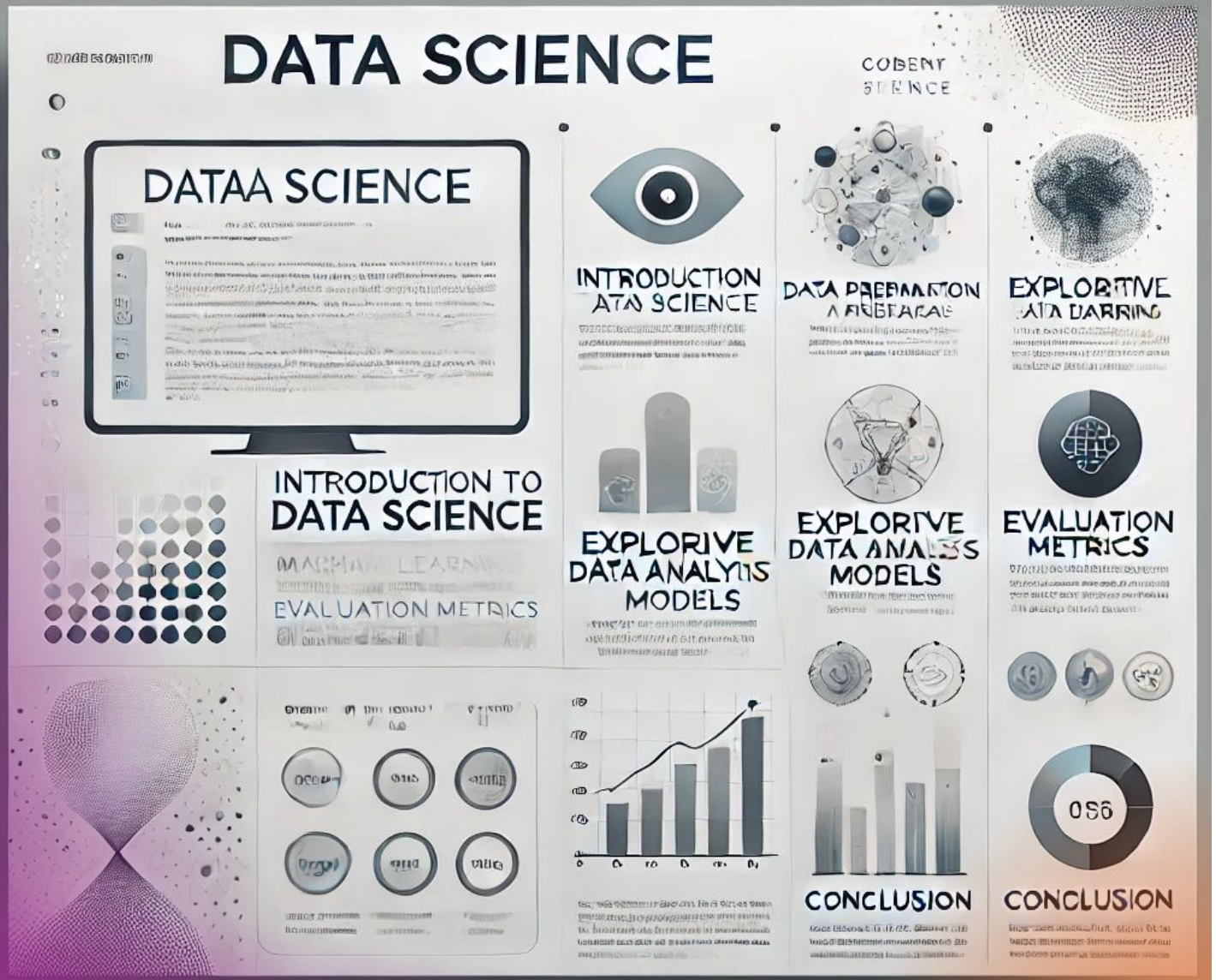
CAPSTONE PROJECT

FORECASTING HOUSE PRICE IN JOHOR BAHRU

Prepared by : Amy Anak Pirah
Supervisor : Dr Mohd Zamri Othman

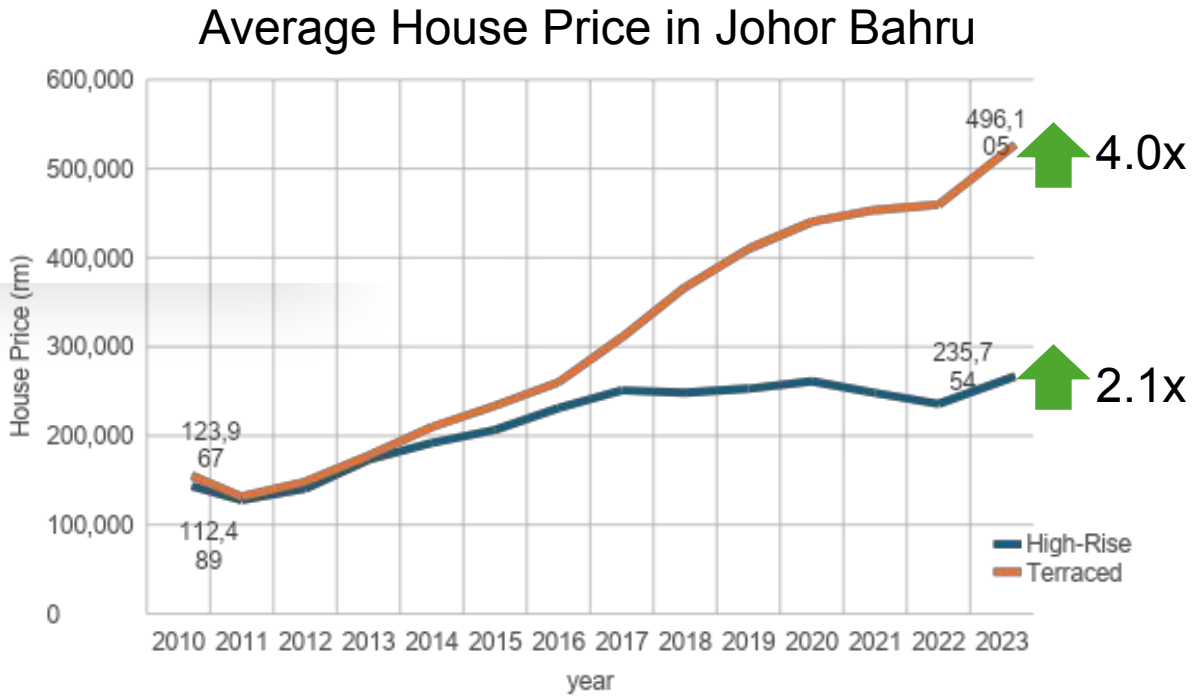


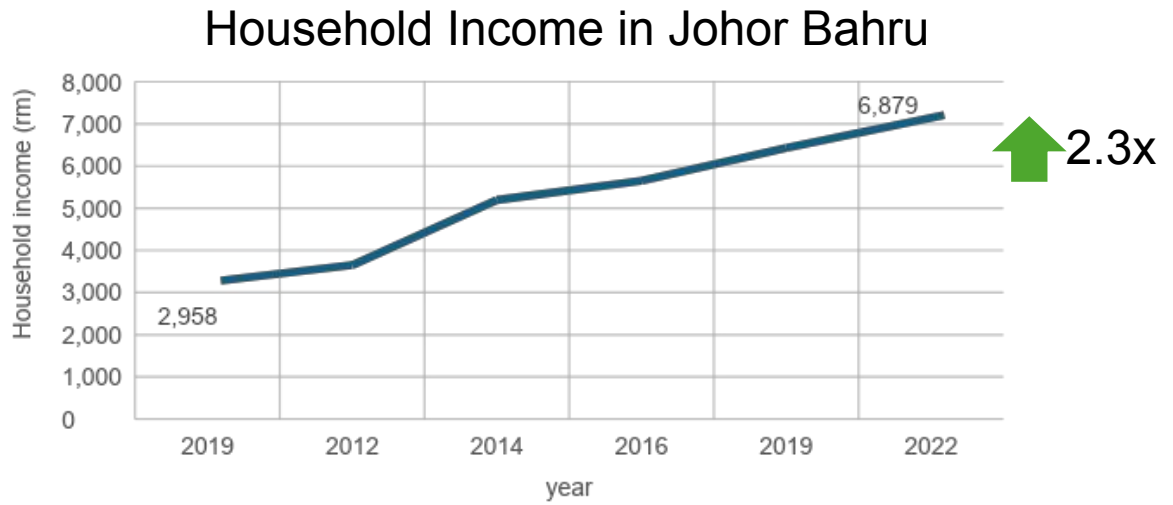
OUTLINE :

- PROBLEM STATEMENT
- DATASET
- DATA CLEANING & PREPROCESSING
- EXPLORATORY DATA ANALYSIS
- MACHINE LEARNING
- DASHBOARD
- CONCLUSION

PROBLEM STATEMENT

- The property market in Johor Bahru experiences fluctuations, making it challenging for investors, developers, and policymakers to predict future trends accurately.
- These challenges highlight the need for reliable forecasting models based on historical transactions to support decision-making.
- Accurate forecasts are essential for policymakers to design and implement sustainable housing strategies, ensuring balanced supply and affordability. Precise predictions help investors, developers, and other stakeholders make informed decisions, optimize investments, and align development strategies with market trends.





DATASET

- Historical house price data from 2014 to 2023 sourced from Form PDS15 provided by the IRB(Inland Revenue Board) and Sales & Purchase Agreements collected by the Valuation and Property Services Department (JPPH).
- Excel Format

H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
LOT	KOD_J	NO_HA	J_HAR	PEGAI	PAJAK	STATU	SKIM	KOD_K	KEGUI	LUAS_	UNIT_I	KOD_K	JARAK	J_HAR	JENIS_	KEGUI	KOD_K
197539	30	502172	A	F		N	SIERRA PERDANA	3	1	130	mp	1	24	187		46	2
97288	75	22255	A	F		N	TMN BUKIT DAHLIA	4	1	130	mp	2	30	187		46	2
93431	129	329748	A	F		N	TMN DESA TEBRAU	3	1	132.851	mp	1	17	187		46	2
21007	129	119420	A	F		N	TMN DAYA	4	1	143.071	mp	1	15	187		46	2
124669	30	345772	A	F		N	TMN SKUDAI RIA	2	1	178	mp	1	24	187		46	2
112564	30	383142	A	F		N	TMN MEGAH RIA	3	1	143	mp	2	22	187		46	2

- Terraced House : 82609 rows, 42 columns
- Highrise : 17056 rows, 42 columns
- All House : 193615 rows, 42 columns

DATA PREPROCESSING

1. • **Data Reduction**
2. • **Data Cleaning**
3. • **Data Transformation**

Example process of Data Preprocessing using Google Colab

Column Selection & Dropping Irrelevant Columns

```
# Get the list of columns to drop by excluding the ones you want
columns_to_keep = ['DAERAH','MUKBAN','PEGANGAN','STATUS_LOT', 'SKIM',
'KOD_KLAS_KAW', 'LUAS_LOT', 'KOD_KATEGORI_KAW', 'JARAK_BDR', 'KOD_KEADAAN_BGN',
'KOD_JEN_BINAAN', 'LUAS_LOT_BGN', 'B_TINGKAT', 'BILIK_TDR', 'TKH_SIAP',
'SYER2', 'HARGA_B', 'KATEGORI_LPR', 'TKH_NILAI', 'PM_PERTAMA']

```

```
# Drop the columns that are not in the list of columns to keep
df = df[columns_to_keep]
```

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82609 entries, 0 to 82608
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   DAERAH            82609 non-null  int64
 1   MUKBAN            82609 non-null  int64
 2   PEGANGAN          82609 non-null  object
 3   STATUS_LOT        82609 non-null  object
 4   SKIM              82605 non-null  object
 5   KOD_KLAS_KAW      82609 non-null  int64
 6   LUAS_LOT          82609 non-null  float64
 7   KOD_KATEGORI_KAW  82609 non-null  int64
 8   JARAK_BDR         82607 non-null  float64
 9   KOD_KEADAAN_BGN   82609 non-null  int64
 10  KOD_JEN_BINAAN    82609 non-null  int64
 11  LUAS_LOT_BGN      82609 non-null  float64
 12  B_TINGKAT         82609 non-null  int64
 13  BILIK_TDR         82609 non-null  int64
 14  TKH_SIAP          82583 non-null  float64
 15  SYER2             82609 non-null  int64
 16  HARGA_B           82564 non-null  object
 17  KATEGORI_LPR      82609 non-null  int64
 18  TKH_NILAI         82609 non-null  datetime64[ns]
 19  PM_PERTAMA        82609 non-null  object
dtypes: datetime64[ns](1), float64(4), int64(10), object(5)
memory usage: 12.6+ MB
None
```

Converting Data Types

```
# Convert to Integer
columns_to_convert = ['KOD_KLAS_KAW', 'KOD_KATEGORI_KAW', 'KOD_KEADAAN_BGN',
'KOD_JEN_BINAAN', 'B_TINGKAT', 'KATEGORI_LPR']

df[columns_to_convert] = df[columns_to_convert].fillna(0).astype(int)
print(df)
```

```
# Convert 'USIA_BGN' to integer after filling NaN values with 0 (if any)
df['USIA_BGN'] = df['USIA_BGN'].fillna(0).astype(int)
```

```
df['HARGA_B']= df['HARGA_B'].apply(lambda x: str(x).strip())
df['HARGA_B']= df['HARGA_B'].apply(lambda x: str(x).replace(',',''))
```

```
# Convert 'HARGA_B' to float
df['HARGA_B'] = df['HARGA_B'].astype(float)
```

Converting to date time

```
# Convert to datetime
df.loc[:, 'TKH_NILAI'] = pd.to_datetime(df['TKH_NILAI'], format='%d/%m/%Y',
dayfirst=True, errors='coerce')

# Check the data type of the column
print(df['TKH_NILAI'].dtypes)

# Display the first few rows
print(df['TKH_NILAI'].head())
```

```
object
0    2019-01-01 00:00:00
1    2019-01-01 00:00:00
2    2019-01-04 00:00:00
3    2019-01-01 00:00:00
4    2019-01-02 00:00:00
Name: TKH_NILAI, dtype: object
```

```
df['TKH_NILAI'] = pd.to_datetime(df['TKH_NILAI'], format='%d/%m/%Y',
dayfirst=True, errors='coerce')

```

