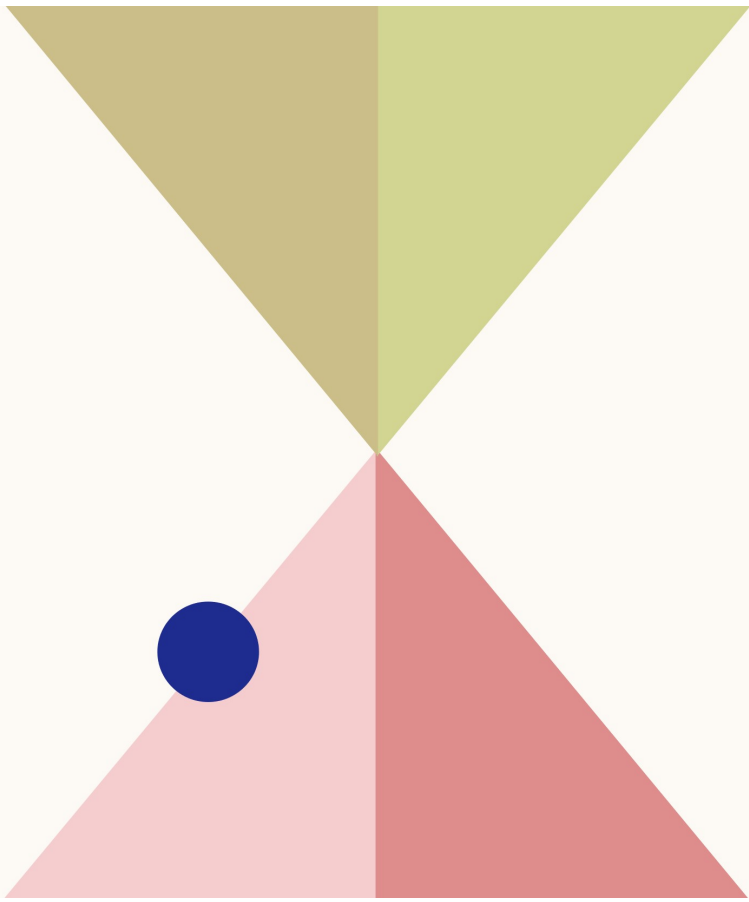
HOUSE SALARY PREDICTION IN PARIS

Instructor Name / Doctor : Sameh Zaref
Instructor Name / Doctor : Esraa Afify
Student Name And ID : Mohamed Hesham Emam 205090
Student Name And ID : Habiba Ahmed Saad 205056

AGENDA

TABLE OF CONTENTS



INTRODUCTION

- The goal of this data appears to be to provide information about individual homes sold in a particular area, including sale price, number of bedrooms and bathrooms, square footage, and other features.
- This data can be used to analyze trends in the real estate market, including forecasting home prices based on available features, identifying factors affecting home prices, developing predictive models for future sales and performing statistical analysis to better understand the housing market.

1)IMPORT LIBRARY
2)READ DATASET
3)UNDERSTAND

```
data.describe()
```

Out[5]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.00
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.65
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.17
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.00
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.00
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.00
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.00
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.00

In [6]:
```
# Statistic approach to numerical variables of the dataset
data.describe().astype(int)
```

Out[6]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_r
count	21613	21613	21613	21613	21613	21613	21613	21613	21613	21613	21613	21611	21613	21613	
mean	-2147483648	540088	3	2	2079	15106	1	0	0	3	7	1788	291	1971	
std	-2147483648	367127	0	0	918	41420	0	0	0	0	1	828	442	29	
min	1000102	75000	0	0	290	520	1	0	0	1	1	290	0	1900	
25%	2123049194	321950	3	1	1427	5040	1	0	0	3	7	1190	0	1951	
50%	-2147483648	450000	3	2	1910	7618	1	0	0	3	7	1560	0	1975	
75%	-2147483648	645000	4	2	2550	10688	2	0	0	4	8	2210	560	1997	
max	-2147483648	7700000	33	8	13540	1651359	3	1	4	5	13	9410	4820	2015	

```
# Checking for '0' in salaries
data.query("price == 0")
```

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	la

0 rows × 21 columns

```
#Change format to standarize the dataset describe() output
pd.set_option('display.float_format', lambda x: '%.5f' % x) # Set 5 decimals to eliminate numerical notation
data.describe()
```

