House Prices in Tehran

Project Documentation

Machine Learning Project 2020-21

Professor : P. Velardi

Student: Parham Membari

Student Number: 1966736

Contact: My Email

SAPIENZA
UNIVERSITÀ DI ROMA

Table of Contents

Introduction:

Problem:

Nowadays, the Prediction of house prices is one of the hot subjects in Machine Learning tasks. Unfortunately, there is no data set like those data set in Kaggle for predicting houses of Tehran which is the capital of Iran.

Data set & Tools:

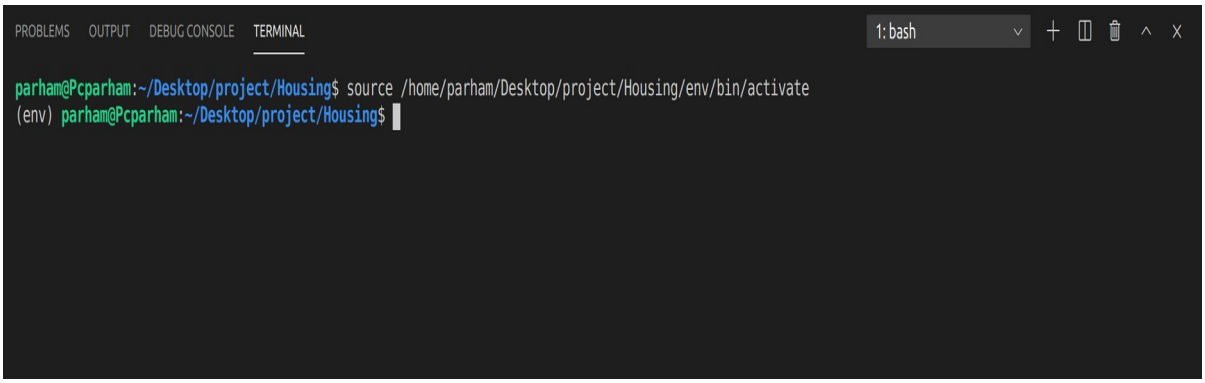
In this project, A data set with 4506 records and 8 columns is aggregated by taking advantage of web crawling from ihome website. One of the challenging barriers I cope with is Persian Font which is used in the website.
I used scrapy library for web crawling to aggregate data. Subsequently, A CSV file is created to store the data in which named posts.csv. Then I used Jupyter notebook to analyze the data.
I used Python 3.8.5 as an interpreter and some packages such as scikit-learn, Numpy, Pandas(for loading data set and creating data frame).

Barriers:

The most challenging barrier was coping with Persian font. Also, most columns have not identical data type for the records which contains (e.g. the price records have type of integer,float, Nonetype and string)

Web Crawling:

first I created a virtualenv named env and I activate the env with the command bellow:



When the virtualenv is activated we have “env” in parenthesis.
Then I start an scrapy project (named Housing) to create posts_spider.py (located in spiders directory). This file is responsible for crawling data from website.

In the posts_spider.py there is a class which is inherited from **scrapy.Spider** it has two attributes “name” and “start_urls” (the urls which are crawled).
The class has a method which name is parse (those xpath(s) should be crawled). I find the xpath of each section of website though “inspect elements” of website.

The features which are crawled are:
- Zone
- Price
- How many years the apartment is constructed
- Number of Rooms
- Area(m^2)
- The floor which the apartment is located
- Number of elevators
- side of apartment(direction)

```
def parse(self,response):

    for post, option in zip(response.xpath('//*[@id="__layout"]/div/div[1]/div/div[1]/div[2]/div'),response.xpath('//*[@id="__layout"]/div/div[1]/div/div[3]/div/div[2]')):
        is_for_sale = post.xpath('div[2]/h1/text()').extract()
        if 'فروش' in is_for_sale[0]:
            Zone = post.xpath('div[2]/h1/text()').extract()
            Zone = cleanTitle(Zone[0])

            price = post.xpath('div[3]/div/div[2]/strong/text()').extract()


            if post.xpath('div[4]/div[1]/div/div/span[1]/span/text()').extract() :
                Years = post.xpath('div[4]/div[1]/div/div/span[1]/span/text()').extract()
            else:
                Years = 0

            Rooms = post.xpath('div[4]/div[2]/div/div/span[1]/span/text()').extract()
            Rooms = int(Rooms[0])

            Area = post.xpath('div[4]/div[3]/div/div/span[1]/span/text()').extract()
            Area = int(Area[0])

        ## '//*[@id="__layout"]/div/div[1]/div/div[3]/div/div[2]/div[1]/span[2]'
        # further options
            NumberOfElevators = option.xpath('div[1]/span/text()').extract()
            # if NumberOfElevators is None:
            #     NumberOfElevators[0] = ['0']
            # NumberOfElevators = int(Rooms[0])

            floor = option.xpath('div[4]/span[1]/text()').extract()

            if floor[0] == '—' or floor[0] =='-':
                floor = None


            direction = option.xpath('div[5]/span[1]/text()').extract()

            direction = str(direction[0])
            if direction == '-':
                direction = None
            yield{
                'Zone':Zone,
                'Price': price,
                'Years' : Years,
                'Rooms': Rooms,
                'Area' : Area,
                'NumberOfElevators': NumberOfElevators,
                'floor': floor,
                'direction':direction
```

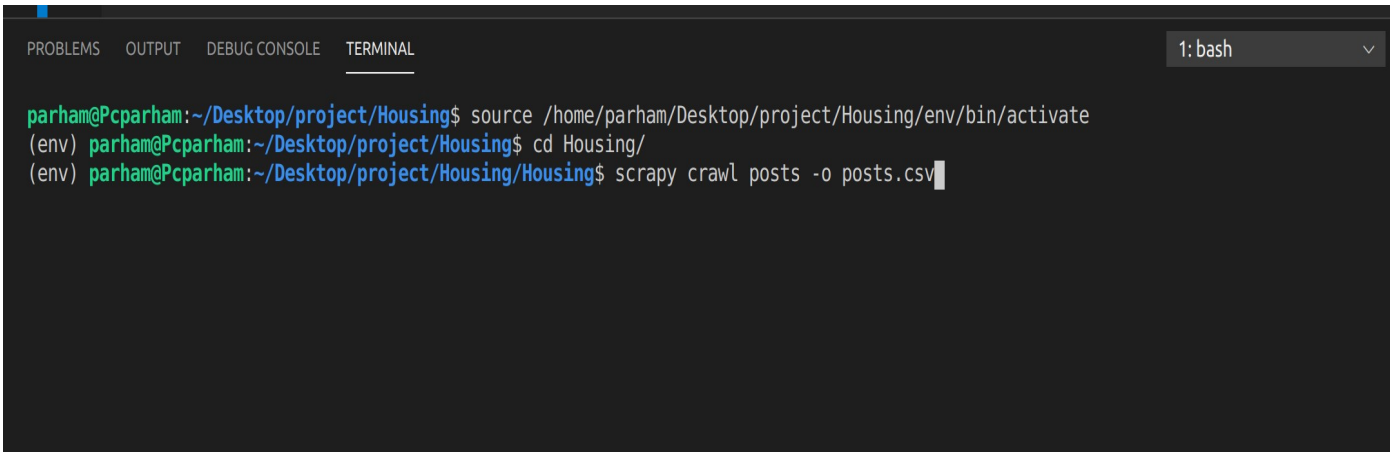
```
OUTPUT  DEBUG CONSOLE  TERMINAL

nt already satisfied: numpy<1.23.0,>=1.16.5 in /home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages (from scipy) (1.20.1)
ham@Pcparham:~/Desktop/project/Housing/Housing$
```