```
df.head()
```

	DAERAH	MUKBAN	PEGANGAN	STATUS_LOT	SKIM	KOD_KLAS_KAW	LUAS_LOT	KOD_KATEGORI_KAW	JARAK_BDR	KOD_KEADAAN_BGN	KOD_JEN_BINAAN	LUAS_LOT_BGN	B_TINGKAT	BILIK_TDR	TKH_SIAP	SYER2	HARGA_B	KATEGORI_LPR	TKH_NILAI	PM_PERTAMA
0	2	2	F	N	SIERRA PERDANA	3	130.000	1	24.0	2	1	133.360	2	3	2008.0	1	0	4	2019-01-01	T
1	2	2	F	N	TMN BUKIT DAHLIA	4	130.000	2	30.0	2	1	141.100	2	3	2003.0	2	0	4	2019-01-01	T
2	2	8	F	N	TMN DESA TEBRAU	3	132.851	1	17.0	2	1	113.032	2	3	2000.0	1	260,000	3	2019-01-04	T
3	2	8	F	N	TMN DAYA	4	143.071	1	15.0	2	1	83.330	1	3	1996.0	2	0	4	2019-01-01	T
4	2	3	F	N	TMN SKUDAI RIA	2	178.000	1	24.0	2	1	198.830	2	4	2003.0	1	0	4	2019-01-02	T

Handling Missing Data

```
df.shape
```

```
(31695, 21)
```

```
df.isna().sum()
```

	0
DAERAH	0
MUKBAN	0
PEGANGAN	0
STATUS_LOT	0
SKIM	1
KOD_KLAS_KAW	0
LUAS_LOT	0
KOD_KATEGORI_KAW	0
JARAK_BDR	1
KOD_KEADAAN_BGN	0
KOD_JEN_BINAAN	0
LUAS_LOT_BGN	0
B_TINGKAT	0
BILIK_TDR	0
TKH_SIAP	4
SYER2	0
HARGA_B	0
KATEGORI_LPR	0
TKH_NILAI	0
PM_PERTAMA	0
USIA_BGN	0

dtype: int64

```
df = df.dropna()
```

```
df.shape
```

```
(31690, 21)
```

Sorting and Filtering

```python
filtered_df = df[
    (df['KOD_JEN_BINAAN'] == 1) &
    (df['B_TINGKAT'] == 2) &
    (df['BILIK_TDR'].isin([2, 3, 4, 5, 6])) &
    (df['SYER2'] == 1) &
    (df['STATUS_LOT'] == 'N') &
    (df['DAERAH'] == 2) &
    (df['LUAS_LOT'] >= 50) & (df['LUAS_LOT'] <= 300) &
    (df['LUAS_LOT_BGN'] > 50) &
    (df['HARGA_B'] > 42000) &
    (df['USIA_BGN'] >= 0) &
    (df['KATEGORI_LPR'].isin([1, 2]))
]

# Display the filtered DataFrame
print(filtered_df)

```

Data Aggregation

```python

# Ensure 'quarter' is in datetime format
df['quarter'] = pd.PeriodIndex(df['quarter'], freq='Q').to_timestamp()

```

```python
#Aggregation by Quarter and Average Price
df['quarter'] = pd.PeriodIndex(df['quarter'], freq='Q').to_timestamp()
df_avg = df.groupby('quarter')['HARGA_B'].mean().reset_index()
```

Encoding categorical variables

```python
# Encode 'PM_PERTAMA' and 'PEGANGAN' columns, ignoring any values that do not match
if 'PEGANGAN' in df.columns:
    df['PEGANGAN'] = df['PEGANGAN'].map({'F': 1, 'L': 0}).fillna(df['PEGANGAN'])
if 'PM_PERTAMA' in df.columns:
    df['PM_PERTAMA'] = df['PM_PERTAMA'].map({'Y': 1, 'T': 0}).fillna(df['PM_PERTAMA'])
```

Creating New Features

```python
# Calculate the difference in years and create a new column 'USIA_BGN'
df['USIA_BGN'] = df['TKH_NILAI'].dt.year - df['TKH_SIAP'].dt.year
```

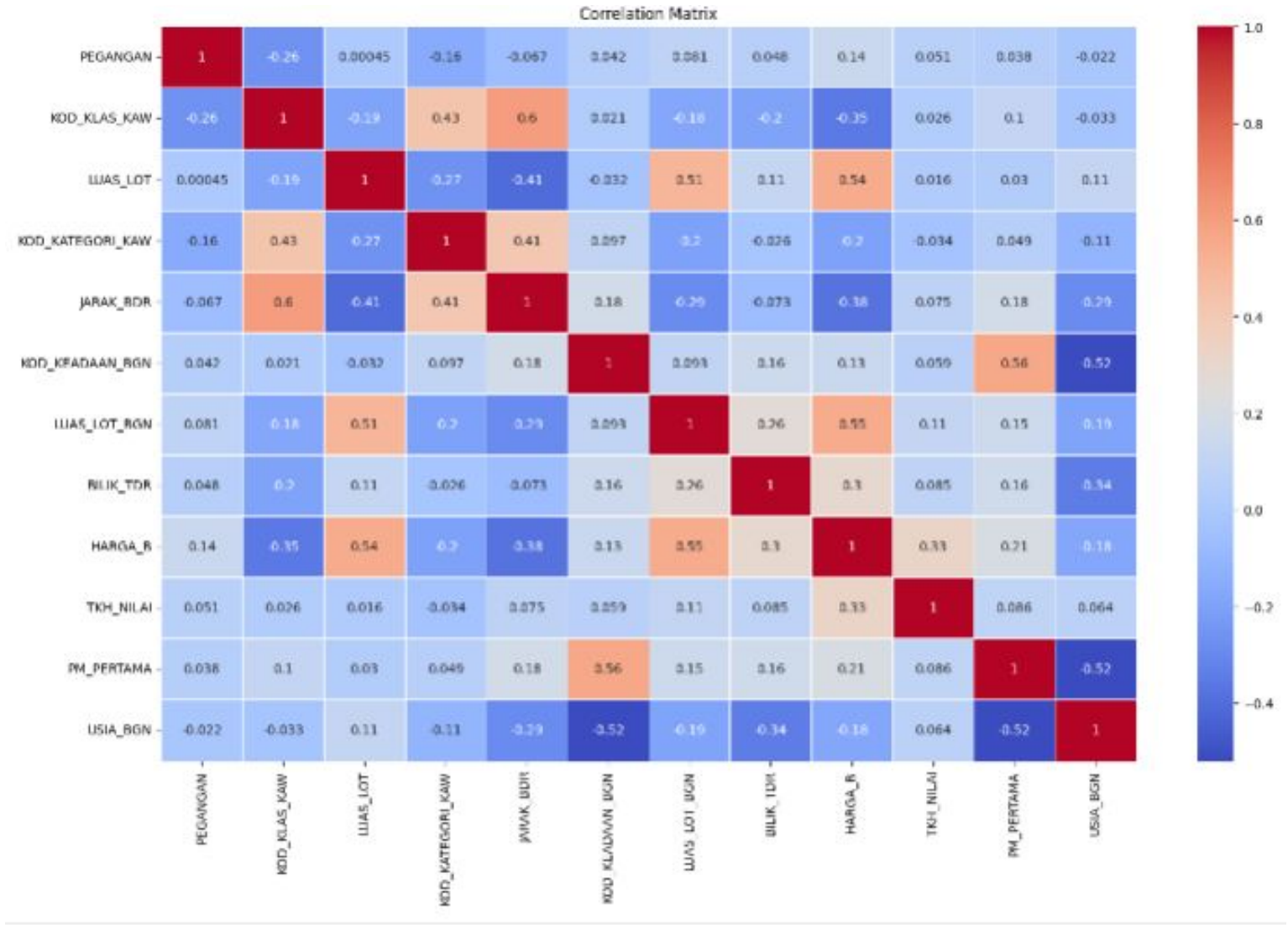
Exploratory Data Analysis – Using Python

Correlation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

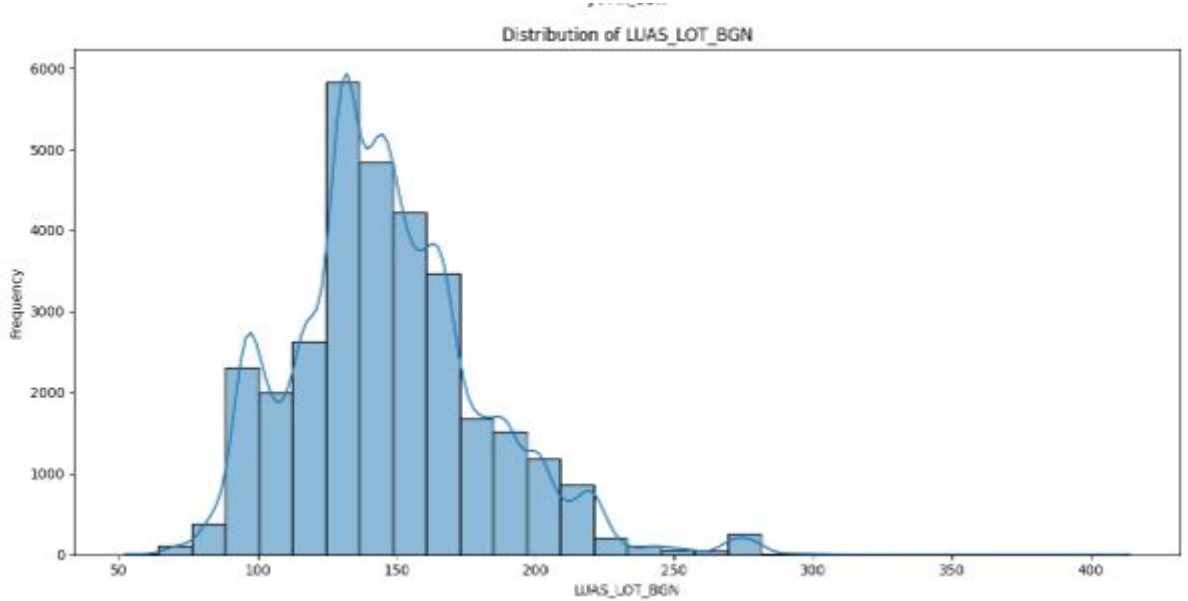
```
# Convert non-numeric columns to numeric where possible for correlation analysis
df_numeric = df.apply(pd.to_numeric, errors='coerce')

# Exclude the 'I_HARTA2' feature by dropping it
df_numeric = df_numeric.drop(columns=['quarter','MUKIM', 'SKIM'])

# Plot the correlation matrix
plt.figure(figsize=(15, 10))
sns.heatmap(df_numeric.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.tight_layout()
plt.show()
```

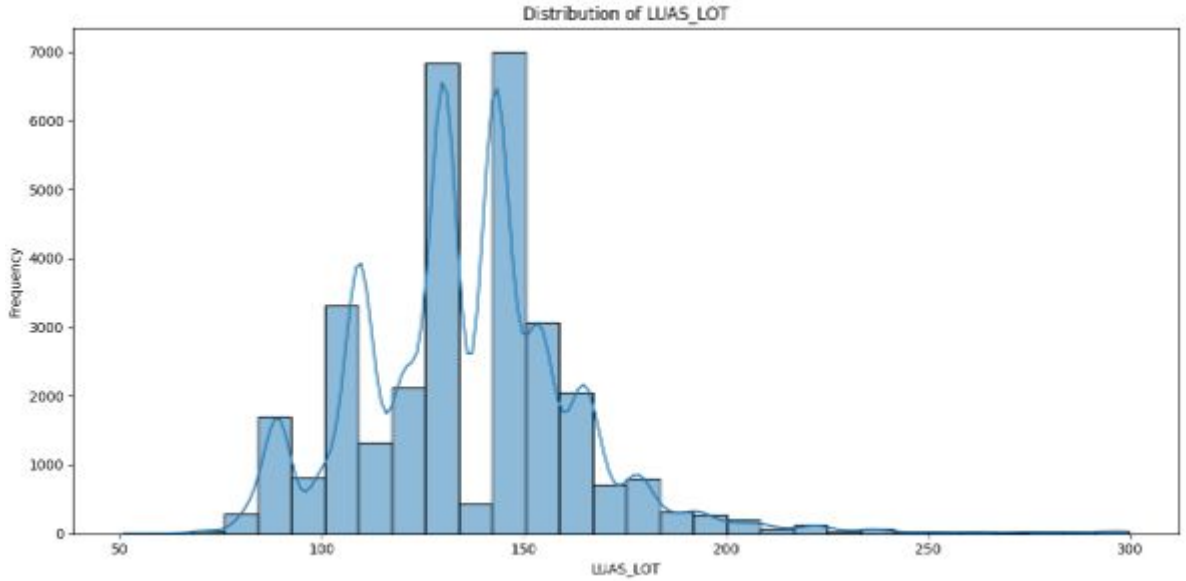
Terraced House

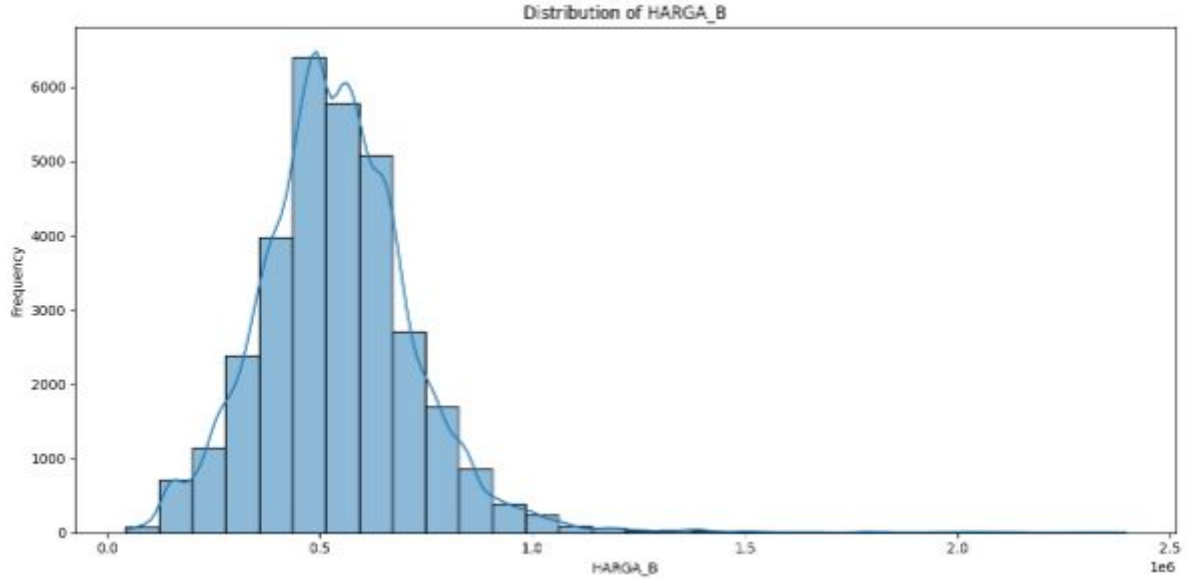


Data Distribution

