PROJECT REPORT

Group Number : 8

Member Name	Enrollment Number
Pooja Allampallewar	20114071
Paritosh Kabra	20114066
Richa	20114081
Ishu Gupta	20114041

Problem

Determining the rent price of the house is very important nowadays as people might need to know the price of the house in a locality to take a decision. Thus there is a need of a simple technique to predict the house price. The predicted price of house helps the buyer to compare the prices of the house as per their demands like number of bedrooms, locality, city, etc. The right price of the house helps the customer to elect the house and go for the bidding of that house.

Problem parameters

There are several factors that affect the price of the house such as:

- Seller type
- Number of bedrooms
- Layout(BHK/RK)
- Property type
- Locality
- Area
- Furnish type
- Number of Bathrooms
- City
- Address

With these preferences of the user, we can build a model which eases the task of the user in electing the house, by predicting the price of the house in the locality.

Data analysis

Original Data source:
https://www.kaggle.com/datasets/saisaathvik/house-rent-prices-of-metropolitan-cities-in-india
(CSV for the modified data with x,y,z coordinates and filtering is submitted in the zip.)

	Unnamed: 0	seller_type	bedroom	layout_type	property_type	locality	price	area	furnish_type	bathroom	city	address	lat_lng	
0	0	OWNER	2.0	BHK	Apartment	Bodakdev	20000.0	1450.0	Furnished	2.0	Ahmedabad	Bodakdev, Ahmedabad	["23° 2' 40.53156" N", "72° 31' 2.43840" E"]	40.5
1	1	OWNER	1.0	RK	Studio Apartment	CG Road	7350.0	210.0	Semi-Furnished	1.0	Ahmedabad	CG Road, Ahmedabad	["23° 1' 33.64752" N", "72° 33' 24.04548" E"]	33.6
2	2	OWNER	3.0	BHK	Apartment	Jodhpur	22000.0	1900.0	Unfurnished	3.0	Ahmedabad	Jodhpur, Ahmedabad	["23° 1' 0.93648" N", "72° 31' 13.55412" E"]	0.9
3	3	OWNER	2.0	BHK	Independent House	Sanand	13000.0	1285.0	Semi-Furnished	2.0	Ahmedabad	Sanand, Ahmedabad	["23° 1' 25.99536" N", "72° 23' 6.53100" E"]	25.9
4	4	OWNER	2.0	BHK	Independent House	Navrangpura	18000.0	1600.0	Furnished	2.0	Ahmedabad	Navrangpura, Ahmedabad	["23° 2' 9.59928" N", "72° 33' 51.63444" E"]	9.5

The data set contains two types of data values:

1. **Numerical data**: The parameters such as bedroom, area, bathroom are numerical and these are needed for mapping the user's preferences.
2. **Categorical data:** The parameters such seller_type, layout_type, property_type, locality, furnish_type, city, address are categorical and need some preprocessing before it can be used in our model.

   Apart from these there are several irrelevant fields which need to be dropped before working with the data.

Methodology

Preprocessing

- *Preparing the training and the test data*

  We partition the data to get the training data on which the model is trained. This training data is 90% of the original dataset.

- *Encoding the categorical data*

  The dataset contains the different types of variables, the models used internally need the data in numerical format instead of categorical. So, to solve this issue and pass the data to our model we encoded the categorical data to numerical.

  There were two options to do this:
  - One Hot Encoder
  - Label Encoding

  The model uses One Hot Encoding.

  The motivation to use this is since after label encoding, we might confuse our model into thinking that a column has data with some kind of order or hierarchy when we clearly don't have it. To avoid this, we 'OneHotEncode' that column.

  What one hot encoding does is, it takes a column which has categorical data, which has been label encoded and then splits the

```
from sklearn.preprocessing import OneHotEncoder

def prepareDataForLinReg(cluster):
    cluster = cluster.reset_index()
    st_df = cluster.drop(["lat","lng", "lat_lng", "index","locality", "Unnamed: 0",'seller_type','layout_type','property_typ
    enc = OneHotEncoder(handle_unknown='ignore')
    en_frame = enc.fit_transform(cluster[['seller_type','layout_type','property_type','furnish_type']])
    column_name = enc.get_feature_names_out(['seller_type','layout_type','property_type','furnish_type'])
    en_df = pd.DataFrame.sparse.from_spmatrix(en_frame, columns=column_name)
    en_final_cluster = pd.concat([en_df,st_df], axis=1)
    return en_final_cluster
```

  column into multiple columns. The numbers are replaced by 1s and 0s, depending on which column has what value.