Ln 72, Col 18 Spaces: 4

In order to clean the Zone of each apartment in order to point to exact distinct of Tehran, I coded a method which named CleanTitle:

```
def cleanTitle(zone):
    title = zone
    if 'متری' in str(title):
        title = title.split('متری')

    title[-1].replace('r\n                ','')
    new_title = ''
    if '-' in title[-1] :
        index = title[-1].index('-')
        for i in range(0,index):
            new_title = new_title+title[-1][i]
        title[-1] = new_title

    title = str(title[-1])
    return title
```

Then I saved the records in a .CSV file which named is posts.csv by the command bellow(in terminal):



this process(web crawling) takes a while to save records in .CSV file.

Finally the result is a .CSV file with 4506 records in 8 columns.

Machine Learning Analysis:

the most important section of this project is predicting the house price in Tehran with taking advantage of Machine Learning algorithms. I use **Jupyter notebook** and **Python 3.8.5** as interpreter. All codes associated with Machine Learning are in a **Jupyter notebook** named **learning.ipynb.**

Instructions:

1- Read the CSV file.

2- Preprocessing:

- Getting ready the records for learning the data

3- Data splitting.

3- Model Selection:

- SVM
- Ensemble (Random forest)
- MLP classifier
- Decision Tree

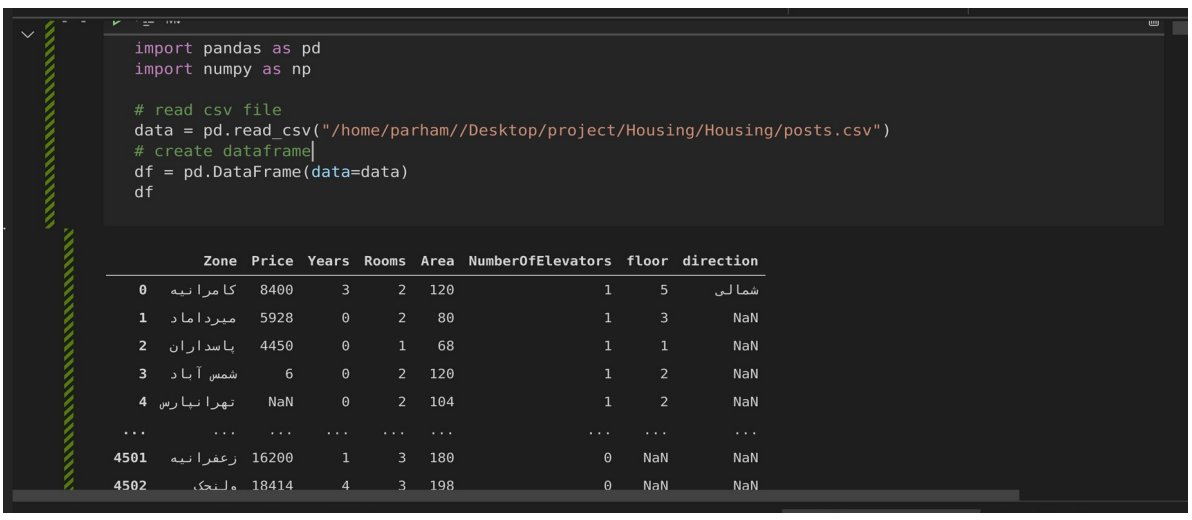
4- Tuning Hyper parameters.

5- Evaluation

1- Read the CSV file:

First I import some open-source libraries such as Pandas and Numpy.

The CSV file is read by one of **Pandas** library function called "**read_csv**" and then I created a Pandas data frame (named df) as bellow:



```
import pandas as pd
import numpy as np

# read csv file
data = pd.read_csv("/home/parham//Desktop/project/Housing/Housing/posts.csv")
# create dataframe
df = pd.DataFrame(data=data)
df
```

	Zone	Price	Years	Rooms	Area	NumberOfElevators	floor	direction
0	کامرانیه	8400	3	2	120	1	5	شمالی
1	میرداماد	5928	0	2	80	1	3	NaN
2	پاسداران	4450	0	1	68	1	1	NaN
3	شمس آباد	6	0	2	120	1	2	NaN
4	تهرانپارس	NaN	0	2	104	1	2	NaN
...	...	...	...	...	...	...	...	...
4501	زعفرانیه	16200	1	3	180	0	NaN	NaN
4502	ولنجک	18414	4	3	198	0	NaN	NaN

2- Preprocessing:

2-1) Dealing with data types:

Most of our data have not ideal type for Machine learning Classifiers. For Example in Price column which is the target feature we face with a lot of empty records. These records are ignored by **dropna** function. After implementing this method our data base reduces to **4470** . Also most of price records have wrong scale (indeed all records must be represented in scale of billion Toman (the Monetary unit in Iran)).

Ex:

1st record: 8400 (8 billion and 4 hundred million)

4th record: 6 (6 billion)

after converting:

8400 ==> 8.4 (billion Toman) type:float64

6 ==> 6.0 (billion Toman) type:float64

thereby, I converted the price column as bellow:

```
### Lets clean the data
df=df.dropna(subset = ['Price']) # drop the rows that the target value has not any value.
df.index = range(0,df.shape[0]) # re index the dataframe
############################################################
############################################################
### convert numeric string to int or float:

#    ### convert Price column to float value
df["Price"] = df["Price"].str.replace(",","")
df["Price"] = df["Price"].astype(float)

price= df["Price"]
#Some of data prices are like 8400.0 but some of them are like 6.0 or 37.0 How ever both of them shold be represent in scale
of billion Toman so I re-scale all of data As bellow.
for i in range(0,len(price)):
    if price[i] > 100.0:
        price[i] = price[i]/1000

df
```

```
copy
  exec(code_obj, self.user_global_ns, self.user_ns)
```