```python
# 5. Distribution Analysis for All Float Features
float_columns = df.select_dtypes(include=['float64']).columns
for col in float_columns:
    plt.figure(figsize=(12, 6))
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.tight_layout()
    plt.show()
```

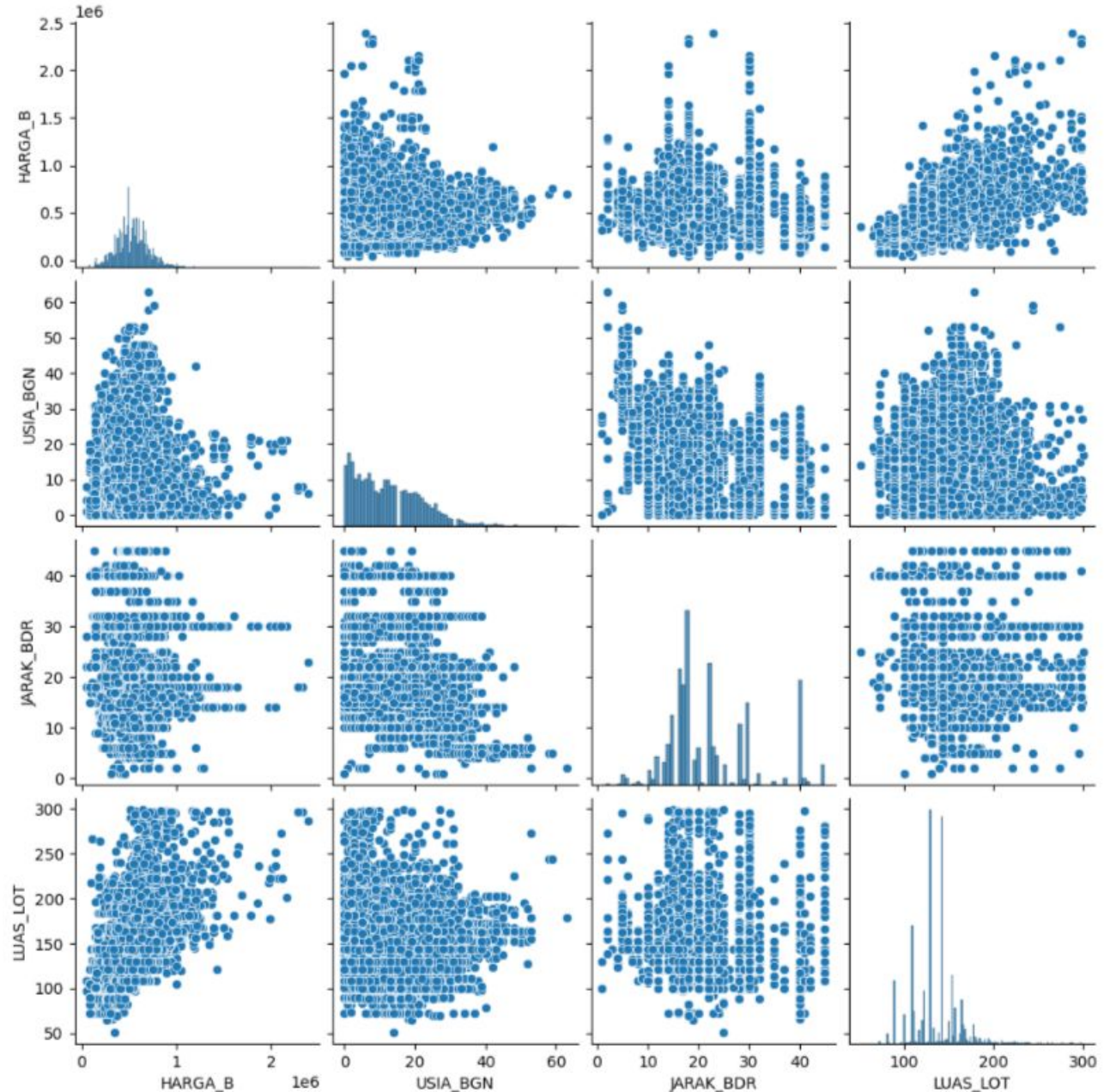






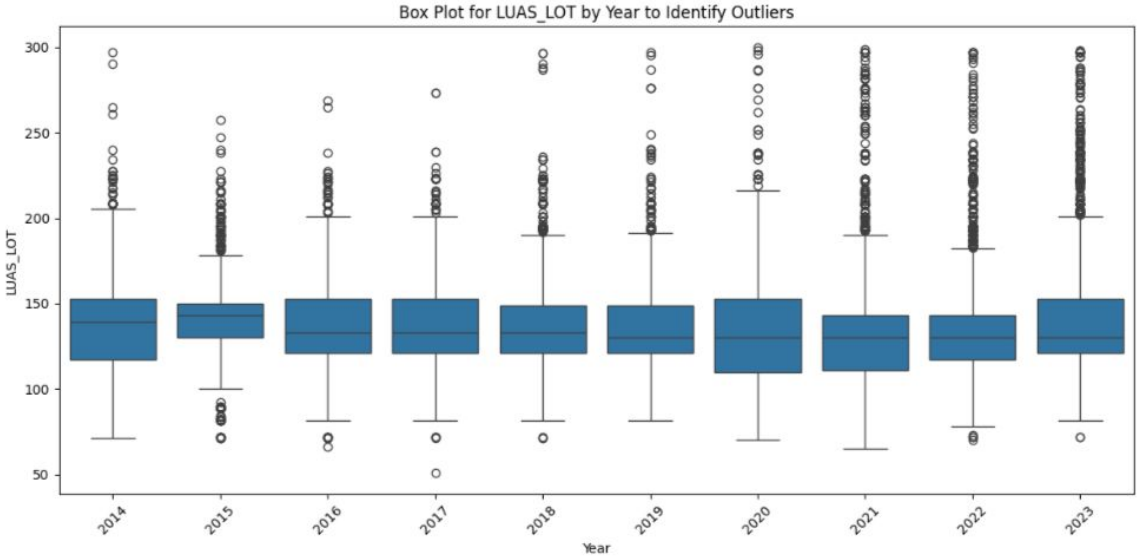
Pairplot

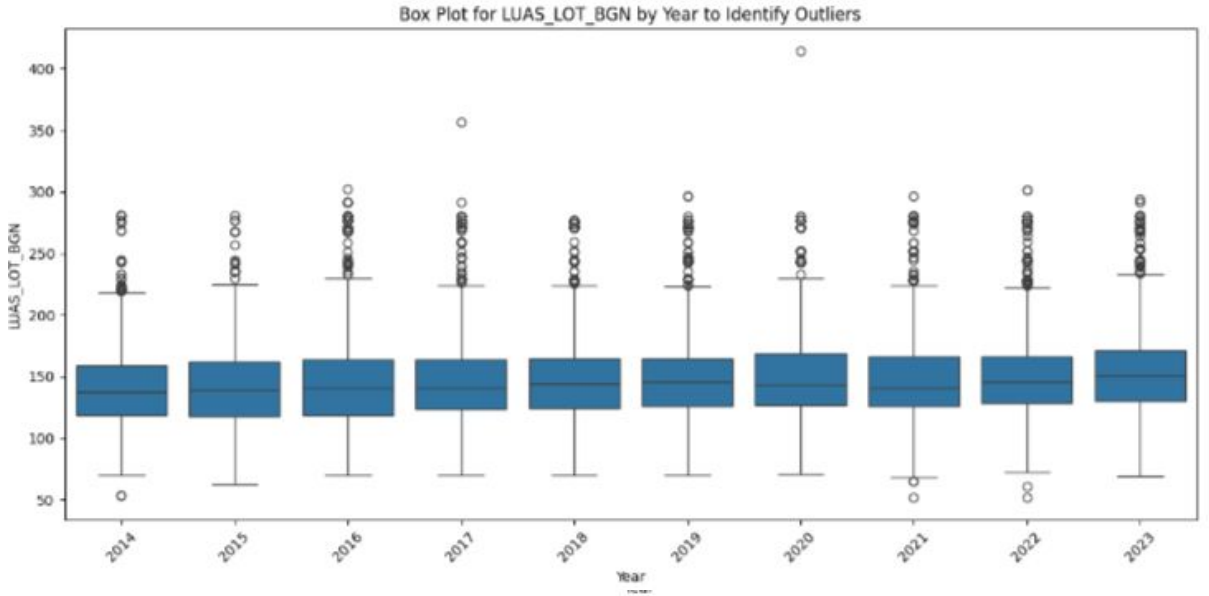
```
# Pair Plot for Selected Features
sns.pairplot(df[['HARGA_B', 'USIA_BGN', 'JARAK_BDR', 'LUAS_LOT']])
plt.show()
```

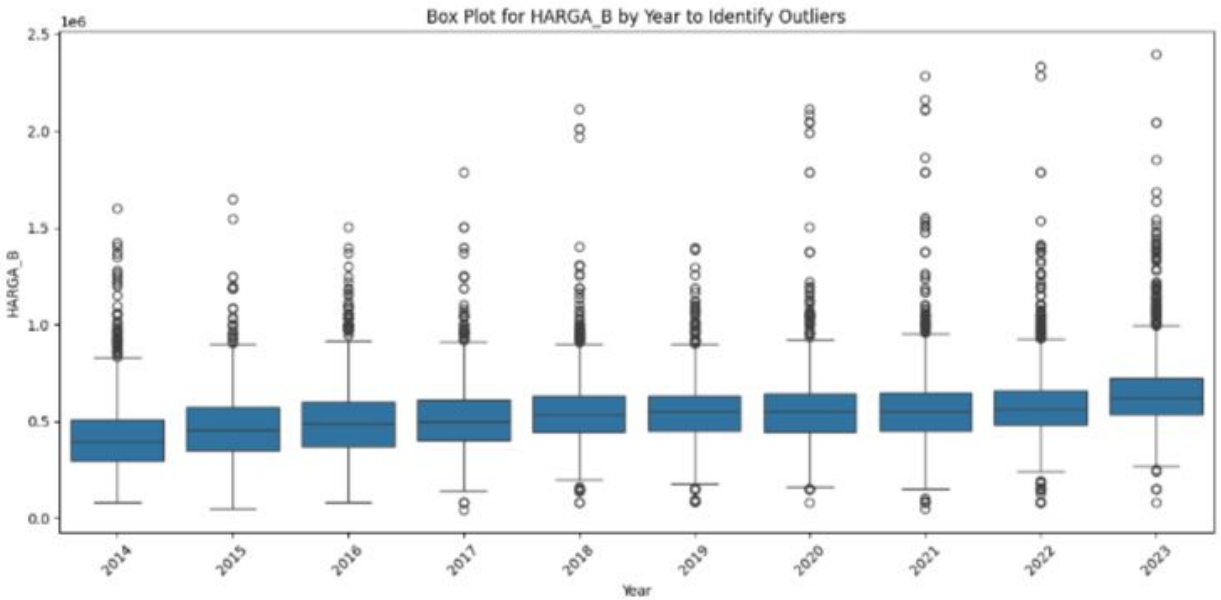


Boxplot

```python
# 1. Box Plot to Visually Inspect Outliers by Year
for col in float_columns:
    plt.figure(figsize=(12, 6))
    sns.boxplot(x='year', y=col, data=df)
    plt.title(f'Box Plot for {col} by Year to Identify Outliers')
    plt.xlabel('Year')
    plt.ylabel(col)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

