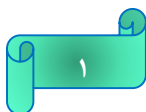


فهرست

۲	چکیده.....
۲	مقدمه و تعریف مسئله
۲	فاز اول.....
۲	خواندن داده و معرفی آن
۴	پیش‌پردازش داده
۴	feature engineering
۵	Encoding Categorical Features
۶	Imputation
۶	Outlier Detection
۸	Scaling
۸	ارائه مدل پیش‌بینی قیمت.....
۸	مرحله اول بررسی
۱۳	مرحله‌ی دوم بررسی
۱۴	پیش‌بینی داده‌های Test
۱۶	فاز دوم.....
۲۲	محاسبه‌ی فاصله
۲۴	پاسخ سیستم در صورت عدم وجود ملک مناسب



چکیده

هدف اصلی این پروژه ایجاد یک سیستم هوشمند خرید خانه است که در فاز اول سعی بر این است تا مدلی طراحی شود تا با کمترین خطای ممکن قیمت ملک را پیش‌بینی کند و در فاز دوم recommendation system طراحی گردد تا بهترین گزینه را با توجه به معیارهای کاربر به او پیشنهاد دهد.

مقدمه و تعریف مسئله

یکی از اصلی‌ترین معضلاتی که در دنیای پرشتاب امروز با آن درگیر هستیم پیش‌بینی تغییرات آینده و آماده کردن شرایط خود برای مقابله با آنها است. یکی از کاراترین ابزارهای که علم در اختیار ما قرار داده‌است تا بتوانیم در راستای حل چالش موجود قدم برداریم علم داده‌کاوی و علوم وابسته به آن است که بر اساس تجربه‌ی گذشته و الگوهای پنهان در داده‌های موجود می‌تواند ما را برای رویایی با آینده تجهیز نماید. به همین علت نمی‌توان برای این علم مرزی قائل شد و به عبارت دیگر در هر حرفه و کسب‌وکاری می‌توان از آن استفاده کرد.

چالشی که در حال حاضر با آن روبه‌رو هستیم، در وهله اول برآورد قیمت املاک با موقعیت‌های جغرافیایی و کاربری متفاوت است و سپس باید recommendation system طراحی شود تا بر اساس معیارهای مورد نظر کاربر، املاکی را به او پیشنهاد دهد. برای حل این مسئله ابتدا باید بررسی دقیقی بر روی ویژگی‌های موجود برای املاک (attributes) انجام شود و در ادامه باید داده‌ها را برای طراحی یک مدل آماده کنیم که این کار در مرحله‌ی پیش‌پردازش انجام می‌شود. سپس باید با توجه به داده‌هایی که در اختیار داریم و تنوع موجود از بین مدل‌های مناسب برای پیش‌بینی قیمت، مدل بهینه را برای برآورد انتخاب کنیم.

در ادامه نیز باید با توجه به معیارهایی که کاربر به عنوان ورودی در اختیار سیستم قرار می‌دهد، املاکی با توجه به نزدیکی قیمت و سایر معیارهای درخواستی به او نمایش داده شود. این کار کمک خواهد کرد تا کاربر با استناد به دو دیدگاه کیفیت ملک و محدودیت مالی، بهترین انتخاب را انجام دهد.

فاز اول

خواندن داده و معرفی آن

در این مرحله نگاهی اجمالی به داده‌های موجود خواهیم داشت و کتابخانه‌های مورد نیاز برای این فاز را وارد کد می‌کنیم.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import impute
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import Lasso
from sklearn import ensemble as en
from sklearn.model_selection import cross_val_score
from tensorflow import keras
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error as mae
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import IsolationForest
import seaborn as sns
import folium
```

به صورت کلی از کتابخانه‌های numpy, pandas, sklearn, seaborn استفاده شده است.

```
df_train = pd.read_csv('C:/Users/user/Downloads/Telegram Desktop/train_data.csv')
df_train.shape
df_train.describe()
df_train.dtypes
```

```
>>> df_train.describe()
count    MSSubClass    LotFrontage    ...    YrSold    SalePrice
mean      56.815068      70.04797    ...    2007.792998    181307.531202
std       42.143591      24.55054    ...      1.321132     78402.449093
min       20.000000      21.00000    ...    2006.000000     34900.000000
25%       20.000000      59.00000    ...    2007.000000    130000.000000
50%       50.000000      69.00000    ...    2008.000000    163250.000000
75%       70.000000      80.00000    ...    2009.000000    214375.000000
max      190.000000     313.00000    ...    2010.000000    745000.000000
```

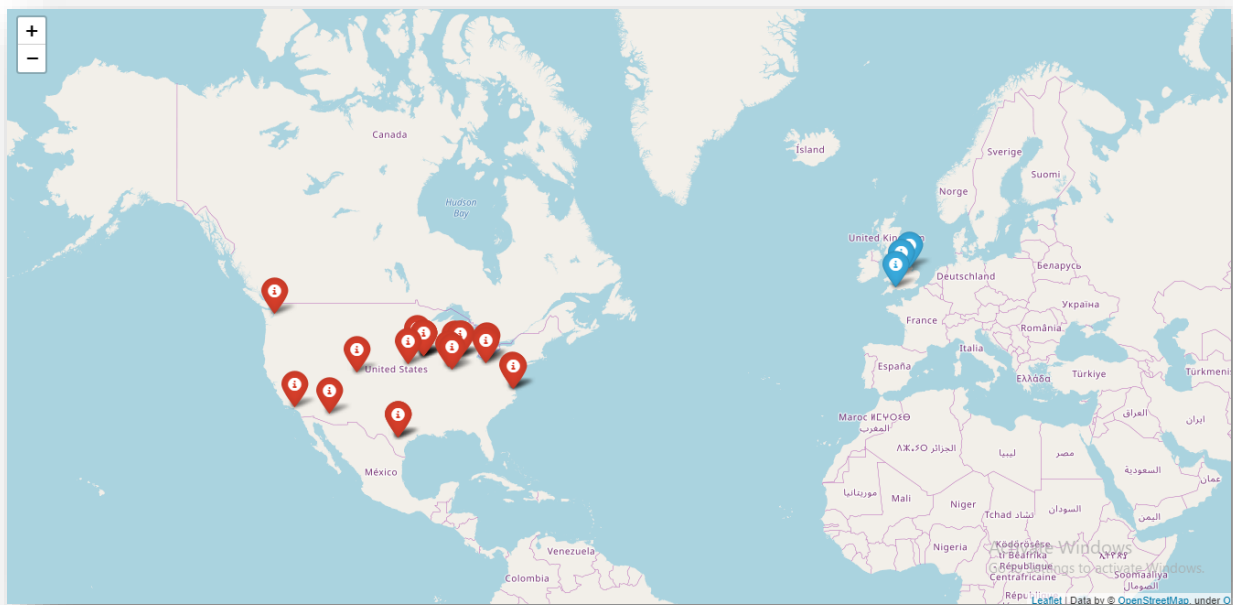
```
>>> df_train.dtypes
MSSubClass    int64
MSZoning      object
LotFrontage   float64
LotArea       int64
Street        object
Alley         object
LotShape      object
LandContour   object
```

```
>>> df_train.shape
(1314, 80)
```

همانطور که مشاهده می‌شود داده‌های موجود دارای ۸۰ attribute و ۱۳۱۴ record است. type داده‌های موجود را نیز با بررسی اولیه به دو دسته‌ی numeric و categorical می‌توان تقسیم بندی کرد.

حال به بررسی املاک موجود بر اساس موقعیت جغرافیایی آنها می‌پردازیم. در این مرحله از کتابخانه‌ی folium جهت ترسیم نقشه و تعیین موقعیت دقیق محل املاک استفاده شده است. (بخشی از کد با توجه به طولانی بودن آن ذکر شده است)

```
map=df_train.groupby('Neighborhood').count().iloc[:,0:1]
m = folium.Map(location=[41,-90],zoom_start=5)
folium.Marker(location=[40.479717,-89.033002],
    popup='Bloomington Heights',icon=folium.Icon(color='red', icon='info-sign')).add_to(m)
folium.Marker(location=[40.054218, -88.192292],
    popup='Bluestem',icon=folium.Icon(color='red', icon='info-sign')).add_to(m)
folium.Marker(location=[41.607025, -81.509672],
    popup='Briardale',icon=folium.Icon(color='red', icon='info-sign')).add_to(m)
folium.Marker(location=[41.446835, -81.702149],
    popup='Brookside',icon=folium.Icon(color='red', icon='info-sign')).add_to(m)
```



با توجه به نتیجه‌ی حاصل می‌توان از همین ابتدا پیشبینی کرد که تعداد کمی از داده‌ها را باید جهت بهبود عملکرد حذف کنیم. زیرا به صورت قابل توجهی از سایر املاک و چگالی جغرافیایی دور هستند. البته این قضیه نیازمند بررسی بیشتر است. تعداد املاک بر اساس محل جغرافیایی آنها در map ذخیره گشته‌است.

پیش‌پردازش داده feature engineering