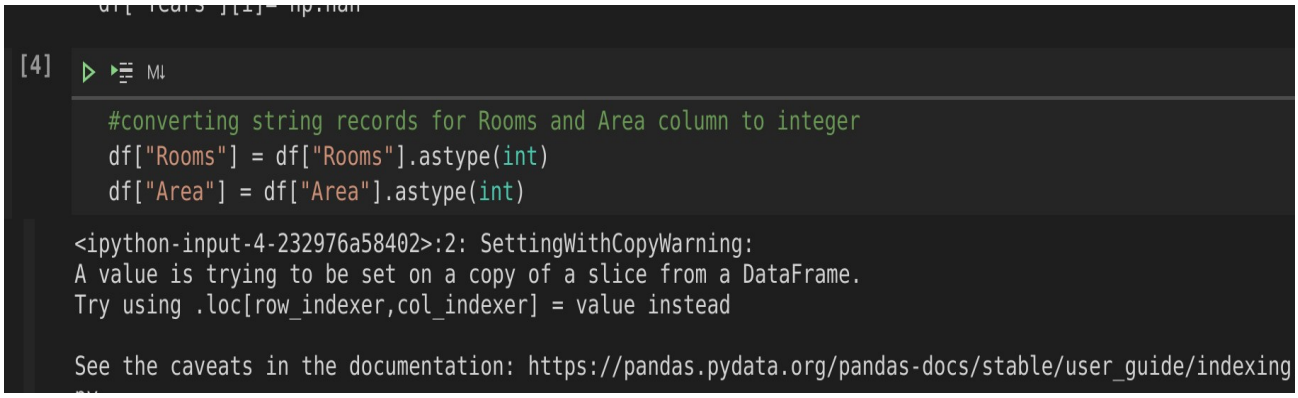
	Zone	Price	Years	Rooms	Area	NumberOfElevators	floor	direction
0	کامرانیه	8.400	3	2	120	1	5	شمالی
1	میرداماد	5.928	0	2	80	1	3	NaN
2	پاسداران	4.450	0	1	68	1	1	NaN
3	شمس آباد	6.000	0	2	120	1	2	NaN
4	[illegible]	7.700	8	3	131	1	1	NaN

also other columns such as **Years,Rooms,Area,NumberOfElevators and floor** has some records that have string type or None Type. I convert this records to integer type and I show them as bellow respectively:

Converting Years column:

```
3]
# finding how many data are null
isnull = df['Years'].isnull()
isnull = list(isnull)

# converting those records in "Years" column which are string,NoneType and integer to float type
for i in range(0,len(df['Years'])):
    if df['Years'][i] == '-':

        df['Years'][i] = None
    if type(df['Years'][i])==float or type(df['Years'][i])==int or type(df['Years'][i])==str:
        if isnull[i] == False and (df['Years'][i] is not None or df['Years'][i]!= np.nan):

            df['Years'][i]= int(df['Years'][i])
    else:

        df['Years'][i]= None
for i in range(0,len(df['Years'])):
    if df['Years'][i]== None:
        print(i,"   " ,df['Years'][i])
        df['Years'][i]= np.nan
```

Converting **Rooms & Area** column:



```
[4]
#converting string records for Rooms and Area column to integer
df["Rooms"] = df["Rooms"].astype(int)
df["Area"] = df["Area"].astype(int)

<ipython-input-4-232976a58402>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing
```

Converting **NumberOfElevators** column:

```
[5]
# converting those records in "NumberOfElevators" column which are string,NoneType to integer type
for i in range(0,len(df['NumberOfElevators'])):
    if df['NumberOfElevators'][i] == '-':
        df['NumberOfElevators'][i] = None
    if df['NumberOfElevators'][i] =='دارد':
        df['NumberOfElevators'][i] = 1


    if type(df['NumberOfElevators'][i])== str :

        df['NumberOfElevators'][i]= int(df['NumberOfElevators'][i])
    if type(df['NumberOfElevators'][i]) == int:
        pass
    else:

        df['NumberOfElevators'][i] = 0

<ipython-input-5-e86f992ebaad>:11: SettingWithCopyWarning:
```

Converting **floor** column:

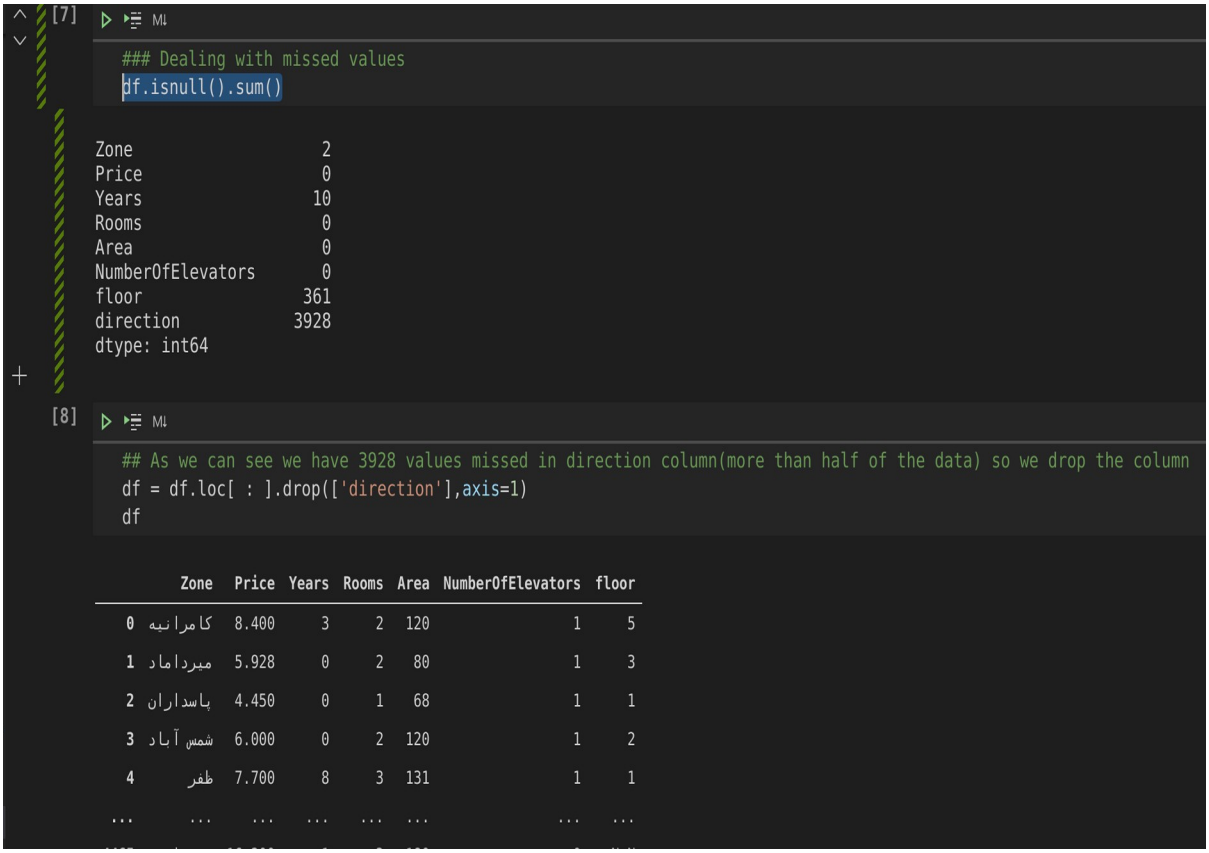
```
[6]
import math
import numpy as np
# converting those records in "floor" column which are string,NoneType to integer type
isnull = df['floor'].isnull()
isnull = list(isnull)
for i in range(0,len(df['floor'])):
    if df['floor'][i] == '-':
        df['floor'][i] = None
    if df['floor'][i] == 'دارد':
        df['floor'][i] = 1

    if type(df['floor'][i])==float or type(df['floor'][i])==int or type(df['floor'][i])==str:
        if isnull[i] == False:
            df['floor'][i]= int(df['floor'][i])


    else:
        df['floor'][i] = None
for i in range(0,len(df['floor'])):
    if df['floor'][i] == None:
        df['floor'][i] = np.nan

<ipython-input-6-7c7fb8b8c569>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.h
copy
  df['floor'][i]= int(df['floor'][i])
```