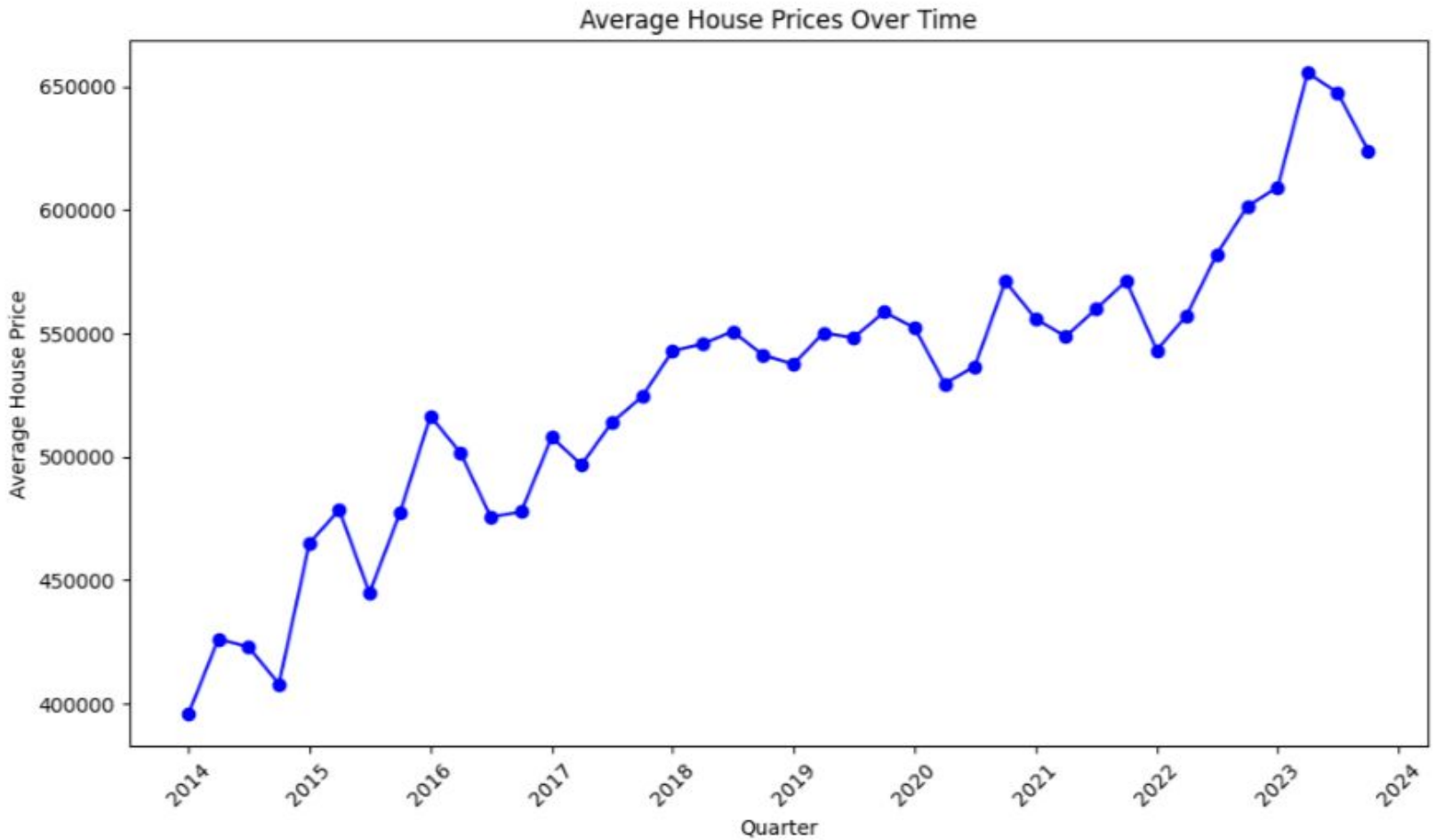




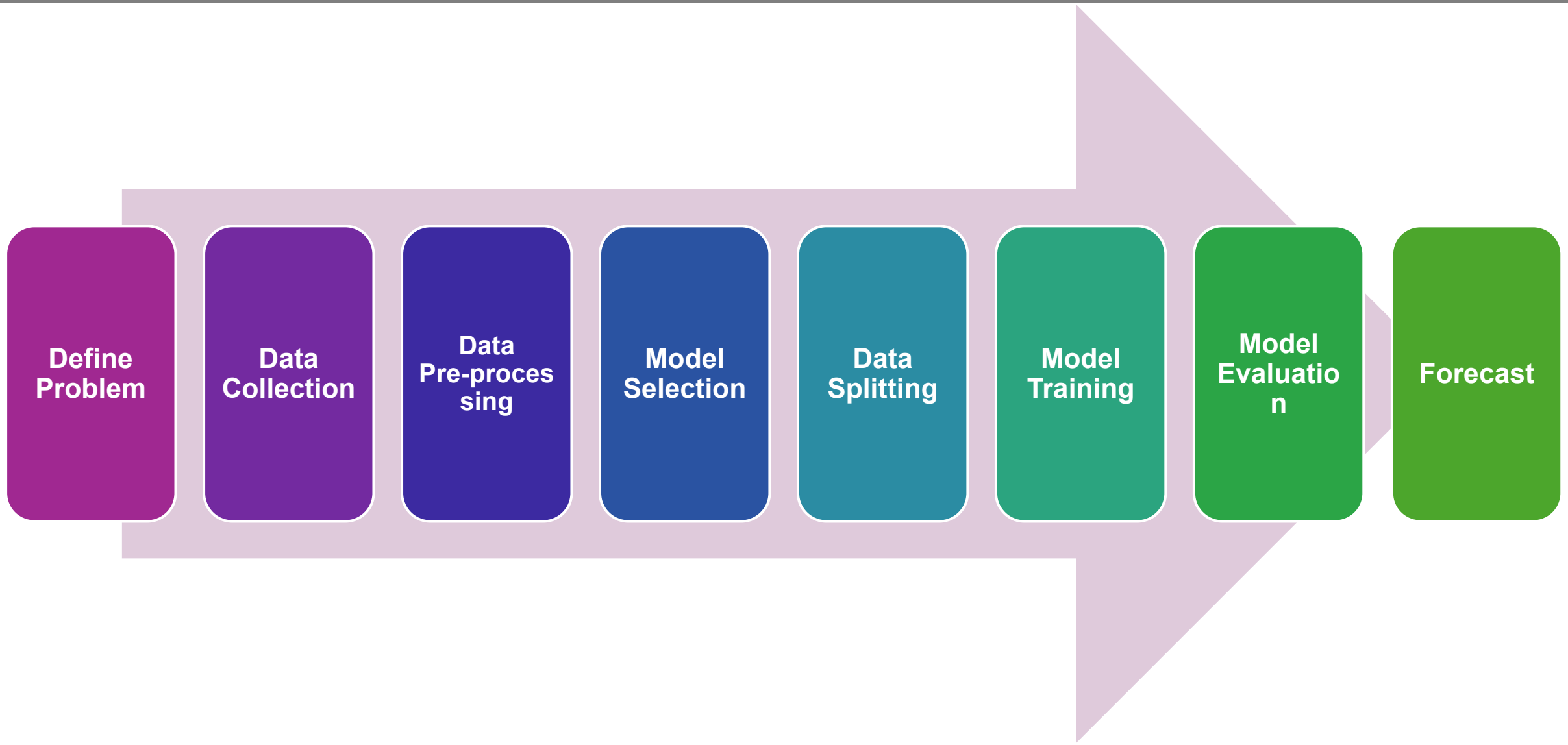


Time Series

```python
#Aggregation by Quarter and Average Price
df['quarter'] = pd.PeriodIndex(df['quarter'], freq='Q').to_timestamp()
df_avg = df.groupby('quarter')['HARGA_B'].mean().reset_index()
# 6. Time Series Plot of House Prices Over Time
plt.figure(figsize=(10, 6))
plt.plot(df_avg['quarter'], df_avg['HARGA_B'], marker='o', linestyle='-',
color='b')
plt.title('Average House Prices Over Time')
plt.xlabel('Quarter')
plt.ylabel('Average House Price')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Machine Learning Using Python



Model Selection- LSTM

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
from sklearn.metrics import mean_squared_error, mean_absolute_error,
root_mean_squared_error
```

```
# Preprocess the data
# Convert 'quarter' to datetime and calculate average house price per quarter
df['quarter'] = df['quarter'].dt.to_period('Q')
df_avg = df.groupby('quarter')['HARGA_B'].mean().reset_index()

```

Feature Scaling

```
# Feature Scaling
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(df_avg['HARGA_B'].values.reshape(-1, 1))

```

Set Random Seed

```
#Set a Random Seed:
import numpy as np
import tensorflow as tf
import random

# Set random seeds for reproducibility
np.random.seed(1)
tf.random.set_seed(1)
random.seed(1)
```

Prepare data for LSTM

```
# Prepare the data for LSTM
n_input = 8
n_features = 1
X, y = [], []
for i in range(n_input, len(data_scaled)):
    X.append(data_scaled[i-n_input:i, 0])
    y.append(data_scaled[i, 0])
X, y = np.array(X), np.array(y)
X = X.reshape((X.shape[0], X.shape[1], n_features))
```

Splitting the Data

```
from sklearn.model_selection import train_test_split

# Fixing random_state ensures consistent splitting
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Split the data into training and testing sets
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

Building the LSTM Model:

```
# Build the LSTM model
model = Sequential()
model.add(LSTM(20, activation='relu', input_shape=(n_input, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not
  super().__init__(**kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 20)	1,760
dense (Dense)	(None, 1)	21

```
Total params: 1,781 (6.96 KB)
Trainable params: 1,781 (6.96 KB)
Non-trainable params: 0 (0.00 B)
```

Training the LSTM model using early stopping

```
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
model.fit(X_train, y_train, epochs=150, batch_size=X_train.shape[0], validation_data=(X_test, y_test), callbacks=[early_stopping])
```

Making Predictions

```
# Make predictions
y_pred_scaled = model.predict(X_test)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 283ms/step
```

Inverse Transforming the Predictions

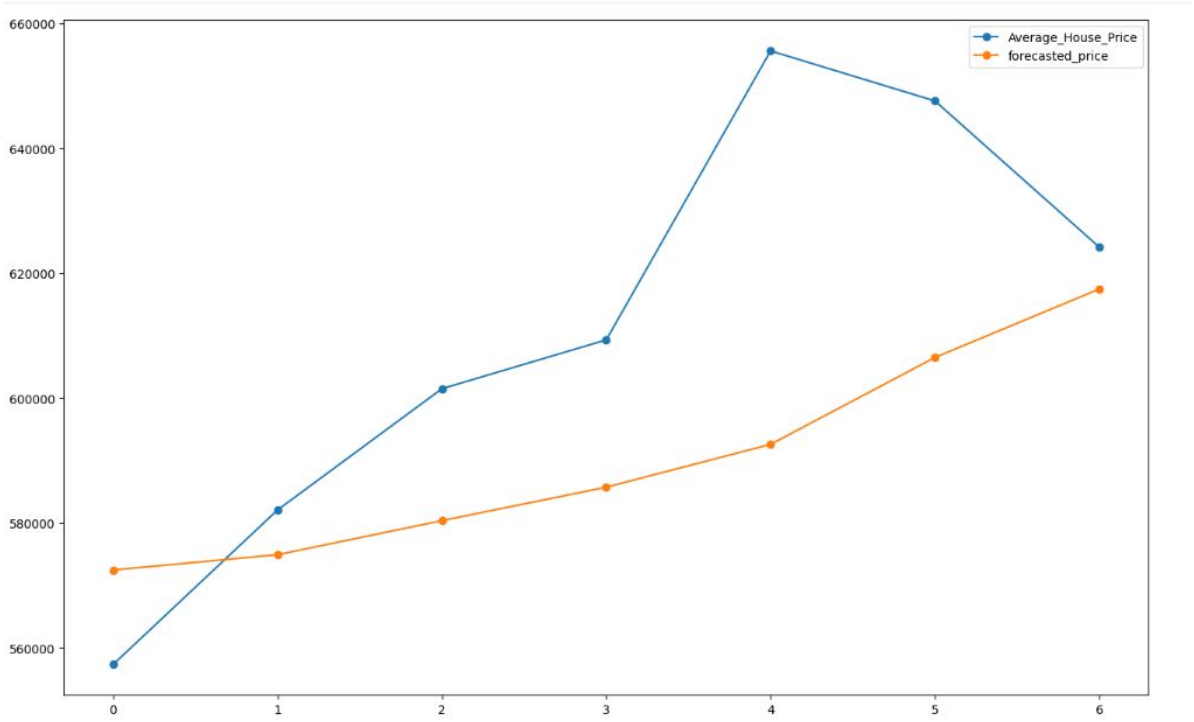
```
# Inverse transform the predictions
y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))
y_pred_inv = scaler.inverse_transform(y_pred_scaled)
```

Model Evaluation

```
# Model Evaluation
mse = mean_squared_error(y_test_inv, y_pred_inv)
mae = mean_absolute_error(y_test_inv, y_pred_inv)
rmse = root_mean_squared_error(y_test_inv, y_pred_inv)
print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")
print(f"Root Mean Squared Error: {rmse}")
```

```
Mean Squared Error: 997314117.9474686
Mean Absolute Error: 25391.10289850234
Root Mean Squared Error: 31580.280523571488
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(16,10))
plt.plot(y_test_inv, label='Average_House_Price', marker='o')
plt.plot(y_pred_inv, label='forecasted_price', marker='o')
plt.legend()
plt.show()
```



Forecasting

```python
# Forecast future values (Q1 2024 - Q4 2024)
last_4_quarters = data_scaled[-n_input:]
forecast = []
input_seq = last_4_quarters.reshape((1, n_input, n_features))
for _ in range(4):
    next_pred_scaled = model.predict(input_seq)[0][0]
    forecast.append(next_pred_scaled)
    input_seq = np.append(input_seq[:, 1:, :], [[[next_pred_scaled]]], axis=1)
```

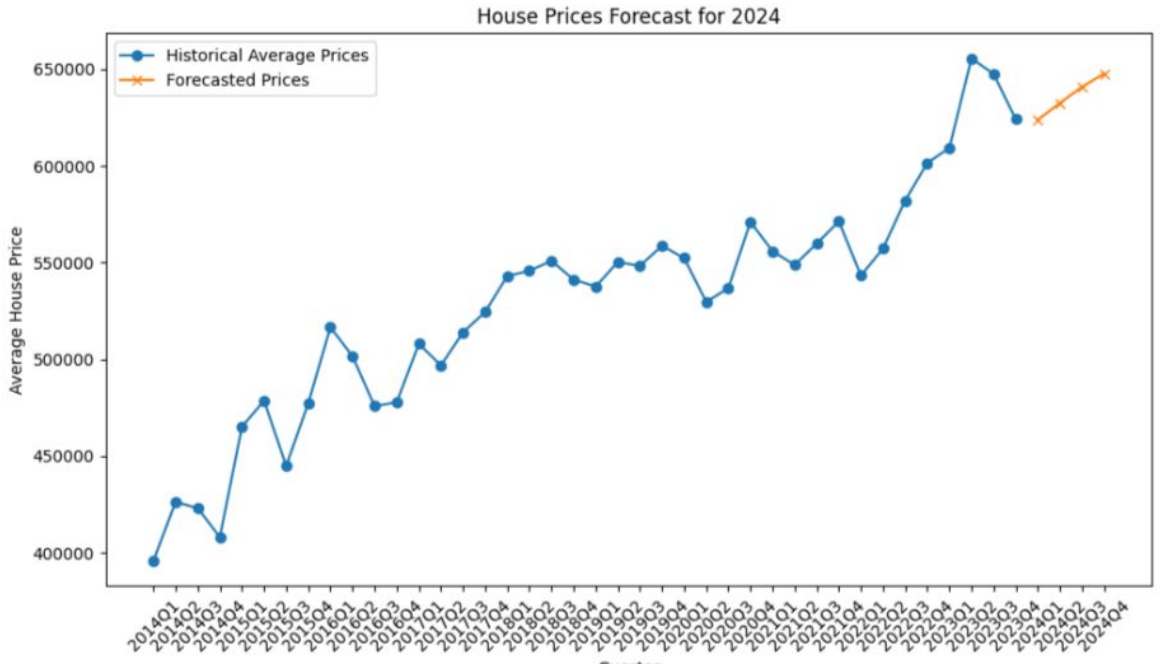
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 309ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
```

```python
forecast_inv = scaler.inverse_transform(np.array(forecast).reshape(-1, 1))
```

```
Start coding or generate with AI.
```

```python
# Display the forecasted values
dates = pd.date_range(start='2024-01-01', periods=4, freq='Q')
forecast_df = pd.DataFrame({'quarter': dates, 'forecasted_price': forecast_inv.flatten()})
print("\nForecasted Average House Prices for 2024:")
print(forecast_df)
```

```
Forecasted Average House Prices for 2024:
     quarter  forecasted_price
0 2024-03-31       623803.1250
1 2024-06-30       632467.1250
2 2024-09-30       640837.5000
3 2024-12-31       647662.3125
```



Model Selection- ARIMA

Stationarity Test using Augmented Dickey-Fuller (ADF) Test

```
from statsmodels.tsa.stattools import adfuller
test_result=adfuller(df_quarterly['Average_Price'])
from statsmodels.tsa.stattools import adfuller

def adfuller_test(price):
    result = adfuller(price)
    labels = ['ADF Test Statistic', 'p-value', '#Lags Used', 'Number of Observations']
    for value, label in zip(result, labels):
        print(label + ' : ' + str(value))