```
df_train['ROldness'] = 2015 - df_train['YearRemodAdd']
df_train = df_train.drop(['YearRemodAdd'], axis=1)
df_train['Oldness'] = 2015 - df_train['YearBuilt']
df_train = df_train.drop(['YearBuilt'], axis=1)
df_train['GOldness'] = 2015 - df_train['GarageYrBlt']
df_train = df_train.drop(['GarageYrBlt'], axis=1)
df_train['Floor'] = df_train['2ndFlrSF'].apply(lambda x:2 if x > 0 else 1)
```

در این مرحله با توجه به ماهیت برخی از ویژگی‌ها، می‌توانیم ویژگی‌های جدیدی را برای داده ایجاد و جایگزین کنیم. در این قسمت با توجه به اینکه سال ساخت ملک، بازسازی آن و سال ساخت گاراژ را داریم می‌توانیم سن و قدمت آنها را حساب و سپس با اطلاعات پیشین جایگزین کنیم. البته شایان ذکر است با توجه به اینکه جدیدترین سال موجود در داده‌ها ۲۰۱۰ است، سال معیار برای محاسبه‌ی سن‌ها، ۲۰۱۵ در نظر گرفته شده‌است.

Encoding Categorical Features

در مرحله‌ی بعدی پیش‌پردازش داده باید تمامی داده‌ها را به عدد تبدیل کنیم. با بررسی‌های انجام شده می‌توان به این نتیجه رسید که برخی از **categorical** دارای داده‌های **ordinal** و برخی دارای داده‌های **nominal** هستند. در نتیجه باید با روش‌های جداگانه‌ای با آنها برخورد شود و به عدد تبدیل کرد.

با بررسی دقیق‌تر داده‌های **ordinal** می‌توان دریافت که برخی از کلید واژه‌ها در آنها مشترک است و می‌توان آنها را در کل داده‌ها جایگزین کرد. برای مثال می‌توان به عبارت‌های لیست **check** اشاره کرد که در تعداد قابل توجهی از **attribute**ها وجود دارد و به صورت **ordinal** نیز هستند. برخی از **attribute**ها نیز استثنا هستند و باید مورد به مورد تبدیل و جایگزینی برای آنها انجام شود که به شرح زیر قابل مشاهده است.

```
check = ['Po','Fa','TA','Gd','Ex']

i_count = np.arange(0, df_train.shape[0])
j_count = np.arange(0, df_train.shape[1])
k_count = np.arange(0, 5)

header = []
for i in i_count:
    for j in j_count:
        for k in k_count:
            if df_train.iloc[i,j] == check[k]:
                header.append([df_train.columns[j]])
            continue
header = pd.DataFrame(header)
header = pd.unique(header[0])

df_train['BsmtExposure'] = df_train['BsmtExposure'].replace(['No','Mn','Av','Gd'],[1,2,3,4])
df_train = df_train.replace(['Po','Fa','TA','Gd','Ex'],[1,2,3,4,5])

i_count = np.arange(0, header.shape[0])
for i in i_count:
    df_train[header[i]] = df_train[header[i]].replace(np.nan,0)

df_train = df_train.replace(['MnWw','GdWo','MnPrv','GdPrv'],[1,2,3,4])
df_train['Fence'] = df_train['Fence'].replace(np.nan,0)

df_train['GarageFinish'] = df_train['GarageFinish'].replace(['Unf','RFn','Fin'],[1,2,3])
df_train['GarageFinish'] = df_train['GarageFinish'].replace(np.nan,0)

df_train = df_train.replace(['Unf','LwQ','Rec','BLQ','ALQ','GLQ'],[1,2,3,4,5,6])
df_train['BsmtFinType1'] = df_train['BsmtFinType1'].replace(np.nan,0)
df_train['BsmtFinType2'] = df_train['BsmtFinType2'].replace(np.nan,0)
```

حال باید به بررسی و جایگزینی داده‌های **nominal** بپردازیم. به همین منظور ابتدا باید آنها را پیدا کنیم و سپس به ازای هر داده‌ی منحصر به فرد با توجه به ستون آن، یک ستون و **attribute** جدید تعریف کنیم که به صورت ۰ و ۱ عمل کند. به این معنا که در صورت وجود داشتن عدد ۱ و در صورت عدم وجود ۰ را نمایش دهد. این قست با کمک تابع **get_dummies** انجام شده‌است.

```
label = []
for y in df_train.columns:
    if(df_train[y].dtype == np.object):
        label.append(y)

label = pd.DataFrame(label)

df_train = pd.concat([df_train, pd.get_dummies(df_train[label[0]],prefix=label[0]), axis=1)
df_train = df_train.drop(label[0], axis=1)
```

Imputation

برای مقابله با داده‌های miss ابتدا باید دید کلی نسبت به شرایط و تعداد آنها داشته باشیم.

```
df_count_isnaR = pd.DataFrame(df_train.isna().sum(axis=1))
df_count_isnaR['percentage']=round(df_count_isnaR[0]/df_train.shape[1],3)
df_count_isnaR= df_count_isnaR.sort_values(0,ascending=False)

df_count_isnaA = pd.DataFrame(df_train.isna().sum())
df_count_isnaA['percentage']=round(df_count_isnaA[0]/df_train.shape[0],3)
df_count_isnaA= df_count_isnaA.sort_values(0,ascending=False)
```

	0	percentage
471	2	0.00800
1028	2	0.00800
929	2	0.00800
347	2	0.00800
1266	2	0.00800
208	2	0.00800
274	2	0.00800
257	2	0.00800
331	2	0.00800
1024	1	0.00400

جدول ۱

	0	percentage
LotFrontage	230	0.17500
GOldness	73	0.05600
MasVnrArea	7	0.00500
Exterior2nd_AsbShng	0	0.00000
Exterior1st_BrkFace	0	0.00000
Exterior1st_CBlock	0	0.00000

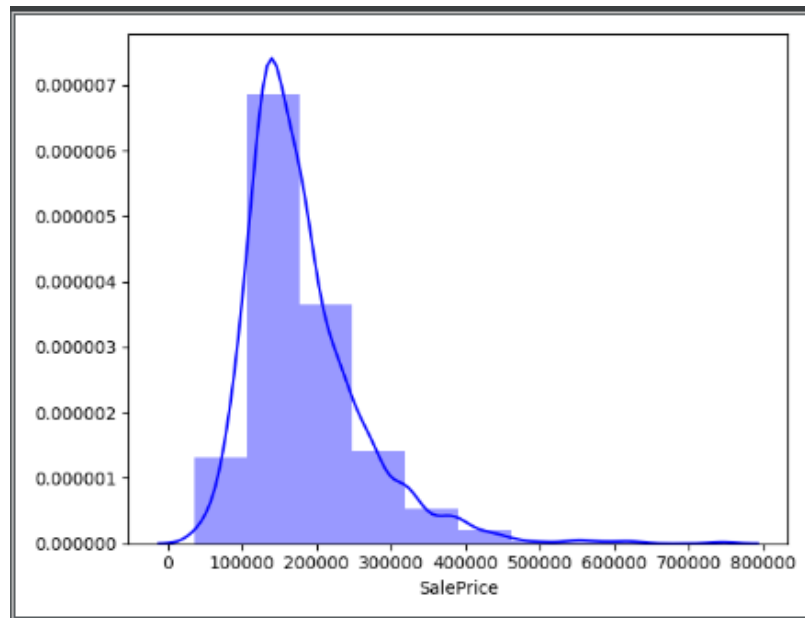
جدول ۲

همانطور که مشاهده می‌شود بیشترین مقدار missing برای ستون «LotFrontage» است که در ۱۷,۵ درصد از داده‌ها نیاز به تغییرات دارد. بیشترین مقدار missing به ازای سطرهای موجود نیز ۲ است، که این امر هم بیانگر این است که به صورت کلی داده‌های مورد بررسی در این قسمت آنچنان مشکل‌آفرین نیستند. به همین منظور برای impute داده‌ها از تابع «impyute» که عملکردی مشابه با تابع mice در R را دارد استفاده شده‌است.