2-2) Dealing with missed values:

Most of time we face with missed value in our data set. There are three scenarios to deal with this problem.

- Ignore the feature if the number of missed values is greater than 40 percent of whole records (in this case: 4470*0.4 = 1788).
- Uni-variate vs Multivariate imputation.
- Nearest neighbors imputation.

So the policy for this step is, counting missed values for each columns as bellow:



```
[7]
### Dealing with missed values
df.isnull().sum()
```

```
Zone                   2
Price                  0
Years                 10
Rooms                  0
Area                   0
NumberOfElevators      0
floor                361
direction           3928
dtype: int64
```

```
[8]
## As we can see we have 3928 values missed in direction column(more than half of the data) so we drop the column
df = df.loc[ : ].drop(['direction'],axis=1)
df
```

	Zone	Price	Years	Rooms	Area	NumberOfElevators	floor
0	کامرانیه	8.400	3	2	120	1	5
1	میرداماد	5.928	0	2	80	1	3
2	پاسداران	4.450	0	1	68	1	1
3	شمس آباد	6.000	0	2	120	1	2
4	ظفر	7.700	8	3	131	1	1
...	...	...	...	...	...	...	...

As we can see in picture the **direction** columns has more than 40 percent of number of data set records (3928) . Thereby, I drop the column.

For columns **Years** and **Zone** I use SimpelImputer method which is the Uni-variate algorithms for missing data imputation. The codes are represented as bellow:

```
[11]
import pandas as pd
from sklearn.impute import SimpleImputer
data = df[['Years', 'Price']]
data
imp = SimpleImputer(strategy="most_frequent")
imp = imp.fit_transform(data)
for i in range(0,len(imp)):
    df['Years'][i] = imp[i][0]
```

```
<ipython-input-11-8ddc842711d3>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pan
py
  df['Years'][i] = imp[i][0]
```

```
[12]
import pandas as pd
from sklearn.impute import SimpleImputer
data = df[['Zone', 'Price']]
data
imp = SimpleImputer(strategy="most_frequent")
imp = imp.fit_transform(data)
for i in range(0,len(imp)):
    df['Zone'][i] = imp[i][0]
```

```
<ipython-input-12-e32e8c0e1c89>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

2-3) Dealing with categorical values:

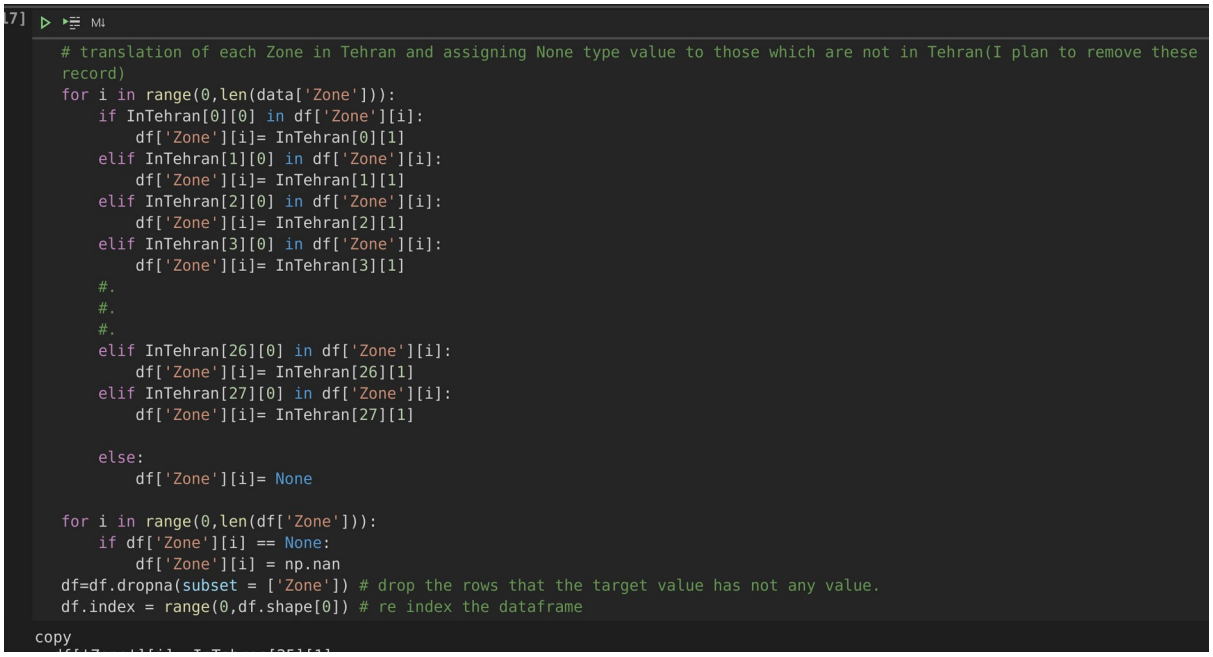
In our data set there was two categorical features. But one of them were ignored (**direction**).

The other one is Zone. But, there are a lot of Zones in our data set which are not belongs to Tehran. Such as “گلسار”. This zone is located in north of Iran(I ignore these records as follow). Thereby, I create a list of tuples which are the zones in Tehran and their translation in to English (if each record is not in this list, it is assigned as None then the row which contains None will be dropped by using dropna (because it used to have a wrong data which not belongs to Tehran city).

The List is defined as bellow:

