Final Project Machine Learning
Clustering and Classification (Airbnb)

Akmal Ariq Santoso (1301174378)

TELKOM UNIVERSITY
SCHOOL OF COMPUTING
INFORMATICS ENGINEERING
2020

1. Classification

From Problem to ML Solution

a) Articulate Your Problem Clearly

The problem is a 3-class, single-label classification, which predicts whether the room_type of the listing is one of the three classes ('Entire home/apt', 'Private room', 'Shared room')

b) Identify Your Data Sources

In this experiment we will use 22552 labeled data. The decision here is to find out the trend between room_type in the dataset. So here we want to classify which are categorized as such

The features are:

- Id : id of the rented space, represented by int64, each id is unique
- Name : name of the rented space, represented by a string, has 59 null values
- Host_id : id of the user, represented by int64 there are 19180 host_id in 22552 data
- Host_name : name of the user, represented by a string, has 29 null values
- Neighbourhood_group : 12 categorical feature represented by
- Neighbourhood : 136 unique categorical feature
- Latitude : Unique set of float64
- Longitude : Unique set of float64
- Price : 295 prices represented by int64
- Minimum_nights : 102 unique int64
- Number_of_reviews : 306 unique int64
- Last_review : 1313 different dates of the last review, contains 3908 NULL Values
- Reviews_per_month : 769 unique float64
- Calculated_host_listings_count : 23 unique int64, how many listings a host has.
- Availability_365 : 366 unique int64

c) Identify Potential Learning Problems

- The dataset is unbalanced between room_type:

Private room	0.511440
Entire home/apt	0.475435
Shared room	0.013125

- The features are noisy and has outliers

d) Think About Potential Bias and Ethics

The writer thinks that there are no Bias and Ethics concerns in this experiment

A. Experiment 1 (KNN)

1. Set the research goal

The goal of this experiment is to use K-Nearest Neighbour to predict the classification process.

2. Make a hypothesis

In the dataset there are three types of category for room_type. They are: Entire home/apt, Private Room, and Shared room. From my understanding, what would be informative features to predict the label are Neighbourhood_group, Neighbourhood, Price, Minimum_nights, Number_of_reviews, Calculated_host_listings_count, and Availability_365.

3. Collect the data

The dataset air_bnb.csv is provided by the lecturer.

4. Test your hypothesis

1. KNN

In this model the selected features are:

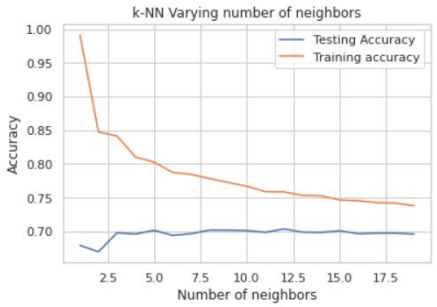
1. neighbourhood_group
2. neighbourhood
3. calculated_host_listings_count
4. price/minimum_nights
5. number_of_reviews
6. availability_365

The target is:

- room_type

In [0]:
```
#Setup arrays to store training and test accuracies
neighbors = np.arange(1,20)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    #Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    #Fit the model
    knn.fit(X_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)

    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test)
```

In [0]:
```
#Generate plot
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```



In [0]:
```
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(n_neighbors=12)
```

In [0]:
```
#Fit the model
knn.fit(X_train,y_train)
```

Out[0]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=12, p=2,
                     weights='uniform')
```

In [0]:
```
#Get accuracy. Note: In case of classification algorithms score method represents accuracy.
knn.score(X_test,y_test)
```

Out[0]:
```
0.7036653857522909
```

5. Analyze the results

Evaluation

```
In [0]: pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
```

Out[0]:

Predicted	0	1	2	All
True				
0	2245	971	1	3217
1	959	2496	5	3460
2	26	43	20	89
All	3230	3510	26	6766

```
In [0]: print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.70      0.70      0.70      3217
           1       0.71      0.72      0.72      3460
           2       0.77      0.22      0.35        89

    accuracy                           0.70      6766
   macro avg       0.73      0.55      0.59      6766
weighted avg       0.70      0.70      0.70      6766
```

6. Reach a conclusion

The accuracy score for this model is 70%, which is not that great. In the next step the writer will use select the best feature and tune the model for better accuracy.

