You can find some introduction about the project in the code file .ipyj

Introduction

Data analysis and price prediction are essential in the real estate industry, particularly for housing markets. These practices enable sta insights into the factors influencing housing prices, make informed decisions, and maximize opportunities. Data analysis helps uncover historical transaction data, while price prediction utilizes statistical and machine learning techniques to forecast future prices.

Accurate price predictions benefit buyers, sellers, and investors by aiding decision-making, optimizing returns, and minimizing risks. Pc leverage data analysis and price prediction to inform housing policies and promote sustainable growth. Overall, these practices empow data-driven insights, leading to better outcomes in the dynamic housing market.

Loading Modules & Data

```
[1]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from scipy.stats import f_oneway
from scipy.stats import wilcoxon
from scipy.stats import ttest_ind
from sklearn.cluster import KMeans
import matplotlib.gridspec as gridspec

# Suppressing Unusal warnings
import warnings
warnings.simplefilter("ignore")

# Changing default pandas setting to custom
pd.options.display.max_rows = 50
pd.options.display.max_columns = 50

# Setting Theme
```

The project starts importing the libraries need use to perform different functionality, then also suppresses the warnings so that the output should look very simply. So, it shouldn't present a lot of warnings.

I also, custom the default pandas setting to display only 50 rows and 50 columns.

```
: data = pd.read_csv('new.csv',
                   encoding='utf_8_sig',
                   encoding_errors='ignore')

housing = data.copy()
```

After that I read the data then I add a copy of the original data to keep it safe as it is.

```
print(housing.shape)
housing.head()
```

```
(318851, 26)
```

, then I check the shape of the data as you can see it has 3,8????? 36 columns

Statistics Analysis

a) Descriptive Statistics

[4]:

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 318851 entries, 0 to 318850
Data columns (total 26 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   url                  318851 non-null  object
 1   id                   318851 non-null  object
 2   Lng                  318851 non-null  float64
 3   Lat                  318851 non-null  float64
 4   Cid                  318851 non-null  int64
 5   tradeTime            318851 non-null  object
 6   DOM                  160874 non-null  float64
 7   followers            318851 non-null  int64
 8   totalPrice           318851 non-null  float64
 9   price                318851 non-null  int64
 10  square               318851 non-null  float64
 11  livingRoom           318851 non-null  object
 12  drawingRoom          318851 non-null  int64
 13  kitchen              318851 non-null  int64
 14  bathRoom             318851 non-null  object
 15  floor                318851 non-null  object
 16  buildingType         316830 non-null  float64
 17  constructionTime     318851 non-null  object
 18  renovationCondition  318851 non-null  int64
 19  buildingStructure    318851 non-null  int64
 20  ladderRatio          318851 non-null  float64
 21  elevator             318819 non-null  float64
 22  fiveYearsProperty    318819 non-null  float64
 23  subway               318819 non-null  float64
 24  district             318851 non-null  int64
 25  communityAverage     318388 non-null  float64
dtypes: float64(11), int64(8), object(7)
memory usage: 63.2+ MB
```

Following on the requirement I've separate the Statistics analysis accordingly.
First, I start from different statistics.
Basically, the purpose of descriptive analysis is to check the distribution of the data like how are data is distributed and it looks like such kind of characteristics we can find though the analysis. E.g Data types and so on.

After that I check this statistics properties for numerical features using the .describe function. You can performance the whole statistical analysis using only this descriptive function

First displayed is numerical features then I checked for categorical features.

```
# Statistical Properties for numerical features
housing.describe()
```

	Lng	Lat	Cid	DOM	followers	totalPrice	price	square	drawin
count	318851.000000	318851.000000	3.188510e+05	160874.000000	318851.000000	318851.000000	318851.000000	318851.000000	318851.
mean	116.418459	39.949591	1.129113e+12	28.822339	16.731508	349.030201	43530.436379	83.240597	1.
std	0.112054	0.091983	2.363447e+12	50.237343	34.209185	230.780778	21709.024204	37.234661	0.
min	116.072514	39.627030	1.111027e+12	1.000000	0.000000	0.100000	1.000000	6.900000	0.
25%	116.344985	39.893200	1.111027e+12	1.000000	0.000000	205.000000	28050.000000	57.900000	1.
50%	116.416780	39.934527	1.111027e+12	6.000000	5.000000	294.000000	38737.000000	74.260000	1.
75%	116.477581	40.003018	1.111027e+12	37.000000	18.000000	425.500000	53819.500000	98.710000	1.
max	116.732378	40.252758	1.114620e+15	1677.000000	1143.000000	18130.000000	156250.000000	1745.500000	28.