```
# Some records for Zone column are not an Area in Tehran such as 'گلسار' so
#I create a list of district for Tehran and Also translate each Zone to English.
InTehran = [('الهیه','Elahie'),('نیاوران','Niavaran'),('جلفا','Jolfa'),('دیباجی','Dibaji'),
('دولت','Dowlat'),('تهرانپارس','Tehran_pars'),
('بوکان','Bukan'),('قلهک','Qolhak'),('جمشیدیه','Jamshidieh'),('سید خندان','Sed_Khandan'),
('اقدسیه','Aqdasieh'),('اندرزگو','Andarzgu'),('دزاشیب','Dezashib'),('نبرد','Nabard'),
('نیروی هوایی','Niru_havai'),('ونک','Vanak'),('مهران','Mehran'),('زعفرانیه','Zaferanieh'),
('اختیاریه','Ekhtiarieh'),('فرمانیه','Farmanieh'),('هروی','Heravi'),('قیطریه','Qeitarieh'),
('ولنجک','Velenjak'),('میرداماد','Mirdamad'),('کامرانیه','Kamranieh'),
('پاسداران','Pasdaran'),('ظفر','Zafar'),('شمس آباد','ShamsAbad')]
```

Translation and removing unwanted data are coded as bellow:

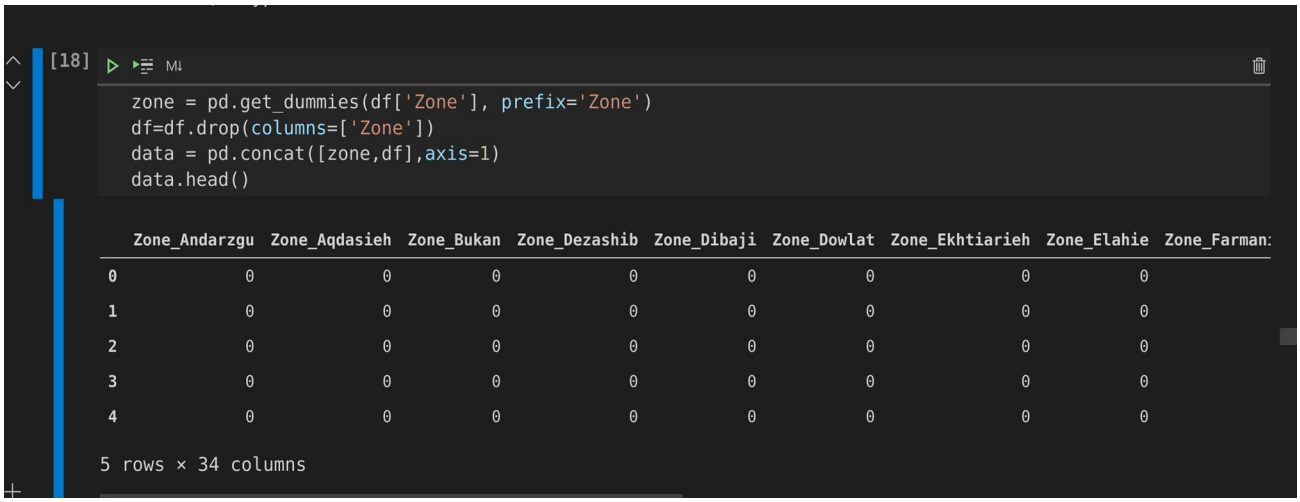


```
# translation of each Zone in Tehran and assigning None type value to those which are not in Tehran(I plan to remove these
record)
for i in range(0,len(data['Zone'])):
    if InTehran[0][0] in df['Zone'][i]:
        df['Zone'][i]= InTehran[0][1]
    elif InTehran[1][0] in df['Zone'][i]:
        df['Zone'][i]= InTehran[1][1]
    elif InTehran[2][0] in df['Zone'][i]:
        df['Zone'][i]= InTehran[2][1]
    elif InTehran[3][0] in df['Zone'][i]:
        df['Zone'][i]= InTehran[3][1]
    #.
    #.
    #.
    elif InTehran[26][0] in df['Zone'][i]:
        df['Zone'][i]= InTehran[26][1]
    elif InTehran[27][0] in df['Zone'][i]:
        df['Zone'][i]= InTehran[27][1]

    else:
        df['Zone'][i]= None

for i in range(0,len(df['Zone'])):
    if df['Zone'][i] == None:
        df['Zone'][i] = np.nan
df=df.dropna(subset = ['Zone']) # drop the rows that the target value has not any value.
df.index = range(0,df.shape[0]) # re index the dataframe
```

Notice: in above figure I remove some lines of codes and put dots in order to show my codes in a single picture. But you can see the original code in **learning.py**.

By implementing above codes the number of our data set's records is reduced to **3758**.

In Next step I use get_dummies function in pandas library to encode the Zone column (based on one-hot encoding)



```
zone = pd.get_dummies(df['Zone'], prefix='Zone')
df=df.drop(columns=['Zone'])
data = pd.concat([zone,df],axis=1)
data.head()
```

	Zone_Andarzgu	Zone_Aqdasieh	Zone_Bukan	Zone_Dezashib	Zone_Dibaji	Zone_Dowlat	Zone_Ekhtiarieh	Zone_Elahie	Zone_Farman
0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	

5 rows × 34 columns

So the shape of our data set is changed to **3758 rows * 34 columns**

2-4)Discretization of data:

Our target value should be discretized in to some bins. In this case I use **KbinsDiscretizer** method which is defined in **sklearn** library an the target value (**Price**) is discretized in to 6 bins(the range of our value is changed in to Zero to 6)

```
[19]
from sklearn.preprocessing import KBinsDiscretizer
# kbins is created
kbins = KBinsDiscretizer(n_bins=6, encode='ordinal', strategy='quantile')
# target feature is Discretized
df_y = pd.DataFrame(
    kbins.fit_transform(data[['Price']]),
    columns=['Price']
)
df_y
```

	Price
0	3.0
1	1.0
2	0.0
3	1.0
4	2.0
...	...
3753	5.0
3754	5.0
3755	5.0
3756	5.0
3757	5.0

3- Data splitting:

in this step I split the data set in to training set and test set (training: 75% of data set and test: 25% of data set).