- *Clustering the data set*

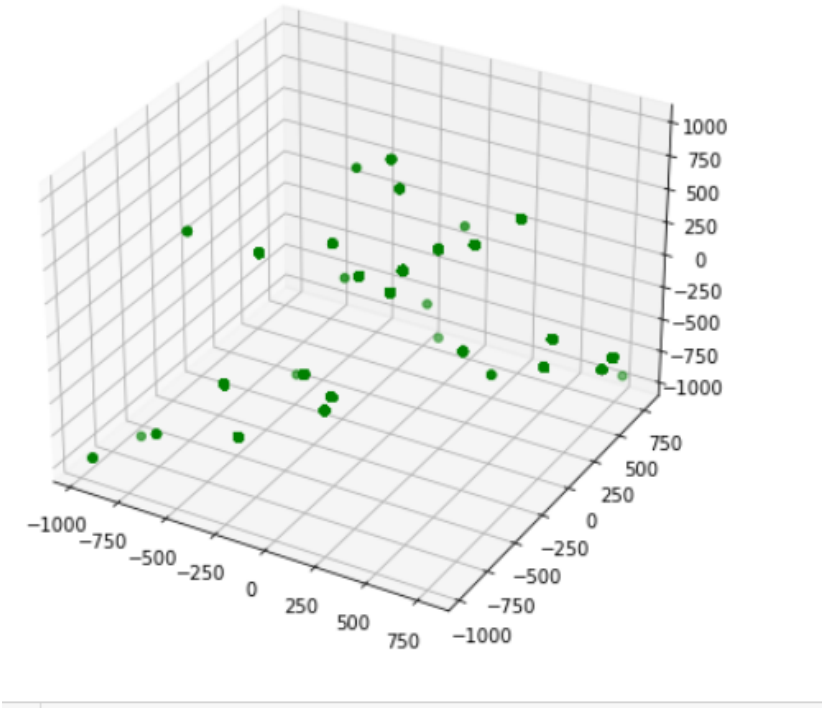
The dataset contains data for several cities and localities. We train dthe dataset separately for different cities since the price would be varying for the city type. We can cluster the data based on the localities, in this way the areas with nearby locality would be in the same cluster hence the prediction would be more close to accurate. We also trained the data without clustering to compare the result.

In order to cluster the data appropriately:

- The localities are converted to their longitudes and latitudes. For this we used *GeoCoding API.* This returns the latitudes and longitudes of the locality entered.

```
In [3]:
from opencage.geocoder import OpenCageGeocode

In [7]:
def get_lat_lng(place, geocoder):
    query = place
    results = geocoder.geocode(query)
    lat = results[0]['annotations']['DMS']['lat']
    lng = results[0]['annotations']['DMS']['lng']
    return [lat, lng]

In [8]:
key1 = '7712da193a1a4b57a398a6d398dffec5'
geocoder1 = OpenCageGeocode(key1)

In [9]:
get_lat_lng("Ravet, Pune", geocoder1)
Out[9]:
["18° 38' 35.76840'' N", "73° 44' 42.20052'' E"]

In [ ]:
```

- Next the latitudes and longitudes are converted to their equivalent x, y, z coordinates which are used for clustering the data using **k-means clustering.**



Scatter Plot For the 3-D coordinates obtained from (lat, long)

Feature Engineering

- Add new variable promising feature based on knowledge of the domain like the x,y,z coordinates.
- Remove trivial variables of very low prediction value.
- Adjust variables as required to ensure that their values or types are fit for regression purposes

Algorithms

The project utilizes two models for finding the model with the best prediction results.