```
# Statistical Properties for Categorical features
housing.describe(include="O")
```

	url	id	tradeTime	livingRoom	bathRoom	floor	constructionTime
count	318851	318851	318851	318851	318851	318851	318851
unique	318851	318851	2560	21	26	71	74
top	https://bj.lianjia.com/chengjiao/101084782030....	101084782030	2016-02-28	2	1	6	2004
freq	1	1	1096	83333	206915	107530	21145

```python
# Calculate the mean of a specific column (e.g., 'price')
mean_price = housing['price'].mean()
print("Mean price:", mean_price)

# Calculate the median of a specific column (e.g., 'price')
median_price = housing['price'].median()
print("Median price:", median_price)

# Calculate the standard deviation of a specific column (e.g., 'price')
std_price = housing['price'].std()
print("Standard deviation of price:", std_price)

# Calculate the skewness of a specific column (e.g., 'price')
skewness_price = housing['price'].skew()
print("Skewness of price:", skewness_price)

# Calculate the kurtosis of a specific column (e.g., 'price')
kurtosis_price = housing['price'].kurt()
print("Kurtosis of price:", kurtosis_price)
```

```
Mean price: 43530.43637937469
Median price: 38737.0
Standard deviation of price: 21709.02420359375
Skewness of price: 1.3028650069857575
Kurtosis of price: 2.1735120294568038
```

After that I performed some descriptive analysis on our target features that is basically is the price.
It was checked the mean, median standard deviation, skewness and kurtosis of the price feature.

Then I checked the correlation numerical that you can see here.

```python
# Calculate the correlation matrix for numerical columns
correlation_matrix = housing.corr()
print("Correlation matrix:")
display(correlation_matrix)
```

Correlation matrix:

	Lng	Lat	Cid	DOM	followers	totalPrice	price	square	drawingRoo
Lng	1.000000	0.040847	-0.007301	-0.014274	-0.012846	-0.069831	-0.153212	0.064499	0.06664
Lat	0.040847	1.000000	-0.000257	0.022363	-0.005676	0.019969	-0.052004	0.119889	0.05657
Cid	-0.007301	-0.000257	1.000000	0.000952	0.001264	0.000071	-0.000387	-0.000413	0.00098
DOM	-0.014274	0.022363	0.000952	1.000000	0.465489	0.225404	0.215473	0.080909	0.00938
followers	-0.012846	-0.005676	0.001264	0.465489	1.000000	0.152681	0.257173	-0.050814	-0.05356
totalPrice	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

```
# Group the data by a specific column (e.g., 'district') and calculate summary statistic
grouped_district = housing.groupby('district')['price'].describe()
print("Summary statistics of price by district:")
display(grouped_district)
```

```
Summary statistics of price by district:
```

district	count	mean	std	min	25%	50%	75%	max
1	17086.0	62024.151235	24296.229204	1.0	43957.50	56769.5	77252.50	150000.0
2	29338.0	38173.677994	13232.607510	1.0	28998.00	35543.0	44815.00	123830.0
3	2537.0	31312.978715	10955.282409	3.0	23298.00	29060.0	37543.00	78263.0
4	15313.0	30022.898126	12338.158785	2.0	21415.00	26897.0	36855.00	131308.0
5	2955.0	28329.428088	9666.255263	1997.0	20221.50	26833.0	34838.00	67969.0
6	38634.0	29380.379277	11528.106693	2.0	21489.00	27237.5	35478.75	142928.0
7	107244.0	43628.537438	16986.575549	2.0	31431.00	40146.5	52776.00	156250.0
8	38200.0	54855.085654	21486.112239	1.0	39104.00	50810.0	67310.75	149967.0
9	11371.0	35171.638642	12021.013170	2.0	27163.50	32179.0	41246.00	95996.0
10	31293.0	67615.931582	27581.058929	3.0	46305.00	62582.0	84765.00	150000.0

Accordingly, with the district was performed for each district I get statical properties separately
E.g mean, standard deviation and so on.

It was the descriptive analyze performed. I've also, added some other things in descriptive analyze like correlation matrix and district wise properties the values for price feature.