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	gra
count	21613.00000	21613.00000	21613.00000	21613.00000	21613.00000	21613.00000	21613.00000	21613.00000	21613.00000	21613.00000	21613.000
mean	4580301520.86499	540088.14177	3.37084	2.11476	2079.89974	15106.96757	1.49431	0.00754	0.23430	3.40943	7.656
std	2876565571.31205	367127.19648	0.93006	0.77016	918.44090	41420.51152	0.53999	0.08652	0.76632	0.65074	1.175
min	1000102.00000	75000.00000	0.00000	0.00000	290.00000	520.00000	1.00000	0.00000	0.00000	1.00000	1.000
25%	2123049194.00000	321950.00000	3.00000	1.75000	1427.00000	5040.00000	1.00000	0.00000	0.00000	3.00000	7.000
50%	3904930410.00000	450000.00000	3.00000	2.25000	1910.00000	7618.00000	1.50000	0.00000	0.00000	3.00000	7.000
75%	7308900445.00000	645000.00000	4.00000	2.50000	2550.00000	10688.00000	2.00000	0.00000	0.00000	4.00000	8.000
max	9900000190.00000	7700000.00000	33.00000	8.00000	13540.00000	1651359.00000	3.50000	1.00000	4.00000	5.00000	13.000

ZIPCODE

```
# Create an instance of the SearchEngine class
engine = SearchEngine()

# Record the start time
start = time.time()

# Define a function to get the location information for a given zipcode and add it to a DataFrame
def get_location(zipcode, data):
    # Use the SearchEngine instance to get the location information for the given zipcode
    location = engine.by_zipcode(zipcode)

    # Add the location information to the DataFrame
    data["city"] = location.major_city
    data["state"] = location.state
    data["county"] = location.county
    data["population"] = location.population
    data["population_density"] = location.population_density

    # Return the updated DataFrame
    return data

# Apply the get_location function to each row of the DataFrame using the apply method
data = data.apply(lambda x: get_location(x['zipcode'], x), axis=1)

# Record the end time
end = time.time()

# Print the execution time and the updated DataFrame
print(f"The time of execution of above program is :{end-start}\n")
data
```

```
The time of execution of above program is :193.742769241333
```

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	...	zipcode	lat	long	sqft
0	20141013T000000	221900.00000	3	1.00000	1180	5650	1.00000	0	0	3	...	98178	47.51120	-122.25700	
1	20141209T000000	538000.00000	3	2.25000	2570	7242	2.00000	0	0	3	...	98125	47.72100	-122.31900	
2	20150225T000000	180000.00000	2	1.00000	770	10000	1.00000	0	0	3	...	98028	47.73790	-122.23300	
3	20141209T000000	604000.00000	4	3.00000	1960	5000	1.00000	0	0	5	...	98136	47.52080	-122.39300	

ManipulateColumnsWrongFormating

date
20141013T000000
20141209T000000
20150225T000000
20141209T000000
20150218T000000
...
20140521T000000
20150223T000000
20140623T000000
20150116T000000
20141015T000000

date
2014-10
2014-12
2015-02
2014-12
2015-02
...
2014-05
2015-02
2014-06
2015-01
2014-10

CHANGE DATE

```
# Convert the date column to a datetime format
data['date'] = pd.to_datetime(data['date'])

# Add a new column for the year of the transaction
data["tr_year"] = data["date"].dt.year

# Add a new column for the month of the transaction
data["tr_month"] = data["date"].dt.month

# Change the date column to a string format with only year and month
data["date"] = data["date"].dt.strftime('%Y-%m')

# Print the updated DataFrame
data
```

tr_year	tr_month
2014	10
2014	12
2015	2
2014	12
2015	2
...	...
2014	5
2015	2
2014	6
2015	1
2014	10