    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis (Ho), reject the null hypothesis. Data is stationary")
    else:
        print("Weak evidence against the null hypothesis, indicating it is non-stationary")

adfuller_test(df_quarterly['Average_Price'])
```

Autocorrelation Analysis

```
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(df_quarterly['Average_Price'])
plt.show()
```

Model Fitting and Model Summary

```
# fit an ARIMA model and plot residual errors
from statsmodels.tsa.arima.model import ARIMA
from matplotlib import pyplot
# fit model
model = ARIMA (df_quarterly['Average_Price'], order = (8,1,0))
model_fit = model.fit()
# summary of fit model
print(model_fit.summary())
```

SARIMAX Results

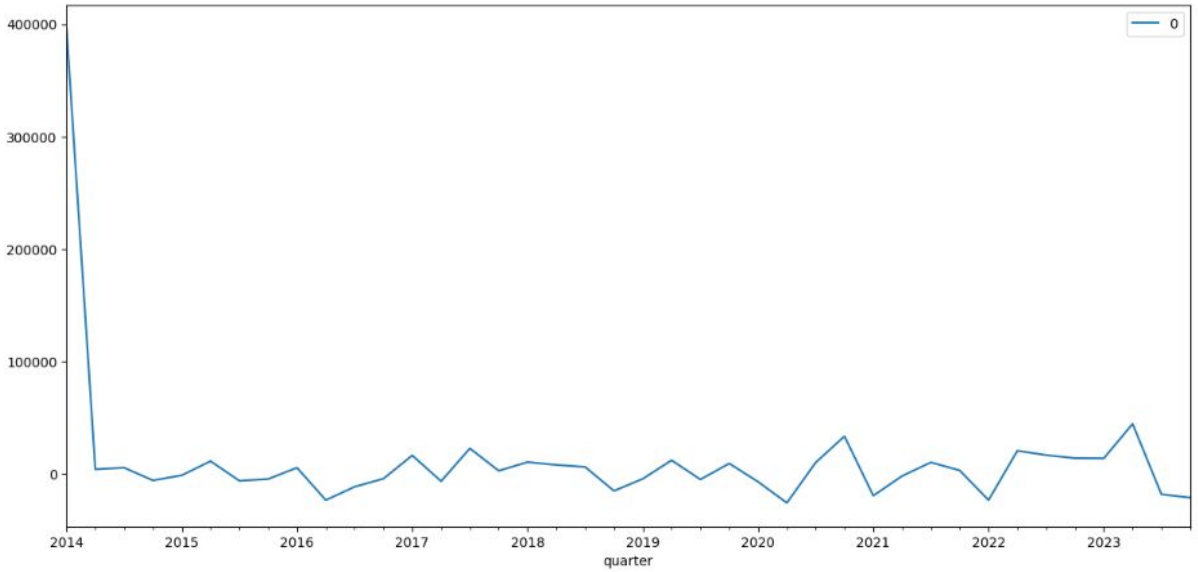
Dep. Variable:	Average_Price	No. Observations:	40
Model:	ARIMA(8, 1, 0)	Log Likelihood	-431.778
Date:	Tue, 22 Oct 2024	AIC	881.556
Time:	02:14:52	BIC	896.528
Sample:	03-31-2014	HQIC	886.928
	- 12-31-2023		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0744	0.104	0.716	0.474	-0.129	0.278
ar.L2	-0.0067	0.078	-0.086	0.931	-0.160	0.146
ar.L3	-0.0516	0.098	-0.526	0.599	-0.244	0.141
ar.L4	0.1486	0.163	0.912	0.362	-0.171	0.468
ar.L5	-0.0094	0.051	-0.186	0.852	-0.109	0.090
ar.L6	-0.0781	0.087	-0.895	0.371	-0.249	0.093
ar.L7	0.1161	0.104	1.115	0.265	-0.088	0.320
ar.L8	0.0489	0.098	0.501	0.617	-0.143	0.240
sigma2	2.324e+08	2.83e-10	8.22e+17	0.000	2.32e+08	2.32e+08

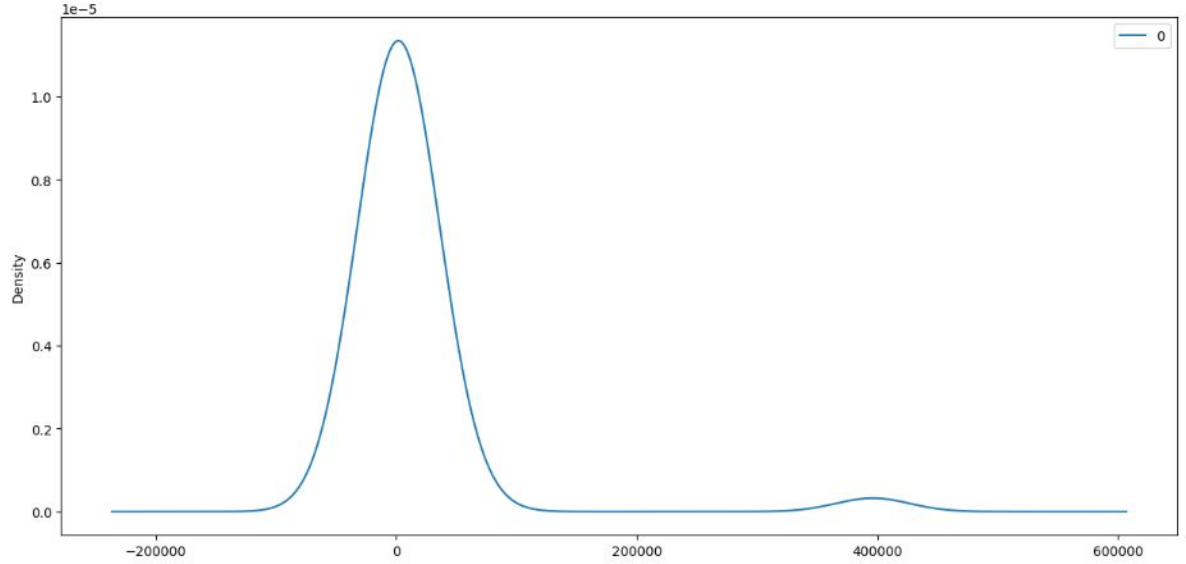
Ljung-Box (L1) (Q):	0.07	Jarque-Bera (JB):	0.67
Prob(Q):	0.80	Prob(JB):	0.72
Heteroskedasticity (H):	4.66	Skew:	0.31
Prob(H) (two-sided):	0.01	Kurtosis:	3.20

Residual Analysis (Model Diagnostics)

```
1  # line plot of residuals
2  from pandas import DataFrame
3  residuals = DataFrame(model_fit.resid)
4  residuals.plot()
5  pyplot.show()
6  # density plot of residuals
7  residuals.plot(kind='kde')
8  pyplot.show()
9  # summary stats of residuals
10 print (residuals.describe())
```




```
                0
count      40.000000
mean    11967.835179
std     64149.418744
min    -25615.459377
25%     -6076.715142
50%      3748.866944
75%     11691.463490
max    396015.222834
```



Import Libraries

```python
from sklearn.metrics import mean_squared_error
from math import sqrt
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
```

Split the Data into Train and Test Sets

```python
# Split the data into train and test sets
X = df_quarterly['Average_Price']
size = int(len(X) * 0.80)

# Reset the index to avoid index mismatch errors
train, test = X.iloc[:size].reset_index(drop=True), X.iloc[size:].reset_index(drop=True)
```

Initialize Variables

```python
# Initialize history with all observations from the training data
history = list(train)  # Use all the data points for forecasting
predictions = []  # List to store predictions

```

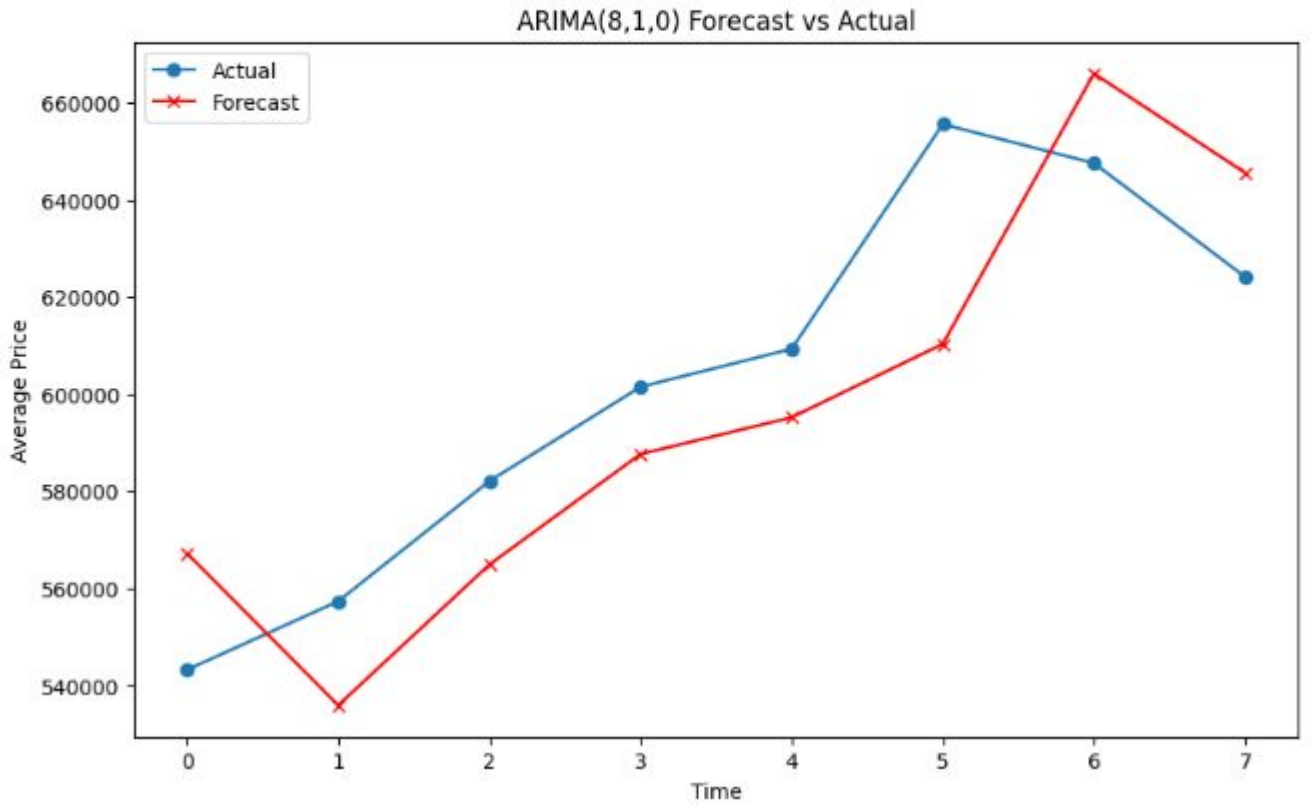
Walk-Forward Validation

```python
# Walk-forward validation using ARIMA(8,1,0)
for t in range(len(test)):
    # Use the entire history for training the ARIMA model
    model = ARIMA(history, order=(8, 1, 0))
    model_fit = model.fit()

    # Forecast the next step (1 quarter ahead)
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)  # Store the prediction

    # Get the actual observation from the test set
    obs = test[t]

    # Add the observed value to the history (update for next iteration)
    history.append(obs)

    print(f'predicted={yhat:.3f}, expected={obs:.3f}')

```

Evaluate Model

```python
# Evaluate the forecasts
rmse = sqrt(mean_squared_error(test, predictions))
print(f'Test RMSE: {rmse:.3f}')

# Plot the forecasts against actual outcomes
plt.figure(figsize=(10, 6))
plt.plot(test, label='Actual', marker='o')
plt.plot(predictions, color='red', marker='x', label='Forecast')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Average Price')
plt.title('ARIMA(8,1,0) Forecast vs Actual')
plt.show()
```

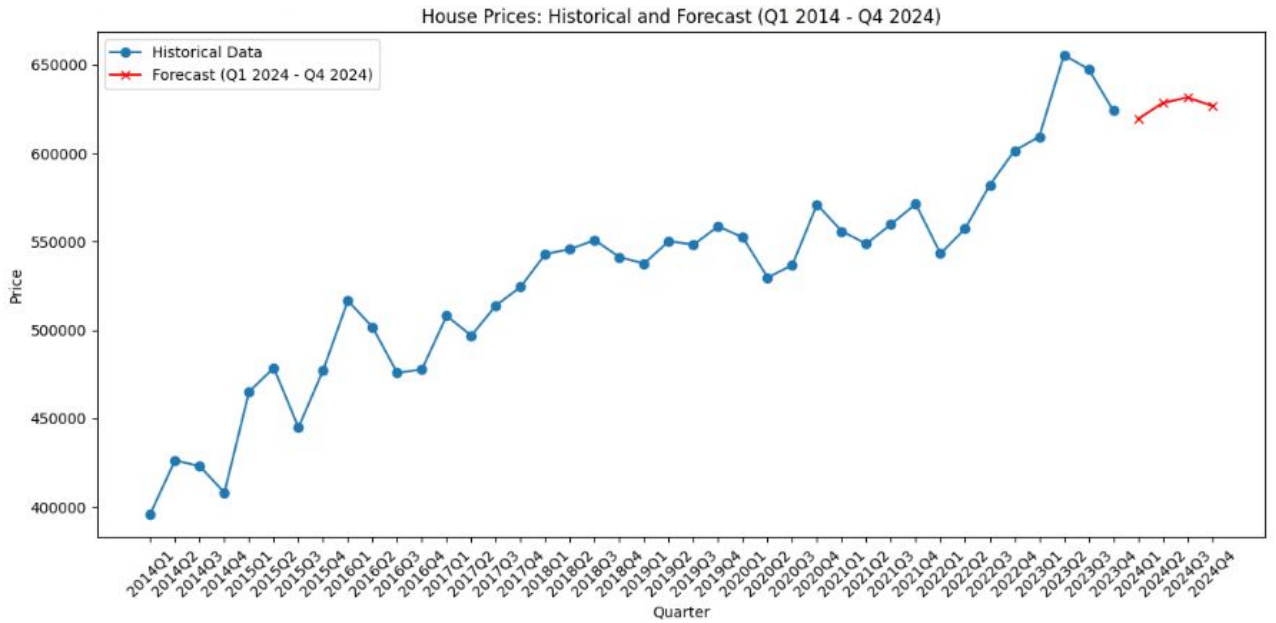
```
predicted=567180.612, expected=543211.790
predicted=535896.079, expected=557318.515
predicted=564855.965, expected=582047.627
predicted=587614.509, expected=601457.896
predicted=595170.976, expected=609277.379
predicted=610294.738, expected=655543.045
predicted=665980.291, expected=647528.498
predicted=645553.269, expected=624110.540
Test RMSE: 23893.715
```



Forecast