```
# Calculate the correlation matrix for numerical columns
correlation_matrix = housing.corr()
print("Correlation matrix:")
display(correlation_matrix)
```

```
Correlation matrix:
```

	Lng	Lat	Cid	DOM	followers	totalPrice	
Lng	1.000000	0.040847	-0.007301	-0.014274	-0.012846	-0.069831	-0.1
Lat	0.040847	1.000000	-0.000257	0.022363	-0.005676	0.019969	-0.0
Cid	-0.007301	-0.000257	1.000000	0.000952	0.001264	0.000071	-0.0
DOM	-0.014274	0.022363	0.000952	1.000000	0.465489	0.225404	0.2
followers	-0.012846	-0.005676	0.001264	0.465489	1.000000	0.152681	0.2
totalPrice	0.069831	0.019969	0.000071	0.225404	0.152681	1.000000	0.6


```
# Group the data by a specific column (e.g., 'district') and calculate summary
grouped_district = housing.groupby('district')['price'].describe()
print("Summary statistics of price by district:")
display(grouped_district)
```

```
Summary statistics of price by district:
```

district	count	mean	std	min	25%	50%	75%	max
1	17086.0	62024.151235	24296.229204	1.0	43957.50	56769.5	77252.50	150000.0
2	29338.0	38173.677994	13232.607510	1.0	28998.00	35543.0	44815.00	123830.0

Hypothesis Analysis

T-Test for District A vs. District B

The t-test was performed to compare the average prices between District A and District B. The t-st
average prices. The p-value of 0.0 further supports this finding. The null hypothesis is rejected, inc
between District A and District B. This information is crucial for understanding the variations in hou
buyers, sellers, and policymakers in making informed decisions.

.1]:

```
district_A_prices = housing[housing['district'] == 1]['price']
district_B_prices = housing[housing['district'] == 2]['price']

t_statistic, p_value = ttest_ind(district_A_prices, district_B_prices)

print(f"T-test Results for District A vs. District B:")
print(f"T-Statistic: {t_statistic}")
print(f"P-Value: {p_value}")

if p_value < 0.05:
    print("The null hypothesis is rejected.")
    print("There is a significant difference in average prices between Dist
else:
    print("The null hypothesis is accepted.")
    print("There is no significant difference in average prices between Dis
```

```
T-test Results for District A vs. District B:
T-Statistic: 136.86177572656226
P-Value: 0.0
The null hypothesis is rejected.
There is a significant difference in average prices between District A and
```

Then Hypothesis Analysis where hypothesis test was performed and try to figure out the relationship between different features of the data.

You can find the analysis in detail

I want to check the price of two different districts. E.g. I own a house in district A and also, own a house in district B then will the price change or not?
For this kind of relationship T-Test was used.

You can see by the result obtained that the hypothesis was rejected. There is no relationship identified, there is no change between the price but there is a significant difference in average price between District A and B according with the hypothesis analysis.

One Way ANOVA - to check the difference between numerical values of multiple groups.

You can find also, the Wilcoxon test, Correlation analyses and so on.

One-Way ANOVA for Building Types ¶

The one-way ANOVA test was conducted to examine the difference in average prices among different building ty
significant, and the corresponding p-value of 9.17e-257 indicates a strong evidence of a difference in average pri
rejected, signifying a significant difference in average prices among different building types. This insight is valuab
are interested in specific building types and want to assess their pricing dynamics.

2]:

```
tower_prices = housing[housing['buildingType'] == 1]['price']
bungalow_prices = housing[housing['buildingType'] == 2]['price']
combo_prices = housing[housing['buildingType'] == 3]['price']

f_statistic, p_value = f_oneway(tower_prices, bungalow_prices, combo_prices)

print(f"ANOVA Results for Building Types:")
print(f"F-Statistic: {f_statistic}")
print(f"P-Value: {p_value}")

if p_value < 0.05:
    print("The null hypothesis is rejected.")
    print("There is a significant difference in average prices among different building
else:
    print("The null hypothesis is accepted.")
    print("There is no significant difference in average prices among different buildir
```

```
ANOVA Results for Building Types:
F-Statistic: 591.9625811703552
P-Value: 9.16598942572325e-257
The null hypothesis is rejected.
There is a significant difference in average prices among different building types.
```

Wilcoxon Test for Houses with Elevator vs. Houses without Elevator

The Wilcoxon test aimed to compare the price distributions between houses with and without an elevator. The tes
3.60e-279 suggest a significant difference in price distributions. The null hypothesis is rejected, indicating that the
distributions between houses with and without an elevator. This information is essential for buyers and sellers, as
property values.

3]:

```
elevator_prices = housing[housing['elevator'] == 1]['price']
no_elevator_prices = housing[housing['elevator'] == 0]['price']

# Randomly sample from the larger sample to make lengths equal
if len(elevator_prices) > len(no_elevator_prices):
    elevator_prices = elevator_prices.sample(n=len(no_elevator_prices), random_state=42
```

After the hypothesis analysis we need to explore the data through visualization.
For this I’ve made different types of visualization. I also, performed cluster analysis and there are a lot of exploration using different kind of graphs .

First of all, I made different plot for this columns – followers, totalPrice, price, square, communityAverage.

To check the distribution of our numerical features, I used Shapiro test to check the normal distribution of each numerical feature.

```python
import scipy.stats as stats

# Select the numerical features for the normality test
numerical_features = ['followers', 'totalPrice', 'price', 'square', 'communityAverage']

# Perform normality test and create plots
fig, axes = plt.subplots(len(numerical_features), 2, figsize=(12, 18))
fig.subplots_adjust(hspace=0.4)

for i, feature in enumerate(numerical_features):
    # Perform Shapiro-Wilk test for normality
    stat, p_value = stats.shapiro(housing[feature])

     # Print normality test result
    if p_value > 0.05:
        normality = 'Normal'
    else:
        normality = 'Not Normal'
    print(f'{feature}: p-value = {p_value:.4f} ({normality})')

    # Plot histogram
    axes[i, 0].hist(housing[feature], bins=20, color='lightblue', edgecolor='black')
    axes[i, 0].set_xlabel(feature)
    axes[i, 0].set_ylabel('Frequency')
```

This is normal distribution that I tested. A single feature was selected and created plots. After that I treated the feature one by one and performed Shapiro test on that particular feature if is that in normal distribution or not. If it is in normal distribution value will be 'normal' otherwise will be 'not normal'.

Then for this feature I made two plots, first plot was histogram, and second plot was Q-Q plot. It basically shows how the data varying from average value.

```
# Perform normality test and create plots
fig, axes = plt.subplots(len(numerical_features), 2, figsize=(12, 18))
fig.subplots_adjust(hspace=0.4)

for i, feature in enumerate(numerical_features):
    # Perform Shapiro-Wilk test for normality
    stat, p_value = stats.shapiro(housing[feature])

     # Print normality test result
    if p_value > 0.05:
        normality = 'Normal'
    else:
        normality = 'Not Normal'
    print(f'{feature}: p-value = {p_value:.4f} ({normality})')

    # Plot histogram
    axes[i, 0].hist(housing[feature], bins=20, color='lightblue', edgecolor='black')
    axes[i, 0].set_xlabel(feature)
    axes[i, 0].set_ylabel('Frequency')
    axes[i, 0].set_title('Histogram')

    # Plot Q-Q plot
    stats.probplot(housing[feature], dist='norm', plot=axes[i, 1])
    axes[i, 1].set_xlabel('Theoretical Quantiles')
    axes[i, 1].set_ylabel('Ordered Values')
    axes[i, 1].set_title('Q-Q Plot')

