-3.208208e-14	4.607762e-15	-8.343494e-17	8.774126e-16	2.952815e-16	-6.661338e-17	-3.818494e-17	1.258253e-16
std	1.092771e+03	5.454232e+02	4.271585e+02	2.999140e+02	3.537086e+01	1.764415e+01	6.915432e+00	5.414478e+00	5.211867e+00	3.011891e+00	2.874407e+00
min	-1.863590e+03	-9.820918e+02	-8.102046e+02	-4.479630e+02	-6.600044e+01	-3.099613e+01	-1.218283e+01	-1.003448e+01	-9.457977e+00	-7.803889e+00	-7.684056e+00
25%	-9.446152e+02	-4.484154e+02	-3.702030e+02	-2.354495e+02	-3.032896e+01	-1.519135e+01	-5.964083e+00	-4.580904e+00	-4.380382e+00	-2.131187e+00	-2.002156e+00
50%	5.135264e+01	-5.682489e+01	-1.208844e+01	-4.712391e+01	1.371655e+00	-1.114909e+00	1.812214e-01	2.837725e-01	-2.335453e-01	-3.308698e-01	-3.282976e-01
75%	9.539060e+02	3.931910e+02	3.653546e+02	1.996855e+02	3.042924e+01	1.440806e+01	5.606328e+00	4.596321e+00	3.956577e+00	1.691350e+00	1.711348e+00
max	1.891555e+03	1.411000e+03	9.542153e+02	9.919747e+02	6.716432e+01	3.371877e+01	1.639453e+01	1.009572e+01	1.376282e+01	1.132052e+01	9.457782e+00

We will drop features that have a low degree of relationship with the target variable and are considered not significant. Therefore, the features "wifi" and "touch_screen" were dropped.

```
In [56]: pc_test_set = pd.DataFrame(principalComponents_test, columns=test_set.iloc[:,:20].columns)

In [57]: pc_test_set.drop(['wifi', 'touch_screen'],axis='columns', inplace=True)

In [58]: pc_test_set
```

Out[58]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g
0	-1424.198088	-176.083055	370.504398	579.328873	53.093795	-6.846215	-5.808881	0.200373	-1.832538	-2.588587	-1.775419	-1.894428	1.022216	-0.592308	0.728806	0.077399	0.170073	0.400642
1	457.292599	-450.989061	-116.462488	52.791362	-4.722579	-22.833012	6.644669	-7.427386	12.979405	-0.963953	-2.397750	3.014689	-0.889627	-0.364226	0.689520	-0.112725	-0.894577	0.328325
2	-790.430951	-596.715308	660.059444	52.235178	49.479469	-17.190235	7.780091	4.405176	-5.478666	0.837087	-0.367304	3.139138	1.455361	0.153109	0.723825	0.350222	0.336750	0.343741
3	1783.079472	-137.514329	-275.977895	350.777886	25.207954	-14.932395	7.362611	-7.462265	4.813272	5.707155	1.263716	-1.688182	0.540163	0.042043	0.652719	-0.510613	0.637673	-0.135123
4	-219.654233	815.313041	-513.937300	239.669228	-6.389967	25.278049	0.200084	5.217294	-1.302451	-1.674858	2.555882	0.958129	-0.929160	0.378452	-0.347770	-0.855896	-0.159884	0.244634
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
655	-1617.559803	-503.902956	0.503254	70.541626	53.527534	15.778342	3.195427	2.361493	-6.783370	-3.216958	-1.800056	-3.048048	-0.196986	0.838138	0.024941	0.104123	0.540069	0.877236
656	252.837870	689.927968	-626.839096	-200.099387	28.867169	22.383306	-0.860272	8.293823	5.345302	1.503279	-1.192643	-2.926588	1.381539	0.503324	-0.352431	-0.778484	-0.202697	0.307588
657	1844.583817	51.717465	258.895042	-277.236680	-15.824017	-13.779872	-11.385066	6.072517	5.461704	-2.067517	-0.682492	0.509878	-1.100532	0.122010	0.267229	-0.040173	0.621781	-0.579184
658	1849.576130	81.815882	542.520389	-147.858578	7.078790	-18.450419	-2.756018	9.030846	6.923135	-3.338744	0.525534	-1.053976	0.105808	-0.252041	-0.571461	0.412937	-0.686872	0.316217
659	21.241293	-399.130948	129.023676	35.118618	22.587447	13.851334	10.697978	-0.074184	2.682958	-0.922173	-1.156599	-0.373061	-0.950587	-0.519617	-0.028172	-0.231159	-0.432496	-0.819744

660 rows × 18 columns

Finally, we add the "y_pred" column to visualize the predictions in the dataset for each of the features. To conclude the analysis, we consider the analysis is finished and print the final prediction.

```
In [59]: y_pred=svm_model.predict(pc_test_set)

In [60]: svm_test=pc_test_set

In [61]: svm_test["y_pred"]=y_pred

In [62]: svm_test
```