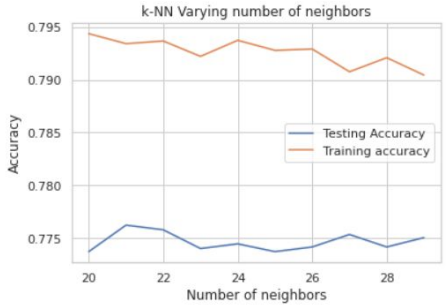
7. Refine hypothesis and repeat

Model Tuning

Feature Selection

In [0]:
```
data = df_split
X = data[['price','minimum_nights','availability_365','number_of_reviews']].values  #independent columns
y = data['room_type'].values    #target column i.e price range

# Split the data
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=42, stratify=y)
```

In [0]:
```
#Setup arrays to store training and test accuracies
neighbors = np.arange(20,30)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    #Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    #Fit the model
    knn.fit(X_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)

    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test)


#Generate plot
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```



In [0]:
```
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(n_neighbors=21)

#Fit the model
knn.fit(X_train,y_train)

#Get accuracy. Note: In case of classification algorithms score method represents accuracy.
knn.score(X_test,y_test)
```

Out[0]:
```
0.7762341117351463
```

Through intuition and iteration, the best features are:

- price
- minimum_nights
- availability_365
- number_of_reviews

The best k is 21

The score is 0.7762341117351463

In [0]:
```
#List Hyperparameters that we want to tune.
leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
p=[1,2]
#Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
#Create new KNN object
knn_2 = KNeighborsClassifier()
#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10)
#Fit the model
best_model = clf.fit(X,y)
#Print The value of best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

```
Best leaf_size: 1
Best p: 1
Best n_neighbors: 29
```

Best leaf_size: 1

Best p: 1

Best n_neighbors: 29

In [0]:
```
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(n_neighbors=29, leaf_size=1 , p=1)

#Fit the model
knn.fit(X_train,y_train)

#Get accuracy. Note: In case of classification algorithms score method represents accuracy.
knn.score(X_test,y_test)
```

Out[0]:
```
0.7778598876736624
```

Score is 0.7778598876736624

In conclusion the result accuracy of this model is 77%

B. Experiment 2 (Random Forest)

1. Set the research goal

The goal of this experiment is to useRandom Forest to predict the classification process.

2. Make a hypothesis

In the dataset there are three types of category for room_type. They are: Entire home/apt, Private Room, and Shared room. From my understanding, what would be informative features to predict the label are Neighbourhood_group, Neighbourhood, Price, Minimum_nights, Number_of_reviews, Calculated_host_listings_count, and Availability_365.

3. Collect the data

The dataset air_bnb.csv is provided by the lecturer.

4. Test your hypothesis

2. Random Forest

In [0]:
```
df_randomforest = df_split
```

In [0]:
```
df_randomforest
```

Out[0]:

	neighbourhood_group	neighbourhood	room_type	price	minimum_nights	number_of_reviews	calculated_host_listings_count	availability_365
0	4	18	0	-0.032433	-0.077637	2.724030	0.567676	0.512268
1	6	95	1	-0.227655	-0.126821	-0.322031	-0.250393	-0.668977
2	6	98	0	0.103769	1.348680	3.403954	-0.250393	1.174101
3	10	110	1	-0.186795	-0.053046	0.194712	-0.250393	1.819178
4	6	49	1	-0.114154	-0.126821	4.872590	-0.250393	-0.451159
...	...	...	...	...	...	...	...	...
22547	4	18	0	-0.032433	-0.126821	-0.485212	-0.250393	1.961598
22548	10	110	2	-0.214035	-0.151412	-0.485212	1.113055	-0.015522
22549	6	98	0	0.081069	-0.102229	-0.485212	0.022297	-0.543313
22550	4	2	1	0.144630	-0.151412	-0.485212	0.294986	-0.618712
22551	5	105	1	-0.100534	-0.053046	-0.485212	-0.250393	-0.493047

22552 rows × 8 columns

In [0]:
```
data = df_randomforest
# Split data to X and y
X = data.drop(columns='room_type').values   #independent columns
y = data['room_type'].values                #target column i.e price range

# Split data to X_train,X_test,y_train,y_test
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42, stratify=y)
```

In [0]:
```
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
```

Out[0]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [0]:
```
clf.score(X_test,y_test)
```

