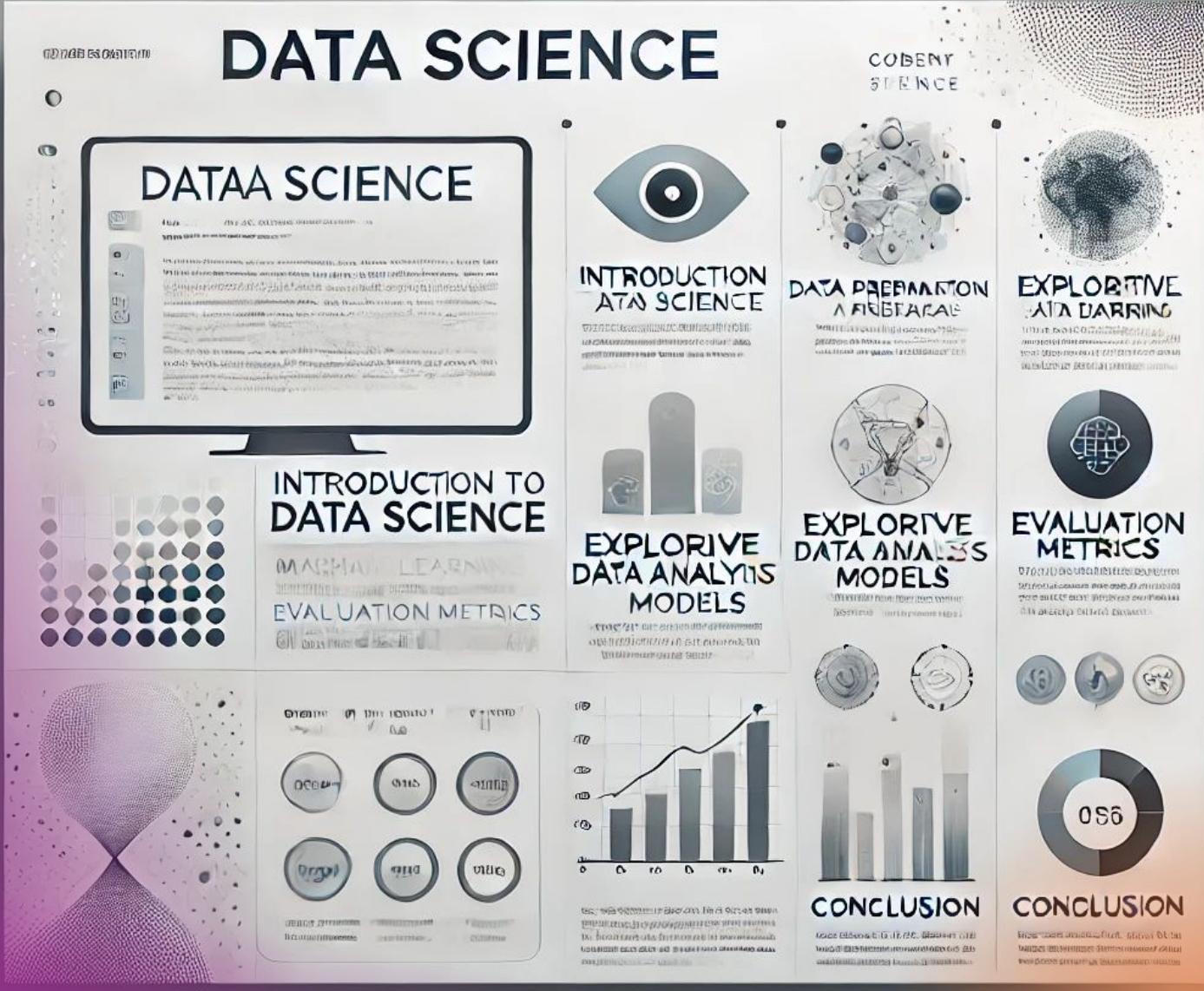




CAPSTONE PROJECT

FORECASTING HOUSE PRICE IN JOHOR BAHRU

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

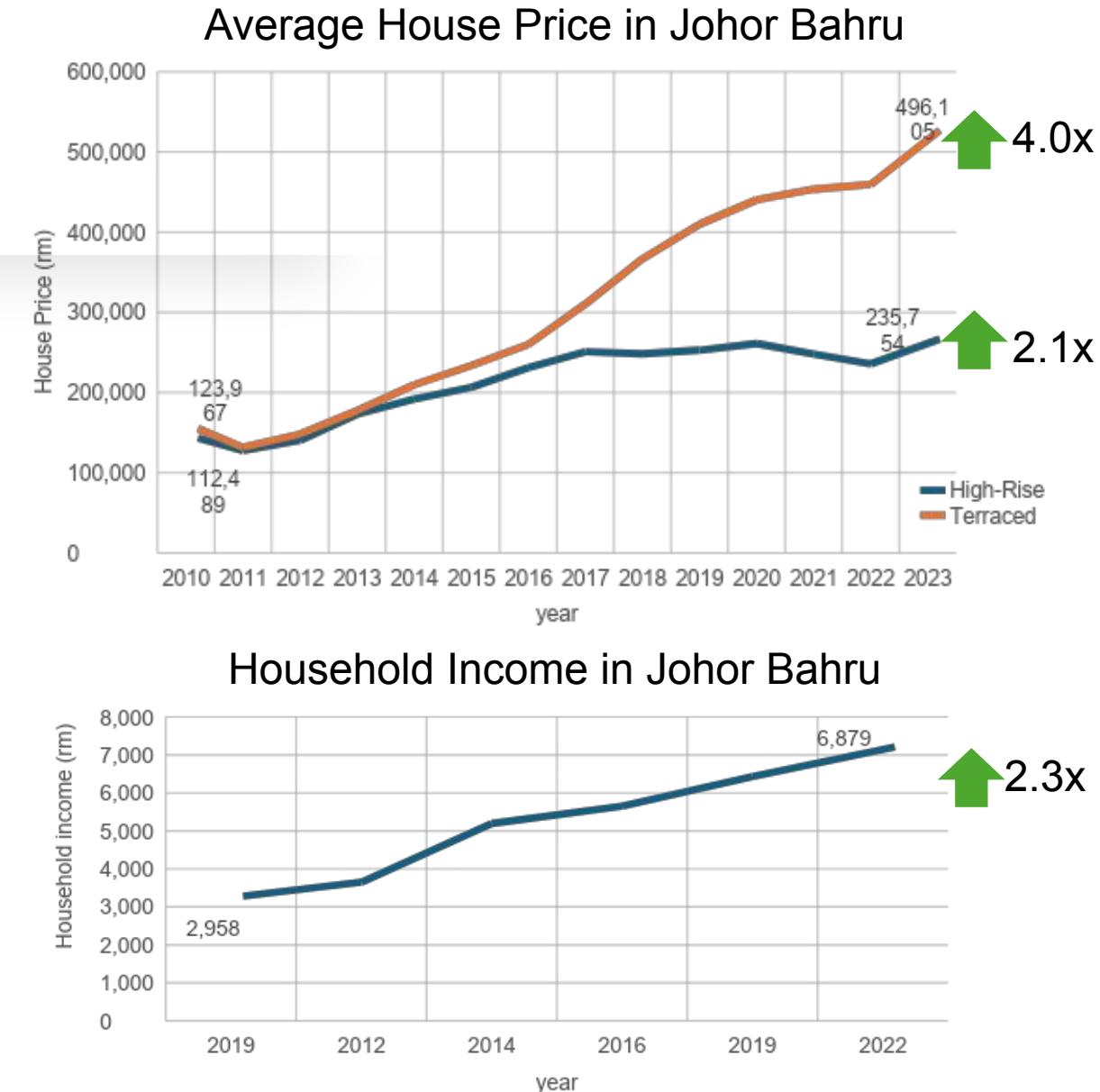


OUTLINE :

- 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	Z
LOT	KOD_J	NO_HA	J_HAR	PEGAN	PAJAK	STATUS	SKIM	KOD_F	KEGUI	LUAS	UNIT_I	KOD_F	JARAK	J_HAR	JENIS	KEGUI	KOD_F	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

```
1 # Get the list of columns to drop by excluding the ones you want
2 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']
3
```

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

```
1 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

```
1 # Convert to Integer
2 columns_to_convert = ['KOD_KLAS_KAW', 'KOD_KATEGORI_KAW', 'KOD KEADAAN_BGN',
'KOD_JEN_BINAAN', 'B_TINGKAT', 'KATEGORI_LPR']
3
4 df[columns_to_convert] = df[columns_to_convert].fillna(0).astype(int)
5 print(df)
```

```
1 # Convert 'USIA_BGN' to integer after filling NaN values with 0 (if any)
2 df['USIA_BGN'] = df['USIA_BGN'].fillna(0).astype(int)
-
1 df['HARGA_B']= df['HARGA_B'].apply(lambda x: str(x).strip())
2 df['HARGA_B']= df['HARGA_B'].apply(lambda x: str(x).replace(',',''))
```

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

Converting to date time

```
1 # Convert to datetime
2 df.loc[:, 'TKH_NILAI'] = pd.to_datetime(df['TKH_NILAI'], format='%d/%m/%Y',
3                                         dayfirst=True, errors='coerce')
4
5 # Check the data type of the column
6 print(df['TKH_NILAI'].dtypes)
7
8 # Display the first few rows
9 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
```

```
1 df['TKH_NILAI'] = pd.to_datetime(df['TKH_NILAI'], format='%d/%m/%Y',
2                                     dayfirst=True, errors='coerce')
3
```