DROP DATA

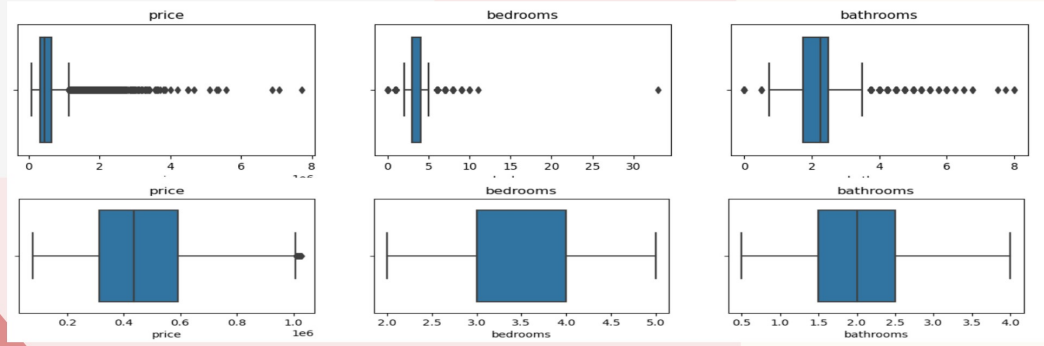
- What is missing Data?
- Missing data is a data where value == 0 or null
- How Handle missing Data?
- If Missing Less Than 10% we can remove it
- If more than 10% we can impute it by mean or mode
- Python code
- Df.dropna(axis =1 , inplace = true)
- Df.fillna(df[columnsName].mean)

OUTLIERS

handle outliers in a DataFrame

```python
# Define a function to handle outliers in a DataFrame
def handling_outliers(df, display=False, drop=False, drop_order=1, columns_to_drop=[]):

    # Get a list of numerical columns in the DataFrame
    numerical_columns = list((df.select_dtypes(include=np.number)).columns)

    # If display is True, plot boxplots for each numerical column
    if display == True:
        x = math.ceil(len(numerical_columns)/3)
        plt.figure(figsize=(15, 25))
        plt.subplots_adjust(hspace=0.5)
        plt.suptitle("Outliers Detection")
        for i in numerical_columns:
            y = numerical_columns.index(i) + 1
            ax = plt.subplot(x, 3, y)
            ax = sns.boxplot(x=df[i], data=df)
            ax.set_title(i)

    # If drop is True, remove outliers from the DataFrame
    if drop == True:

        # If columns_to_drop is not empty, use those columns
        if (len(columns_to_drop) != 0):
            numerical_columns = columns_to_drop

        # If drop_order is less than 1, set it to 1
        elif drop_order < 1:
            drop_order = 1

        # Remove outliers drop_order times using the interquartile range (IQR) method
        while drop_order != 0:

    # Remove outliers drop_order times using the interquartile range (IQR) method
    while drop_order != 0:

        for i in numerical_columns:
            q1 = df[i].quantile(0.25)
            q3 = df[i].quantile(0.75)
            iqr = q3 - q1
            lower = q1 - 1.5*iqr
            if lower < 0:
                lower = 0
            higher = q3 + 1.5*iqr
            df = df[df[i] >= lower]
            df = df[df[i] <= higher]

        drop_order = drop_order - 1

# Return the updated DataFrame
return df
```