- **Multiple Linear Regression**: The price varies based on the dependency of the price on multiple variables such as number of bedrooms, layout type, property type etc. Hence, we trained the model to recognise the linear dependency of the price on the variables obtained using one hot encoding and other numerical variables using multiple linear regression.
- **Tensorflow with keras:** ANNs gather their knowledge by detecting the patterns and relationships in data and learn (or are trained) through

experience, not from programming. The behavior of a neural network is determined by the transfer functions of its neurons, by the learning rule, and by the architecture itself. The weights are the adjustable parameters and, in that sense, a neural network is a parameterized system.

We used Keras Sequential model with two densely connected hidden layers, and an output layer that returns a single, continuous value.

In this case, we picked 64 hidden units for the first and second layer of our model. We are setting the amount of freedom that we are allowing the network to have when it's learning representations. If we would allow more hidden units, our network will be able to learn more complex representations but it will also be a more expensive operation that can be prone to overfitting. For the activation function, we will use *Rectified Linear Unit (relu)* which is one of the most common activation functions.

Result

- **With neural networks**

  The cluster wise result after using tensorflow with keras is **(for AHMEDABAD city):**

```
(None, 1)
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_51 (Dense)            (None, 64)                1152

 dense_52 (Dense)            (None, 64)                4160

 dense_53 (Dense)            (None, 1)                 65

=================================================================
Total params: 5,377
Trainable params: 5,377
Non-trainable params: 0
_________________________________________________________________
None
472/472 [==============================] - 2s 3ms/step - loss: 280552672.0000 - mae: 9825.5156
472/472 [==============================] - 1s 2ms/step
472/472 [==============================] - 1s 2ms/step - loss: 172680400.0000 - mae: 7418.4336
Testing set Root Mean Squared Error: RM13140.791452572406
Testing set Mean Absolute Error: RM7418.43359375
Tensorflow with Keras Sequential model R-squared: 0.47050123800155974
(None, 1)
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_54 (Dense)            (None, 64)                1152

 dense_55 (Dense)            (None, 64)                4160

 dense_56 (Dense)            (None, 1)                 65

=================================================================
Total params: 5,377
Trainable params: 5,377
Non-trainable params: 0
_________________________________________________________________
None
```

```
Trainable params: 5,377
Non-trainable params: 0
_________________________________________________________________
None
48/48 [==============================] - 1s 4ms/step - loss: 1636224896.0000 - mae: 31470.0996
48/48 [==============================] - 0s 2ms/step
48/48 [==============================] - 0s 2ms/step - loss: 1529337344.0000 - mae: 30181.8320
Testing set Root Mean Squared Error: RM39106.742947987885
Testing set Mean Absolute Error: RM30181.83203125
Tensorflow with Keras Sequential model R-squared: -1.2964913719082358
(None, 1)
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_57 (Dense)            (None, 64)                1152

 dense_58 (Dense)            (None, 64)                4160

 dense_59 (Dense)            (None, 1)                 65

=================================================================
Total params: 5,377
Trainable params: 5,377
Non-trainable params: 0
_________________________________________________________________
None
1/1 [==============================] - 1s 802ms/step - loss: 629921856.0000 - mae: 25098.2441
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 114ms/step - loss: 627961920.0000 - mae: 25059.1680
Testing set Root Mean Squared Error: RM25059.16838205131
Testing set Mean Absolute Error: RM25059.16796875
Tensorflow with Keras Sequential model R-squared: nan
(None, 1)
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_60 (Dense)            (None, 64)                1152
```

Applying the model for the whole data gives the r2 score as 48.99%

In [170]:
```
# Assuming X_Train is made structured before.
X_whole = X_train.drop(['price'], axis=1)
y_whole = X_train.loc[:,'price']
ann_model = create_model(X_whole)
ann_model.compile(
    loss="mse",
    optimizer='adam',
    metrics=["mae"]
)
ann_model.fit(X_whole, y_whole)
y_pred_whole = ann_model.predict(X_whole)
```

```
(None, 1)
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_36 (Dense)            (None, 64)                1152

 dense_37 (Dense)            (None, 64)                4160

 dense_38 (Dense)            (None, 1)                 65

=================================================================
Total params: 5,377
Trainable params: 5,377
Non-trainable params: 0
_________________________________________________________________
None
519/519 [==============================] - 2s 2ms/step - loss: 340771072.0000 - mae: 10997.0059
519/519 [==============================] - 1s 2ms/step
```

