



**ÇANKAYA UNIVERSITY**  
**FACULTY OF ENGINEERING**  
**COMPUTER ENGINEERING DEPARTMENT**

**CENG 474**

Introduction to Data Science

**Machine Learning with Airbnb Istanbul Data**

*201514039 Merve KARAKAYA*

*201411063 Doğa ÜÇÜNCÜ*

# Table of Contents

1.	Overview	3
1.1	Group Members	3
1.2	Project Data Set	3
1.3	Literature	4
2.	Methods	5
2.1	Logistic Regression Model	8
2.2	k Nearest Neighbor (kNN) Model	12
2.3	Decision Tree Model	16
3.	Results	17
3.1	Logistic Regression Model	17
3.2	k Nearest Neighbor (kNN) Model	18
3.3	Decision Tree Model	19
4.	Conclusion	20
5.	References	21

# Machine Learning with Airbnb Istanbul Data

## 1. Overview

This is a project for implementing machine learning processes. Airbnb Istanbul data was chosen for the machine learning application. Classification models were applied among the machine learning models. Accordingly, the first stage, exploratory data analysis and visualization, was completed. Thanks to this stage, the preliminary stages for model development were completed in order. In the next stage, the model development was made and its applications were specified in the method section. Here, the classification models were used and compared with each other. In addition, the feature *low\_price* is selected for the models to predict. The consequences of the models were interpreted in the results section.

### 1.1 Group members

- Merve Karakaya:

Data Exploration  
Data Cleaning  
Model Building  
Present Results

- Doğa Üçüncü:

Data Exploration  
Data Cleaning  
Model Building  
Present Results

### 1.2 Project Data Set

The data required for the project is from kaggle.com. This data is Airbnb Istanbul.[2]

### 1.3 Literature

Airbnb is a platform for organizing or offering accommodation. It is American vacation rental online marketplace company based in San Francisco, California, United States. Airbnb offers an opportunity to stay for those who prefer home to a hotel. Due to Airbnb's unique nature, hosting pricing strategies are very different from the traditional hospitality industry. For example, price determination criteria in hotels may not have a sufficient effect on Airbnb prices. Thus, there are quite different criteria in determining the prices in Airbnb compared to traditional accommodation types.

With this project, we wanted to examine the Airbnb pricing determinants. Therefore, we chose Airbnb Istanbul data which is has most lists in Turkey. Before working on this data, we did a literature review. The study we examined as a result of this research is briefly described below.[1]

The dataset is taken from kaggle.com. This dataset contains 16251 rows and 16 columns. Some rows contain missing values. Therefore, these missing values must be corrected or cleared. For example, there is no data available in the *neighbourhood\_group* column. This column needs to be removed. Additionally, 8484 missing values appear in the *last\_review* and *reviews\_per\_month* columns. Thus, it will be sufficient to fill in these columns with a value of zero.

The libraries used in this study are as follows: Pandas library was used for data analysis. Seaborn and Matplotlib libraries were used for visualization. Folium library was used to show the visualization on the map. Shortly, in this study, exploratory data analysis and visualization were performed. Firstly, analyzes were made about the data set. Later, the missing values in the data set were corrected. Visualizations were used while doing these operations. According to these, the results of the study are as follows:[1]

- The neighborhoods of the most frequently found lists are from Beyoglu, Sisli, Fatih, Kadiköy and Besiktas.
- The most listed room type is private room with number of 8565. The Entire home/apt and shared room follow with 7191 and 495 numbers.
- The host with id "21907588" has the most listings with 77 number of listings.