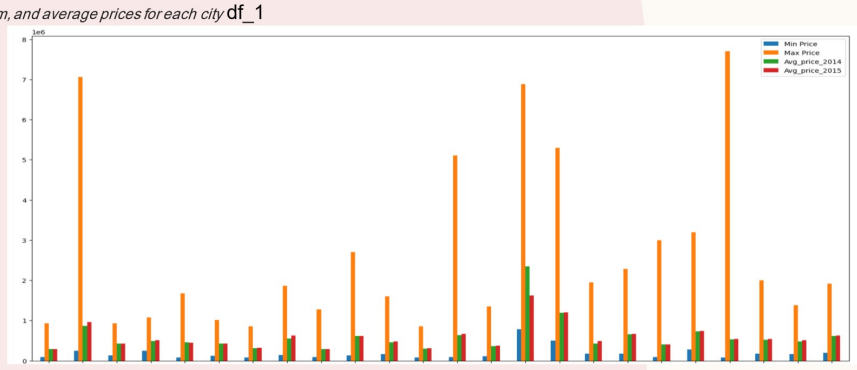
Before



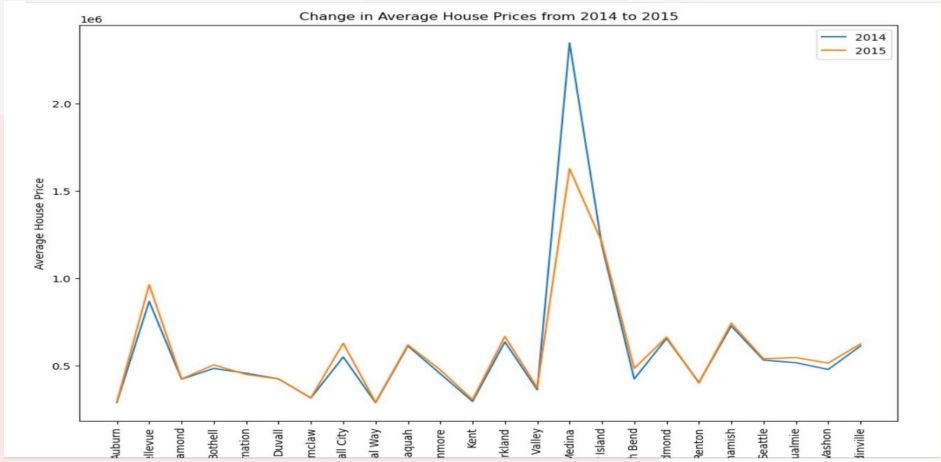
After

DESCRIPTIVE STATISTICS ANALYSIS

```
# Calculate the minimum and maximum prices for each city in df_copy
df_1 = [df_copy.groupby("city")["price"].min(), df_copy.groupby("city")["price"].max()]
df_1 = pd.DataFrame(df_1).round() df_1.index = ['Min Price', 'Max Price']
df_1 = df_1.T
# Calculate the average price for each city in 2014 and 2015
avg_2014 = (df_copy[df_copy['tr_year'] == 2014]).groupby('city')['price'].mean()
avg_2015 = (df_copy[df_copy['tr_year'] == 2015]).groupby('city')['price'].mean()
# Combine the average prices for 2014 and 2015 into a single DataFrame
avg = pd.DataFrame({'Avg_price_2014': avg_2014, 'Avg_price_2015': avg_2015}).round()
# Merge the minimum and maximum prices and average prices into a single
DataFrame df_1 = pd.merge(df_1, avg, right_index=True, left_index=True)
# Return the DataFrame with the minimum, maximum, and average prices for each city df_1
```



```python
# Calculate the percentage change in average house prices from 2014 to 2015 for each city
df_1['change_percentage'] = round(((df_1['Avg_price_2015'] - df_1['Avg_price_2014'])/df_1['Avg_price_2014'])*100 , 2)

# Reset the index of the DataFrame
df_1 = df_1.reset_index()

# Create a line plot showing the average house prices for each city in 2014 and 2015
# create fig size
plt.figure(figsize=(12,8))
# select per city and average price in 2014
plt.plot(df_1['city'], df_1['Avg_price_2014'], label='2014')
# select per city and average price in 2015
plt.plot(df_1['city'], df_1['Avg_price_2015'], label='2015')
# create rotation =90
plt.xticks(rotation=90)
plt.legend()
# X Label Name 'City'
plt.xlabel('City')
# Y Label Name 'Average House Price'
plt.ylabel('Average House Price')
# Title NAme 'Change in Average House Prices from 2014 to 2015'
plt.title('Change in Average House Prices from 2014 to 2015')
# display
plt.show()
```