```
complete_data = impyute.mice(df_train)
complete_data.columns = df_train.columns
```

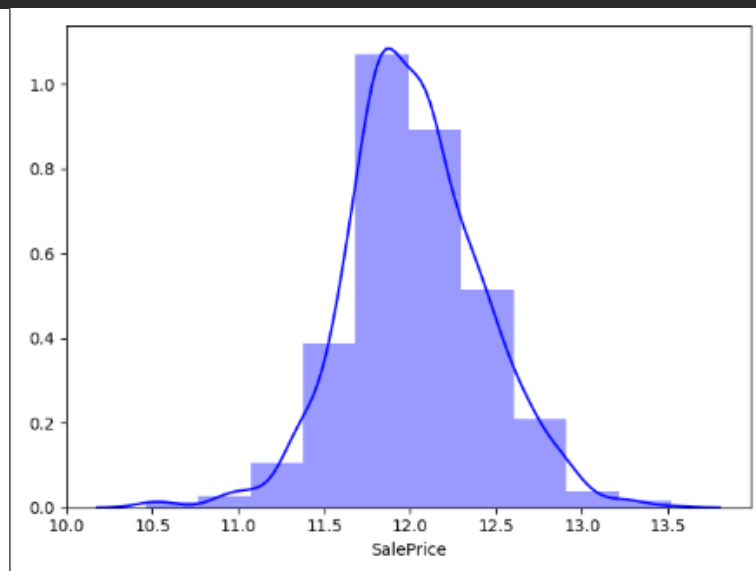
Outlier Detection

در اولین گام پیش از بررسی داده‌های پرت، توزیع attribute هدف را بررسی کنیم. به صورت قابل توجهی می‌توان چولگی این توزیع را مشاهده کرد که این امر در صورت وجود داده‌های پرت می‌تواند دقت نهایی مدل را کاهش دهد. به این منظور با استفاده از تابع لگاریتم این چولگی را از بین می‌بریم.



نمودار ۱

```
sns.distplot(df_train['SalePrice'], bins=10, kde=True,color='b')
plt.show()
```



نمودار ۲

```
sns.distplot(df_train['SalePrice'], bins=10, kde=True,color='b')
plt.show()
```

حال به حذف داده‌های پرت می‌پردازیم. برای انجام این کار از تابع پیشرفته‌ی IsolationForest استفاده می‌کنیم که مبنای کار آن الگوریتم Random Forest است. داده‌هایی با متغیر ۱- مشخص شده‌اند، پرت هستند و آنها را حذف می‌کنیم.

```
complete_data.SalePrice = np.log(complete_data.SalePrice)
```

```
clf = IsolationForest(max_samples=100)
clf.fit(complete_data)
y=clf.predict(complete_data)
list(y).count(-1)
complete_data = complete_data[np.where(y == 1, True, False)]
```

Scaling

آخرین اقدامی که با بررسی‌های انجام شده باید در مرحله‌ی پیش‌پردازش انجام شود، یکسان‌سازی دامنه‌ی داده‌ها است. به همین منظور تمامی داده‌ها را به بازه‌ی ۰ الی ۱ منتقل می‌کنیم. البته بررسی نهایی برای مدل Ridge که بهترین مدل است، این را نشان داد که یکسان‌سازی دامنه داده‌ها دقت را کاهش می‌دهد. به همین دلیل این عملیات در نهایت بر روی داده‌ها اعمال نشده‌است.

```
complete_data.SalePrice = np.log(complete_data.SalePrice)
y_train = complete_data.SalePrice
x_train = complete_data.drop(['SalePrice'], axis = 1)

test=complete_data.columns
scaler = MinMaxScaler(copy=True, feature_range=(0, 1))
scaler.fit(complete_data)
complete_data = scaler.transform(complete_data)
complete_data = pd.DataFrame(complete_data)
complete_data.columns = test
```

اقداماتی دیگری نیز مانند feature selectin، PCA، correlation analysis و موارد مشابه نیز بر روی داده‌ها انجام شد اما بهبودی در عملکرد مدل‌های نهایی نداشتند. به همین منظور از توضیح این موارد در بخش پیش‌پردازش اجتناب شده‌است.

ارائه مدل پیش‌بینی قیمت

این بخش از پروژه در دو مرحله انجام شده‌است. در مرحله اول به صورت اجمالی تعدادی از مدل‌ها بررسی شده‌اند و با استفاده از شاخص «mean absolute error» دو مدل برای بررسی بیشتر انتخاب شده‌است. در مرحله‌ی دوم این دو مدل با بررسی دقیق‌تر و با استفاده از شاخص RMSE مقایسه شده‌اند و مدل نهایی انتخاب شده‌است. لازم به ذکر است که در مرحله‌ی اول پارامترهای بهینه با استفاده از cross validation محاسبه می‌شود.

مرحله اول بررسی

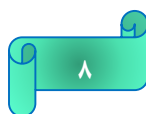
در ابتدا متغیر هدف و پیش‌بینی کننده را جدا می‌کنیم و سپس داده‌ها را به دو قسمت test و train تقسیم می‌کنیم.

```
y_train = complete_data.SalePrice
x_train = complete_data.drop(['SalePrice'], axis = 1)

X_train, X_test, Y_train, Y_test = train_test_split(x_train, y_train, test_size=0.2)
```

Ridge ➤

```
score_RR = []
for i in np.arange(800,2000,200):
    for k in np.arange(0.1,1.5,0.1):
        clf_1 = Ridge(alpha=k,normalize=True,solver='svd',max_iter=i)
        clf_1.fit(X_train, Y_train)
        score_RR.append([i,k, np.mean(cross_val_score(clf_1, X_train, Y_train,scoring='neg_mean_absolute_error',cv=10))])
```




```
score_RR = pd.DataFrame(score_RR)
score_RR = score_RR.sort_values(by = 2 ,ascending=False)
```

	0	1	2
71	1800	0.20000	-0.07672
57	1600	0.20000	-0.07672
43	1400	0.20000	-0.07672
1	800	0.20000	-0.07672
29	1200	0.20000	-0.07672
15	1000	0.20000	-0.07672
16	1000	0.30000	-0.07694

جدول ۳

همانطور که مشاهده می‌شود پارامترهای بهینه‌ی این مدل مطابق جدول بالا می‌باشد. حال باید با داده‌های test مدل را ارزیابی کنیم.

```
clf_1 = Ridge(alpha=0.2, normalize=True, solver='svd', max_iter=1800)
clf_1.fit(X_train, Y_train)
mae(Y_test, clf_1.predict(X_test))
```

```
>>> mae(Y_test, clf_1.predict(X_test))
0.07215835335337893
```

GradientBoosting ➤

```
score_GBR = []
for i in np.arange(1,6):
    for j in np.arange(0.1, 1.3, 0.1):
        for k in np.arange(50,500,50):
            clf_2 = en.GradientBoostingRegressor(loss='huber',n_estimators=k,learning_rate=j,max_depth=i)
            clf_2.fit(X_train, Y_train)
            score_GBR.append([i,j,k, np.mean(cross_val_score(clf_2, X_train, Y_train,scoring='neg_mean_absolute_error',cv=10))])
```

```
score_GBR = pd.DataFrame(score_GBR)
score_GBR = score_GBR.sort_values(by = 3 ,ascending=False)
```

	0	1	2	3
122	2	0.20000	300	-0.07497
223	3	0.10000	400	-0.07501
221	3	0.10000	300	-0.07516
17	1	0.20000	450	-0.07518
116	2	0.10000	450	-0.07527
224	3	0.10000	450	-0.07533

جدول ۴



همانطور که مشاهده می‌شود پارامترهای بهینه‌ی این مدل مطابق جدول بالا می‌باشد. حال باید با داده‌های test مدل را ارزیابی کنیم.

```
clf_2 = en.GradientBoostingRegressor(loss='huber', n_estimators=300, learning_rate=0.2, max_depth=2)
clf_2.fit(X_train, Y_train)
mae(Y_test, clf_2.predict(X_test))
```

```
>>> mae(Y_test, clf_2.predict(X_test))
0.07097927132158816
```

RandomForest ➤

```
score_RFR = []
for i in np.arange(1, 25):
    for k in np.arange(50, 300, 50):
        clf_3 = en.RandomForestRegressor(oob_score=True, max_features='log2', n_estimators=k, max_depth=i)
        clf_3.fit(X_train, Y_train)
        score_RFR.append([i, k, np.mean(cross_val_score(clf_3, X_train, Y_train, scoring='neg_mean_absolute_error', cv=10))])

score_RFR = pd.DataFrame(score_RFR)
score_RFR = score_RFR.sort_values(by=2, ascending=False)
```

	0	1	2
183	21	200	-0.09126
159	18	350	-0.09131
210	24	200	-0.09132
193	22	250	-0.09146
196	22	400	-0.09149
177	20	350	-0.09152
184	21	250	-0.09167

جدول ۵

همانطور که مشاهده می‌شود پارامترهای بهینه‌ی این مدل مطابق جدول بالا می‌باشد. حال باید با داده‌های test مدل را ارزیابی کنیم.