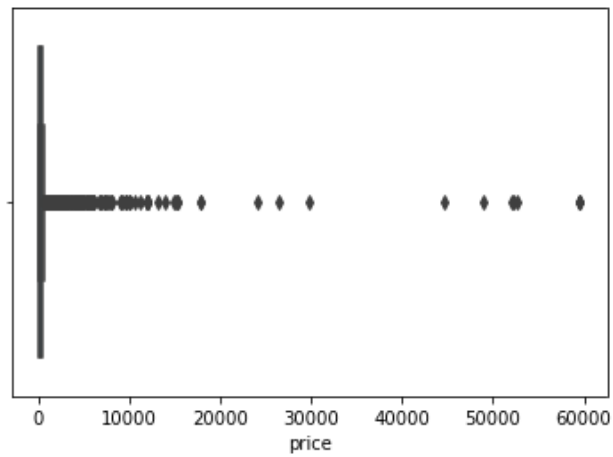
- Average price is highest in Küçükçekmece neighborhood. The average daily price for Küçükçekmece is 1263 TL.
- Pendik has the lowest average price with 153 TL per day.
- The most expensive advertisement is the "3 Rooms 1 Salon - Grand Holiday Istanbul" special room in Küçükçekmece with 59561 TL a day.
- According to the map, 90% of the 10 most expensive ads are located on the European side of Istanbul.
- The most reviewed list is the special room "Atatürk Airport 5 minutes" prepared by Melik Fırat.
- The daily average list price is 207.8 TL.

## 2. Methods

In the method section, model development was made after the preparations in the previous sections. Three different classification models are used in this project. These models are Logistic Regression, K Nearest Neighbor and Decision tree. Three models were trained with the same train data, then the models were estimated with the test data. Finally, the accuracy values were calculated. Accordingly, it was observed which model gave a better result.

However, there is another process that needs to be done before the model is built. As stated before, the feature *price* to be predict is selected. Since the classification models will be applied and also the values in *price* are not in a group, this feature should be grouped first. For this reason, *price* has been grouped as *low\_price* and *high\_price* then added to the *airbnb* data frame. For *low\_price*, values less than 2000 are preferred because this is the range with the highest congestion according to the distribution graph below (Figure 1). This means ensuring that the number of samples is large enough for the models to give better results. The things to do for these operations are listed below. According to the results, while 15929 samples are grouped as *low\_price*, 226 samples are in the *high\_price* group.

```
#Visualizing the price distribution  
sns.boxplot(x = airbnb["price"]);
```



*Figure 1: Price distribution graph*

```
airbnb['low_price']=airbnb['price']-airbnb['price']  
airbnb['high_price']=airbnb['price']-airbnb['price']
```

*Figure 2: Creating new columns*

```
airbnb.columns
```

```
Index(['id', 'name', 'host_id', 'host_name', 'latitude', 'longitude', 'price',  
      'minimum_nights', 'number_of_reviews', 'reviews_per_month',  
      'calculated_host_listings_count', 'availability_365', 'Adalar',  
      'Arnavutkoy', 'Atasehir', 'Avcilar', 'Bagcilar', 'Bahcelievler',  
      'Bakirkoy', 'Basaksehir', 'Bayrampasa', 'Besiktas', 'Beykoz',  
      'Beylikduzu', 'Beyoglu', 'Buyukcekmece', 'Catalca', 'Cekmekoy',  
      'Esenler', 'Esenyurt', 'Eyup', 'Fatih', 'Gaziosmanpasa', 'Gungoren',  
      'Kadikoy', 'Kagithane', 'Kartal', 'Kucukcekmece', 'Maltepe', 'Pendik',  
      'Sancaktepe', 'Sariyer', 'Sile', 'Silivri', 'Sisli', 'Sultanbeyli',  
      'Sultangazi', 'Tuzla', 'Umraniye', 'Uskudar', 'Zeytinburnu',  
      'Entire home/apt', 'Private room', 'Shared room', 'low_price',  
      'high_price'],  
      dtype='object')
```

Figure 3: Checking new columns

```
# Imputation function for low_price
```

```
def impute_low_price(cols):
```

```
    Price = cols[0]
```

```
    Low_Price = cols[1]
```

```
    if (Price<2000):
```

```
        return 1
```

```
    else:
```

```
        return Low_Price
```

```
# Apply the function to the low_price column
```

```
airbnb['low_price']=airbnb[['price','low_price']].apply(impute_low_price, axis =1 )
```

Figure 4: Assign appropriate values into low\_price

```
airbnb['low_price'].value_counts()
```

```
1    15929
```

```
0      226
```

```
Name: low_price, dtype: int64
```