```
[21]
# split the data frame into train set and test set
from sklearn.model_selection import train_test_split
import sklearn

x_feature=data.loc[ : , data.columns !='Price']
y_feature=data['Price']
x_train,x_test,y_train,y_test = train_test_split(x_feature,y_feature,test_size=0.25,random_state=0)
```

Model Selection, Tuning hyper parameter and Evaluation:

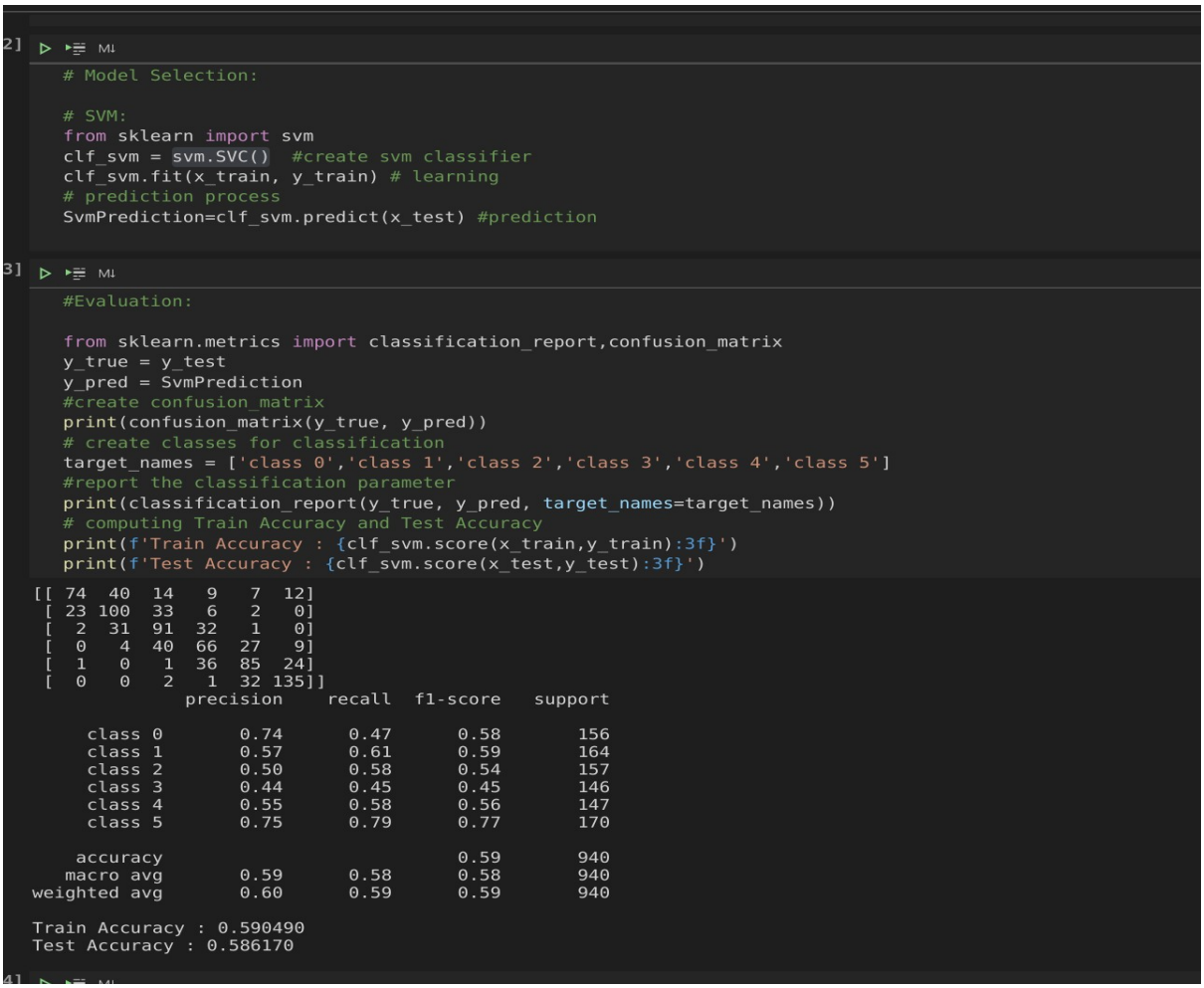
This step is a mixture of Model selection, Tuning hyper parameter and Evaluation for each Machine learning algorithms that I use in the project. The instructions for each Machine learning are the same but there are a few difference between the function I use of each algorithms.

i.e:

- SVM ===> Method I use: svm.SVC
- Ensemble (Random forest) ====> Method I use: RandomForestClassifier
- MLP classifier ======> Method I use: MLPClassifier
- Decision Tree ======> Method I use: DecisionTreeClassifier

Model selection: SVM

learning the data, predicting the target value of test and calculating precision, recall and Accuracy with **confusion matrix.** Finally, the calculated parameters were shown by **classification_report** method also the Train accuracy and Test accuracy is calculated by **ens_classifier.score** method.



```
2]
# Model Selection:

# SVM:
from sklearn import svm
clf_svm = svm.SVC()  #create svm classifier
clf_svm.fit(x_train, y_train) # learning
# prediction process
SvmPrediction=clf_svm.predict(x_test) #prediction
```

```
3]
#Evaluation:

from sklearn.metrics import classification_report,confusion_matrix
y_true = y_test
y_pred = SvmPrediction
#create confusion matrix
print(confusion_matrix(y_true, y_pred))
# create classes for classification
target_names = ['class 0','class 1','class 2','class 3','class 4','class 5']
#report the classification parameter
print(classification_report(y_true, y_pred, target_names=target_names))
# computing Train Accuracy and Test Accuracy
print(f'Train Accuracy : {clf_svm.score(x_train,y_train):3f}')
print(f'Test Accuracy : {clf_svm.score(x_test,y_test):3f}')
```

```
[[ 74  40  14   9   7  12]
 [ 23 100  33   6   2   0]
 [  2  31  91  32   1   0]
 [  0   4  40  66  27   9]
 [  1   0   1  36  85  24]
 [  0   0   2   1  32 135]]
              precision    recall  f1-score   support

     class 0       0.74      0.47      0.58       156
     class 1       0.57      0.61      0.59       164
     class 2       0.50      0.58      0.54       157
     class 3       0.44      0.45      0.45       146
     class 4       0.55      0.58      0.56       147
     class 5       0.75      0.79      0.77       170

    accuracy                           0.59       940
   macro avg       0.59      0.58      0.58       940
weighted avg       0.60      0.59      0.59       940

Train Accuracy : 0.590490
Test Accuracy : 0.586170
```

```
4]
```

As we can see the Train accuracy is calculated 59.04% and Test accuracy: 58.61%

we use the above instructions for other algorithms. The results are:

Random Forest:

Train accuracy: 53.12% Test accuracy: 52.12%

MLP:

Train accuracy: 53.12% Test accuracy: 52.12%

Decision Tree:

Train accuracy: 64.12% Test accuracy: 63.61%

Hyper Tuning:

we must improve our result through finding the best set of hyper parameters. There are two ways:

- Randomized Search CV
- Grid Search CV

In this project I used the second way to find **best_estimator_.**

In SVM:

SVC(kernel=’linear’)

In Random Forest:

RandomForestClassifier(max_depth=100, max_features='log2')