```
clf_3 = en.RandomForestRegressor(oob_score=True, max_features='log2', n_estimators=200, max_depth=21)
clf_3.fit(X_train, Y_train)
mae(Y_test, clf_3.predict(X_test))
```

```
>>> mae(Y_test, clf_3.predict(X_test))
0.10257657264997816
```

AdaBoost ➤

```
score_ABR = []
for j in np.arange(0.1, 1.3, 0.1):
```



```
for k in np.arange(50,400,50):
    clf_4 = en.AdaBoostRegressor(n_estimators=k,learning_rate=j)
    clf_4.fit(X_train, Y_train)
    score_ABR.append([j,k, np.mean(cross_val_score(clf_4, X_train, Y_train,scoring='neg_mean_absolute_error',cv=10))])

score_ABR = pd.DataFrame(score_ABR)
score_ABR = score_ABR.sort_values(by = 2 ,ascending=False)
```

	0	1	2
8	0.20000	100	-0.11149
82	1.20000	300	-0.11156
72	1.10000	150	-0.11161
83	1.20000	350	-0.11177
22	0.40000	100	-0.11180
30	0.50000	150	-0.11185

جدول ۶

همانطور که مشاهده می شود پارامترهای بهینه ی این مدل مطابق جدول بالا می باشد. حال باید با داده های test مدل را ارزیابی کنیم.

```
clf_4 = en.AdaBoostRegressor(n_estimators=100, learning_rate=0.2)
clf_4.fit(X_train, Y_train)
mae(Y_test,clf_4.predict(X_test))
```

```
>>> mae(Y_test,clf_4.predict(X_test))
0.11959827386319843
```

ElasticNet ➤

```
score_ELR = []
for i in np.arange(0.1,2, 0.1):
    for j in np.arange(0.1, 2, 0.1):
        for k in np.arange(200,2000,200):
            clf_5 = ElasticNet(alpha=i, l1_ratio=j,max_iter=k, normalize=True)
            clf_5.fit(X_train,Y_train)
            score_ELR.append([i,j,k, np.mean(cross_val_score(clf_5, X_train, Y_train,scoring='neg_mean_absolute_error',cv=10))])

score_ELR = pd.DataFrame(score_ELR)
score_ELR = score_ELR.sort_values(by = 3 ,ascending=False)
```

	0	1	2	3
0	0.10000	0.10000	200	-0.29263
2	0.10000	0.10000	600	-0.29263
3	0.10000	0.10000	800	-0.29263
4	0.10000	0.10000	1000	-0.29263
5	0.10000	0.10000	1200	-0.29263
6	0.10000	0.10000	1400	-0.29263

جدول ۷



همانطور که مشاهده می‌شود پارامترهای بهینه‌ی این مدل مطابق جدول بالا می‌باشد. حال باید با داده‌های test مدل را ارزیابی

کنیم.

```
clf_5 = ElasticNet(alpha=0.1, l1_ratio=0.1, max_iter=200, normalize=True)
clf_5.fit(X_train, Y_train)
mae(Y_test, clf_5.predict(X_test))
```

```
>>> mae(Y_test, clf_5.predict(X_test))
0.2954637194492807
```

Lasso ➤

```
score_LasR = []
for i in np.arange(0.1, 2, 0.1):
    for k in np.arange(100, 2000, 100):
        clf_6 = Lasso(alpha=i, normalize=True, max_iter=k)
        clf_6.fit(X_train, Y_train)
        score_LasR.append([i, k, np.mean(cross_val_score(clf_6, X_train, Y_train, scoring='neg_mean_absolute_error', cv=10))])

score_LasR = pd.DataFrame(score_LasR)
score_LasR = score_LasR.sort_values(by=2, ascending=False)
```

	0	1	2
0	0.10000	100	-0.29280
248	1.40000	200	-0.29280
246	1.30000	1900	-0.29280
245	1.30000	1800	-0.29280
244	1.30000	1700	-0.29280
243	1.30000	1600	-0.29280

جدول ۸

همانطور که مشاهده می‌شود پارامترهای بهینه‌ی این مدل مطابق جدول بالا می‌باشد. حال باید با داده‌های test مدل را ارزیابی

کنیم.

```
clf_6 = Lasso(alpha=0.1, normalize=True, max_iter=100)
clf_6.fit(X_train, Y_train)
mae(Y_test, clf_6.predict(X_test))
```

```
>>> mae(Y_test, clf_6.predict(X_test))
0.2954917313959888
```

در پایان این مرحله باید دو مدل را جهت بررسی بیشتر در مرحله‌ی دوم انتخاب کنیم. نتیجه‌ی بررسی‌های مرحله‌ی اول به

طور خلاصه به شرح زیر می‌باشد. در نتیجه دو مدل GradientBoosting و Ridge برای بررسی بیشتر انتخاب می‌شود.



ردیف	نام مدل	MAE	CV Result
۱	GradientBoosting	۰.۰۷۰۹۷	۰.۰۷۴۹۷
۲	Ridge	۰.۰۷۲۲۲	۰.۰۷۶۷۲
۳	RandomForest	۰.۱۰۲۵۷	۰.۰۹۱۲۶
۴	AdaBoost	۰.۱۱۹۵۹	۰.۱۱۱۴۹
۵	ElasticNet	۰.۲۹۵۴۳	۰.۲۹۲۶۳
۶	Lasso	۰.۲۹۲۸۰	۰.۲۹۵۴۹

جدول ۹

مرحله‌ی دوم بررسی

همانطور که پیش از این نیز اشاره شد در این مرحله دو مدل نهایی را با شاخص RMSE ارزیابی می‌کنیم. این شاخص نسبت انحراف مدل به انحراف در صورت عدم وجود مدل است. در نتیجه هرچه این عبارت کوچکتر باشد، کارایی و اثربخشی آن بهتر است.

در ابتدا مدل GradientBoosting را بررسی می‌کنیم. به این صورت که مدل را با استفاده از پارامترهایی که در مرحله‌ی قبل به دست آمد، ۱۰۰۰ بار بر داده‌ها اعمال می‌کنیم و میانگین RMSE را برای این ۱۰۰۰ بار محاسبه می‌کنیم.

```
from sklearn.metrics import mean_squared_error as mse
final_score_G=[]
for i in np.arange(1,1000,1):
    X_train, X_test, Y_train, Y_test = train_test_split(x_train, y_train, test_size=0.2)
    clf_2 = en.GradientBoostingRegressor(loss='huber', n_estimators=450, learning_rate=0.1, max_depth=2)
    clf_2.fit(X_train, Y_train)
    final_score_G.append([i,np.sqrt(mse(Y_test,clf_2.predict(X_test)))/np.sqrt(np.mean((Y_test - np.mean(Y_test))**2))])

final_score_G=pd.DataFrame(final_score_G)
np.mean(final_score_G)
```

```
>>> np.mean(final_score_G)
0    500.00000
1     0.30477
dtype: float64
```

سپس مدل Ridge را بررسی می‌کنیم:

```
final_score_R=[]
for i in np.arange(1,1000,1):
    X_train, X_test, Y_train, Y_test = train_test_split(x_train, y_train, test_size=0.2)
    clf_1 = Ridge(alpha=0.2, normalize=True, solver='svd', max_iter=1800)
    clf_1.fit(X_train, Y_train)
    final_score_R.append([i,np.sqrt(mse(Y_test,clf_1.predict(X_test)))/np.sqrt(np.mean((Y_test - np.mean(Y_test))**2))])

final_score_R=pd.DataFrame(final_score_R)
np.mean(final_score_R)
```

```
>>> np.mean(final_score_R)
0      500.000000
1       0.298616
dtype: float64
```

با نتایج به دست آمده می‌توان این مورد را گفت که مدل Ridge با پارامترهای به دست آمده بیشترین کارایی را دارد و به عنوان مدل نهایی انتخاب می‌شود.

```
Ridge(alpha=0.2, normalize=True, solver='svd', max_iter=1800)
```

پیشینه داده‌های Test

```
datas_test = pd.read_csv('C:/Users/user/Downloads/Telegram Desktop/test_data.csv')

datas_test['ROldness'] = 2015 - datas_test['YearRemodAdd']
datas_test = datas_test.drop(['YearRemodAdd'], axis=1)
datas_test['Oldness'] = 2015 - datas_test['YearBuilt']
datas_test = datas_test.drop(['YearBuilt'], axis=1)
datas_test['GarageYrBlt']=datas_test['GarageYrBlt'].replace('Hamed',1915)
datas_test['GarageYrBlt']=datas_test['GarageYrBlt'].astype('float64')
datas_test['GOldness'] = 2015 - datas_test['GarageYrBlt']
datas_test = datas_test.drop(['GarageYrBlt'], axis=1)
datas_test['Floor'] = datas_test['2ndFlrSF'].apply(lambda x:2 if x > 0 else 1)
```

در ابتدا باید تغییراتی را در داده‌های test ایجاد کنیم. بعد از بررسی داده‌های test، می‌توان دید که برخی از داده‌ها با عبارت NA پر شده‌اند و این باعث خطا در خواندن اطلاعات در پایتون می‌شود. باید به این نکته توجه کرد که NA در دیتاست موجود به معنای داده‌ای از دست رفته نیست و به معنای عدم وجود آن ویژگی در آن ملک است. به همین دلیل تمامی NAها را با عبارت «Hamed» جایگزین کردیم تا مشکلی پیش نیاید.

دومین مورد تغییر هم مربوط به attribute «GarageYrBlt» است. برخی از ملک‌ها «Garage» ندارند و به همین دلیل نمی‌توان سن را برای آن حساب کرد. به همین دلیل چون رابطه‌ی سن گاراژ و قیمت ملک عکس است، شرایطی را ایجاد می‌کنیم که عدم وجود گاراژ برای ملک یک فاکتور منفی به حساب بیاید. به همین علت مواردی که گاراژ ندارند را معادل با گاراژ با قدمت بسیار زیاد می‌گیریم.