Figure 5: The low\_price group values

```
# Imputation function for high_price
def impute_high_price(cols):
    Price = cols[0]
    High_Price = cols[1]

    if (Price>2000):
        return 1
    else:
        return High_Price

# Apply the function to the high_price column
airbnb['high_price']=airbnb[['price','high_price']].apply(impute_high_price, axis =1 )
```

Figure 6: Assign appropriate values into high\_price

```
airbnb['high_price'].value_counts()
0    15929
1      226
Name: high_price, dtype: int64
```

Figure 7: The high\_price group values

## 2.1 Logistic Regression Model

Logistics regression is common and is a useful regression method for solving the binary classification problem. Since the values in *low\_price* are also in this form, accuracy value gave a good result. [3].

Before developing the model, the data must be splitted randomly into train and test. The variable *x* shows the features in the data, and the *y* variable indicates the feature desired to predict. The values for *x* are selected as those other than the features *low\_price*, *host\_name*, *name*, *id*, *reviews\_per\_month*, because when the correlation graph below is examined, there is a similarity between *host\_id* and *id*, as well as *number\_of\_reviews* and *reviews\_per\_month*. Therefore, it will be useful to choose one of these features. While *low\_price* is dropped because it is the property to



predict, *host\_name* and *name* are not an important variable for price. In addition, for distinction, *train\_test\_split* () function is used. While 30 percent of the data is reserved for testing, 70 percent is reserved for the training.



Figure 8: Correlation graph

```
#Separate data into x and y variables
x = airbnb.drop(['low_price', 'host_name', 'name', 'id', 'reviews_per_month'],axis = 1)
#x is everything else(except all dropped features)
y = airbnb['low_price']
#y is the column to predict

#Use x and y variables to split the data into training and test sets
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
#30% test 70% train
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state= 101)
#print (rows,columns) of x
print(x_train.shape)
print(x_test.shape)

(11308, 51)
(4847, 51)
```

Figure 9: Splitting data as train and test

After preparing the data in this way, the model is built. The *max\_iter* number here is chosen as 500. It has been tested in high values, but the same results were observed.

```
#Import LogisticRegression model
from sklearn.linear_model import LogisticRegression

#Create instance of model
logmodel = LogisticRegression(solver='lbfgs',max_iter=500)
```

Figure 10: Building logistic regression model

Then the model is trained using the *fit()* function:

```
#Training Logmodel with train datas
logmodel.fit(x_train,y_train)
```

Figure 11: Training model on train data

The *predict()* function was used to see the predictive power of the trained model. Accordingly, 4780 of the 4847 *low\_price* samples were estimated correctly. This value is quite good. Therefore, the accuracy value was found to be a high value of 98 percent. In short, the logistic regression model has shown a good result. Applications required for these results are shown below:

```
#Make predictions on test data
predictions = logmodel.predict(x_test)
print(predictions)
print(y_test)
```

```
[1 1 1 ... 1 1 1]
10889    0
6894     1
11540    1
14932    1
8505     1
..
5243     1
3145     1
5214     1
7994     1
10309    1
Name: low_price, Length: 4847, dtype: int64
```

Figure 12: Predicting on test data

```
#Show predictions with a table
pf = pd.DataFrame({'Actual' : y_test, 'Predicted':predictions})
pf.head(10)
```

	Actual	Predicted
10889	0	1
6894	1	1
11540	1	1
14932	1	1
8505	1	1
11553	1	1
14912	1	1
4525	1	1
6086	1	1
960	0	1

Figure 13: Showing predictions with a table

```
#Total test samples
pf.shape
```

```
(4847, 2)
```

```
#How many correctly samples our model predicted
pf[pf['Actual']==pf['Predicted']].count()
#As a result, out of 4847 samples, 4780 were predicted correctly
```

```
Actual      4780
Predicted   4780
dtype: int64
```

Figure 14: Actual and predicted outputs

```
#Finding accuracy
#Thus, 4780 out of 4847 samples are correctly predict
accuracy=(4780/4847)*100
print(accuracy)
```

```
98.61770167113679
```