```
1 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

```
1 df.shape
```

```
(31695, 21)
```

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

	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
object	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
int64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
dtype: int64
```

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

```
1 df.shape
```

```
(31690, 21)
```

Sorting and Filtering

```
1 filtered_df = df[  
2     (df['KOD_JEN_BINAAN'] == 1) &  
3     (df['B_TINGKAT'] == 2) &  
4     (df['BILIK_TDR'].isin([2, 3, 4, 5, 6])) &  
5     (df['SYER2'] == 1) &  
6     (df['STATUS_LOT'] == 'N') &  
7     (df['DAERAH'] == 2) &  
8     (df['LUAS_LOT'] >= 50) & (df['LUAS_LOT'] <= 300) &  
9     (df['LUAS_LOT_BGN'] > 50) &  
10    (df['HARGA_B'] > 42000) &  
11    (df['USIA_BGN'] >= 0) &  
12    (df['KATEGORI_LPR'].isin([1, 2]))  
13 ]  
14  
15 # Display the filtered DataFrame  
16 print(filtered_df)  
17
```

Data Aggregation

```
1 # Ensure 'quarter' is in datetime format  
2 df['quarter'] = pd.PeriodIndex(df['quarter'], freq='Q').to_timestamp()  
3  
1 #Aggregation by Quarter and Average Price  
2 df['quarter'] = pd.PeriodIndex(df['quarter'], freq='Q').to_timestamp()  
3 df_avg = df.groupby('quarter')['HARGA_B'].mean().reset_index()
```

Encoding categorical variables

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

Creating New Features

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

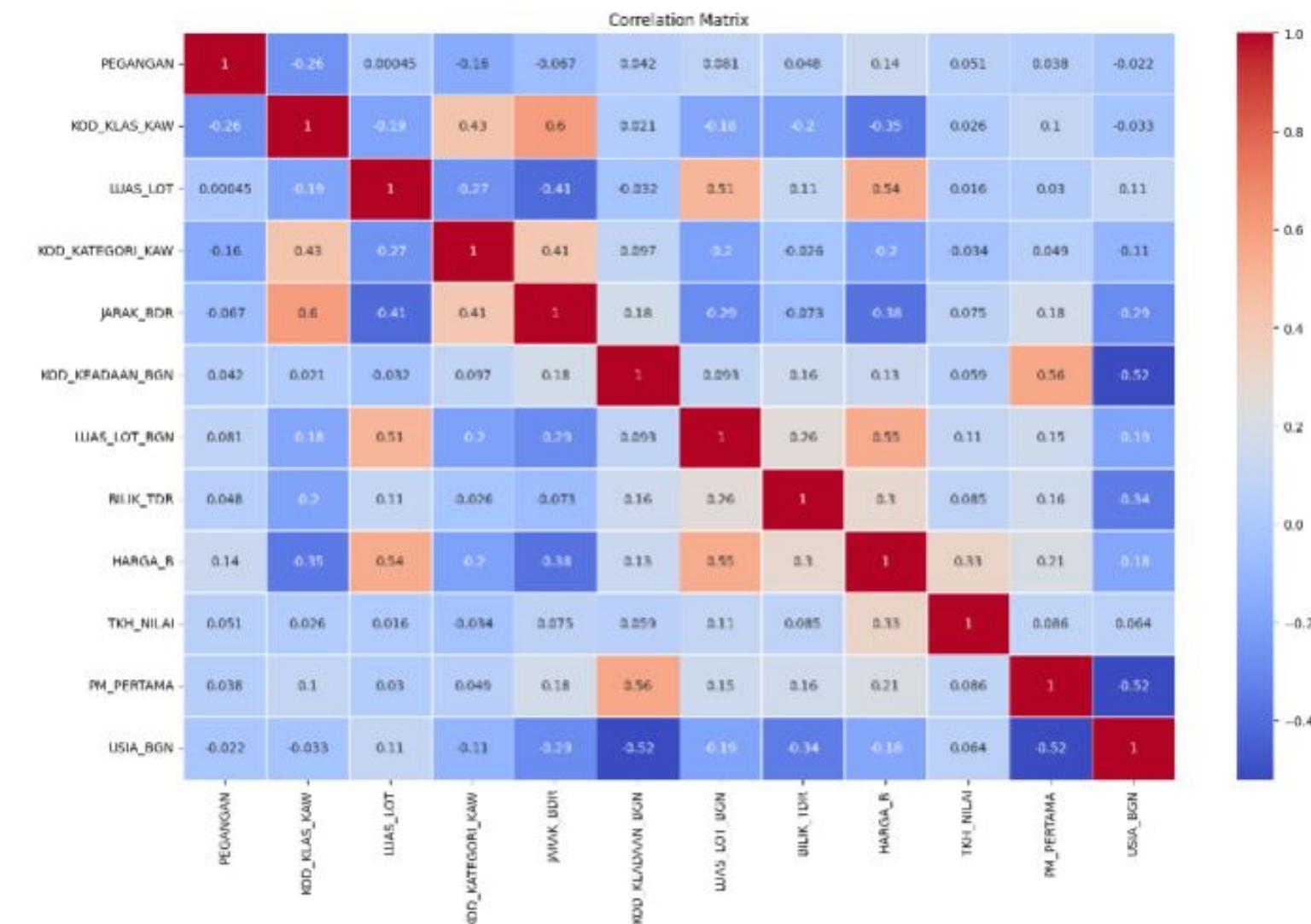
Exploratory Data Analysis – Using Python

Correlation

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns

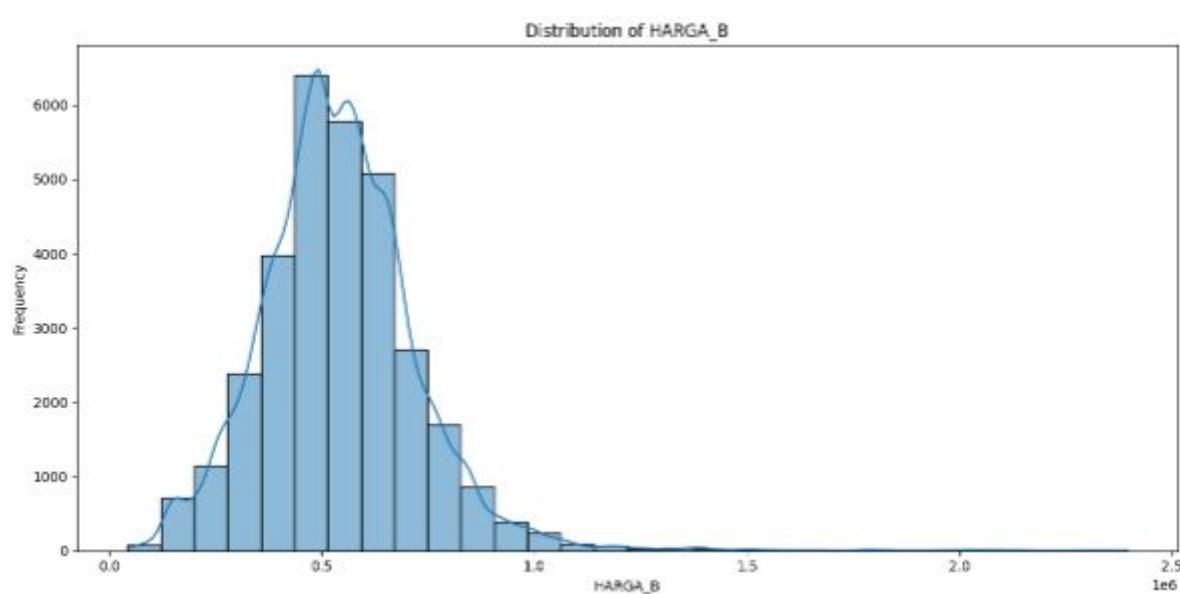
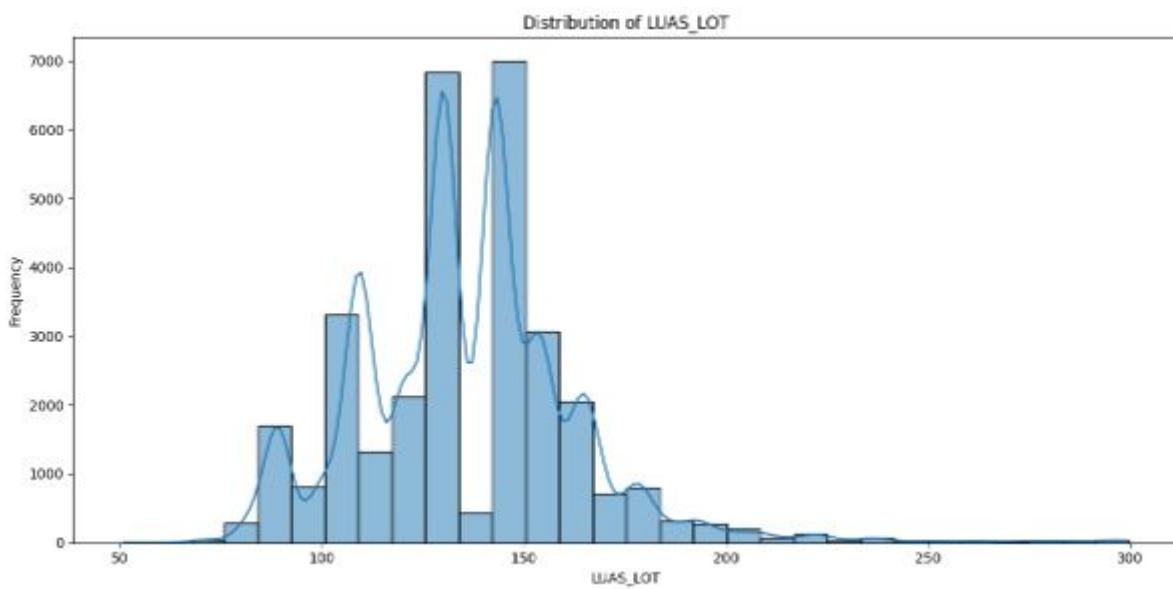
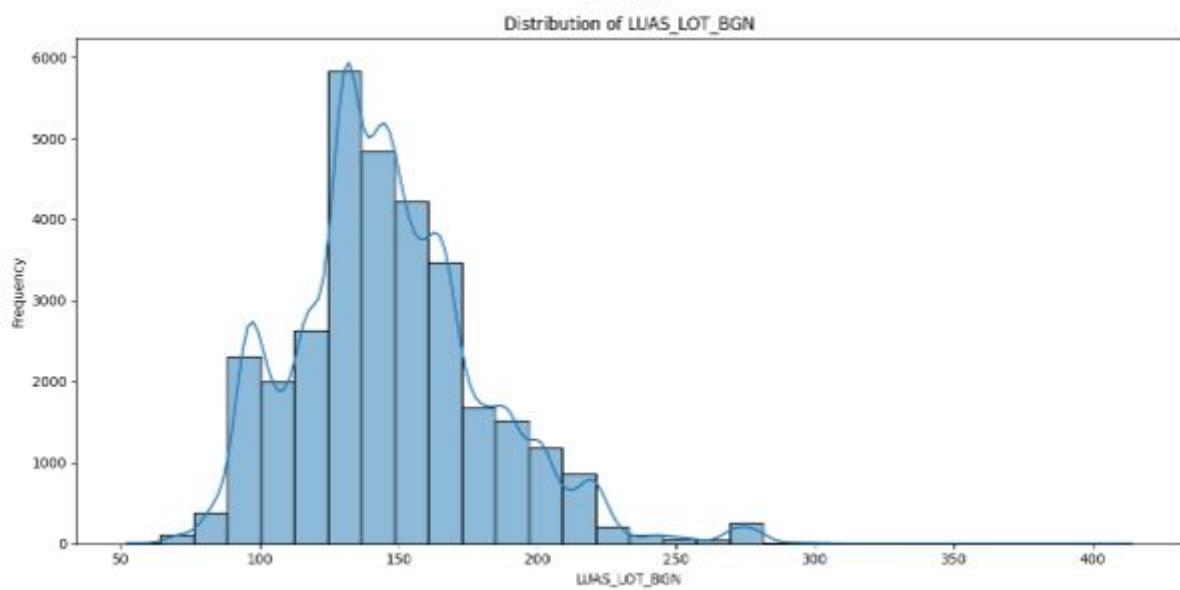
1 # Convert non-numeric columns to numeric where possible for correlation analysis
2 df_numeric = df.apply(pd.to_numeric, errors='coerce')
3
4 # Exclude the 'I_HARIAZ' feature by dropping it
5 df_numeric = df_numeric.drop(columns=['quarter', 'MUKBAN', 'SKIM'])
6
7 # Plot the correlation matrix
8 plt.figure(figsize=(15, 10))
9 sns.heatmap(df_numeric.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
10 plt.title('Correlation Matrix')
11 plt.tight_layout()
12 plt.show()
13
```