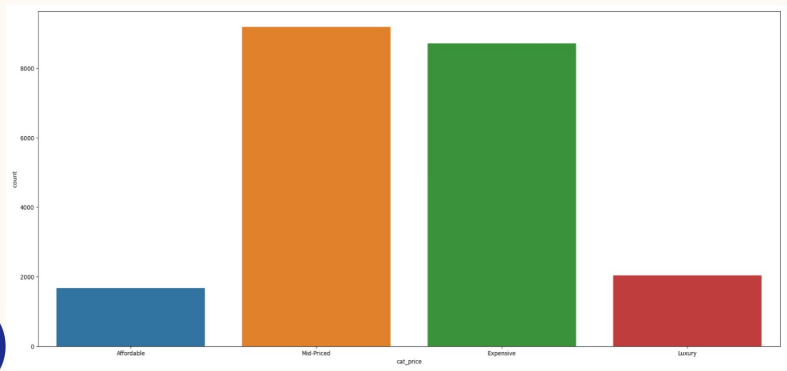


CATEGORISE PRICE

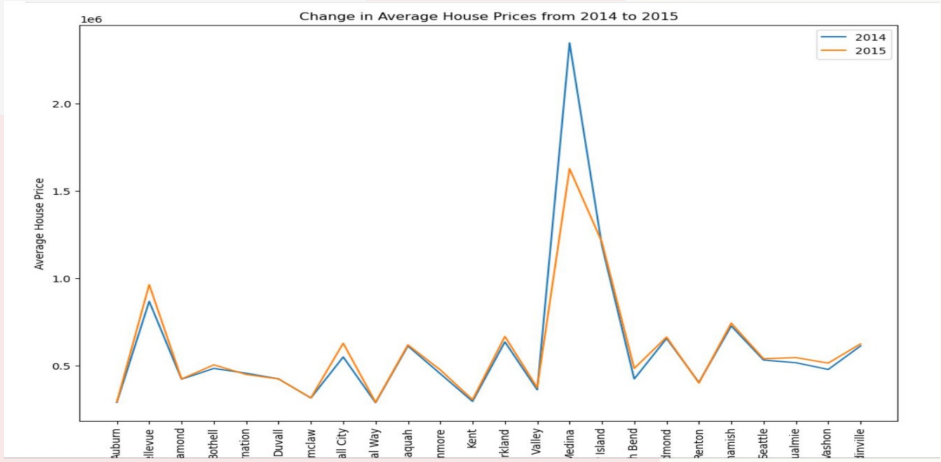
```
# Create a deep copy of the DataFrame df_copy
df_3 = df_copy.copy(deep=True)

# Create a new column in the DataFrame df_3 that categorizes the prices of the houses
df_3['cat_price'] = pd.cut(x=df_copy['price'], bins=[0,230000,450000,900000,df_copy['price'].max()],
                           labels=['Affordable', 'Mid-Priced', 'Expensive', 'Luxury'])
```

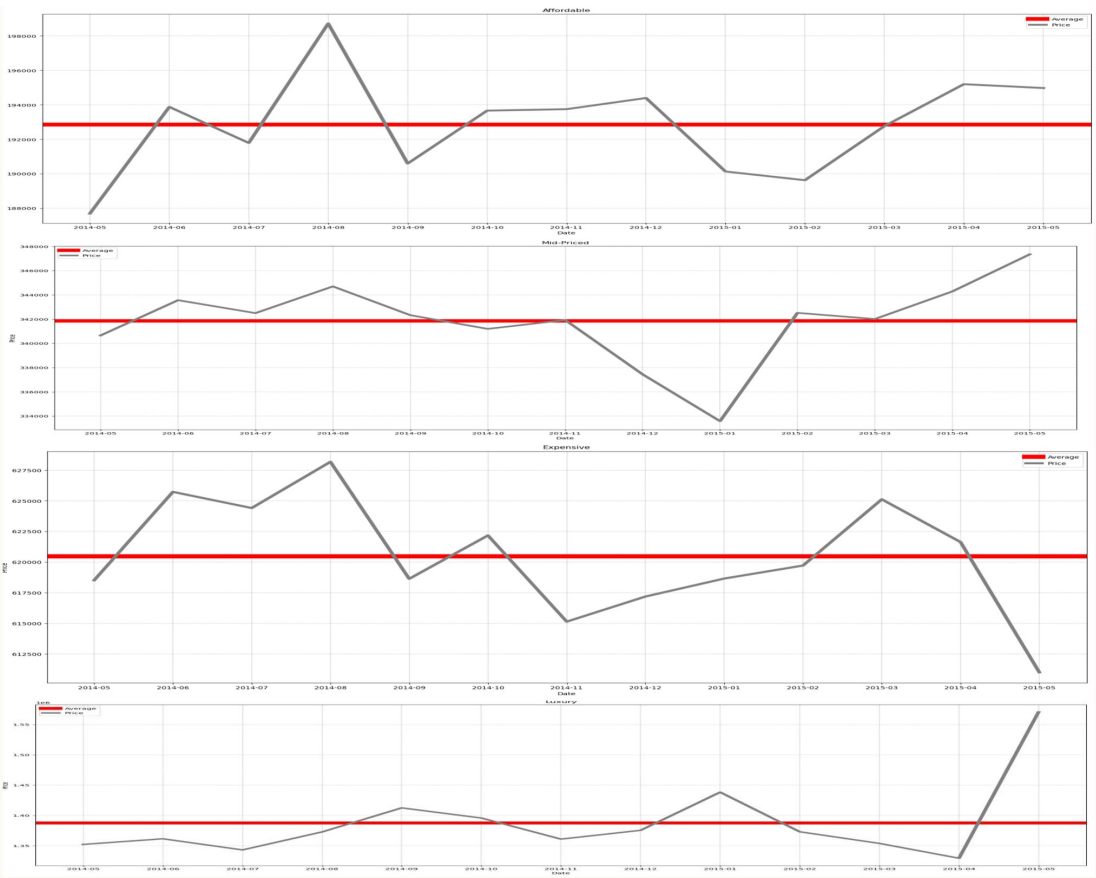
the overall code creates a new DataFrame df_3 that is a deep copy of an existing DataFrame df_copy and adds a new "Expensive", and "Luxury"

```
# Create a new figure with a specified size
plt.figure(figsize=(24,10))

# Create a countplot of the price categories in the DataFrame df_3 using Seaborn
sns.countplot(x='cat_price', data=df_3);
```



```python
# Calculate the percentage change in average house prices from 2014 to 2015 for each city
df_1['change_percentage'] = round(((df_1['Avg_price_2015'] - df_1['Avg_price_2014'])/df_1['Avg_price_2014'])*100 , 2)

# Reset the index of the DataFrame
df_1 = df_1.reset_index()

# Create a line plot showing the average house prices for each city in 2014 and 2015
# create fig size
plt.figure(figsize=(12,8))
# select per city and average price in 2014
plt.plot(df_1['city'], df_1['Avg_price_2014'], label='2014')
# select per city and average price in 2015
plt.plot(df_1['city'], df_1['Avg_price_2015'], label='2015')
# create rotation =90
plt.xticks(rotation=90)
plt.legend()
# X Label Name 'City'
plt.xlabel('City')
# Y Label Name 'Average House Price'
plt.ylabel('Average House Price')
# Title NAme 'Change in Average House Prices from 2014 to 2015'
plt.title('Change in Average House Prices from 2014 to 2015')
# display
plt.show()
```