Figure 15: Accuracy result

Besides, classification report and confusion matrix can be used in addition to accuracy value to evaluate the model. The accuracy value according to the classification report is shown below. Additionally, values in the confusion matrix show how many values are wrong and how many values are correct as a result of predict. It also provides information about error values. According to this, there are 67 false predict values, while 4780 are correct.

```
#Classification report results
from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	67
1	0.99	1.00	0.99	4780
accuracy			0.99	4847
macro avg	0.49	0.50	0.50	4847
weighted avg	0.97	0.99	0.98	4847

Figure 16: Classification report results

```
#Confusion matrix results
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,predictions)

array([[ 0, 67],
       [ 0, 4780]], dtype=int64)
```

Figure 17: Confusion matrix results

## 2.2 k Nearest Neighbor (kNN) Model

k-Nearest Neighbor is a lazy learning algorithm which stores all instances correspond to training data points in n-dimensional space. When an unknown discrete data is received, it analyzes the closest k number of instances saved nearest neighbors and returns the most common class as the prediction and for real-valued data it returns the mean of k nearest neighbors.[3]

The same train and test data allocated previously are used in this model. First of all, the model is built.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 1)
```

Figure 18: Building kNN model with  $k=1$

After the model was built, training was performed for different  $k$  values. The accuracy values of the trained data were also examined for each increasing  $k$  value. Accordingly, it was observed that the accuracy values generally decreased as the  $k$  values increased.

```
k=10

knn = KNeighborsClassifier(n_neighbors = 10)

#The accuracy for the knn k=10 model
knn.fit(x_train,y_train)
Actual = y_test
Predicted = knn.predict(x_test)
print(accuracy_score(Actual,Predicted)*100)

98.59707035279554

k=20

knn = KNeighborsClassifier(n_neighbors = 20)

#The accuracy for the knn k=20 model
knn.fit(x_train,y_train)
Actual = y_test
Predicted = knn.predict(x_test)
print(accuracy_score(Actual,Predicted)*100)

98.61770167113679
```

Figure 19: Sample  $k$  values results

However, it will not be good to reach a definite result with a few trials in this way. Therefore, the results of  $k$  values up to 40 values using the Elbow Method were listed as prediction and error. According to the results, the value of  $k$  with the lowest error and highest accuracy is 3.

Thus, the model was re-trained for the value of  $k = 3$ . According to the evaluation of this model, the resulting accuracy value is higher than the logistic regression value. In other words, this model gave a better result than logistic regression.

```
#Find a more accurate value of k
# Function
error_rate = []

for i in range (1,40):

    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(x_train, y_train)
    pred_i = knn.predict(x_test)
    error_rate.append(np.mean(pred_i != y_test))

# Plot error rate
plt.figure(figsize = (10,6))
plt.plot(range(1,40), error_rate, color = 'blue', linestyle = '--', marker = 'o',
        markerfacecolor = 'green', markersize = 10)

plt.title('Error Rate vs K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Figure 20: Elbow Method functions

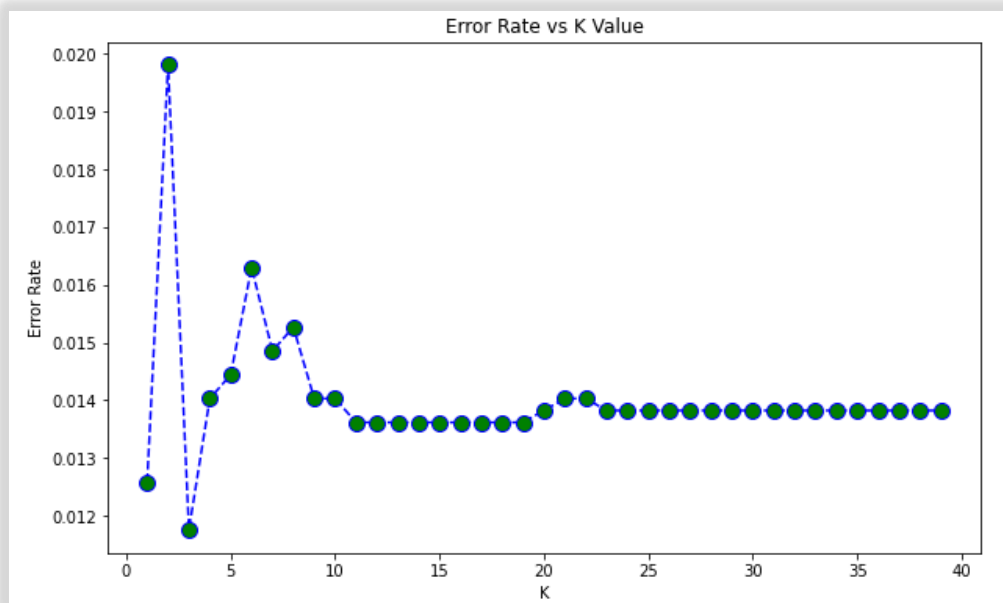


Figure 21: Elbow Method results