Terraced House



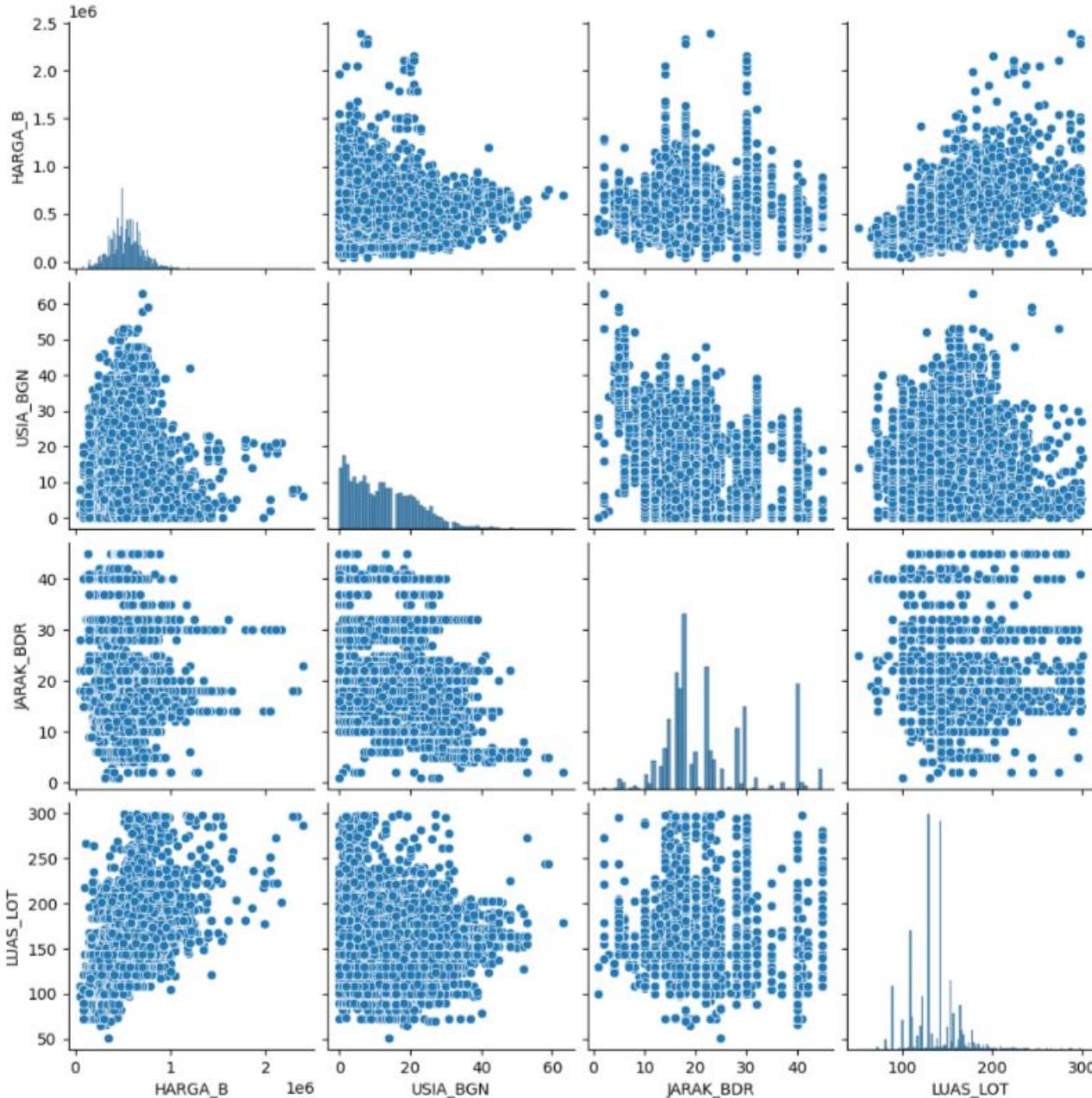
Data Distribution

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



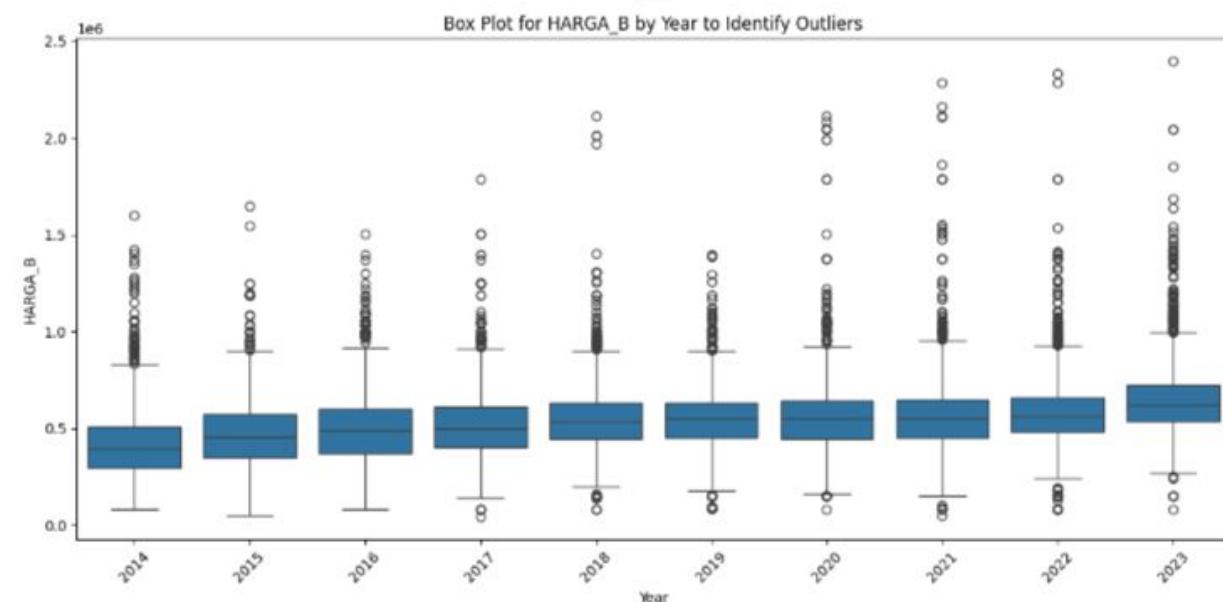
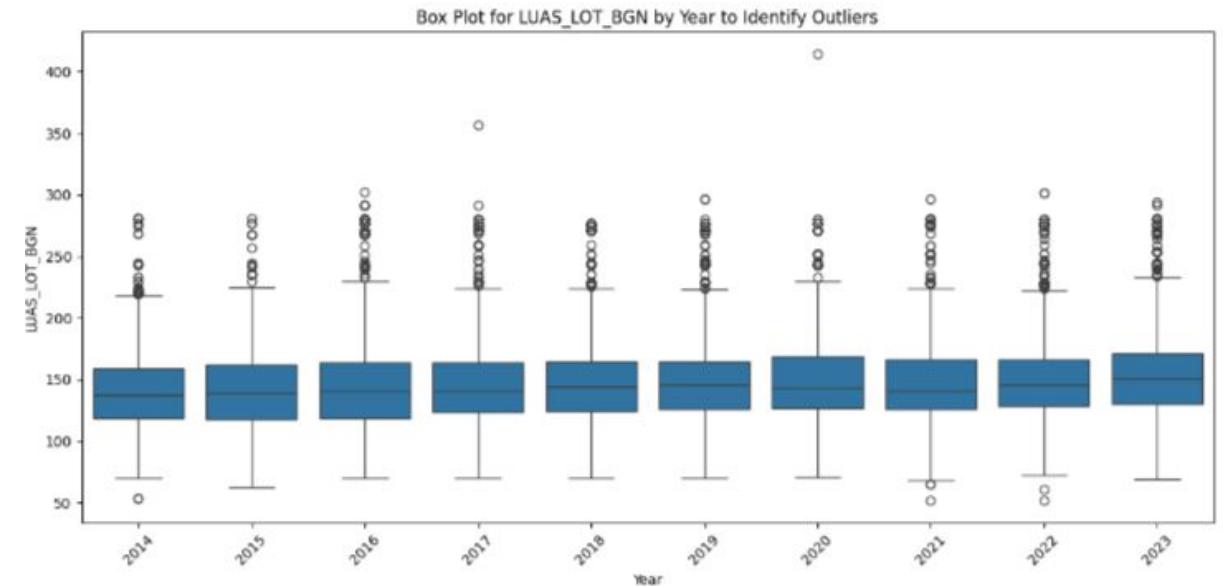
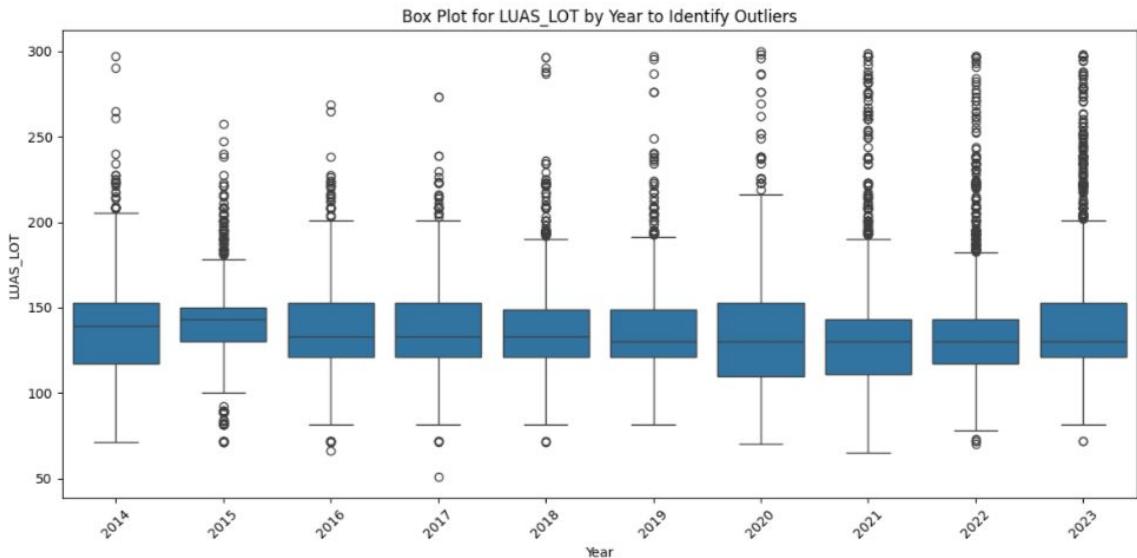
Pairplot