```
df_count_isnaRT = pd.DataFrame(datas_test.isna().sum(axis=1))
df_count_isnaRT['percentage']=round(df_count_isnaRT[0]/datas_test.shape[1],3)
df_count_isnaRT= df_count_isnaRT.sort_values(0,ascending=False)

df_count_isnaAT = pd.DataFrame(datas_test.isna().sum())
df_count_isnaAT['percentage']=round(df_count_isnaAT[0]/datas_test.shape[0],3)
df_count_isnaAT= df_count_isnaAT.sort_values(0,ascending=False)

i_count_test = np.arange(0, datas_test.shape[0])
j_count_test = np.arange(0, datas_test.shape[1])
```

```

k_count_test = np.arange(0, 5)

header_T=[]
check = ['Po','Fa','TA','Gd','Ex']

for i in i_count_test:
    for j in j_count_test:
        for k in k_count_test:
            if datas_test.iloc[i,j] == check[k]:
                header_T.append([datas_test.columns[j]])
            continue
header_T = pd.DataFrame(header_T)
header_T = pd.unique(header_T[0])

datas_test['BsmtExposure']= datas_test['BsmtExposure'].replace(['Hamed','No','Mn','Av','Gd'],[0,1,2,3,4])
datas_test= datas_test.replace(['Hamed','Po','Fa','TA','Gd','Ex'],[0,1,2,3,4,5])

datas_test= datas_test.replace(['Hamed','MnWw','GdWo','MnPrv','GdPrv'],[0,1,2,3,4])
datas_test['GarageFinish']= datas_test['GarageFinish'].replace(['Hamed','Unf','Rfn','Fin'],[0,1,2,3])
datas_test= datas_test.replace(['Hamed','Unf','LwQ','Rec','BLQ','ALQ','GLQ'],[0,1,2,3,4,5,6])

float = ['MasVnrArea','BsmtFullBath','GarageArea','BsmtFinSF1','LotFrontage',
        'BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','GarageCars','BsmtHalfBath']

for i in np.arange(0,10,1):
    datas_test[float[i]] = datas_test[float[i]].replace('Hamed',0)
    datas_test[float[i]] = datas_test[float[i]].astype("float64")

label_t = []
for y in datas_test.columns:
    if(datas_test[y].dtype == np.object):
        label_t.append(y)
label_t = pd.DataFrame(label_t)

datas_test = pd.concat([datas_test, pd.get_dummies(datas_test[label_t[0]],prefix=label_t[0]), axis=1)
datas_test = datas_test.drop(label_t[0], axis=1)

complete_data_T = impute.mice(datas_test)
complete_data_T.columns = datas_test.columns

```

حال باید بررسی کنیم که attribute های موجود در دو دیتاست آیا یکسان هستند و یا خیر، در صورت وجود مغایرت نیز باید تغییراتی را اعمال کنیم. اگر ستونی نیز وجود نداشت، آن را ایجاد و به ازای تمامی سطرها صفر قرار دهیم.

```

temp=list(set(list(complete_data_T.columns))-set(list(x_train.columns)))
complete_data_T = complete_data_T.drop(temp,axis=1)

temp=list(set(list(x_train.columns))-set(list(complete_data_T.columns)))
for i in temp:
    complete_data_T[i]=[0]*1605

```

برای اینکه بتوانیم از مدلی که ساخته ایم استفاده کنیم باید attribute ها را مطابق داده های train مرتب کنیم. به همین منظور به صورت زیر عمل می کنیم:

```
complete_data_T = complete_data_T[x_train.columns]
```

سپس مدل را می‌سازیم و پیش‌بینی را انجام می‌دهیم:

```
clf_1 = Ridge(alpha=0.2, normalize=True, solver='svd', max_iter=1800)
clf_1.fit(x_train, y_train)
```

```
complete_data_T['SalePrice'] = clf_1.predict(complete_data_T)
```

در پایان هم ستون PricePrediction را ایجاد و به دیتاست اضافه می‌کنیم. این داده‌ی نهایی را تحت عنوان «test_data» ذخیره می‌کنیم. البته شایان ذکر است که ستون ذکر شده به دیتاستی که تغییر نکرده‌است اضافه شده‌است.

```
datas_test = pd.read_csv('C:/Users/user/Downloads/Telegram Desktop/test_data (2).csv')
datas_test['PricePrediction'] = np.exp(complete_data_T['SalePrice'])
datas_test.to_csv('C:/Users/user/Desktop/test_data')
```

```
describe=datas_test['PricePrediction'].describe()
```

count	1605.00000
mean	177533.75183
std	82498.40903
min	46915.34613
25%	126288.46478
50%	156070.33920
75%	207785.41852
max	1503273.12347

فایل کامل کد فاز اول در پیوست تحت عنوان «First Phase Code» آورده شده‌است.

فاز دوم

برای انجام این فاز که طراحی یک recommendation system است، کتابخانه‌های مورد استفاده شده در این فاز به شرح

زیر است:

```
import numpy as np
import pandas as pd
import scipy
from scipy.spatial import distance
from sklearn.preprocessing import MinMaxScaler
import easygui as eg
from easygui import *
```

در مرحله‌ی بعدی، پس از خواندن داده‌ها، پایگاه داده df را تشکیل می‌دهیم که دربر دارنده‌ی ستون‌های قدمت ساختمان، تعداد طبقات، تعداد اتاق خواب، قیمت، کاربری و منطقه‌ی داده‌های موجود است. علت انتخاب این ستون‌ها، هماهنگی داده‌های موجود با داده‌هایی است که کاربر وارد می‌کند.

```
df_train = pd.read_csv('C:/Users/user/Downloads/Telegram Desktop/train_data.csv')
```

```
df = pd.DataFrame()
df['Oldness'] = 2015 - df_train['YearRemodAdd']
df['Floor'] = df_train['2ndFlrSF'].apply(lambda x: 2 if x > 0 else 1)
```



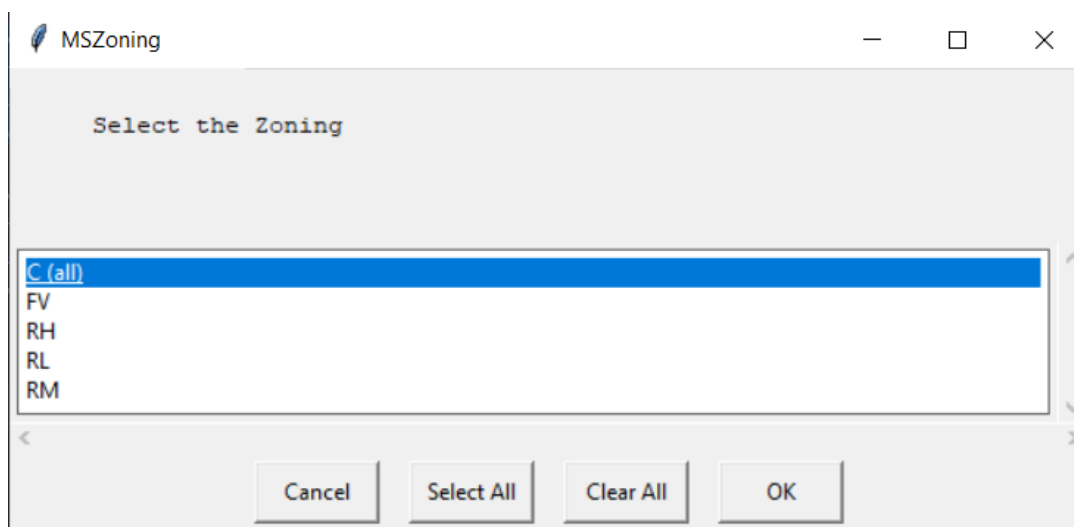
```
df['Area'] = df_train['LotArea']*0.092903
df['Bedrooms'] = df_train['BedroomAbvGr']
df['Price'] = df_train['SalePrice']
df['MSZoning'] = df_train['MSZoning']
df['District'] = df_train['Neighborhood']
```

با توجه به این که منطقه و نوع کاربری که مدنظر مشتری است از اهمیت بالایی برخوردار است و برای مثال در صورتی که مشتری ملک مسکونی احتیاج داشته باشد و به او زمین کشاورزی هم پیشنهاد شود، چندان منطقی نیست به همین منظور ابتدا داده‌های موجود مطابق با مقداری که مشتری برای منطقه و کاربری وارد می‌کند، فیلتر می‌شوند.

برای دریافت داده‌های MSZoning و District از کاربر به صورت زیر، از کتابخانه‌ی easygui استفاده شده‌است. در این حالت گزینه‌های موجود به مشتری نشان داده می‌شود و می‌تواند از بین آنها انتخاب کند.

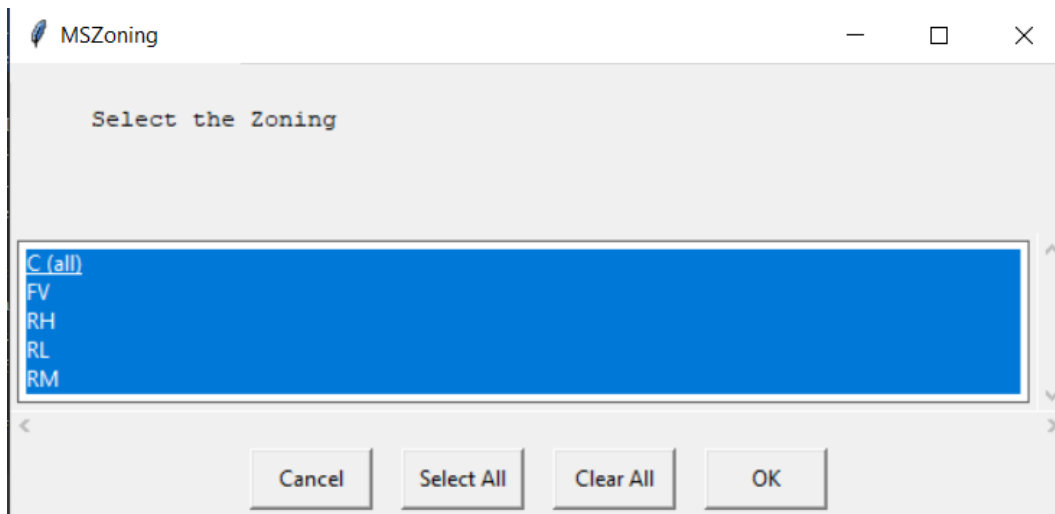
در این قسمت، مشتری کاربری زمین مورد نظر خود را وارد می‌کند.