```
# Forecast for the next 4 quarters (Q1 2024 - Q4 2024)
for t in range(4):
    # Train ARIMA model using the entire dataset
    model = ARIMA(X, order=(8, 1, 0))  # Set ARIMA order to (8, 1, 0)
    model_fit = model.fit()

    # Forecast the next step (1 quarter ahead)
    output = model_fit.forecast()
    yhat = output.iloc[0]  # Use .iloc to access the forecasted value
    future_predictions.append(yhat)  # Store forecasted value

    # Update X with the forecast value
    X = pd.concat([X, pd.Series([yhat])], ignore_index=True)

# Create a DataFrame for the forecasted quarters
future_quarters = pd.date_range(start='2024-01-01', periods=4, freq='Q').
to_period('Q')
forecast_summary = pd.DataFrame({
    'Quarter': future_quarters.astype(str),
    'Forecasted_Price': future_predictions
})

# Display the forecast summary
print(forecast_summary)
```

```
<ipython-input-19-a0ab898565ac>:26: FutureWarning: 'Q'
  future_quarters = pd.date_range(start='2024-01-01',
  Quarter  Forecasted_Price
0  2024Q1     619345.020354
1  2024Q2     628411.675694
2  2024Q3     631554.074747
3  2024Q4     626806.810336
```



ETS Model

Augmented Dickey-Fuller (ADF) Test

```
from statsmodels.tsa.stattools import adfuller
test_result=adfuller(df_quarterly['Average_Price'])
from statsmodels.tsa.stattools import adfuller

def adfuller_test(sales):
    result = adfuller(sales)
    labels = ['ADF Test Statistic', 'p-value', '#Lags Used', 'Number of Observations']
    for value, label in zip(result, labels):
        print(label + ' : ' + str(value))

    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis (Ho), reject the null hypothesis. Data is stationary")
    else:
        print("Weak evidence against the null hypothesis, indicating it is non-stationary")

adfuller_test(df_quarterly['Average_Price'])

```

Autocorrelation

```
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(df_quarterly['Average_Price'])
plt.show()
```

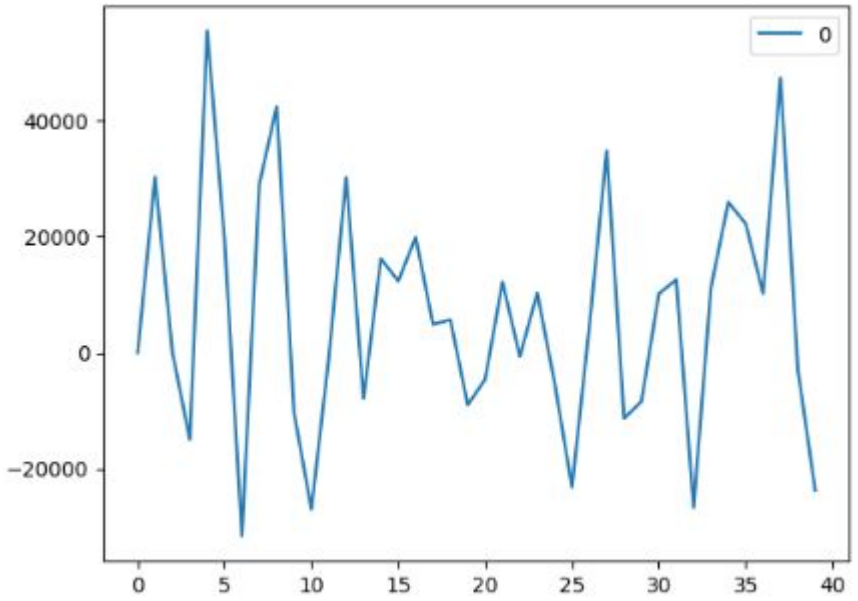
Simple Exponential Smoothing

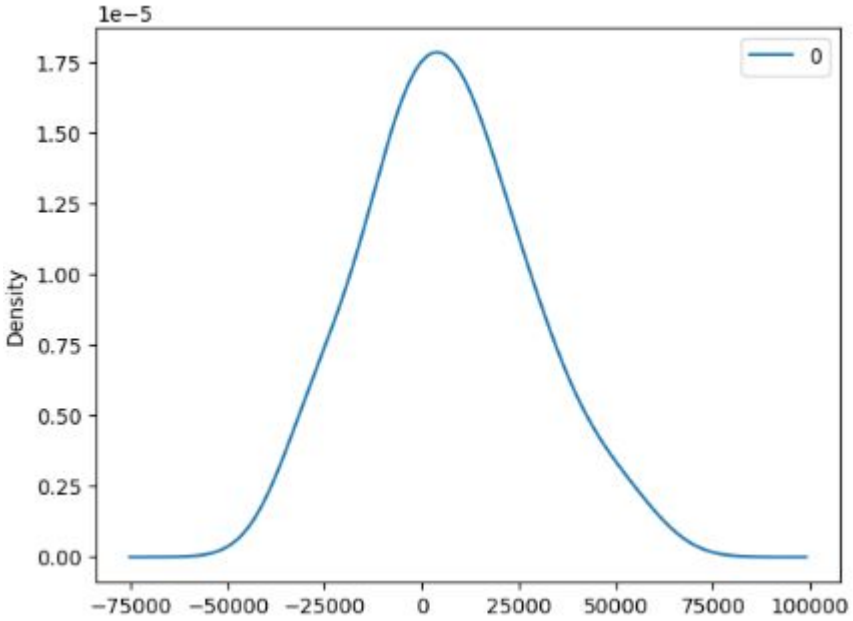
```
# fit an ETS model and plot residual errors
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
from matplotlib import pyplot
# fit model
model = SimpleExpSmoothing(df_quarterly['Average_Price'])
model_fit = model.fit()
# summary of fit model
print(model_fit.summary())
```

```
                       SimpleExpSmoothing Model Results
==============================================================================
Dep. Variable:          Average_Price   No. Observations:                   40
Model:             SimpleExpSmoothing   SSE                 18454956483.145
Optimized:                       True   AIC                         801.989
Trend:                           None   BIC                         805.367
Seasonal:                        None   AICC                        803.132
Seasonal Periods:                None   Date:               Sun, 20 Oct 2024
Box-Cox:                        False   Time:                       02:51:30
Box-Cox Coeff.:                  None
==============================================================================
                        coeff                 code              optimized
------------------------------------------------------------------------------
smoothing_level           0.8913953               alpha                 True
initial_level             3.9602e+05                1.0                 False
------------------------------------------------------------------------------
```

Residual analysis

```
# line plot of residuals
from pandas import DataFrame
residuals = DataFrame(model_fit.resid)
residuals.plot()
pyplot.show()
# density plot of residuals
residuals.plot(kind='kde')
pyplot.show()
# summary stats of residuals
print (residuals.describe())
```





	0
count	40.000000
mean	6469.420624
std	20743.130067
min	-31617.765246
25%	-8024.474056
50%	5272.182820
75%	19578.107051
max	55488.378880

Import Libraries

```
from sklearn.metrics import mean_squared_error
from math import sqrt
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
```

Split Data into Train and Test Sets

```
# Split into train and test sets
X = df_quarterly['Average_Price']
size = int(len(X) * 0.8)
train, test = X[0:size], X[size:len(X)]

# Convert to list to manage data easily for walk-forward validation
history = list(train)
predictions = list()
```

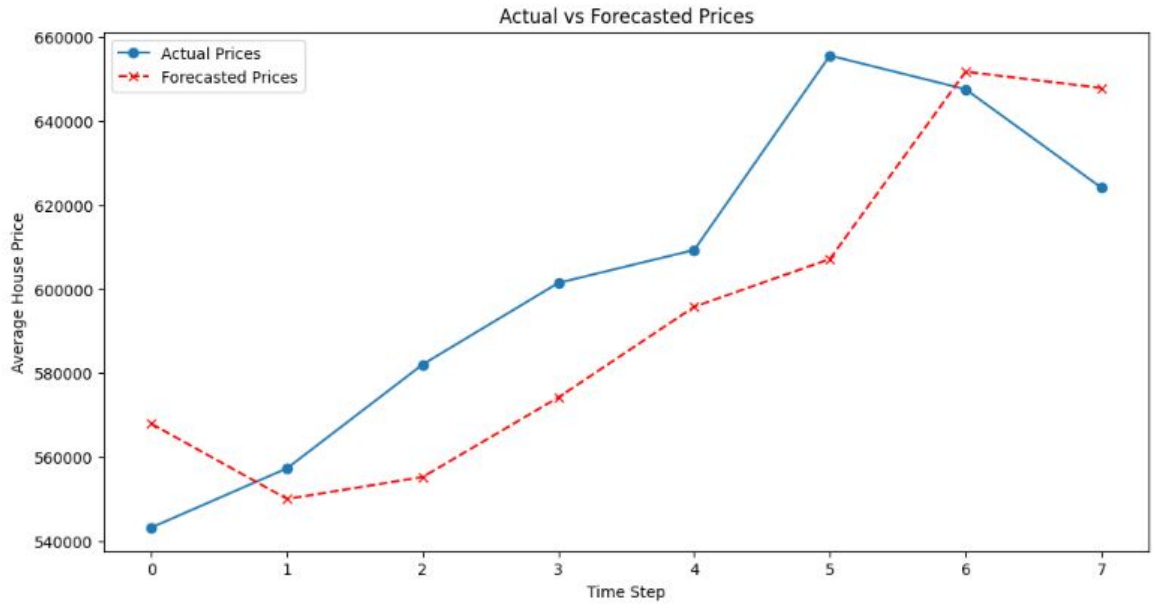
Walk-Forward Validation

```
# Walk-forward validation
for t in range(len(test)):
    # Fit the model to the current history
    model = SimpleExpSmoothing(history)
    model_fit = model.fit(optimized=True)  # Adding optimization for better convergence

    # Forecast next value
    output = model_fit.forecast(1)  # Forecasting 1 step ahead
    yhat = output[0]  # Accessing the value directly from NumPy array

    # Append the prediction to the predictions list
    predictions.append(yhat)

    # Add the observed value to history to continue forecasting
    obs = test.iloc[t]  # Use iloc to access value safely from pandas Series
    history.append(obs)

    # Print predictions for debugging purposes
    print('predicted=%f, expected=%f' % (yhat, obs))
```

Evaluate Model

```
# Evaluate forecasts
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)

# Plot forecasts against actual outcomes
plt.figure(figsize=(12, 6))
plt.plot(test.values, label='Actual Prices', marker='o')
plt.plot(predictions, label='Forecasted Prices', color='red', linestyle='--', marker='x')
plt.legend()
plt.xlabel('Time Step')
plt.ylabel('Average House Price')
plt.title('Actual vs Forecasted Prices')
plt.show()
```

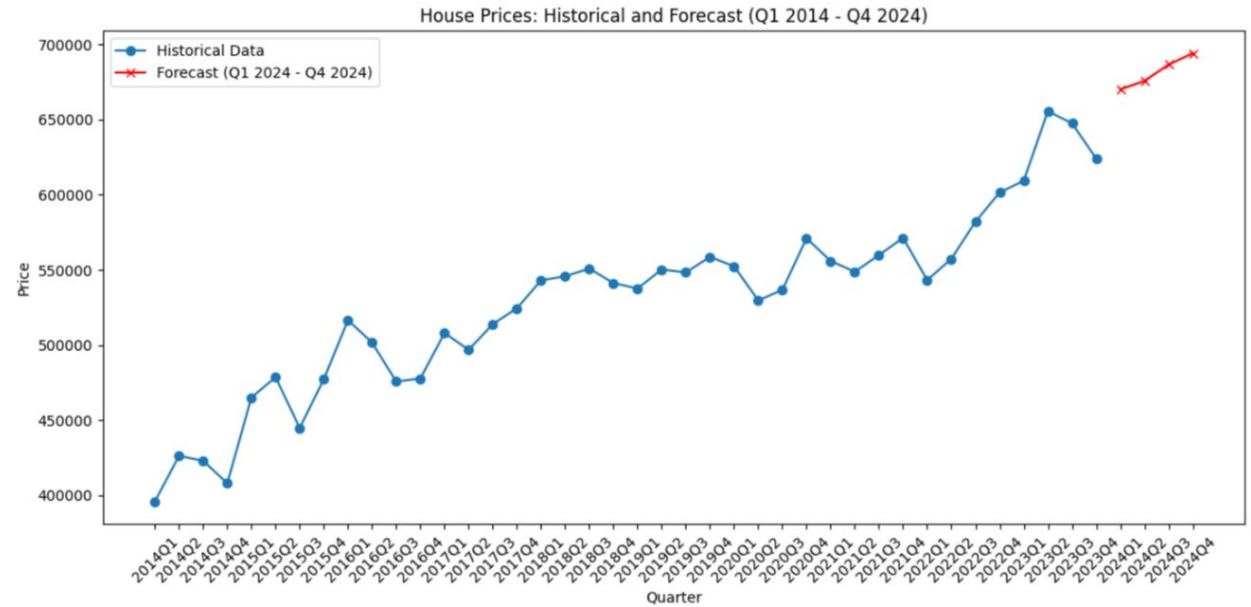
```
  warnings.warn(
predicted=567913.617954, expected=543211.789557
predicted=550009.009561, expected=557318.514653
predicted=555244.240519, expected=582047.626710
predicted=574199.844567, expected=601457.895623
predicted=595753.598364, expected=609277.379430
predicted=607125.167011, expected=655543.045094
predicted=651710.819737, expected=647528.498127
predicted=647840.103976, expected=624110.539959
Test RMSE: 25575.378
```



Forecast

```python
# Forecast for the next 4 quarters (Q1 2024 - Q4 2024)
for t in range(4):
    # Train ETS (Exponential Smoothing) model using only the last 8 observations
    model = ExponentialSmoothing(history[-8:], trend='add', seasonal=None)
    model_fit = model.fit(optimized=True)

    # Forecast the next step (1 quarter ahead)
    output = model_fit.forecast(1)
    yhat = output[0]
    future_predictions.append(yhat)  # Store forecasted value

    # Update history with the forecast (sliding window of 8)
    history = history[1:] + [yhat]  # Keep only the last 8 points

# Create a DataFrame for the forecasted quarters
future_quarters = pd.date_range(start='2024-01-01', periods=4, freq='Q').
to_period('Q')
forecast_summary = pd.DataFrame({
    'Quarter': future_quarters.astype(str),
    'Forecasted_Price': future_predictions
})