In [171]:
```
evaluate_model(ann_model, X_whole, y_whole, y_pred_whole)
```

```
519/519 [==============================] - 1s 2ms/step - loss: 187315904.0000 - mae: 7879.8296
Testing set Root Mean Squared Error: RM13686.340051306632
Testing set Mean Absolute Error: RM7879.82958984375
Tensorflow with Keras Sequential model R-squared: 0.48997220200529457
```

It has a poorer r2_score than the one obtained from LinearRegression, as evident from the above screenshot. Also, we calculated the r2_score for cluster 2 as well, however it came out to be negative, which may violate our ideology to simulate the same using Tensorflow with Keras. Therefore, for cluster 2 as well, LinearRegression seems to be a better option.

Test results

For AHMEDABAD city:

In [322]:
```
for i in range(optimal_k):
    y_test = test_clusters[i]['price']
    x_test = test_clusters[i].drop(['price', 'x',  'y', 'z', 'label', 'index'], axis=1)
    y_pred = ann_models[i].predict(x_test)
    evaluate_model(ann_models[i], x_test, y_test, y_pred)
```

```
52/52 [==============================] - 0s 2ms/step
52/52 [==============================] - 0s 2ms/step - loss: 178993040.0000 - mae: 7608.8516
Testing set Root Mean Squared Error: RM13378.828050318907
Testing set Mean Absolute Error: RM7608.8515625
Tensorflow with Keras Sequential model R-squared: 0.43568355834587624
6/6 [==============================] - 0s 2ms/step
6/6 [==============================] - 0s 3ms/step - loss: 1226542592.0000 - mae: 28705.0352
Testing set Root Mean Squared Error: RM35022.030095355694
Testing set Mean Absolute Error: RM28705.03515625
Tensorflow with Keras Sequential model R-squared: -1.8084760714076733

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_3009/3024770305.py in <module>
      2     y_test = test_clusters[i]['price']
      3     x_test = test_clusters[i].drop(['price', 'x',  'y', 'z', 'label', 'index'], axis=1)
----> 4     y_pred = ann_models[i].predict(x_test)
      5     evaluate_model(ann_models[i], x_test, y_test, y_pred)

~/.local/lib/python3.10/site-packages/keras/utils/traceback_utils.py in error_handler(*args, **kwargs)
     68             # To get the full stack trace, call:
     69             # `tf.debugging.disable_traceback_filtering()`
---> 70             raise e.with_traceback(filtered_tb) from None
     71         finally:
     72             del filtered_tb

~/.local/lib/python3.10/site-packages/keras/engine/training.py in predict(self, x, batch_size, verbose, steps, callbacks, max_q
ueue_size, workers, use_multiprocessing)
   2373                         )
   2374             if batch_outputs is None:
```

In [325]:
```
nnn_whole_test = X_test.drop(['x', 'y', 'z', 'label'], axis=1)
```

In [330]:
```
y_pred_test = ann_model.predict(nnn_whole_test.drop(['price'], axis = 1))
```

```
58/58 [==============================] - 0s 2ms/step
```

Out[330]:
```
array([[17268.994],
       [33388.32 ],
       [20599.045],
       ...,
       [26424.139],
       [13011.395],
       [17785.629]], dtype=float32)
```

In [335]:
```
evaluate_model(ann_model, nnn_whole_test.drop(['price'], axis=1), y_pred_test, nnn_whole_test['price'])
```

```
58/58 [==============================] - 0s 2ms/step - loss: 27597394.0000 - mae: 4359.7891
Testing set Root Mean Squared Error: RM5253.32218695941
Testing set Mean Absolute Error: RM4359.7890625
Tensorflow with Keras Sequential model R-squared: 0.2699047062396911
```

In [ ]:
```

```

- **With multiple linear regression**

  The cluster-wise result for multiple linear regression is as **(for AHMEDABAD city):** :