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



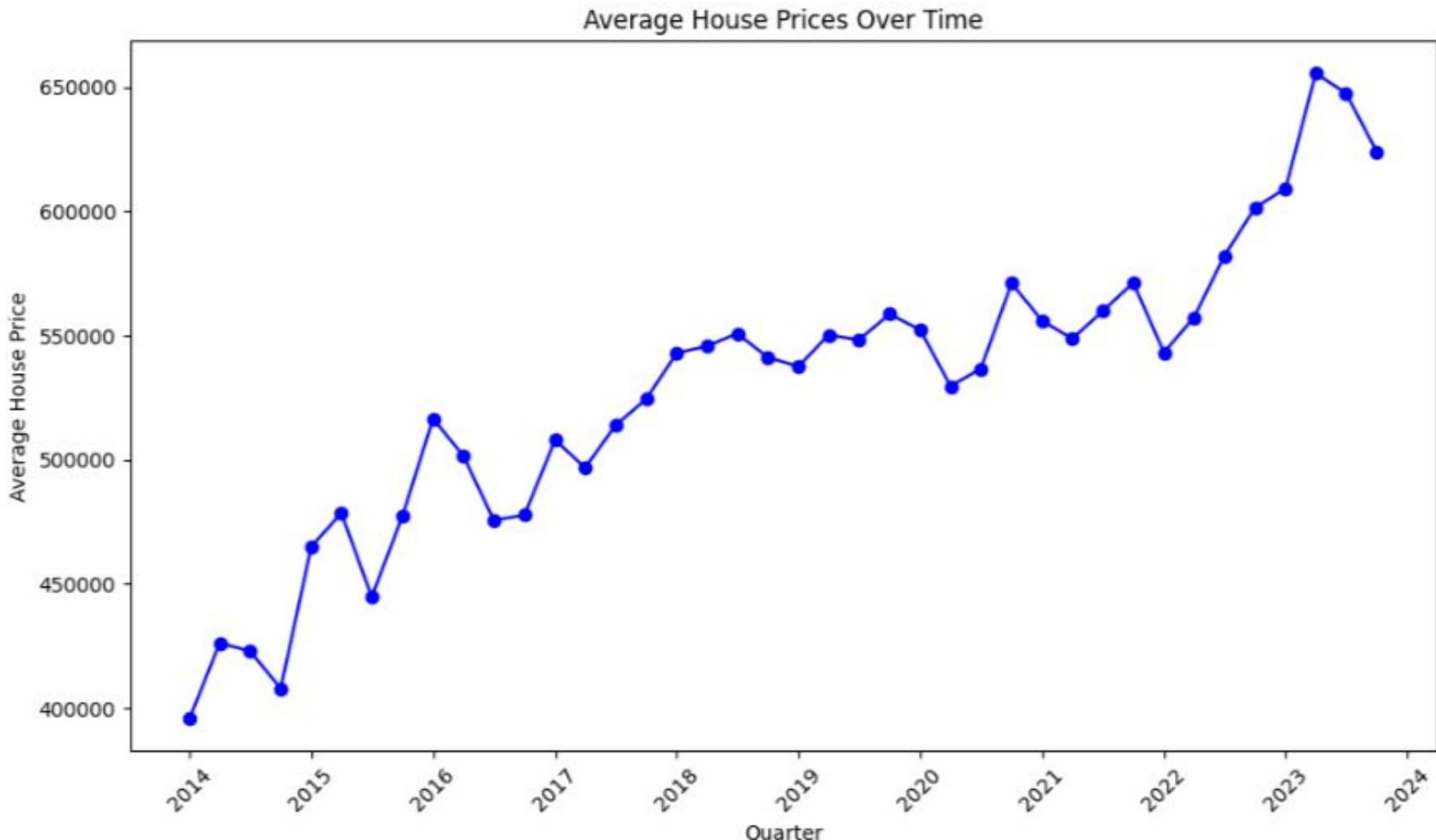
Boxplot

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

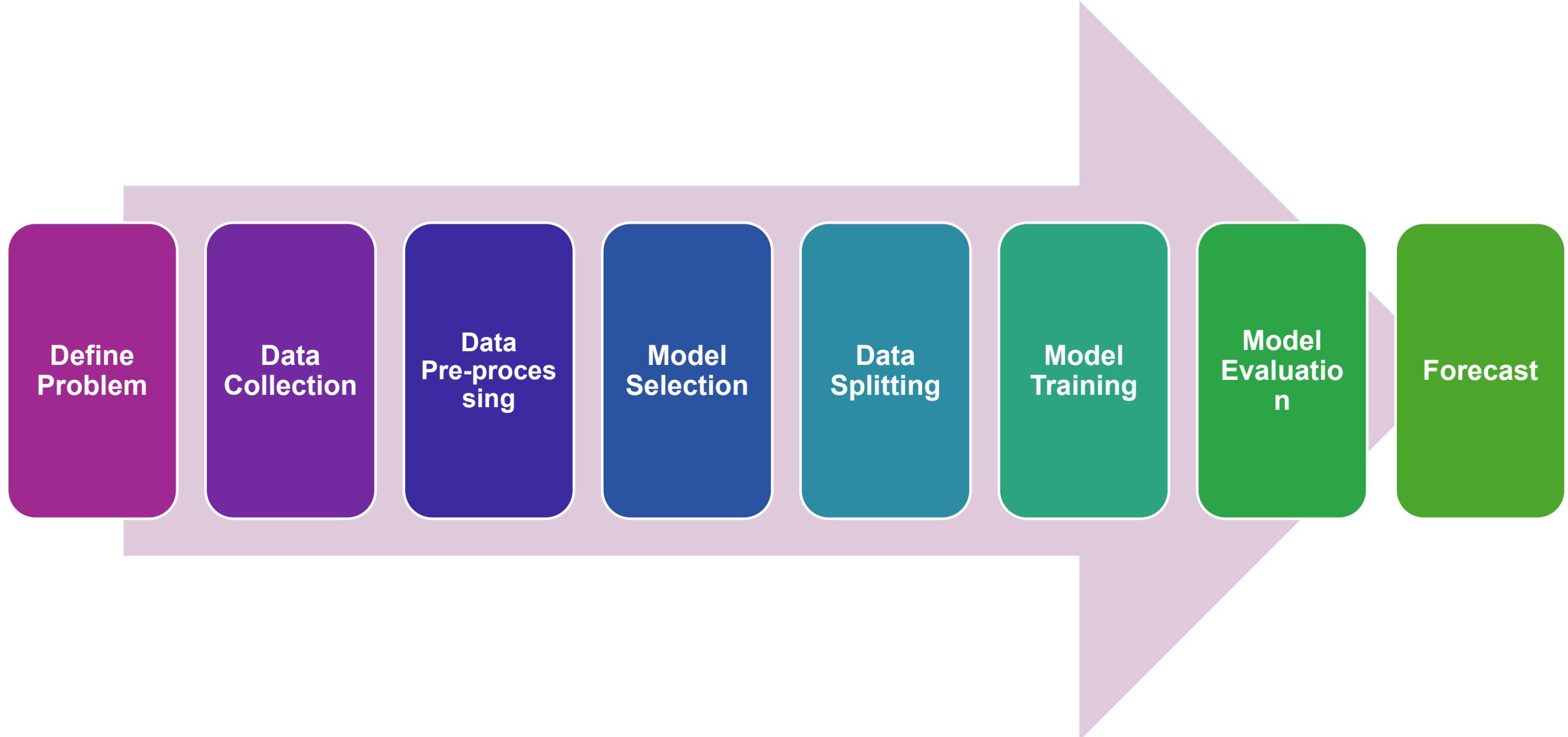


Time Series

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



Machine Learning Using Python



Model Selection- LSTM

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.preprocessing import MinMaxScaler
5 from keras.models import Sequential
6 from keras.layers import Dense, LSTM
7 from sklearn.metrics import mean_squared_error, mean_absolute_error,
root_mean_squared_error

1 # Preprocess the data
2 # Convert 'quarter' to datetime and calculate average house price per quarter
3 df['quarter'] = df['quarter'].dt.to_period('Q')
4 df_avg = df.groupby('quarter')['HARGA_B'].mean().reset_index()
5
```

Feature Scaling

```
1 # Feature Scaling
2 scaler = MinMaxScaler()
3 data_scaled = scaler.fit_transform(df_avg['HARGA_B'].values.reshape(-1, 1))
4
```

Set Random Seed

```
1 #Set a Random Seed:
2 import numpy as np
3 import tensorflow as tf
4 import random
5
6 # Set random seeds for reproducibility
7 np.random.seed(1)
8 tf.random.set_seed(1)
9 random.seed(1)
```

Prepare data for LSTM

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

Splitting the Data

```
1 from sklearn.model_selection import train_test_split
2
3 # Fixing random_state ensures consistent splitting
4 X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

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