```
question1 = "Select the Zoning"
title1 = "MSZoning"
listOfOptions1 = np.unique(df_train['MSZoning'])
choice1 = eg.multichoicebox(question1, title1, listOfOptions1)
```



تصویر ۲

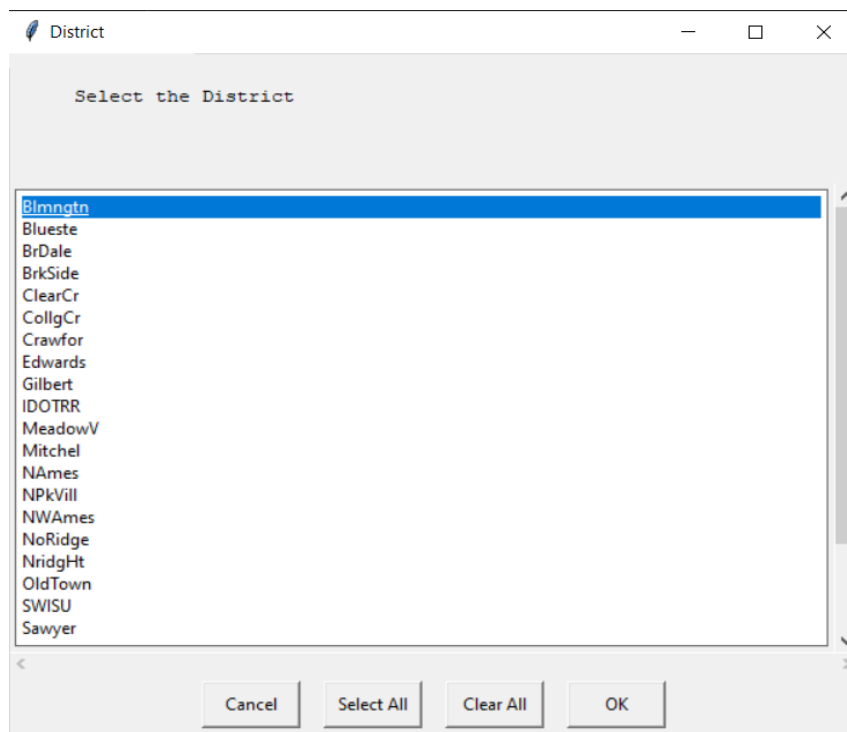
برای مثال اگر مشتری کاربری را به این صورت تعریف کند که زمین کشاورزی نباشد، همه‌ی گزینه‌ها به جز کشاورزی را به صورت زیر انتخاب می‌کنیم:



تصویر ۳

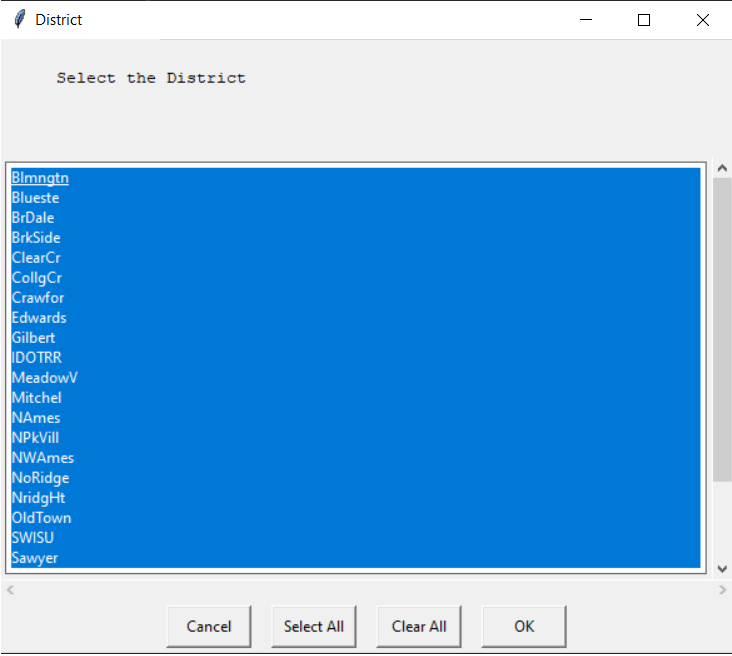
(در داده‌های train اکسل، اطلاعات زمین‌های کشاورزی وجود نداشت و به همین علت در گزینه‌های بالا قابل مشاهده نیست).
در این قسمت مشتری منطقه‌ی مدنظر خود را وارد می‌کند.

```
question2 = "Select the District"
title2 = "District"
listOfOptions2 = np.unique(df_train['Neighborhood'])
choice2 = eg.multichoicebox(question2, title2, listOfOptions2)
```



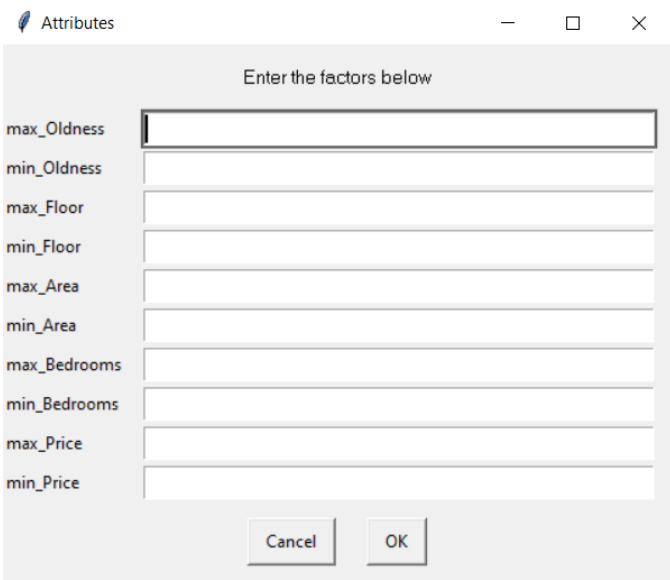
تصویر ۴

اگر منطقه برای شخص بی اهمیت باشد، ورودی به صورت زیر خواهد بود:



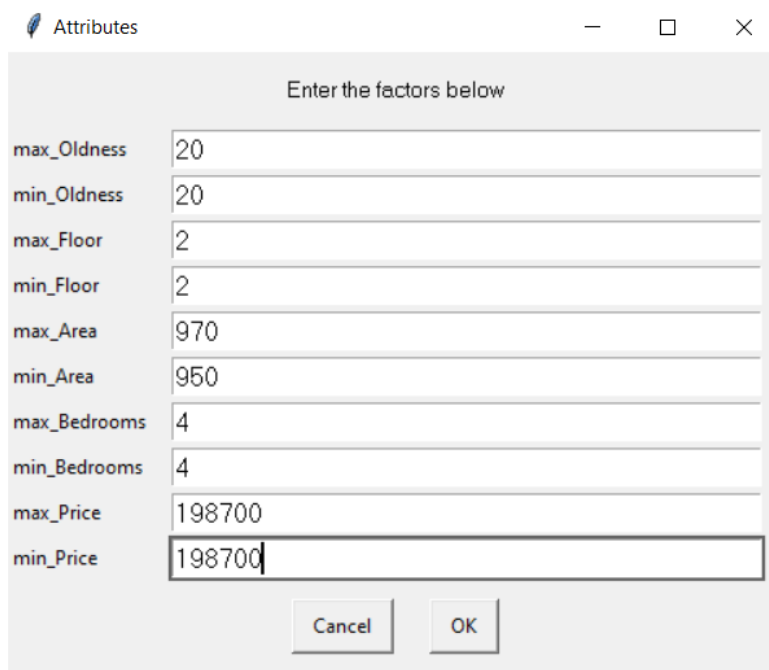
تصویر ۵

در ادامه، کاربر می‌تواند بازه‌ای که برای سایر مشخصات در نظر دارد را با وارد کردن ماکسیمم و مینیمم آن تعیین کند. به این منظور از multenterbox استفاده شده است. لازم به ذکر است که کاربر باید تمام فیلدها را پر کند و در غیر این صورت با خطا مواجه خواهد شد.



تصویر ۶

اطلاعات ورودی در description به صورت زیر است. لازم به ذکر است اگر موردی برای شخص بی اهمیت بود، عبارت «Ni» را وارد می‌کند.



Enter the factors below	
max_Oldness	20
min_Oldness	20
max_Floor	2
min_Floor	2
max_Area	970
min_Area	950
max_Bedrooms	4
min_Bedrooms	4
max_Price	198700
min_Price	198700

تصویر ۷