Out[0]:
```
0.7954478273721549
```

5. Analyze the results

5. Evaluation

In [0]:
```
y_pred = clf.predict(X_test)
confusion_matrix(y_test,y_pred)
```

Out[0]:
```
array([[2599,  616,    2],
       [ 694, 2759,    7],
       [  11,   54,   24]])
```

In [0]:
```
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
```

Out[0]:

Predicted	0	1	2	All
True				
0	2599	616	2	3217
1	694	2759	7	3460
2	11	54	24	89
All	3304	3429	33	6766

In [0]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.81      0.80      3217
           1       0.80      0.80      0.80      3460
           2       0.73      0.27      0.39        89

    accuracy                           0.80      6766
   macro avg       0.77      0.62      0.66      6766
weighted avg       0.80      0.80      0.79      6766
```

6. Reach a conclusion

The accuracy score for this model is 79,5%, which is not that great. In the next step the writer will use select the best feature and tune the model for better accuracy.

7. Refine hypothesis and repeat

- Hyperparameter Tuning

In [0]:
```
data = df_randomforest
# Split data to X and y
X = data.drop(columns='room_type').values  #independent columns
y = data['room_type'].values    #target column i.e price range

# Split the data
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=42, stratify=y)
```

In [0]:
```
rf = RandomForestClassifier(max_features='auto', oob_score=True, random_state=42, n_jobs=-1)

param_grid = { "criterion" : ["gini", "entropy"], "min_samples_leaf" : [1, 5, 10], "min_samples_split" : [2, 4, 10, 12, 16], "n_
estimators": [50, 100, 400, 700, 1000]}

gs = GridSearchCV(estimator=rf, param_grid=param_grid, scoring='accuracy', cv=3, n_jobs=-1)

gs = gs.fit(X, y)
```

```
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_executor.py:706: UserWarning: A worker stopped while some
jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

In [0]:
```
print(gs.best_score_)
print(gs.best_params_)
print(gs.best_estimator_)
```

```
0.7981546518854507
{'criterion': 'entropy', 'min_samples_leaf': 5, 'min_samples_split': 16, 'n_estimators': 1000}
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=16,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=-1, oob_score=True, random_state=42, verbose=0,
                       warm_start=False)


0.7981546518854507
```

In [0]:
```
rf = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=16,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=-1, oob_score=True, random_state=42, verbose=0,
                       warm_start=False)
```

In [0]:
```
rf.fit(X_train, y_train)
```

Out[0]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=16,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=-1, oob_score=True, random_state=42, verbose=0,
                       warm_start=False)
```

In [0]:
```
print("%.4f" % rf.oob_score_)
```

```
0.8083
```

In [0]:
```
rf.score(X_test, y_test)
```

Out[0]:
```
0.8047590895654745
```

In conclusion the result accuracy of this model is 80.4%

2. Clustering

In the clustering experiment, K-Means algorithm is chosen

K Means Algorithm

In [0]:

```python
# https://www.youtube.com/watch?v=HRoeYblYhkg

import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
import numpy as np

colors = 10*['g','r','c','b','k']

class K_means:
  def __init__(self, n_clusters=2, tol=0.001, max_iter=300):
    # The number of clusters to form as well as the number of centroids to generate.
    self.n_clusters = n_clusters
    # Relative tolerance with regards to inertia to declare convergence.
    self.tol = tol
    # Maximum number of iterations of the k-means algorithm for a single run.
    self.max_iter = max_iter

  # Fit functions
  def fit(self, data):

    self.centroids = {}

    # Assigning top n_cluster data into centroids
    for i in range(self.n_clusters):
      self.centroids[i] = data[i]

    # Iterations
    for i in range(self.max_iter):
      # The classification of n_clusters
      self.classifications = {}
      # The labeled data
      self.labels_ = []

      # Assigning classification of each centroids to empty list
      for i in range(self.n_clusters):
        self.classifications[i] = []

      # Computing the distance between centroids, to find the nearest centroid
      for featureset in data:
        # Euclidian distances
        distances = [np.linalg.norm(featureset-self.centroids[centroid]) for centroid in self.centroids]
        # Find the minimum distace between clusters
        classification = distances.index(min(distances))
        self.labels_.append([featureset[0], featureset[1], classification])
        self.classifications[classification].append(featureset)