```
#One of the values of k where accuracy is high and error is low is k = 3
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train, y_train)
pred = knn.predict(x_test)
```

Figure 22: Retrain kNN model with k=3 value

```
#Accuracy for k=3 model
Actual = y_test
Predicted = knn.predict(x_test)
print(accuracy_score(Actual,Predicted)*100)

98.8240148545492
```

Figure 23: Accuracy value for k=3 model

```
#Print confusion matrix
print(confusion_matrix(y_test,pred))
print('\n')
#Print classification report
print(classification_report(y_test, pred))
```

```
[[ 17  50]
 [   7 4773]]
```

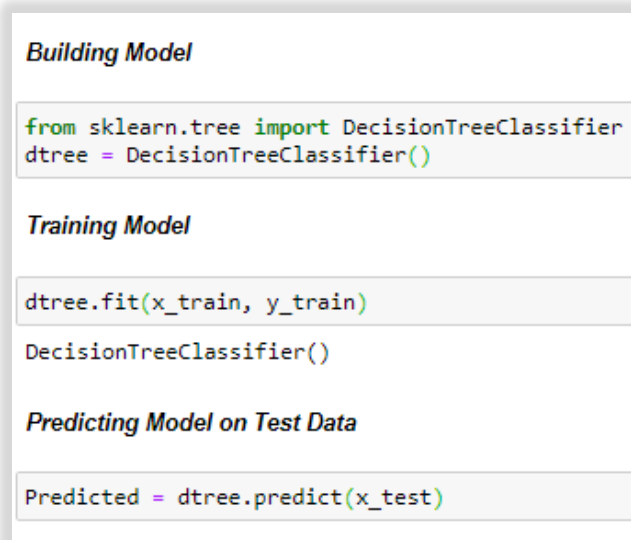
		precision	recall	f1-score	support
	0	0.71	0.25	0.37	67
	1	0.99	1.00	0.99	4780
	accuracy			0.99	4847
	macro avg	0.85	0.63	0.68	4847
	weighted avg	0.99	0.99	0.99	4847

Figure 24: Confusion matrix and classification report results

## 2.3 Decision Tree Model

In Decision Tree Model, rules are learned sequentially using training data one by one. Every time a rule is learned, tuples covered by the rules are removed. This process continues through the training set until a termination condition is met. [3]

Similar to the previous models, the decision tree model was first built. Afterwards, the model was trained. In addition, it was predict with the test data. In the last stage, the model was evaluated and its accuracy value was examined as in every model. In addition, as with kNN, the decision tree model also gives a better accuracy than logistic regression. After this result, it is understood that decision tree gives the highest accuracy value among the three models. The accuracy value is very high, such as one hundred percent, but the fact that this result is as high as a hundred is not always good.