Building the LSTM Model:

```
1 # Build the LSTM model
2 model = Sequential()
3 model.add(LSTM(20, activation='relu', input_shape=(n_input, n_features)))
4 model.add(Dense(1))
5 model.compile(optimizer='adam', loss='mse')
6
7 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

```
1 from keras.callbacks import EarlyStopping  
2  
3 early_stopping = EarlyStopping(monitor='val_loss', patience=5,  
4 restore_best_weights=True)  
5  
6 model.fit(X_train, y_train, epochs=150, batch_size=X_train.shape[0],  
7 validation_data=(X_test, y_test), callbacks=[early_stopping])
```

Model Evaluation

```
1 # Model Evaluation  
2 mse = mean_squared_error(y_test_inv, y_pred_inv)  
3 mae = mean_absolute_error(y_test_inv, y_pred_inv)  
4 rmse = root_mean_squared_error(y_test_inv, y_pred_inv)  
5 print(f"Mean Squared Error: {mse}")  
6 print(f"Mean Absolute Error: {mae}")  
7 print(f"Root Mean Squared Error: {rmse}")
```

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

```
1 import matplotlib.pyplot as plt  
2  
3 plt.figure(figsize=(16,10))  
4 plt.plot(y_test_inv, label='Average_House_Price', marker='o')  
5 plt.plot(y_pred_inv, label='forecasted_price', marker='o')  
6 plt.legend()  
7 plt.show()
```

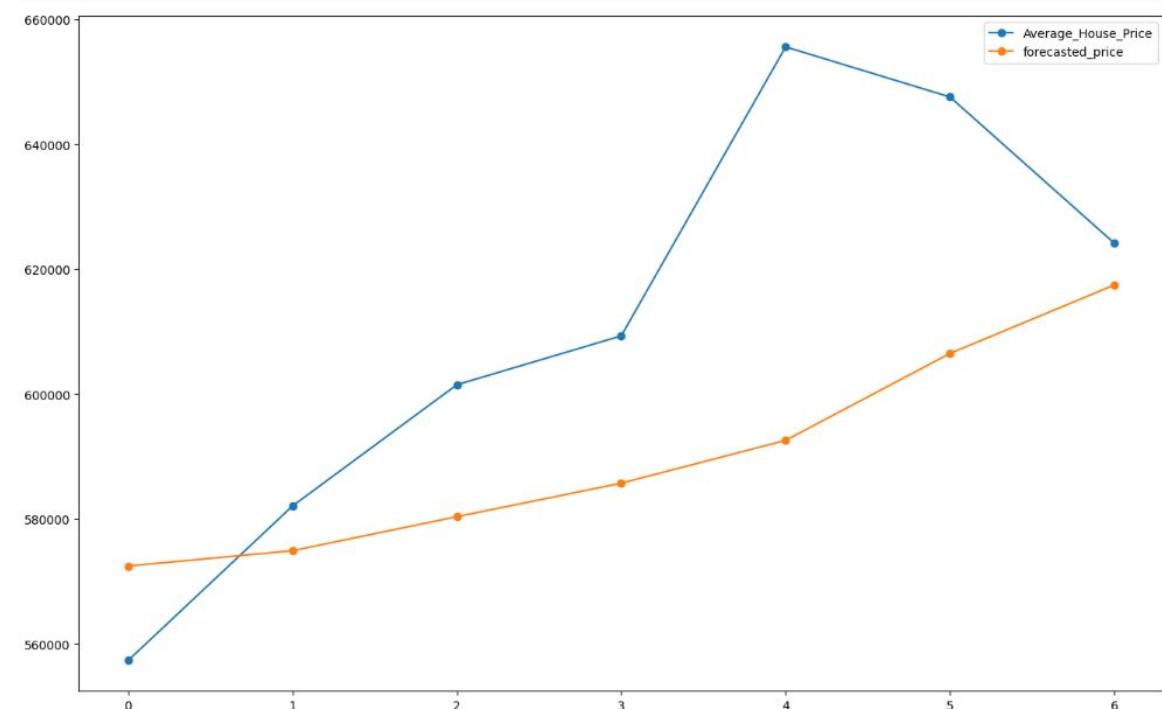
Making Predictions

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

1/1 ━━━━━━━━ 0s 283ms/step

Inverse Transforming the Predictions

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



Forecasting