```
[29]
# Tuning Hyper-parameter (Greedy search):

from sklearn.model_selection import GridSearchCV
tuned_parameters = {
    'n_estimators':[10,100],
    'max_features':('auto', 'sqrt','log2'),
    'bootstrap':(True, False),
    'max_depth':[2,100],
}
ens_classifier=RandomForestClassifier() # create classifier
# estimate the best set of hyper-parameter
ens_cv= GridSearchCV(ens_classifier,tuned_parameters, cv=3, verbose=2,n_jobs=4)



ens_cv.fit(x_train, y_train)
ens_cv.best_estimator_

Fitting 3 folds for each of 24 candidates, totalling 72 fits

RandomForestClassifier(max_depth=100, max_features='log2')

[30]
```

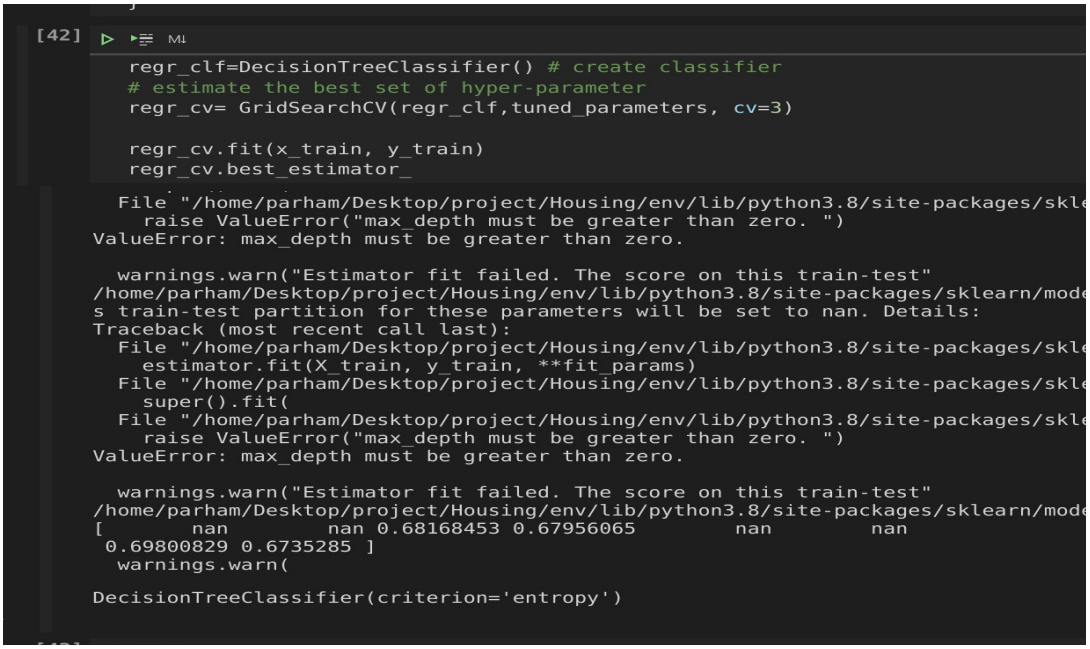
In MLP:

MLPClassifier(activation='tanh', alpha=0.05, hidden_layer_sizes=(10, 30, 10))

```
35]
# Tuning Hyper-parameter (Greedy search):
from sklearn.model_selection import GridSearchCV
tuned_parameters = {
    'hidden_layer_sizes': [(10,30,10),(20,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],

}

MLP_classifier=MLPClassifier() # create classifier
# estimate the best set of hyper-parameter
MLP_cv= GridSearchCV(MLP_classifier,tuned_parameters, cv=3)

MLP_cv.fit(x_train, y_train)
MLP_cv.best_estimator_
ization hasn't converged yet.
  warnings.warn(
/home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_perceptron.py:614: Con
ization hasn't converged yet.
  warnings.warn(
/home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_perceptron.py:614: Con
ization hasn't converged yet.
  warnings.warn(
/home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_perceptron.py:614: Con
ization hasn't converged yet.
  warnings.warn(
/home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_perceptron.py:614: Con
ization hasn't converged yet.
  warnings.warn(
/home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_perceptron.py:614: Con
ization hasn't converged yet.
  warnings.warn(
/home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_perceptron.py:614: Con
ization hasn't converged yet.
  warnings.warn(

MLPClassifier(activation='tanh', hidden_layer_sizes=(10, 30, 10),
              learning_rate='adaptive')
```

In decision Tree:

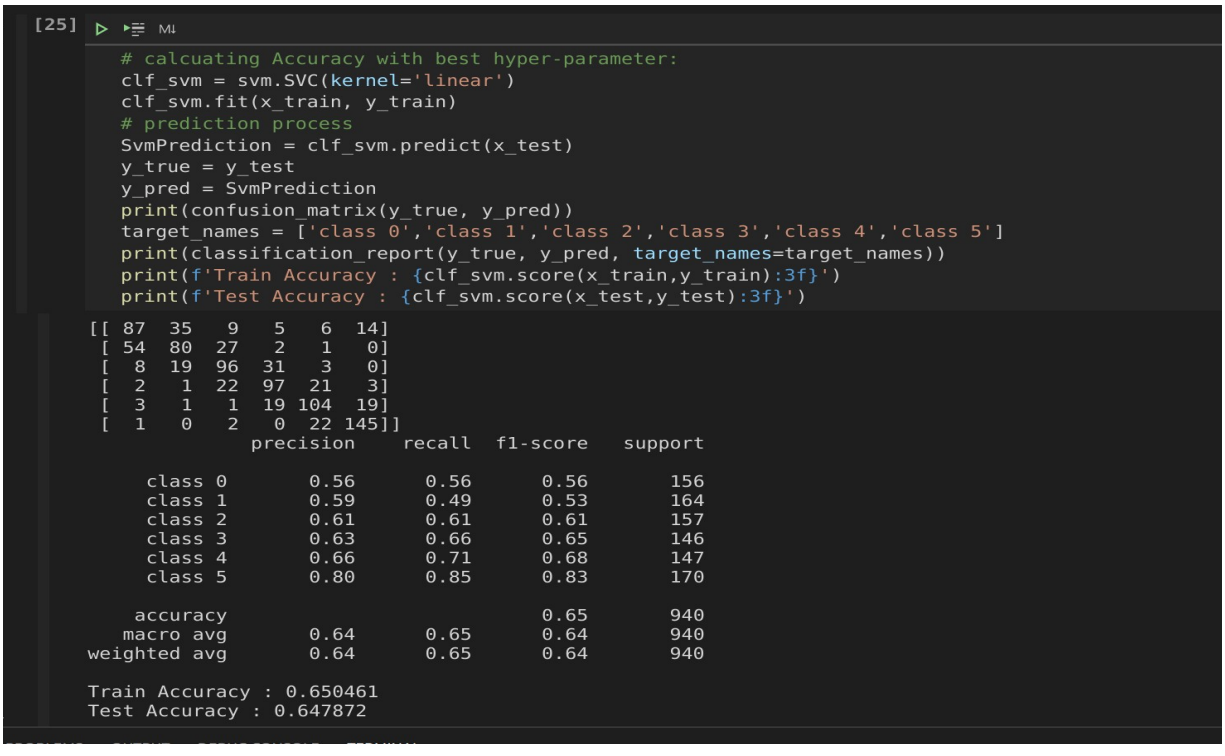
DecisionTreeClassifier(criterion='entropy')