```
Building Model

from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()

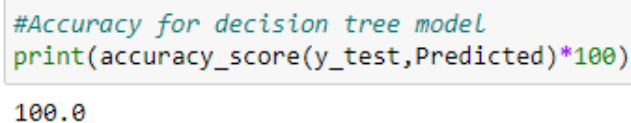
Training Model

dtree.fit(x_train, y_train)
DecisionTreeClassifier()

Predicting Model on Test Data

Predicted = dtree.predict(x_test)
```

Figure 25: Stages of creating the decision tree model



```
#Accuracy for decision tree model
print(accuracy_score(y_test, Predicted)*100)

100.0
```

Figure 26: Accuracy value for decision tree model



```

#Confusion matrix
print(confusion_matrix(y_test,Predicted))

print('\n')

#Classification report
print(classification_report(y_test, Predicted))

```

```

[[ 67   0]
 [  0 4780]]

```

		precision	recall	f1-score	support
	0	1.00	1.00	1.00	67
	1	1.00	1.00	1.00	4780
	accuracy			1.00	4847
	macro avg	1.00	1.00	1.00	4847
	weighted avg	1.00	1.00	1.00	4847

Figure 27: Confusion matrix and classification report results

### 3. Results

In this section, the comparisons of the three classification models with each other are included. Comparisons have been made according to the results in the Method section, but in this section, Cross Validation analysis has also been made. For cross validation analysis, the data were examined from 10 different parts. Inferences were made based on the accuracy values obtained as a result of these investigations. Accordingly, the results are given below.

#### 3.1 Logistic Regression Model

The 10 accuracy values for the logistic regression model are given below. However, for a good result, the average of this accuracy value should be chosen. Accordingly, the result is 98.46124895346745, which is the lowest one compared to the other two models.

```
from sklearn.model_selection import cross_val_score
accuracy = cross_val_score(logmodel,x_train, y_train, cv = 10, scoring = "accuracy")
print(accuracy)
```

Figure 28: Cross Validation score for Logistic Regression Model

```
[0.98585323 0.97082228 0.98585323 0.98143236 0.98585323 0.98585323
 0.99911583 0.98408488 0.98141593 0.98584071]
```

Figure 29: Resulting 10 accuracy values

```
#The average of all this accuracy should be chosen
accuracy.mean()*100
98.46124895346745
```

Figure 30: Average of accuracy values

### 3.2 k Nearest Neighbor (kNN) Model

In kNN model, accuracy values are for different values of k up to 40. Accordingly, the highest accuracy value is 98.94761468823108. As in the previous method part, it gave a better result than the logistic regression model. Additionally, it is seen that this result is appropriate according to the k = 3 model in the method section.

```
#Accuracy for KNN model
acc = []
for i in range (1,40):

    knn = KNeighborsClassifier(n_neighbors = i)
    accuracy = cross_val_score(knn,x_train, y_train, cv = 10, scoring = "accuracy")
    accuracy = accuracy.mean()*100
    acc.append(accuracy)
print(max(acc))

98.94761468823108
```

*Figure 31: Accuracy value for kNN model*

### 3.3 Decision Tree Model

The accuracy value for the Decision tree model is 99.99115826702034. This model gives the best results compared to the other two models, but it should be noted that such high accuracy values may not be desired in some cases.

```
#Accuracy for decision tree model
accuracy = cross_val_score(dtrees,x_train, y_train, cv = 10, scoring = "accuracy")
accuracy.mean()*100

99.99115826702034
```

*Figure 32: Accuracy value for Decision Tree model*

## 4. Conclusion

With this project, we learned the machine learning process better and gained experience in this field. For this, we first selected a data suitable for our project from the Kaggle. Then, we analyzed this data in accordance with exploratory data analysis and tried to understand it. At this stage, we obtained more effective results by using data visualization according to the information in the analysis. After completing the preliminary steps, we trained the models using this data. We used classification models such as Logistic Regression, kNN, Decision Tree. After completing the model training and predict stages according to the test data, we compared these three different models to each other. For this, we used the accuracy values calculated separately for each model. These results are detailed in the method section. In order to make the comparison more effective, we used Cross Validation analysis. Accordingly, the model that gives the best estimation and has the highest accuracy is decision tree. However, it should be noted that such high accuracy values may cause undesirable results in some cases. Therefore, a different suitable model can be selected by examining the accuracy values according to the desired situation.

## 5. References

- [1] <https://www.kaggle.com/fehmiiratpolat/istanbul-airbnb-data-analysis-and-visualization>
- [2] <https://www.kaggle.com/kavanozkafa/airbnb-istanbul-dataset>
- [3] Ceng 474 Introduction to Data Science Classification Lecture Notes, Cankaya University