```
1 # Forecast future values (Q1 2024 - Q4 2024)
2 last_4_quarters = data_scaled[-n_input:]
3 forecast = []
4 input_seq = last_4_quarters.reshape((1, n_input, n_features))
5 for _ in range(4):
6     next_pred_scaled = model.predict(input_seq)[0][0]
7     forecast.append(next_pred_scaled)
8     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
```

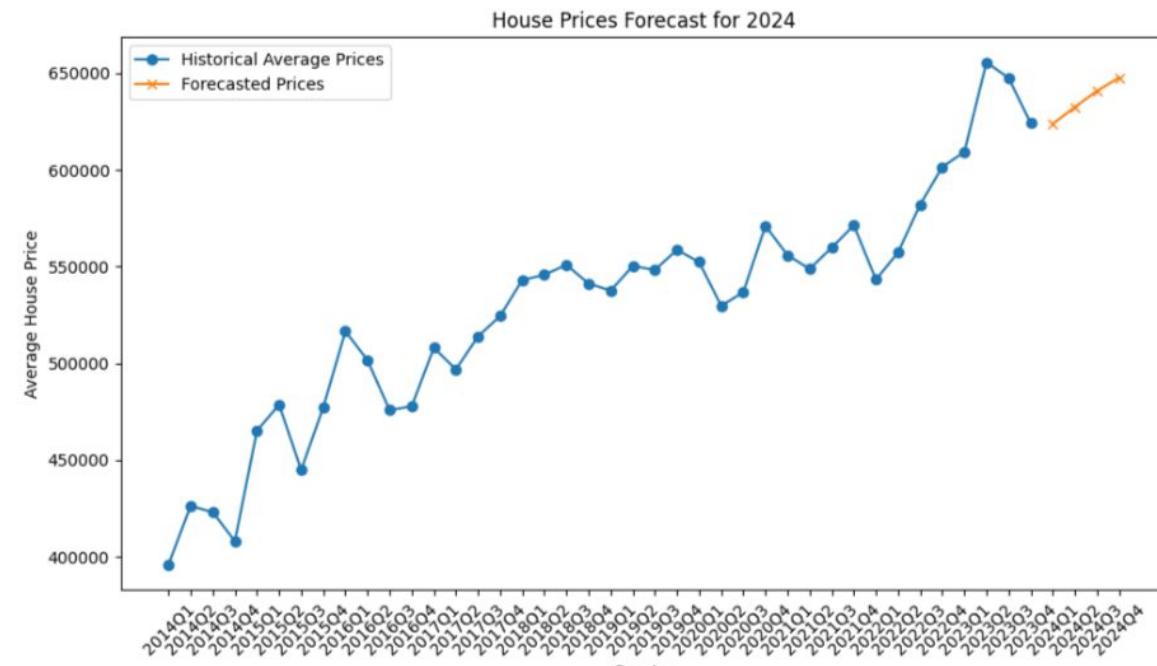
```
1 forecast_inv = scaler.inverse_transform(np.array(forecast).reshape(-1, 1))
```

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

```
1 # Display the forecasted values
2 dates = pd.date_range(start='2024-01-01', periods=4, freq='Q')
3 forecast_df = pd.DataFrame({'quarter': dates, 'forecasted_price': forecast_inv.
flatten()})
4 print("\nForecasted Average House Prices for 2024:")
5 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

```
1 from statsmodels.tsa.stattools import adfuller
2 test_result=adfuller(df_quarterly['Average_Price'])
3 from statsmodels.tsa.stattools import adfuller
4
5 def adfuller_test(price):
6     result = adfuller(price)
7     labels = ['ADF Test Statistic', 'p-value', '#Lags Used', 'Number of Observations']
8     for value, label in zip(result, labels):
9         print(label + ' : ' + str(value))
10
11 if result[1] <= 0.05:
12     print("Strong evidence against the null hypothesis (H0), reject the null hypothesis. Data is stationary")
13 else:
14     print("Weak evidence against the null hypothesis, indicating it is non-stationary")
15
16 adfuller_test(df_quarterly['Average_Price'])
```

Autocorrelation Analysis

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

Model Fitting and Model Summary

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

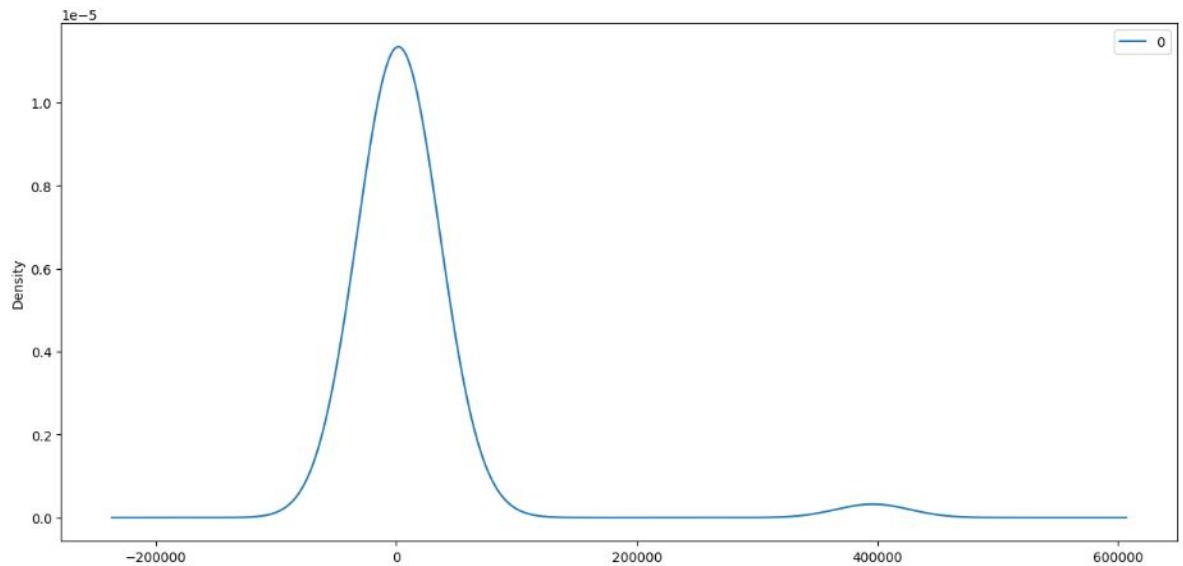
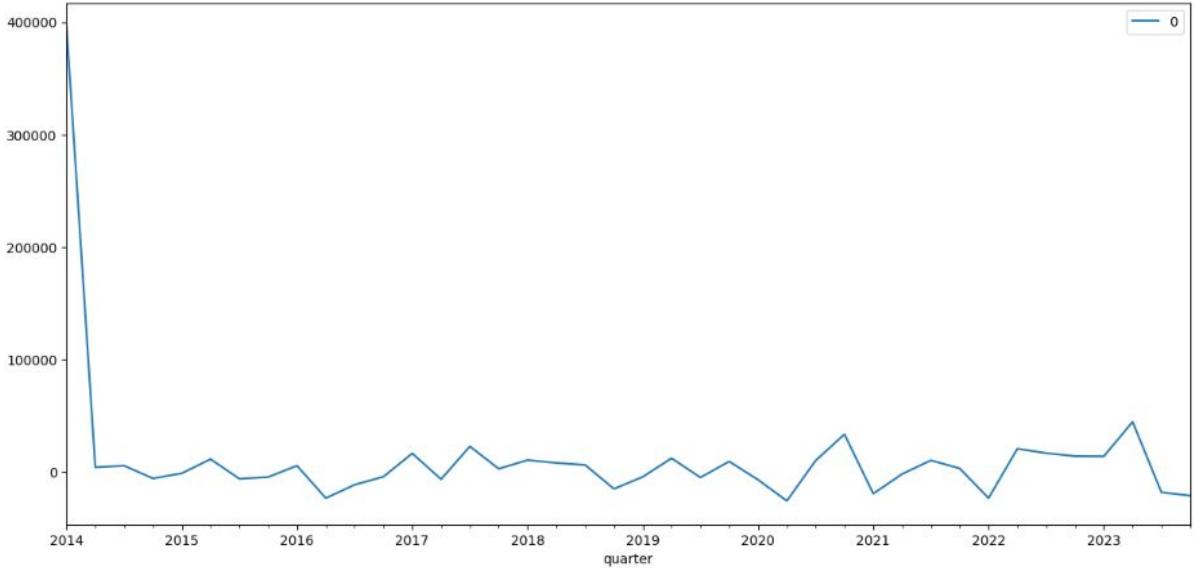
SARIMAX Results

```
=====
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

```
1 from sklearn.metrics import mean_squared_error
2 from math import sqrt
3 from statsmodels.tsa.arima.model import ARIMA
4 import matplotlib.pyplot as plt
```

Split the Data into Train and Test Sets

```
6 # Split the data into train and test sets
7 X = df_quarterly['Average_Price']
8 size = int(len(X) * 0.80)
9
10 # Reset the index to avoid index mismatch errors
11 train, test = X.iloc[:size].reset_index(drop=True), X.iloc[size:].reset_index
   (drop=True)
```

Initialize Variables