```
[42]
    regr_clf=DecisionTreeClassifier() # create classifier
    # estimate the best set of hyper-parameter
    regr_cv= GridSearchCV(regr_clf,tuned_parameters, cv=3)

    regr_cv.fit(x_train, y_train)
    regr_cv.best_estimator_

  File "/home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages/skle
    raise ValueError("max_depth must be greater than zero. ")
ValueError: max_depth must be greater than zero.

  warnings.warn("Estimator fit failed. The score on this train-test"
/home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages/sklearn/mode
s train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "/home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages/skle
    estimator.fit(X_train, y_train, **fit_params)
  File "/home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages/skle
    super().fit(
  File "/home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages/skle
    raise ValueError("max_depth must be greater than zero. ")
ValueError: max_depth must be greater than zero.

  warnings.warn("Estimator fit failed. The score on this train-test"
/home/parham/Desktop/project/Housing/env/lib/python3.8/site-packages/sklearn/mode
[       nan        nan 0.68168453 0.67956065        nan        nan
 0.69800829 0.6735285 ]
  warnings.warn(

DecisionTreeClassifier(criterion='entropy')
```

Evaluation:

In this step I calculate I re calculate the confusion matrix. But this time with new hyper parameter which were found in previous steps for each algorithms.



```
[25]
    # calcuating Accuracy with best hyper-parameter:
    clf_svm = svm.SVC(kernel='linear')
    clf_svm.fit(x_train, y_train)
    # prediction process
    SvmPrediction = clf_svm.predict(x_test)
    y_true = y_test
    y_pred = SvmPrediction
    print(confusion_matrix(y_true, y_pred))
    target_names = ['class 0','class 1','class 2','class 3','class 4','class 5']
    print(classification_report(y_true, y_pred, target_names=target_names))
    print(f'Train Accuracy : {clf_svm.score(x_train,y_train):3f}')
    print(f'Test Accuracy : {clf_svm.score(x_test,y_test):3f}')

[[ 87  35   9   5   6  14]
 [ 54  80  27   2   1   0]
 [  8  19  96  31   3   0]
 [  2   1  22  97  21   3]
 [  3   1   1  19 104  19]
 [  1   0   2   0  22 145]]
              precision    recall  f1-score   support

     class 0       0.56      0.56      0.56       156
     class 1       0.59      0.49      0.53       164
     class 2       0.61      0.61      0.61       157
     class 3       0.63      0.66      0.65       146
     class 4       0.66      0.71      0.68       147
     class 5       0.80      0.85      0.83       170

    accuracy                           0.65       940
   macro avg       0.64      0.65      0.64       940
weighted avg       0.64      0.65      0.64       940

Train Accuracy : 0.650461
Test Accuracy : 0.647872
```

As we can see our results are improved. The results are shown as bellow(before and after tuning hyper parameter):

SVM:

before: Train accuracy: 59.04% Test accuracy: 58.61%

after : Train accuracy: 65.04% Test accuracy: 64.78%

Random Forest:

before: Train accuracy: 53.12% Test accuracy: 52.12%

after : Train accuracy: 96.91% Test accuracy: 77.12%

MLP:

before: Train accuracy: 53.12% Test accuracy: 52.12%

after : Train accuracy: 72.17% Test accuracy: 68.51%

Decision Tree:

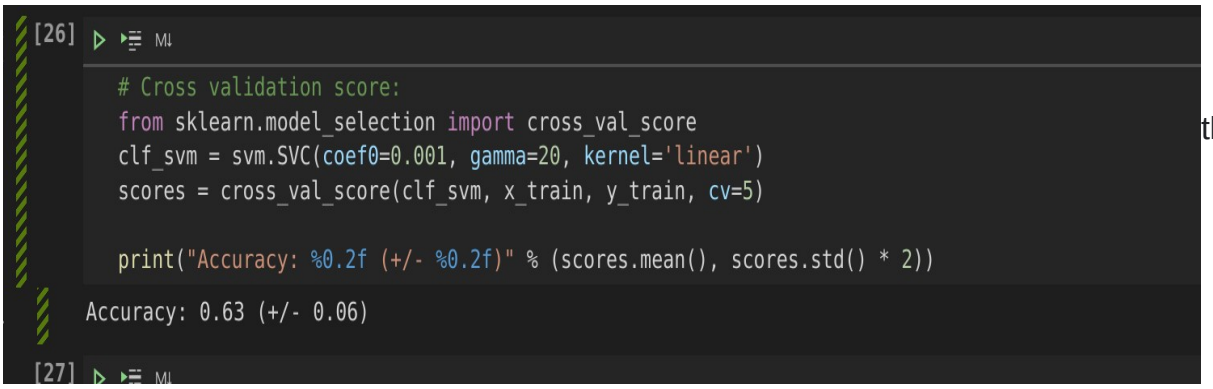
before: Train accuracy: 64.12% Test accuracy: 63.61%

after : Train accuracy: 96.91% Test accuracy: 71.06%

Assessing the analysis generalization:

In order to assessing how the results of a statistical analysis will generalize to an independent data set we use **Cross-validation**

Cross-validation for SVM:



the accuracy is 63% with 0.06 upper and lower boundaries.

The cross-validation is coded like above picture for other Models.

Random Forest: Accuracy: 0.73 (+/- 0.02)

MLP: Accuracy: 0.69 (+/- 0.02)

Decision Tree: Accuracy: 0.69 (+/- 0.01)

Conclusion:

To sum up, The house price precision is one the hottest subjects of Machine Learning tasks. In this project I try to predict the house price in Tehran which is the capital of Iran. The biggest barrier I coped with was converting data set which is crawled from Persian website in to an appropriate types of data.

The Model selection part was collected from the machine learning algorithms such as SVM,Ensemble,MLP,Decision tree. After implementing these algorithms for learning part. The evaluation part shows RandomForestClassifier(ensemble method) is the best algorithms we can taking advantage of to fit our data. The Accuracy of this model was improved up to 96.91% for Train set and 77.12% for Test set.