    # Add normality test result as text
    axes[i, 1].text(0.05, 0.9, f'p-value: {p_value:.4f}', transform=axes[i, 1].transAxes)

plt.tight_layout()
plt.show()
```

```
followers: p-value = 0.0000 (Not Normal)
totalPrice: p-value = 0.0000 (Not Normal)
price: p-value = 0.0000 (Not Normal)
square: p-value = 0.0000 (Not Normal)
communityAverage: p-value = 1.0000 (Normal)
```

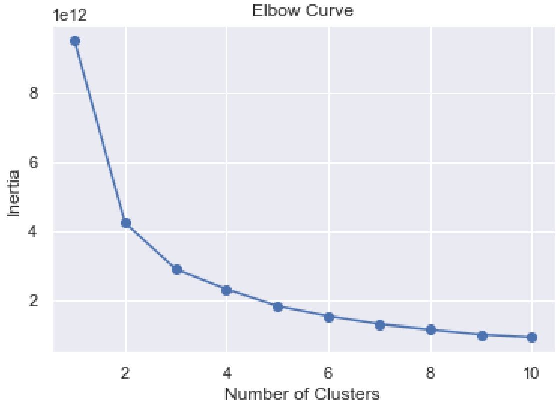
Cluster analysis was performed to find different groups and patterns.
It was only performed on numerical features.
K means clustering. Before applying K clustering was applied elbow curve test.

[17]:

```python
# Select the features for clustering
cols = ['Lng', 'Lat', "followers", "totalPrice", "price", "square", "communityAverage"]
X = housing[cols].sample(10000).dropna()

# Determine the optimal number of clusters using elbow method
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

# Plot the elbow curve
plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Curve')
plt.show()
```



The median was used here to analyze the different between clusters.

```python
X.groupby("cluster").median()[cols]
```

cluster	Lng	Lat	followers	totalPrice	price	square	communityAverage
0	116.427639	39.925069	3.0	230.0	28729.0	84.45	46635.0
1	116.420546	39.941862	6.0	342.0	48070.0	68.30	70655.0
2	116.379101	39.944412	11.0	528.8	80979.0	63.00	97655.0

Data Cleaning & Exploration

```
0]: df = data.copy()
```

```
1]: # Removing Ambigous values
df = df[df['livingRoom'] != '#NAME?']
df = df[df['constructionTime'] != 'δ,']

def to_int(x):
    try:
        x = int(x)
    except:
        x = int(str(x).split(" ")[1])
    return x

df.floor = df.floor.apply(to_int)
```

Data Cleaning for ML modules.
Exploration for visualization

First, I created a copy od the data. Ambiguous values were removed as they were found in the dataset.

Was identified string values not having proper integer data type so I wrote a function and check if it is identified integer values if it is not supply it accordingly to the surpass and only get the value, drop any symbol then the data should get to work in integer.

Some data types and features were not in correct format, so I created setting as below shows for these.

```
# Setting df types
df['livingRoom'] = df['livingRoom'].astype(int)
df['bathRoom'] = df['bathRoom'].astype(float)
df['floor'] = df['floor'].astype(int)
df['constructionTime'] = pd.to_datetime(df['constructionTime'])
```

I wrote a function to calculate the missing values, percentage, data type in each of the features. So, this function take the data frame as an input will read all columns, will calculate the values as shows below.

```
# lets try to check the percentage of missing values,unique values,percentage of one catagory values and type against each co
def statistics(df):
    stats = []
    # Iterating all columns
    for col in df.columns:
        # Calculating different details and storing it into list
        stats.append((col, df[col].nunique(), df[col].isnull().sum(), df[col].isnull().sum() * 100 / df.shape[0], df[col].dty

    # Converting list into table
    stats_df = pd.DataFrame(stats, columns=['Feature', 'Unique_values', 'Missing values', 'Percentage of Missing Values', 'Da
    # Setting column name as index
    stats_df.set_index('Feature', drop=True, inplace=True)
    # Droping features in which no NAN  is present
    stats_df.drop(stats_df[stats_df['Missing values'] == 0].index, axis=0, inplace=True)
    # Sorting table according to number of NAN
    stats_df.sort_values('Percentage of Missing Values', ascending=False, inplace=True)
    return stats_df

statistics(df)
```