      # Assigning original_centroids to prev_centroids
      prev_centroids = dict(self.centroids)

      # Assigning new centroids
      for classification in self.classifications:
        self.centroids[classification] = np.average(self.classifications[classification], axis=0)


      optimum = True

      # Searching optimum centroid, tolerance < self.tol
      for c in self.centroids:
        original_centroid = prev_centroids[c]
        current_centroid = self.centroids[c]
        if np.sum((current_centroid-original_centroid)/original_centroid*100.0)>self.tol:
          optimum = False
        if optimum==True:
          break
```

A. Experiment 1 (price and number_of_reviews)

From Problem to ML Solution

a) Articulate Your Problem Clearly

In this experiment the writer would like to know the amount of clusters in price and number_of_reviews

b) Identify Your Data Sources

In this experiment we will use 22552 unlabeled data. The decision here is to find out the clusters between room_type in the dataset.

c) Identify Potential Learning Problems

The features are noisy and has outliers

d) Think About Potential Bias and Ethics

The writer thinks that there are no Bias and Ethics concerns in this experiment

a. Set the research goal

The goal of this experiment is to find the clusters in price and number_of_reviews.

b. Make a hypothesis

My hypothesis is that there are 3 clusters

c. Collect the data

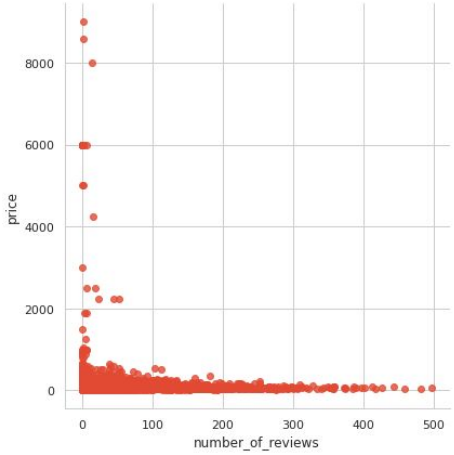
The data is provided by the lecturer air_bnb.csv

In [0]:
```
df_exp1 = df[['number_of_reviews','price']].copy()
```

In [174]:
```
sns.set_style('whitegrid')
sns.lmplot('number_of_reviews','price',data=df_exp1,
           palette='coolwarm',size=6,aspect=1,fit_reg=False)
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/regression.py:574: UserWarning: The `size` parameter has been renamed to `height
`; please update your code.
  warnings.warn(msg, UserWarning)
```

Out[174]:
```
<seaborn.axisgrid.FacetGrid at 0x7f2aae042828>
```



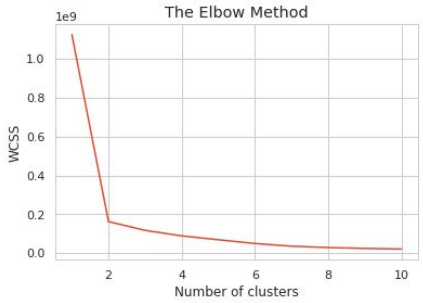
d. Test your hypothesis

Elbow Method

Searching for the optimum number of clusters using elbow method

In [176]:

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



The best number of cluster shown by the elbow method is 2.

K=2

Through the elbow method we can see that the optimum K is 2

e. Analyze the results

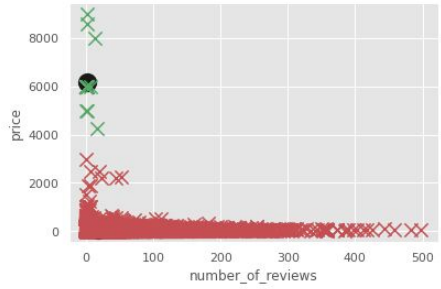
Fit Model

In [0]:
```
clf = K_means(n_clusters=2)
clf.fit(X)
```

Show Clusters (Slow)

PRECAUTION THIS METHOD IS REALLY SLOW

In [76]:
```
for centroid in clf.centroids:
  plt.scatter(clf.centroids[centroid][0], clf.centroids[centroid][1],
              marker='o', color='k', s=150, linewidth=5)

for classification in clf.classifications:
  color = colors[classification]
  for featureset in clf.classifications[classification]:
    plt.scatter(featureset[0], featureset[1], marker='x', color=color, s=150, linewidths=5)