```
13 # Initialize history with all observations from the training data
14 history = list(train) # Use all the data points for forecasting
15 predictions = [] # List to store predictions
16
```

Walk-Forward Validation

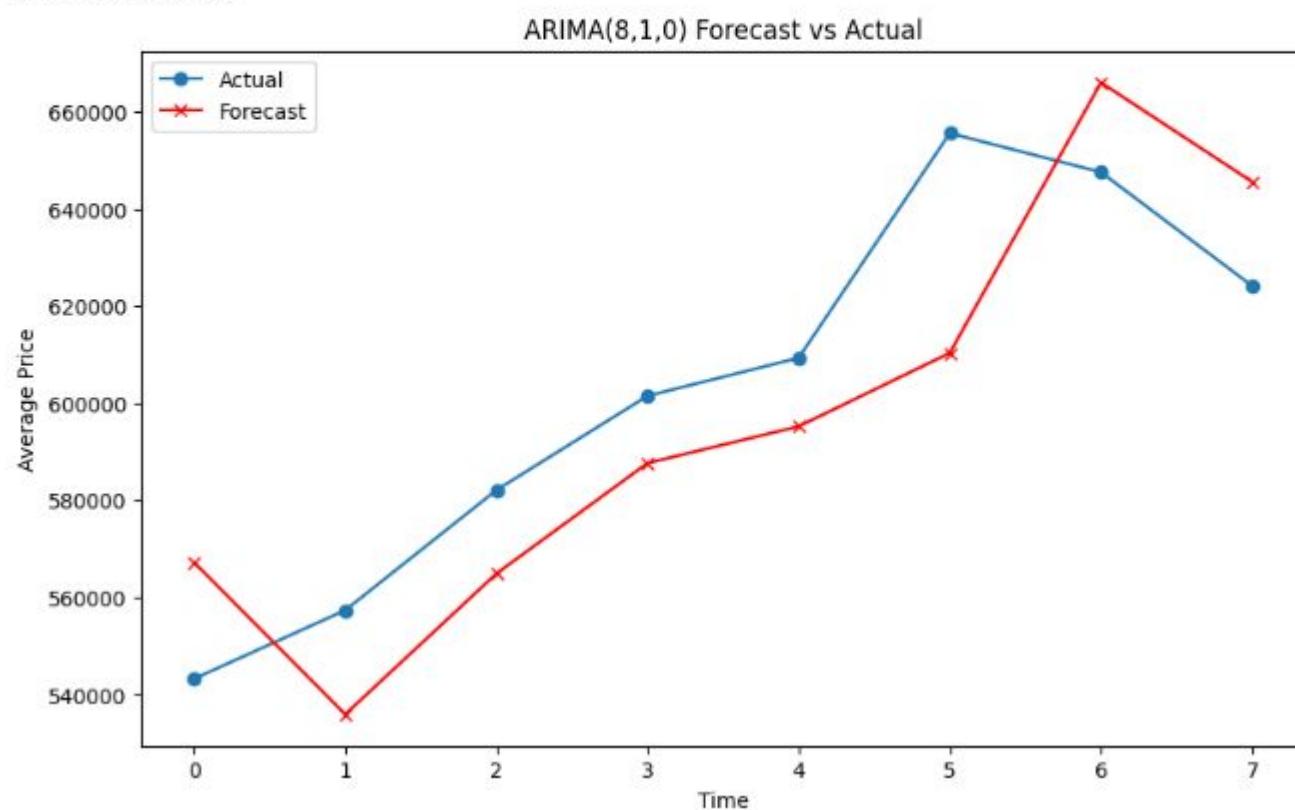
```
17 # Walk-forward validation using ARIMA(8,1,0)
18 for t in range(len(test)):
19     # Use the entire history for training the ARIMA model
20     model = ARIMA(history, order=(8, 1, 0))
21     model_fit = model.fit()
22
23     # Forecast the next step (1 quarter ahead)
24     output = model_fit.forecast()
25     yhat = output[0]
26     predictions.append(yhat) # Store the prediction
27
28     # Get the actual observation from the test set
29     obs = test[t]
30
31     # Add the observed value to the history (update for next iteration)
32     history.append(obs)
33
34     print(f'predicted={yhat:.3f}, expected={obs:.3f}')
35
```

Evaluate Model

```
# 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()
```

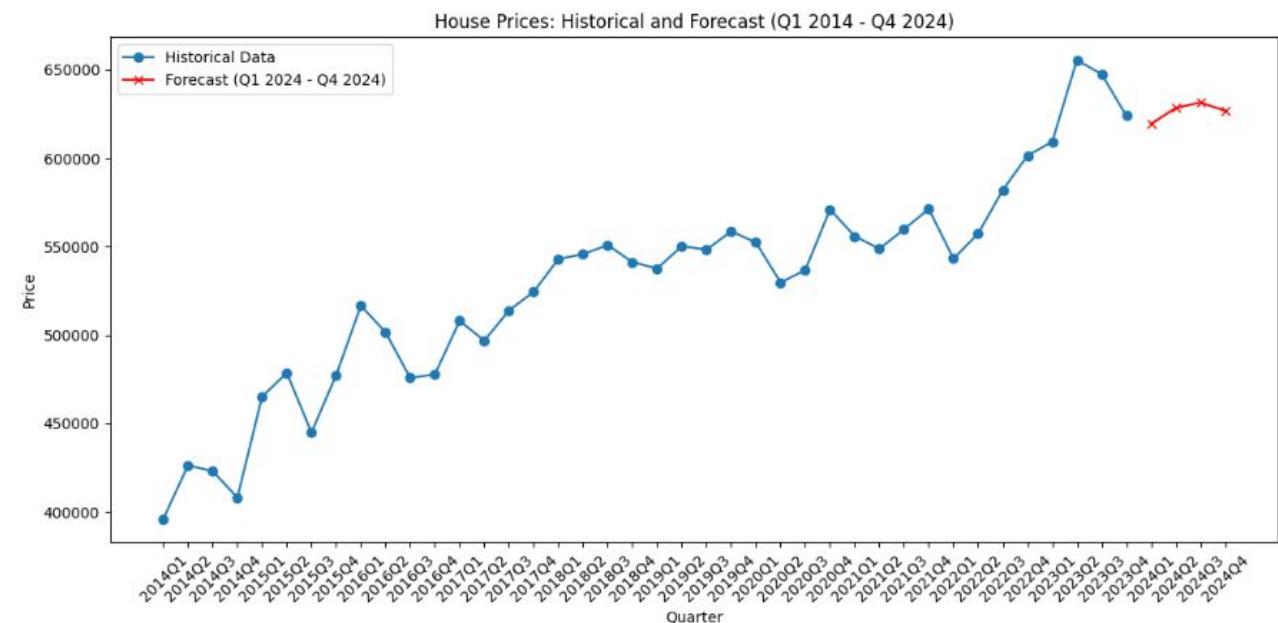
```
predicted=567180.612, expected=543211.790
predicted=535896.079, expected=557318.515
predicted=564855.965, expected=582047.627
predicted=587614.509, expected=601457.896
predicted=595170.976, expected=609277.379
predicted=610294.738, expected=655543.045
predicted=665980.291, expected=647528.498
predicted=645553.269,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



Forecast

```
-- 
11 # Forecast for the next 4 quarters (Q1 2024 - Q4 2024)
12 for t in range(4):
13     # Train ARIMA model using the entire dataset
14     model = ARIMA(X, order=(8, 1, 0)) # Set ARIMA order to (8, 1, 0)
15     model_fit = model.fit()
16
17     # Forecast the next step (1 quarter ahead)
18     output = model_fit.forecast()
19     yhat = output.iloc[0] # Use .iloc to access the forecasted value
20     future_predictions.append(yhat) # Store forecasted value
21
22     # Update X with the forecast value
23     X = pd.concat([X, pd.Series([yhat])], ignore_index=True)
24
25 # Create a DataFrame for the forecasted quarters
26 future_quarters = pd.date_range(start='2024-01-01', periods=4, freq='Q').
27 to_period('Q')
28 forecast_summary = pd.DataFrame({
29     'Quarter': future_quarters.astype(str),
30     'Forecasted_Price': future_predictions
31 })
32
33 # Display the forecast summary
34 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

```
1 from statsmodels.tsa.stattools import adfuller
2 test_result=adfuller(df_quarterly['Average_Price'])
3 from statsmodels.tsa.stattools import adfuller
4
5 def adfuller_test(sales):
6     result = adfuller(sales)
7     labels = ['ADF Test Statistic', 'p-value', '#Lags Used', 'Number of
8     Observations']
8     for value, label in zip(result, labels):
9         print(label + ' : ' + str(value))
10
11 if result[1] <= 0.05:
12     print("Strong evidence against the null hypothesis (H0), reject the
13     null hypothesis. Data is stationary")
14 else:
15     print("Weak evidence against the null hypothesis, indicating it is
16     non-stationary")
17 adfuller_test(df_quarterly['Average_Price'])
```

Autocorrelation

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

Simple Exponential Smoothing