```
msg = "Enter the factors below"
title = "Attributes"
fieldNames = ["max_Oldness", "min_Oldness", "max_Floor", "min_Floor", "max_Area", "min_Area",
              "max_Bedrooms", "min_Bedrooms", "max_Price", "min_Price"]
fieldValues = [] # we start with blanks for the values
fieldValues = multenterbox(msg, title, fieldNames)
# make sure that none of the fields was left blank
while 1:
    if fieldValues == None: break
    errmsg = ""
    for i in range(len(fieldNames)):
        if fieldValues[i].strip() == "":
            errmsg = errmsg + ("%s" is a required field.\n\n' % fieldNames[i])
    if errmsg == "": break # no problems found
    fieldValues = multenterbox(errmsg, title, fieldNames, fieldValues)

max_min=[]
for i in np.arange(0,5,1):
    max_min.append(int(np.max(df.iloc[:,i:i+1])))
    max_min.append(int(np.min(df.iloc[:,i:i+1])))

dis_record=[]
for i in np.arange(0,10,2):
    if fieldValues[i]=='Ni':
        fieldValues[i]=max_min[i]
    if fieldValues[i+1]=='Ni':
```

```
fieldValues[i+1]=max_min[i+1]
dis_record.append((int(fieldValues[i])+int(fieldValues[i+1]))/2)
```

در قسمت max_min[] مقدار maximum و minimum هر attribute محاسبه می‌شود. اگر مشتری در فیلدی مقدار Ni را وارد کند، یعنی مقدار آن attribute برایش بی‌اهمیت بوده‌است بنابراین مقدار max یا min با توجه به شرایط برای او انتخاب می‌شود. برای مثال اگر برای ویژگی age، مقدار بیشینه Ni وارد شود، به جای آن مقدار بیشینه‌ی این ویژگی در داده‌های موجود، جایگزین می‌شود.

```
>>> max_min
[65, 5, 2, 1, 19996, 120, 8, 0, 745000, 34900]
```

در ادامه نیاز داریم که فاصله‌ی مقادیری که کاربر وارد کرده است با دیتاهای موجود در دیتافریم محاسبه شود تا بتوان کمترین فاصله را پیدا و گزینه‌های مناسب را به مشتری پیشنهاد کرد. با توجه به این که کاربر لزوماً یک عدد مشخص را وارد نکرده و بازه‌ای را انتخاب کرده‌است، میانگین این بازه به عنوان ورودی او در نظر گرفته می‌شود که در قسمت dis_record[] قابل مشاهده‌است.

ورودی مشتری به صورت زیر خواهد بود:

```
>>> dis_record
[10.0, 2.0, 965.0, 4.0, 198700.0]
```

```
df_dis=df[(df['MSZoning'].isin(choice1))&(df['District'].isin(choice2))]
```

در این قسمت داده‌ها مطابق با خواست مشتری بر اساس منطقه و کاربری وارد شده، فیلتر و در df_dis ریخته می‌شوند.

Oldness	Floor	Area	Bedrooms	Price	MSZoning	District
12	2	785.03035	3	208500	RL	CollgCr
39	1	891.86880	3	181500	RL	Veenker
13	2	1045.15875	3	223500	RL	CollgCr
45	2	887.22365	3	140000	RL	Crawfor
15	2	1324.79678	4	250000	RL	NoRidge
20	2	1311.32585	1	143000	RL	Mitchel
10	1	936.83385	3	307000	RL	Somerst
65	2	568.56636	2	129900	RM	OldTown
65	1	689.34026	2	118000	RL	BrkSide
9	2	1107.77537	4	345000	RL	NridgHt
53	1	1204.76610	2	144000	RL	Sawyer
8	1	989.60276	3	279500	RL	CollgCr

جدول ۱۰

همان‌طور که انتظار می‌رفت و در داده‌ی df_dis نیز قابل مشاهده است، نیاز است داده‌های موجود scale شوند و تأثیر برابری داشته باشند. برای این کار از تابع MinMaxScaler استفاده شده‌است.

```
df_dis=df[(df['MSZoning'].isin(choice1))&(df['District'].isin(choice2))]
```

```
test=df_dis.iloc[:,5].columns
scaler = MinMaxScaler(copy=True, feature_range=(0, 1))
scaler.fit(df_dis.iloc[:,5])
df_dis.iloc[:,5] = scaler.transform(df_dis.iloc[:,5])
```

```
df_dis.iloc[:,5] = pd.DataFrame(df_dis.iloc[:,5])
df_dis.iloc[:,5].columns = test
```

ورودی مشتری نیز به صورت زیر scale می‌شود:

```
dis_record = scaler.transform(np.array(dis_record).reshape(1,-1))
```

0	1	2	3	4
0.0833333333333333	1.0	0.042222807025801584	0.5	0.2306717363751584

محاسبه فاصله

حال باید فاصله‌ی مواردی که مشتری وارد کرده است با دیتاهای موجود محاسبه شود. فاصله‌ها به صورت ستونی به نام Dis به df_dis اضافه می‌شوند. سپس فاصله‌های به دست آمده را به ترتیب صعودی مرتب می‌کنیم تا گزینه‌هایی که فاصله‌ی نزدیک‌تری به خواسته‌های مشتری دارند مشخص شوند. برای محاسبه‌ی فاصله از کتاب‌خانه‌ی spacy استفاده شده‌است.

```
dis_num= []
for i in np.arange(0,df_dis.shape[0],1):
    dis_num.append(scipy.spatial.distance.cdist(dis_record,df_dis.iloc[i+1,:5], 'euclidean'))

df_dis['Dis']=dis_num
df_dis=df_dis.sort_values(by='Dis',ascending=True)
```

مشتری در این مرحله می‌تواند گزینه‌های پیش روی خود را ببیند. همچنین تفاوت قیمت ملک پیشنهادی با سرمایه‌ی او در ستون Different_Budget مشخص است. اگر نیاز باشد که سرمایه‌ی خود را افزایش دهد در قسمت Fund Raise پیام Yes و در غیر این صورت No را می‌بیند. در صورتی به شخص پیشنهاد افزایش سرمایه داده خواهد شد که اختلاف قیمت ملک پیشنهاد داده شده و سرمایه‌ی مد نظر او از ۵۰۰۰ واحد بیشتر باشد؛ در واقع ملک بیشتر از ۵۰۰۰ واحد از سرمایه‌ی شخص گرانتر است.