Out[289]:

	index	seller_type_AGENT	seller_type_BUILDER	seller_type_OWNER	layout_type_BHK	layout_type_RK	property_type_Apartment	property_type_Indepen F
0	1592	1.0	0.0	0.0	1.0	0.0	1.0	
1	5858	1.0	0.0	0.0	1.0	0.0	0.0	
2	13359	1.0	0.0	0.0	1.0	0.0	0.0	
3	7600	1.0	0.0	0.0	1.0	0.0	1.0	
4	14585	1.0	0.0	0.0	1.0	0.0	1.0	
...	...	...	...	...	...	...	...	
15084	5018	1.0	0.0	0.0	1.0	0.0	1.0	
15085	12749	1.0	0.0	0.0	1.0	0.0	1.0	
15086	11843	1.0	0.0	0.0	1.0	0.0	1.0	
15087	675	1.0	0.0	0.0	1.0	0.0	1.0	
15088	4666	1.0	0.0	0.0	1.0	0.0	1.0	

15089 rows × 23 columns

In [290]:

```
regressors = []
for i in range(len(clusters)):
    y = clusters[i]['price']
    x = clusters[i].drop(['price', 'label', 'x', 'y', 'z', 'index'], axis=1)
    regressor = LinearRegression()
    regressor.fit(x, y)
    regressors.append(regressor)
    y_pred = regressor.predict(x)
    r2 = r2_score(y, y_pred)
    print(r2)
```

```
0.5457032348969448
0.7297691897533612
nan
nan
```

The r2_score for the first cluster is 54.57%, second cluster is 72.93% and nan for the last two clusters.

```
       [90792.13121783],
       ...,
       [12857.64488982],
       [25365.49320244],
       [13486.27645961]])
```

In [300]:

```
y_train
```

Out[300]:

	price
1592	13500.0
5858	23500.0
13359	45000.0
7600	25000.0
14585	16000.0
...	...
5018	60001.0
12749	35000.0
11843	11700.0
675	18000.0
4666	12000.0

16607 rows × 1 columns

In [301]:

```
r2_score(np.array(y_train), y_pred)
```

Out[301]: 0.5633192986840967

In [302]:

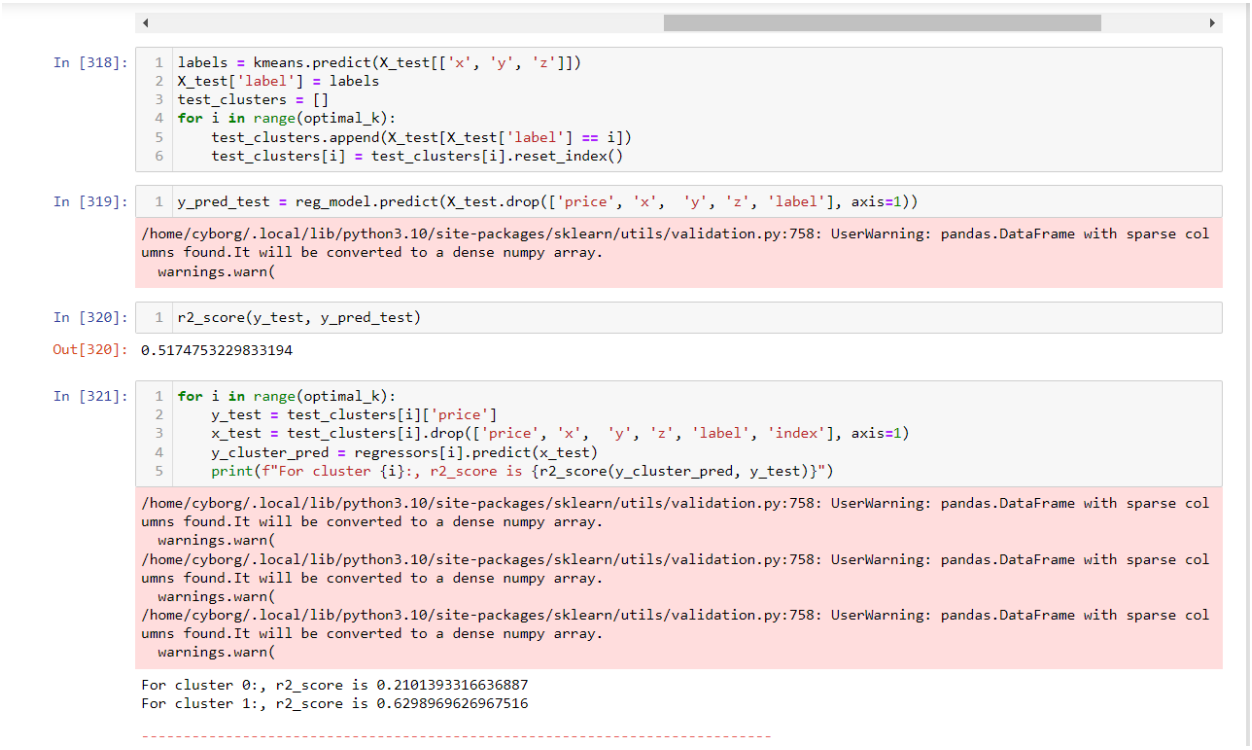
```
y_pred
```

Out[302]:

```
array([[14524.87517819],
       [28487.88235785],
       [90792.13121783],
```

We also applied regression on the entire model without filtering on the basis of locality, and obtained a r2_score of 55.97% as evident from above.

Test results (for AHMEDABAD city):



```
In [318]:
labels = kmeans.predict(X_test[['x', 'y', 'z']])
X_test['label'] = labels
test_clusters = []
for i in range(optimal_k):
    test_clusters.append(X_test[X_test['label'] == i])
    test_clusters[i] = test_clusters[i].reset_index()

In [319]:
y_pred_test = reg_model.predict(X_test.drop(['price', 'x',  'y', 'z', 'label'], axis=1))

/home/cyborg/.local/lib/python3.10/site-packages/sklearn/utils/validation.py:758: UserWarning: pandas.DataFrame with sparse columns found.It will be converted to a dense numpy array.
  warnings.warn(

In [320]:
r2_score(y_test, y_pred_test)

Out[320]:
0.5174753229833194

In [321]:
for i in range(optimal_k):
    y_test = test_clusters[i]['price']
    x_test = test_clusters[i].drop(['price', 'x',  'y', 'z', 'label', 'index'], axis=1)
    y_cluster_pred = regressors[i].predict(x_test)
    print(f"For cluster {i}:, r2_score is {r2_score(y_cluster_pred, y_test)}")

/home/cyborg/.local/lib/python3.10/site-packages/sklearn/utils/validation.py:758: UserWarning: pandas.DataFrame with sparse columns found.It will be converted to a dense numpy array.
  warnings.warn(
/home/cyborg/.local/lib/python3.10/site-packages/sklearn/utils/validation.py:758: UserWarning: pandas.DataFrame with sparse columns found.It will be converted to a dense numpy array.
  warnings.warn(
/home/cyborg/.local/lib/python3.10/site-packages/sklearn/utils/validation.py:758: UserWarning: pandas.DataFrame with sparse columns found.It will be converted to a dense numpy array.
  warnings.warn(

For cluster 0:, r2_score is 0.2101393316636887
For cluster 1:, r2_score is 0.6298969626967516
```

Hence, we can interpret from the scores that the linear regression model worked better than the neural network for the entire test data as well as for clusters based on locality.
However, neural networks worked better for cluster-wise data for some clusters than on the entire data set.
Note: the nan in the result is since some of the clusters just contained a single location (and thus act as outlier and can't be used for prediction).

Most of the localities are largely located around cluster 0 (see below screenshot), we have almost similar r2 score when we run our models for the entire dataset or in particular for this cluster.

Challenges

Applying them individually, had the following challenges:

- Geocode API didn't give exact longitude, latitude like Google, so this may result in a bit of poor clustering.

- Some localities are very densely inhabited. So there may be a locality which does not have sufficient data to train the model (like in our case it was the **third cluster**).
- Somehow there was an empty tuple coming with the train and test data using **train_test_split,** so we ignored the errors which were coming due to the same.