plt.xlabel('number_of_reviews')
plt.ylabel('price')
plt.show()
```



f. Reach a conclusion

In Conclusion
There are 2 clusters:

cluster 0:
- low reviews
- expensive

cluster 1:
- high reviews
- moderate price

B. Experiment 2 ()

From Problem to ML Solution

a) Articulate Your Problem Clearly

In this experiment the writer would like to know the amount of clusters in price and minimum_nights

b) Identify Your Data Sources

In this experiment we will use 22552 unlabeled data. The decision here is to find out the clusters between room_type in the dataset.

c) Identify Potential Learning Problems

The features are noisy and has outliers

d) Think About Potential Bias and Ethics

The writer thinks that there are no Bias and Ethics concerns in this experiment

a. Set the research goal

The goal of this experiment is to find the clusters in price and minimum_nights.

b. Make a hypothesis

My hypothesis is that there are 3 clusters

c. Collect the data

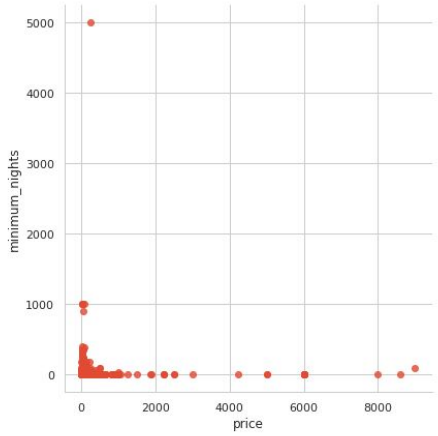
The data is provided by the lecturer air_bnb.csv

```
In [0]:
1 df_exp2 = df[['price','minimum_nights']].copy()
```

```
In [163]:
1 sns.set_style('whitegrid')
2 sns.lmplot('price','minimum_nights',data=df_exp2,
3            palette='coolwarm',size=6,aspect=1,fit_reg=False)
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/regression.py:574: UserWarning: The `size` parameter has been renamed to `height
`; please update your code.
  warnings.warn(msg, UserWarning)
```

```
Out[163]: <seaborn.axisgrid.FacetGrid at 0x7f2ac23db470>
```

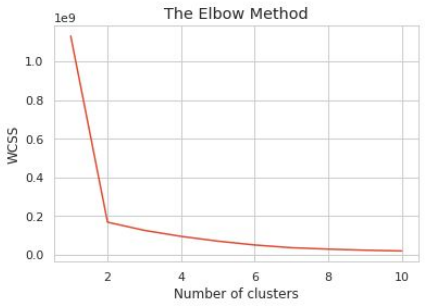


d. Test your hypothesis

Elbow Method

Searching for the optimum number of clusters using elbow method

```
In [165]:
1  from sklearn.cluster import KMeans
2  wcss = []
3  for i in range(1,11):
4      kmeans = KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
5      kmeans.fit(X)
6      wcss.append(kmeans.inertia_)
7  plt.plot(range(1,11),wcss)
8  plt.title('The Elbow Method')
9  plt.xlabel('Number of clusters')
10 plt.ylabel('WCSS')
11 plt.show()
```



Through the elbow method we can see that the optimum K is 2

e. Analyze the results

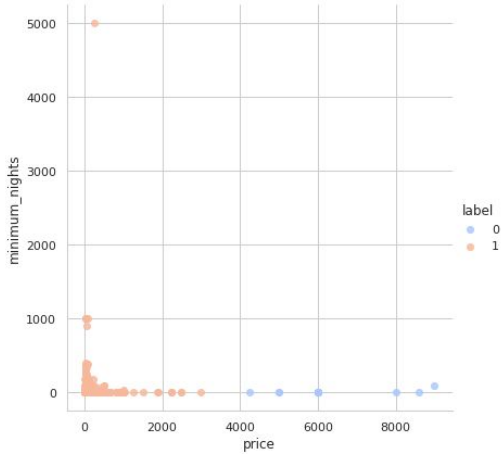
Fit the Model

In [0]:
```
clf = K_means(n_clusters=2)
clf.fit(X)
```

In [171]:
```
sns.set_style('whitegrid')
sns.lmplot('price','minimum_nights',data=df_exp2, hue='label',
           palette='coolwarm',size=6,aspect=1,fit_reg=False)
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/regression.py:574: UserWarning: The `size` parameter has been renamed to `height
`; please update your code.
  warnings.warn(msg, UserWarning)
```

Out[171]: <seaborn.axisgrid.FacetGrid at 0x7f2aaecdd6a0>



f. Reach a conclusion

In conclusion
There are 2 clusters:

cluster 0:
- lower minimum_nights
- expensive

cluster 1:
- moderate to high minimum_nights
- moderate price