# Display the forecast summary
print(forecast_summary)
```

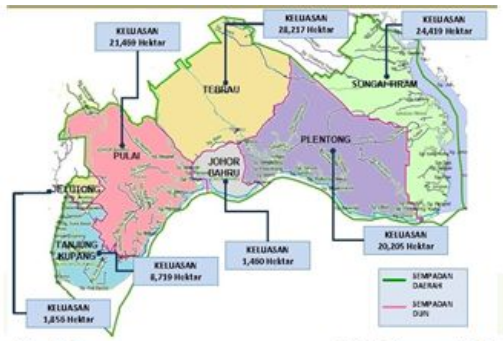
```
  Quarter  Forecasted_Price
0  2024Q1     670006.384177
1  2024Q2     675710.192754
2  2024Q3     686535.519179
3  2024Q4     694056.537357
```



Comparison Model Of Machine learning

Model Type	LSTM (Long Short-Term Memory)	ARIMA (AutoRegressive Integrated Moving Average)	ETS (Exponential Smoothing)
Model Type Definition	LSTM is a type of recurrent neural network used to capture long-term dependencies in sequential data, useful for time series with complex patterns.	ARIMA is a linear model that combines autoregression, differencing, and moving average components to forecast future values in a time series.	ETS methods focus on capturing level, trend, and seasonal components in time series through exponential smoothing.
Model Complexity	High complexity: LSTM requires extensive training and tuning, making it less suitable for a simple dataset with just quarterly averages.	Moderate complexity: ARIMA requires parameter tuning for (p, d, q) values and is well-suited to handle quarterly time series data if it shows a clear trend or seasonality.	Low complexity: ETS is simple and easy to implement, especially for time series that have level, trend, or seasonality without much noise.
Use Cases	Less ideal for quarterly average price data unless there are complex, non-linear patterns in the data. LSTM is usually better suited when more features or dependencies are present.	Well-suited for quarterly average price data with clear trends or seasonal components; ARIMA handles time series data effectively when made stationary.	Very suitable for quarterly average price data with clear seasonality or trends. ETS methods like Holt-Winters are straightforward to apply and interpret in business settings.
Test RMSE	2S Terraced 31,580.28 Highrise 29,448.73 AllHouse 49,792.10	2S Terraced 23,893.72 Highrise 22,171.58 AllHouse 26,410.61	2S Terraced 25,022.35 Highrise 22,123.42 AllHouse 29,935.62

Dashboard– Using Tableau



Johor Bahru Property Market Overview: Boundary Map & Market Activity Trend

All House

Mukim	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023
Bandar Johor Bahru	608	457	443	472	520	473	422	372	576	851
Mukim Jelutong	99	71	90	110	129	108	75	73	124	118
Mukim Plentong	3,508	2,996	3,273	3,111	3,295	3,165	2,970	3,266	4,029	4,311
Mukim Pulai	3,498	3,005	2,896	3,262	3,463	2,981	2,708	3,242	5,110	5,893
Mukim Tebrau	2,651	2,192	2,444	2,405	2,518	2,336	1,869	2,432	3,021	4,201
Mukim Tg Kupang	10	15	24	40	44	62	83	23	40	69

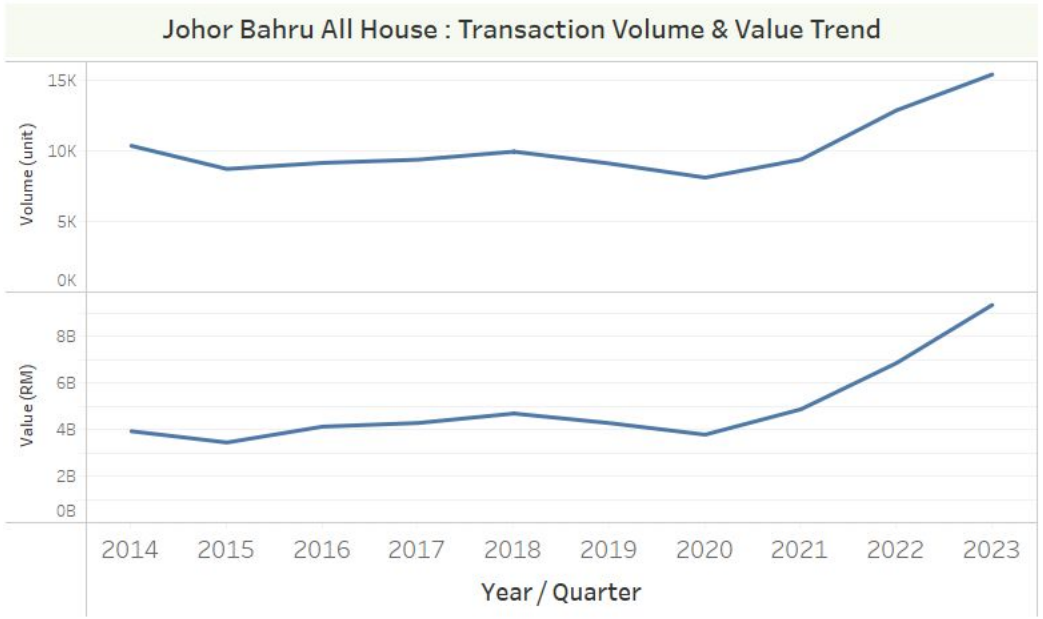
2-Storey Terraced House

Mukim	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023
Bandar Johor Bahru	48	38	45	41	43	27	23	22	39	70
Mukim Jelutong	18	3	6	1	6	11	4	4	6	4
Mukim Plentong	860	684	960	834	877	862	719	985	1,401	1,291
Mukim Pulai	1,016	896	1,045	1,124	1,199	1,047	830	1,159	1,884	2,075
Mukim Tebrau	899	775	949	857	843	802	710	843	1,132	1,669
Mukim Tg Kupang		1						1		

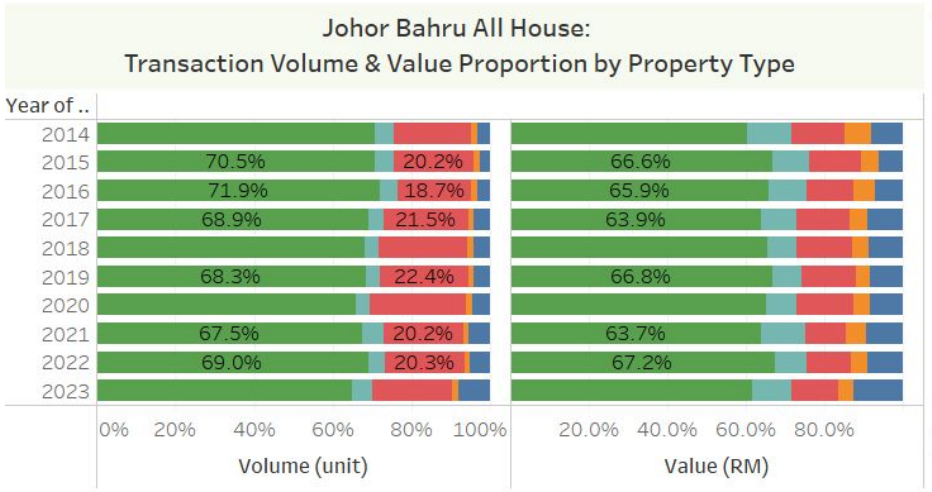
Highrise Unit

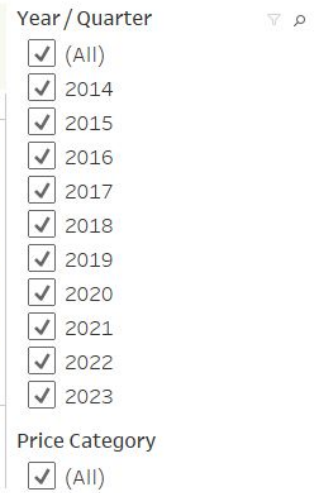
Mukim	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023
Mukim Pulai	421	338	347	419	468	351	330	299	493	598
Mukim Plentong	497	423	320	332	374	299	307	278	405	540
Bandar Johor Bahru	268	193	200	199	234	196	201	159	251	300
Mukim Tebrau	168	166	182	216	215	231	135	179	221	267