```
df_dis.iloc[:,5] = scaler.inverse_transform(df_dis.iloc[:,5])
df_dis.iloc[:,5] = pd.DataFrame(df_dis.iloc[:,5])
df_dis.iloc[:,5].columns = test
df_dis['Diferent_Budget']=(df_dis['Price']-scaler.inverse_transform(dis_record)[0][4])
df_dis['Fund Raise']=df_dis['Diferent_Budget'].apply(lambda x: 'Yes' if x > 5000 else 'No')
```

نتیجه:

	Oldness	Floor	Area	Bedrooms	Price	MSZoning	District	Dis	Diferent_Budget	Fund Raise
1110	10.00000	2.00000	1151.34688	4.00000	195000.00000	RL	CollgCr	[[0.01072604]]	-3700.00000	No
1016	10.00000	2.00000	1095.88379	4.00000	215000.00000	RL	Gilbert	[[0.02388036]]	16300.00000	Yes
172	11.00000	2.00000	694.17122	4.00000	184000.00000	RL	NAMES	[[0.02986612]]	-14700.00000	No
330	11.00000	2.00000	761.80460	4.00000	219500.00000	RL	CollgCr	[[0.03521775]]	20800.00000	Yes
321	12.00000	2.00000	678.28480	4.00000	198500.00000	RL	Edwards	[[0.03632181]]	-200.00000	No
1074	9.00000	2.00000	1305.65876	4.00000	219210.00000	RL	Gilbert	[[0.03749358]]	20510.00000	Yes
865	12.00000	2.00000	1051.29035	4.00000	214900.00000	RL	Timber	[[0.0406254]]	16200.00000	Yes
1100	8.00000	2.00000	1356.19799	4.00000	214000.00000	RL	Somerst	[[0.04430266]]	15300.00000	Yes
1087	10.00000	2.00000	1127.28500	4.00000	164000.00000	RL	Gilbert	[[0.04954377]]	-34700.00000	No
709	11.00000	2.00000	942.22223	4.00000	233000.00000	RL	SawyerW	[[0.05111043]]	34300.00000	Yes
197	13.00000	2.00000	749.54140	4.00000	200000.00000	RL	Gilbert	[[0.05119432]]	1300.00000	No
301	13.00000	2.00000	1318.38647	4.00000	202900.00000	RL	Timber	[[0.05339561]]	4200.00000	No
1287	10.00000	2.00000	2037.36279	4.00000	192140.00000	RL	Gilbert	[[0.05473749]]	-6560.00000	No

جدول ۱۱



در ادامه باید پیشنهادات سیستم به مشتری را مندسی کنیم تا بتوانیم بهترین خروجی را برای کاربر نمایش دهیم. به همین منظور خروجی‌های متفاوتی برای کاربر در نظر گرفته شده‌است که بر اساس قیمت ملک و اختلاف سرمایه او با این قیمت این پیشنهادات مدیریت شده‌اند. در ذیل این پیشنهادات برای ورودی حال حاضر مشخص شده‌است.

```
Result=df_dis.head(3).drop(['Dis','Diferent_Budget'],axis=1)
Result_abs=df_dis[abs(df_dis['Diferent_Budget'])<=5000].head(3).drop(['Dis','Diferent_Budget'],axis=1)
Result_above=df_dis[df_dis['Diferent_Budget']>=5000].head(3).drop(['Dis','Diferent_Budget'],axis=1)
Result_less=df_dis[df_dis['Diferent_Budget']<=-5000].head(3).drop(['Dis','Diferent_Budget'],axis=1)
```

Result:

۳ ملک با کمترین اختلاف از ورودی‌های شخص بدون در نظر گرفتن محدودیت سرمایه:

	Oldness	Floor	Area	Bedrooms	Price	MSZoning	District	Fund Raise
1110	10.00000	2.00000	1151.34688	4.00000	195000.00000	RL	CollgCr	No
1016	10.00000	2.00000	1095.88379	4.00000	215000.00000	RL	Gilbert	Yes
172	11.00000	2.00000	694.17122	4.00000	184000.00000	RL	NAmes	No

جدول ۱۲

Result above:

۳ ملک با کمترین اختلاف از ورودی‌های شخص به صورتی که قیمت ملک نسبت به سرمایه‌ی شخص بیش از ۵۰۰۰ واحد بیشتر باشد:

	Oldness	Floor	Area	Bedrooms	Price	MSZoning	District	Fund Raise
1016	10.00000	2.00000	1095.88379	4.00000	215000.00000	RL	Gilbert	Yes
330	11.00000	2.00000	761.80460	4.00000	219500.00000	RL	CollgCr	Yes
1074	9.00000	2.00000	1305.65876	4.00000	219210.00000	RL	Gilbert	Yes

جدول ۱۳

Result abs:

۳ ملک با کمترین اختلاف از ورودی‌های شخص با در نظر گرفتن این که سرمایه‌ی شخص با قیمت ملک نهایتاً ۵۰۰۰ واحد تفاوت داشته باشد:

	Oldness	Floor	Area	Bedrooms	Price	MSZoning	District	Fund Raise
1110	10.00000	2.00000	1151.34688	4.00000	195000.00000	RL	CollgCr	No
321	12.00000	2.00000	678.28480	4.00000	198500.00000	RL	Edwards	No
197	13.00000	2.00000	749.54140	4.00000	200000.00000	RL	Gilbert	No

جدول ۱۴

Result less:

۳ ملک با کمترین اختلاف از ورودی‌های شخص به صورتی که سرمایه‌ی شخص نسبت به قیمت ملک بیش از ۵۰۰۰ واحد بیشتر باشد :





	Oldness	Floor	Area	Bedrooms	Price	MSZoning	District	Fund Raise
172	11.00000	2.00000	694.17122	4.00000	184000.00000	RL	NAmes	No
1087	10.00000	2.00000	1127.28500	4.00000	164000.00000	RL	Gilbert	No
1287	10.00000	2.00000	2037.36279	4.00000	192140.00000	RL	Gilbert	No

جدول ۱۵

پاسخ سیستم در صورت عدم وجود ملک مناسب

حال فرض کنیم اگر کاربر اطلاعاتی را وارد کند که بر اساس نوع کاربرد آن ملک و منطقه آن، گزینه‌ای وجود نداشته باشد که به او نمایش دهیم. به عبارت دیگر در صورتی که بعد از فیلتر کردن داده‌ها بر اساس منطقه و کاربری، نتیجه‌ای دریافت نکنیم، مشابه عملیات بالا را بر روی داده‌هایی که منطقه و کاربری در آن لحاظ نشده است اعمال می‌کنیم.

```
if Result.empty:

    test = df.iloc[:, :5].columns
    scaler = MinMaxScaler(copy=True, feature_range=(0, 1))
    scaler.fit(df.iloc[:, :5])
    df.iloc[:, :5] = scaler.transform(df.iloc[:, :5])
    df.iloc[:, :5] = pd.DataFrame(df.iloc[:, :5])
    df.iloc[:, :5].columns = test

    dis_record = scaler.transform(np.array(dis_record).reshape(1, -1))

    dis_num = []
    for i in np.arange(0, df.shape[0], 1):
        dis_num.append(scipy.spatial.distance.cdist(dis_record, df.iloc[i:i + 1, :5], 'euclidean'))

    df['Dis'] = dis_num
    df = df.sort_values(by='Dis', ascending=True)

    df.iloc[:, :5] = scaler.inverse_transform(df.iloc[:, :5])
    df.iloc[:, :5] = pd.DataFrame(df.iloc[:, :5])
    df.iloc[:, :5].columns = test

    df['Diferent_Budget'] = (df['Price'] - scaler.inverse_transform(dis_record)[0][4])
    df['Fund Raise'] = df['Diferent_Budget'].apply(lambda x: 'Yes' if x > 5000 else 'No')

    Result = df.head(3).drop(['Dis', 'Diferent_Budget'], axis=1)
    Result_abs = df[abs(df['Diferent_Budget']) <= 5000].head(3).drop(['Dis', 'Diferent_Budget'], axis=1)
    Result_above = df[df['Diferent_Budget'] >= 5000].head(3).drop(['Dis', 'Diferent_Budget'], axis=1)
    Result_less = df[df['Diferent_Budget'] <= -5000].head(3).drop(['Dis', 'Diferent_Budget'], axis=1)
```



Attributes

Enter the factors below

max_Oldness	12
min_Oldness	0
max_Floor	2
min_Floor	0
max_Area	900
min_Area	700
max_Bedrooms	2
min_Bedrooms	1
max_Price	180000
min_Price	0

Cancel OK

برای مثال اگر ورودی کاربر به شرح روبه‌رو باشد با شرایط زیر برخورد خواهد کرد:

```
>>> df_dis.empty
True
```

حال بعد از عدم اعمال فیلتر نتایج به شرح زیر خواهد بود.

	Oldness	Floor	Area	Bedrooms	Price	MSZoning	District	Fund Raise
861	7.00000	1.00000	669.55192	2.00000	116500.00000	RL	BrkSide	Yes
474	8.00000	1.00000	668.90160	2.00000	107500.00000	RL	NAmes	Yes
1012	7.00000	1.00000	873.75271	2.00000	118000.00000	RL	Edwards	Yes

جدول ۱۶

فایل کامل کد فاز دوم در پیوست تحت عنوان «Second Phase Code» آورده شده‌است.