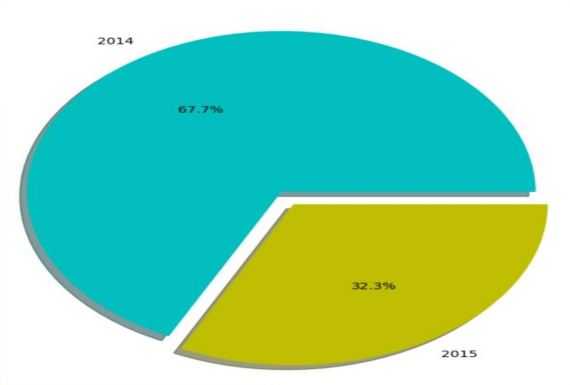



Affordable
Average
Price
198000
196000
194000
192000
190000
188000
2014-05
2014-06
2014-07
2014-08
2014-09
2014-10
2014-11
2014-12
2015-01
2015-02
2015-03
2015-04
2015-05
Date
Mid-Priced
348000
346000
344000
342000
340000
338000
336000
334000
Expensive
627500
625000
622500
620000
617500
615000
612500
Luxury
1.55
1.50
1.45
1.40
1.35

```python
# Get the total house prices over time for the years 2014 and 2015
x = (df_copy[df_copy['tr_year'] == 2014]).groupby('date')['price'].sum()
y = (df_copy[df_copy['tr_year'] == 2015]).groupby('date')['price'].sum()

# Create a new figure with a specified size
plt.figure(figsize=(24, 10))

# Add labels and annotations to the plot
# create Xlabel Name 'Date'
plt.xlabel("Date")
#create Ylabel NAme 'Price'
plt.ylabel("Price")
# create a x and color and wideh
plt.plot(x, color='green', label='T_Price_2014', linewidth=10)
plt.plot(y, color='yellow', label='T_Price_2015', linewidth=10)
# in grid passing Color and line stle and width
plt.grid(color='black', linestyle='--', linewidth=0.2)

plt.legend()
```

Yearly Monthly Transactions

```python
# Calculate the number of rows in the DataFrame for each year
y = [df_copy[df_copy['tr_year'] == 2014].shape[0], df_copy[df_copy['tr_year'] == 2015].shape[0]]

# Specify labels for the pie chart
mylabels = ["2014", "2015"]

# Specify an offset for the first slice of the pie chart
myexplode = [0.09, 0]

# Create a new figure with a specified size
plt.figure(figsize=(12, 8))

# Create a pie chart using Matplotlib
plt.pie(y, labels=mylabels, explode=myexplode, autopct='%1.1f%%', shadow=True, colors=['c', 'y'])

# Display the plot
plt.show()
```