Out[62]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	y_pred
0	-1424.198088	-176.083055	370.504398	579.328873	53.093795	-6.846215	-5.808881	0.200373	-1.832538	-2.588587	-1.775419	-1.894428	1.022216	-0.592308	0.728806	0.077399	0.170073	0.400642	3
1	457.292599	-450.989061	-116.462488	52.791362	-4.722579	-22.833012	6.644669	-7.427386	12.979405	-0.963953	-2.397750	3.014689	-0.889627	-0.364226	0.689520	-0.112725	-0.894577	0.328325	0
2	-790.430951	-596.715308	660.059444	52.235178	49.479469	-17.190235	7.780091	4.405176	-5.478666	0.837087	-0.367304	3.139138	1.455361	0.153109	0.723825	0.350222	0.336750	0.343741	3
3	1783.079472	-137.514329	-275.977895	350.777886	25.207954	-14.932395	7.362611	-7.462265	4.813272	5.707155	1.263716	-1.688182	0.540163	0.042043	0.652719	-0.510613	0.637673	-0.135123	0
4	-219.654233	815.313041	-513.937300	239.669228	-6.389967	25.278049	0.200084	5.217294	-1.302451	-1.674858	2.555882	0.958129	-0.929160	0.378452	-0.347770	-0.855896	-0.159884	0.244634	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
655	-1617.559803	-503.902956	0.503254	70.541626	53.527534	15.778342	3.195427	2.361493	-6.783370	-3.216958	-1.800056	-3.048048	-0.196986	0.838138	0.024941	0.104123	0.540069	0.877236	0
656	252.837870	689.927968	-626.839096	-200.099387	28.867169	22.383306	-0.860272	8.293823	5.345302	1.503279	-1.192643	-2.926588	1.381539	0.503324	-0.352431	-0.778484	-0.202697	0.307588	0
657	1844.583817	51.717465	258.895042	-277.236680	-15.824017	-13.779872	-11.385066	6.072517	5.461704	-2.067517	-0.682492	0.509878	-1.100532	0.122010	0.267229	-0.040173	0.621781	-0.579184	3
658	1849.576130	81.815882	542.520389	-147.858578	7.078790	-18.450419	-2.756018	9.030846	6.923135	-3.338744	0.525534	-1.053976	0.105808	-0.252041	-0.571461	0.412937	-0.686872	0.316217	3
659	21.241293	-399.130948	129.023676	35.118618	22.587447	13.851334	10.697978	-0.074184	2.682958	-0.922173	-1.156599	-0.373061	-0.950587	-0.519617	-0.028172	-0.231159	-0.432496	-0.819744	0

660 rows × 19 columns

```
In [63]: print(y_pred)

[3 0 3 0 0 0 0 0 0 0 0 0 3 3 3 3 3 0 0 0 3 3 3 0 0 3 2 1 3 0 0 2 0 0 0 0
 0 3 3 0 0 3 0 0 3 0 0 3 3 3 0 0 0 0 0 0 3 0 0 3 3 0 0 0 3 3 3 0 3 0 0 3 3
 0 3 3 2 3 0 0 0 0 3 3 0 0 3 0 0 0 3 0 3 3 3 3 3 0 3 0 0 3 0 3 3 0 0 3 0 0
 0 3 3 3 3 3 0 0 0 0 0 3 0 0 3 3 0 3 3 0 0 3 3 3 0 0 0 0 0 0 0 3 0 0 0 0 3
 0 3 3 0 3 0 0 0 0 3 3 0 0 0 3 3 3 3 3 0 3 2 3 0 3 0 3 0 0 0 3 0 0 0 0 0 0
 3 3 0 0 3 0 3 3 2 0 3 0 3 0 0 3 0 3 3 0 3 3 0 3 0 0 3 0 3 3 0 0 0 0 3 3 3
 3 3 3 0 3 3 3 3 3 0 3 0 0 3 0 2 3 0 0 3 0 3 0 3 0 3 3 0 0 3 0 3 0 3 3 0 0
 3 3 3 0 0 3 0 3 3 0 3 0 3 0 0 0 0 0 3 3 3 3 0 3 3 3 0 0 0 0 3 0 3 2 2 0 3
 3 0 0 3 3 3 3 3 0 0 3 2 0 3 3 3 2 0 3 0 0 0 0 0 3 0 3 3 3 0 1 3 0 0 0 0 3
 3 3 3 3 0 0 0 3 3 3 0 3 0 0 0 0 3 2 3 0 3 3 0 0 0 0 3 3 3 0 0 2 3 0 3 0 3
 2 3 3 0 0 0 3 3 0 0 0 0 3 3 3 3 3 3 0 0 0 3 0 0 2 0 3 3 0 3 3 0 0 0 3 3 0
 0 0 3 3 0 0 3 3 0 3 3 0 3 0 3 3 3 0 3 3 0 0 3 3 3 3 2 3 0 0 3 0 0 0 3 0 0
 0 0 0 0 0 3 0 0 2 3 3 0 3 0 0 3 3 0 3 3 0 0 0 3 3 0 3 0 0 3 0 0 2 0 0 3 3
 0 0 0 3 0 0 0 0 0 0 0 0 0 3 0 3 2 3 0 0 0 2 3 3 0 3 3 3 0 0 3 3 3 0 3 0 2
 3 3 0 0 3 2 3 3 3 0 3 0 0 0 0 3 0 3 0 0 0 3 3 3 0 3 3 3 3 3 0 0 3 3 3 3 3
 0 0 2 3 3 0 3 0 0 0 3 0 0 0 0 0 3 0 3 3 0 3 3 0 0 0 0 0 3 1 0 3 3 3 0 0 0
 0 3 0 3 3 0 0 0 0 0 1 3 0 3 3 0 3 3 3 3 0 3 3 3 0 0 3 0 3 0 3 3 0 0 0 3 3
 3 0 0 3 0 3 3 3 3 0 0 3 0 3 0 0 0 0 0 0 3 0 0 0 3 0 0 0 3 3 0]
```