Mukim Tg Kupang	3	7	5	7	13	19	13	10	18	35

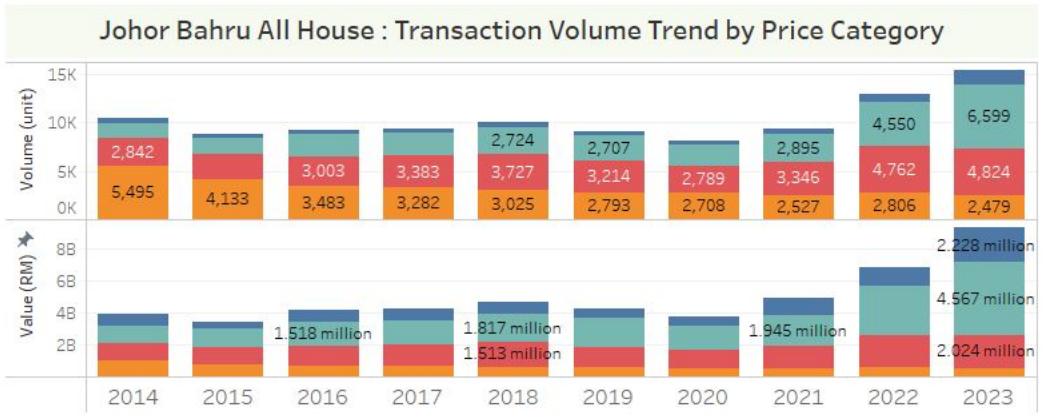
ALL HOUSE

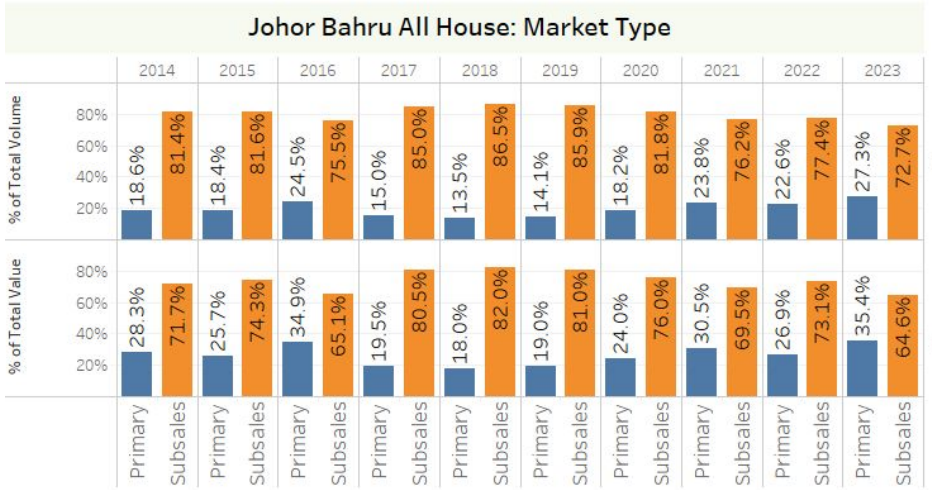

Johor Bahru All House : Transaction Volume & Value Trend
Volume (unit)
15K
10K
5K
0K
Value (RM)
8B
6B
4B
2B
0B
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
Year / Quarter


Year / Quarter
2014
2023
Price Category
(All)
Above RM 1Mil
Below RM300,000
RM300,001 - RM500,000
RM500,001 - RM 1Mil
Price Category
Above RM 1Mil
RM500,001 - RM 1Mil
RM300,001 - RM500,000
Below RM300,000

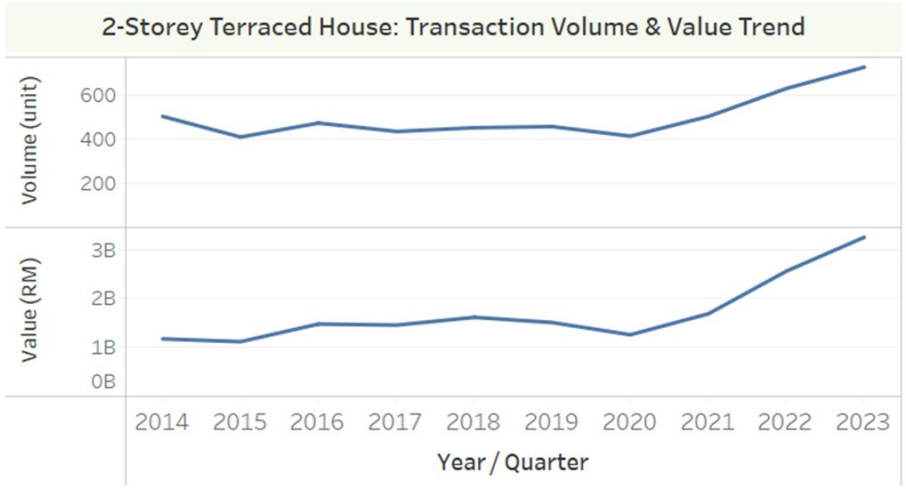

Johor Bahru All House:
Transaction Volume & Value Proportion by Property Type
Year of ..
2014
2015
70.5%
20.2%
66.6%
2016
71.9%
18.7%
65.9%
2017
68.9%
21.5%
63.9%
2018
2019
68.3%
22.4%
66.8%
2020
2021
67.5%
20.2%
63.7%
2022
69.0%
20.3%
67.2%
2023
0%
20%
40%
60%
80%
100%
Volume (unit)
20.0%
40.0%
60.0%
80.0%
Value (RM)

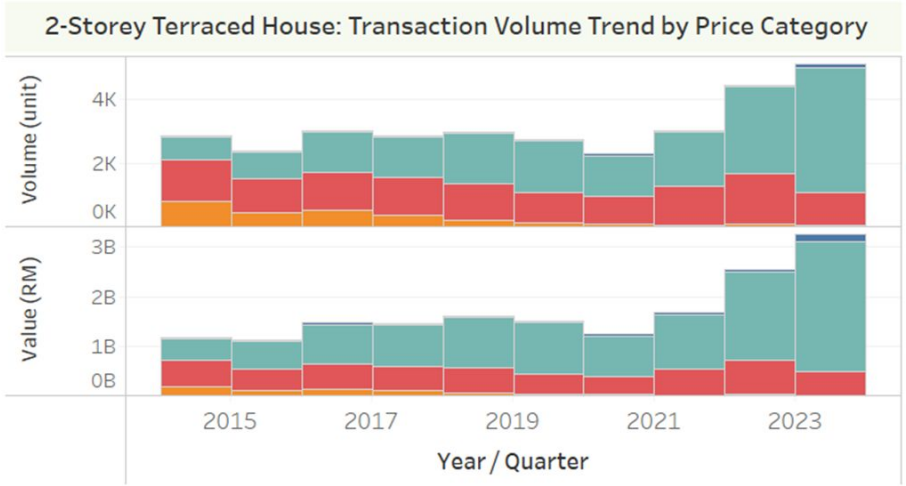

Year / Quarter
(All)
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
Price Category
(All)
Above RM 1Mil
Below RM300,000
RM300,001 - RM500,000
RM500,001 - RM 1Mil
PropType
Clustered
Detached
Highrise
Semi Detached
Terraced
Market Type
Primary
Subsales

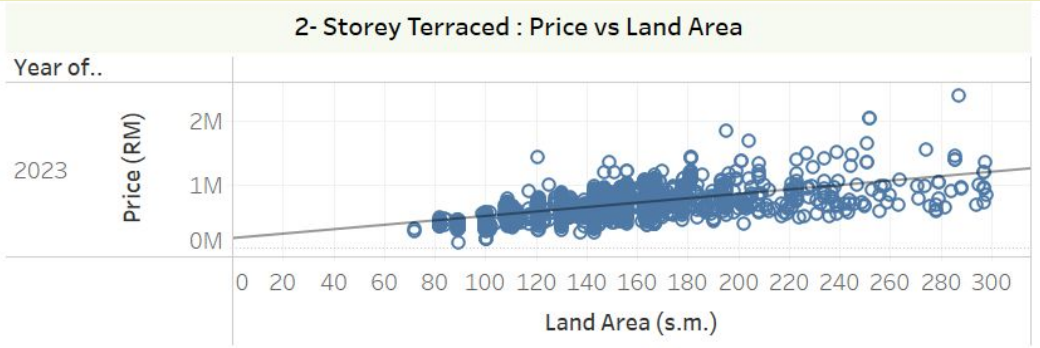

Johor Bahru All House : Transaction Volume Trend by Price Category
Volume (unit)
15K
10K
5K
0K
2,842
5,495
4,133
3,003
3,483
3,383
3,282
2,724
3,727
3,025
2,707
3,214
2,793
2,789
2,708
2,895
3,346
2,527
4,550
4,762
2,806
6,599
4,824
2,479
Value (RM)
8B
6B
4B
2B
1.518 million
1.817 million
1.513 million
1.945 million
2.228 million
4.567 million
2.024 million
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023

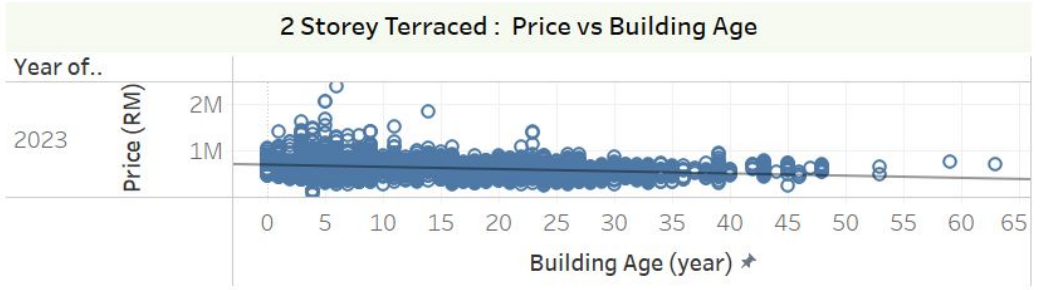

Johor Bahru All House: Market Type
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
% of Total Volume
80%
60%
40%
20%
18.6%
81.4%
18.4%
81.6%
24.5%
75.5%
15.0%
85.0%
13.5%
86.5%
14.1%
85.9%
18.2%
81.8%
23.8%
76.2%
22.6%
77.4%
27.3%
72.7%
% of Total Value
28.3%
71.7%
25.7%
74.3%
34.9%
65.1%
19.5%
80.5%
18.0%
82.0%
19.0%
81.0%
24.0%
76.0%
30.5%
69.5%
26.9%
73.1%
35.4%
64.6%
Primary
Subsales

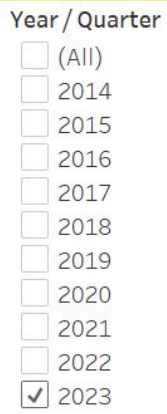
2- STOREY TERRACED HOUSE



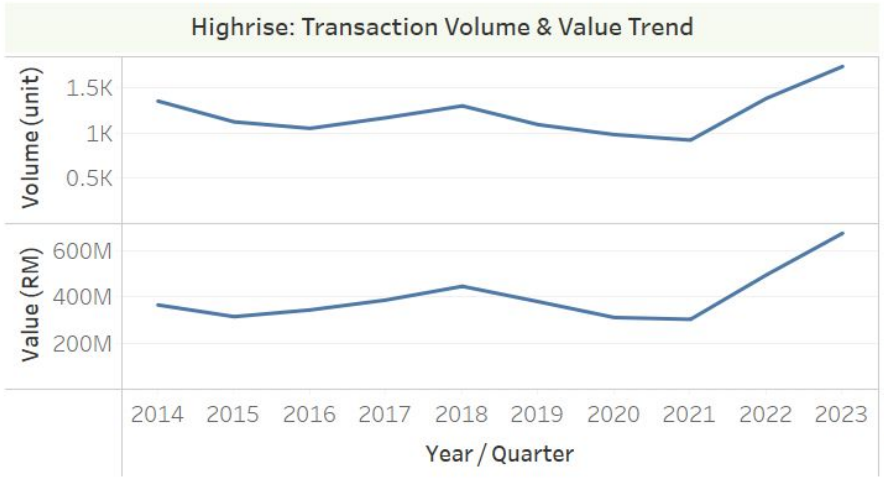


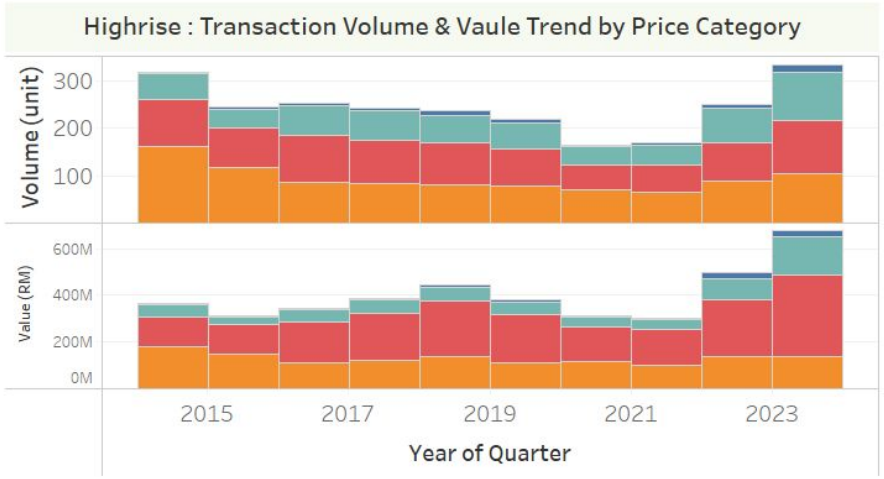


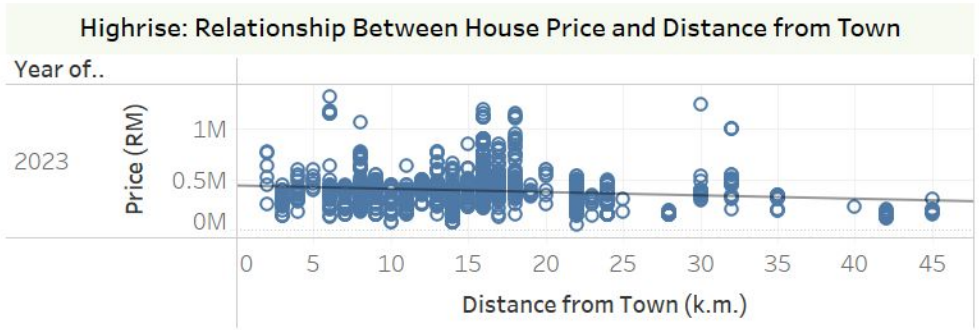


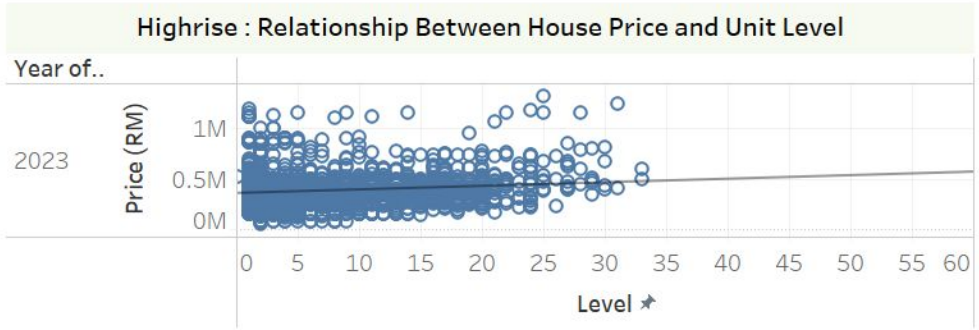


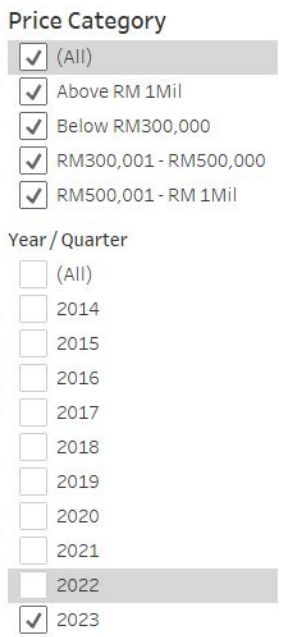
HIGHRISE UNIT



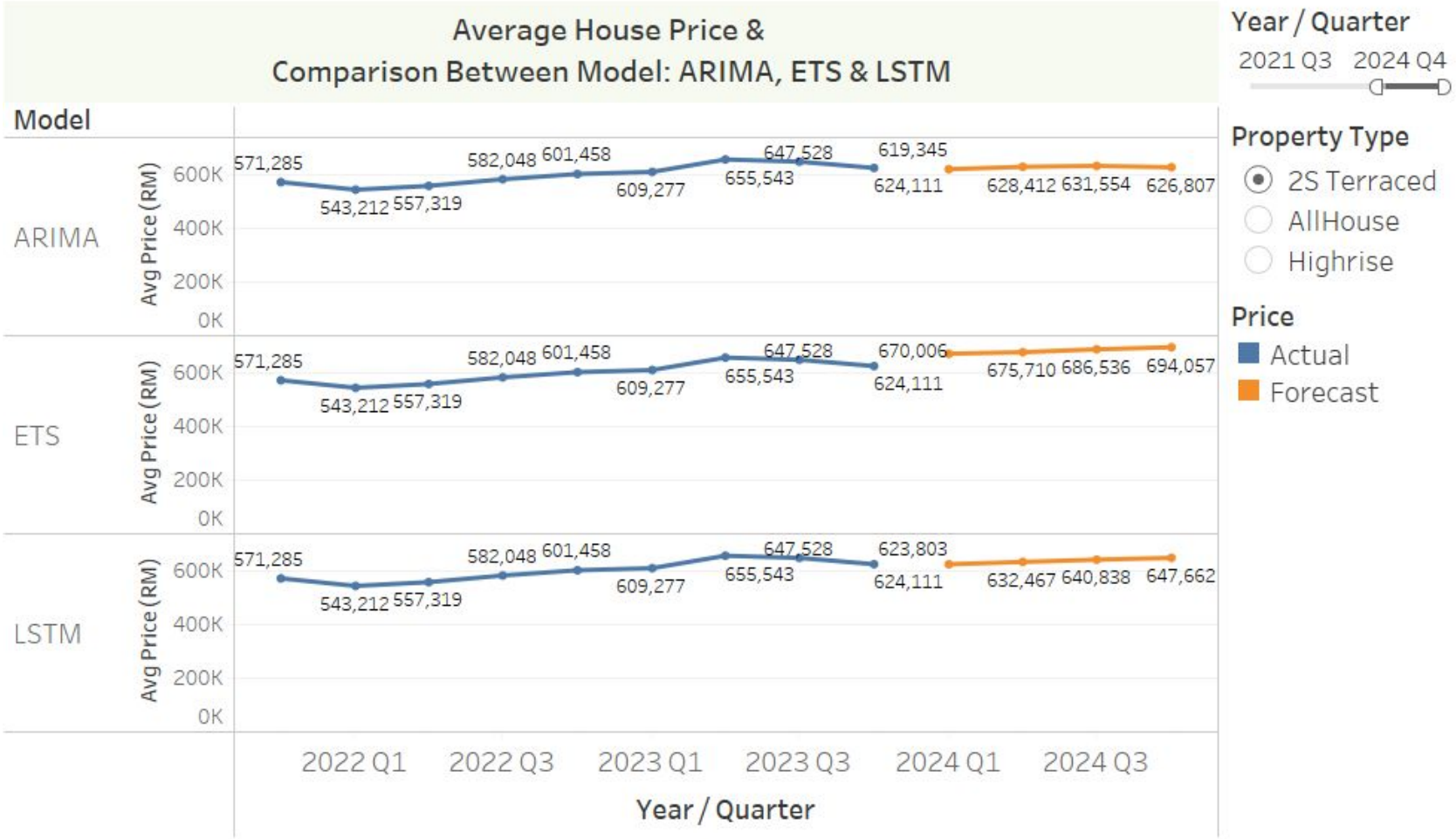








MODEL FORECAST AND RMSE



Root Mean Squared Error (RMSE) by Model and Property Type

Method	2S Terraced	Highrise	AllHouse
ARIMA	23,894	22,172	26,411
ETS	25,022	22,123	29,936
LSTM	31,580	29,449	49,792

CONCLUSION

- Based on the evaluation of various forecasting models, the ARIMA model has been selected as the most suitable approach for forecasting house prices. This decision is grounded on the model's superior performance during the testing phase, as indicated by the lowest Root Mean Square Error (RMSE) among all tested models.

- It is hoped that this forecast will empower policymakers and stakeholders to navigate this environment more effectively using data-driven insights.

- Further exploration and refinement of Machine Learning forecasting techniques while emphasizing the need for more external data and economic indicators to improve the model's accuracy and robustness, while also supporting real-time data updates and continuous model evaluation


THANK YOU!