REGRESSION MODEL SELECTION

LINEAR REGRESSION	GRADIENT BOOSTING	NEURAL NETWORK	RANDOM FOREST	Logistic Model
Predict model	Predict model	Predict model	Predict model The best one	Predict model overfit

CLASSIFICATION MODEL SELECTION

DECISION TREE	LOGISTIC MODEL	GRADIENT BOOSTING	RANDOM FOREST	ADA BOOST
Predict model	predict model	Predict model	Predict model The best one	Predict model

FrontEnd Deployment

Index

```
</style>
</head>
<body>
<form action="/predict" method="post">
  <input type="text" name="bedrooms" placeholder="Number of bedrooms">
  <input type="text" name="bathrooms" placeholder="Number of bathrooms">
  <input type="text" name="sqft_living" placeholder="Square footage of living space">
  <input type="text" name="floors" placeholder="Number of floors">
  <input type="text" name="view" placeholder="View">
  <input type="text" name="grade" placeholder="Grade">
  <input type="text" name="sqft_above" placeholder="Square footage above ground">
  <input type="text" name="sqft_basement" placeholder="Square footage below basement ">
  <input type="text" name="lat" placeholder="Latitude">
  <input type="text" name="sqft_lot15" placeholder="Square footage of lot">
  <input type="text" name="population" placeholder="Population">
  <input type="submit" value="Predict">
</form>

<div class="output">
  <p>The predicted house price is: {{ output }}</p>
  <p>The kind of house price is: {{ output2 }}</p>
</div>
```

BackEnd Deployment

App


```python
from joblib import load
from flask import Flask, request, render_template
app = Flask(__name__, template_folder='C:\\Users\Mohamed\House Salary\\New folder\\template')
model = load("Random_Forest_Model_Regression.pkl")
model2=load('Random_Forset_Model_Classification.pkl')
@app.route("/")
def Home ():
    return render_template('index.html')


@app.route("/predict", methods=["POST"])
def predict():
    bedrooms = request.form["bedrooms"]
    bathrooms = request.form["bathrooms"]
    sqft_living = request.form["sqft_living"]
    floors = request.form["floors"]
    view = request.form["view"]
    grade = request.form["grade"]
    sqft_above = request.form["sqft_above"]
    sqft_basement = request.form["sqft_basement"]
    lat = request.form["lat"]
    sqft_lot15 = request.form["sqft_lot15"]
    population = request.form["population"]

    price = model.predict([[bedrooms, bathrooms, sqft_living, floors, view, grade, sqft_above, sqft_basement, lat, sqft_lot15, popu
    priceKind=model2.predict([[bedrooms, bathrooms, sqft_living, floors, view, grade, sqft_above, sqft_basement, lat, sqft_lot15, p
    return render_template("index.html", output = price,output2=priceKind)

if __name__ == "__main__":
    app.run(debug=True)
```

CONCLUSION

The insights gained from this analysis could be useful for various stakeholders, such as home buyers, real estate agents, and property developers. The developed models provide a way to accurately predict the housing prices or classify the properties based on certain features, which could assist in making informed decisions about buying or selling properties. Overall, this analysis provides a valuable contribution to the field of real estate by identifying the factors that influence housing prices and developing accurate prediction models.

FUTURE WORK

1.Incorporating additional features: The dataset used in this analysis includes various features, but there may be other features that could influence housing prices, such as crime rates, proximity to schools or public transportation, and nearby amenities. Incorporating these features could improve the accuracy of the prediction models.

2.Improving the models' performance: Although the developed models have high accuracy, there is still room for improvement. Techniques such as ensemble learning, feature selection, and hyperparameter tuning could be used to further enhance the models' performance.

3.Evaluating the models' generalizability: The developed models were trained and tested on a specific dataset, so it is important to evaluate their generalizability to other datasets or real-world scenarios. Cross-validation techniques and testing on new datasets could be used to assess the models' generalizability.

4.Exploring interpretability: While the developed models have high accuracy, they may lack interpretability, meaning it may not be clear which features are driving the predictions. Exploring interpretability techniques such as feature importance and partial dependence plots could provide insights into the models' decision-making processes.

THANK YOU

THIS KERNER WRITE BY
ENG : MOHAMED HESHAM EMAM
ENG : HABIBA AHMED SAAD

AGENDA

TABLE OF CONTENTS

Description

Understand and mainpulation

Machine learning

GUI