```
1 # fit an ETS model and plot residual errors
2 from statsmodels.tsa.holtwinters import SimpleExpSmoothing
3 from matplotlib import pyplot
4 # fit model
5 model = SimpleExpSmoothing(df_quarterly['Average_Price'])
6 model_fit = model.fit()
7 # summary of fit model
8 print(model_fit.summary())
```

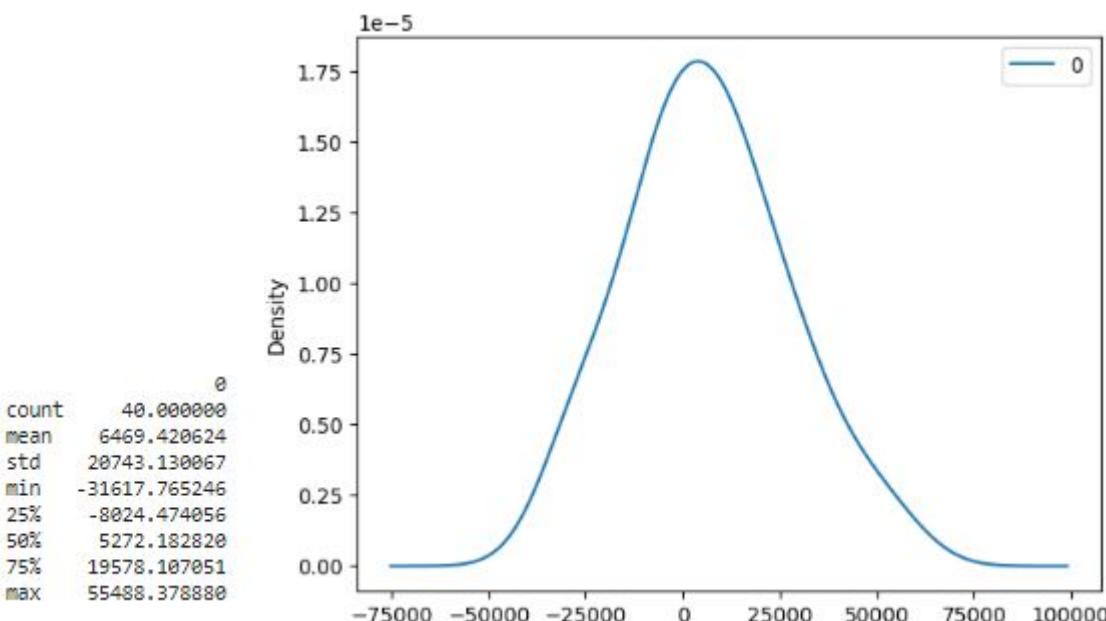
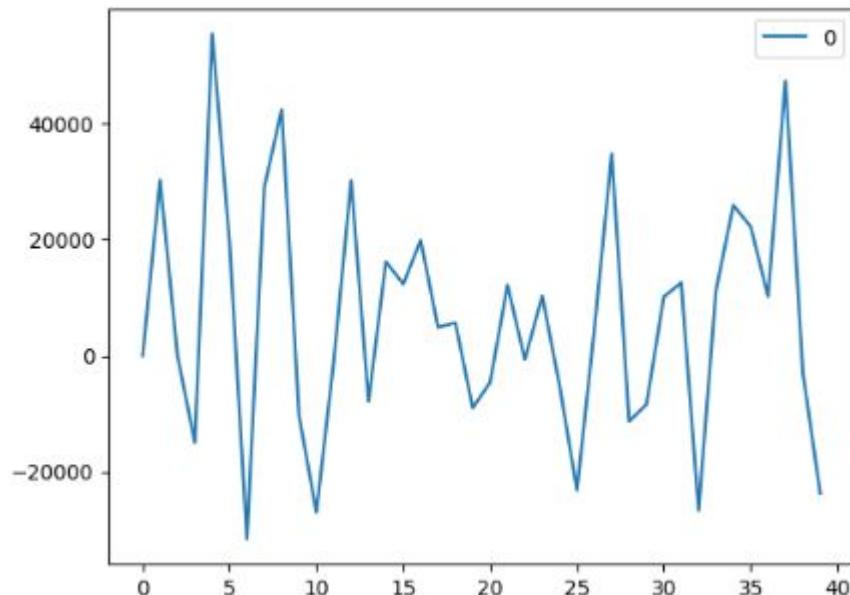
SimpleExpSmoothing Model Results

```
=====
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

```
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())
```



Import Libraries

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

Split Data into Train and Test Sets

```
7 # Split into train and test sets
8 X = df_quarterly['Average_Price']
9 size = int(len(X) * 0.8)
10 train, test = X[0:size], X[size:len(X)]
11
12 # Convert to list to manage data easily for walk-forward validation
13 history = list(train)
14 predictions = list()
15
```

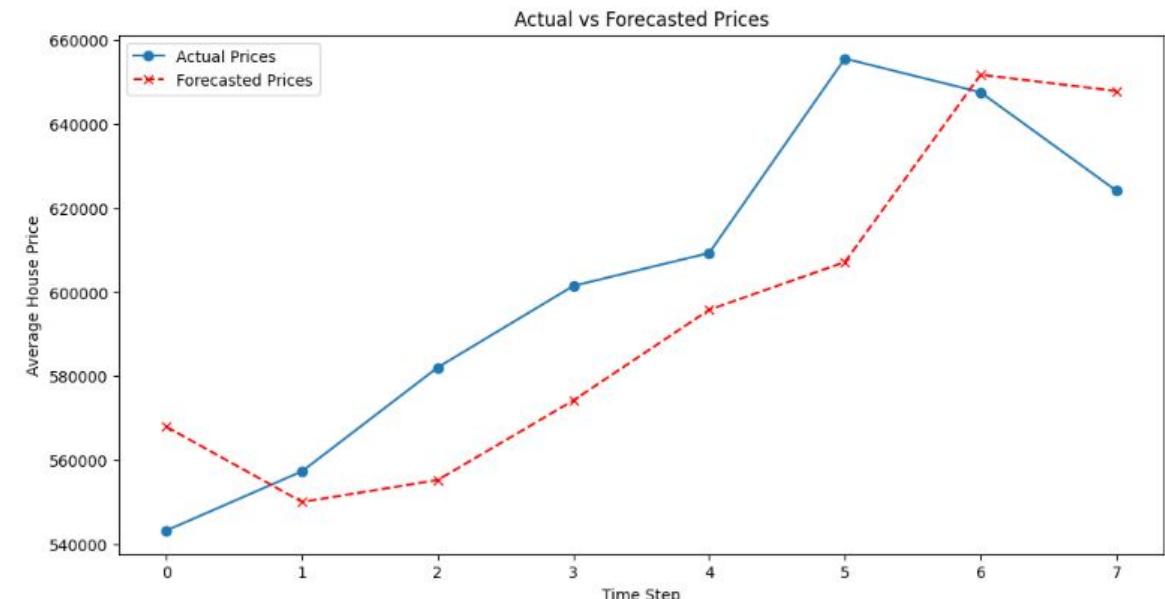
Walk-Forward Validation

```
16 # Walk-forward validation
17 for t in range(len(test)):
18     # Fit the model to the current history
19     model = SimpleExpSmoothing(history)
20     model_fit = model.fit(optimized=True) # Adding optimization for better
21     # convergence
22
23     # Forecast next value
24     output = model_fit.forecast(1) # Forecasting 1 step ahead
25     yhat = output[0] # Accessing the value directly from NumPy array
26
27     # Append the prediction to the predictions list
28     predictions.append(yhat)
29
30     # Add the observed value to history to continue forecasting
31     obs = test.iloc[t] # Use iloc to access value safely from pandas Series
32     history.append(obs)
33
34     # Print predictions for debugging purposes
35     print('predicted=%f, expected=%f' % (yhat, obs))
```

Evaluate Model

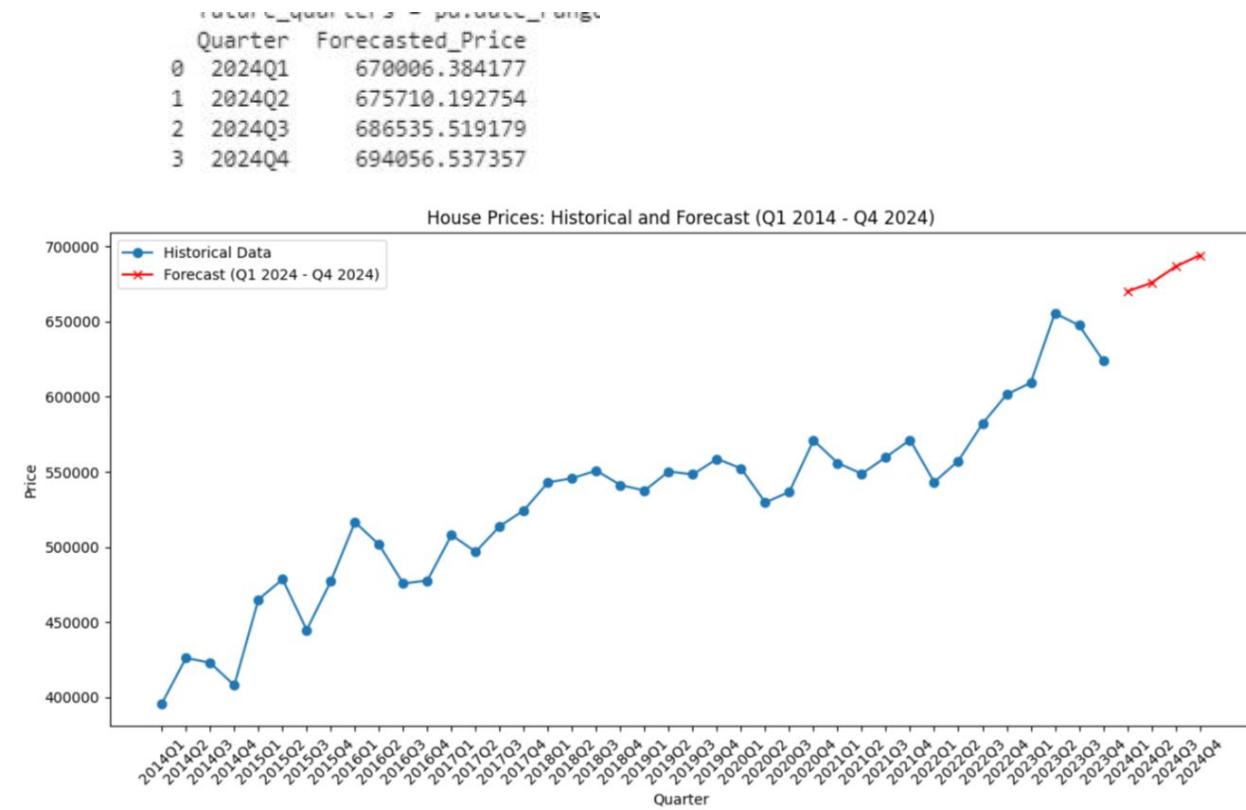
```
55
36 # Evaluate forecasts
37 rmse = sqrt(mean_squared_error(test, predictions))
38 print('Test RMSE: %.3f' % rmse)
39
40 # Plot forecasts against actual outcomes
41 plt.figure(figsize=(12, 6))
42 plt.plot(test.values, label='Actual Prices', marker='o')
43 plt.plot(predictions, label='Forecasted Prices', color='red', linestyle='--',
44 marker='x')
45 plt.legend()
46 plt.xlabel('Time Step')
47 plt.ylabel('Average House Price')
48 plt.title('Actual vs Forecasted Prices')
49 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

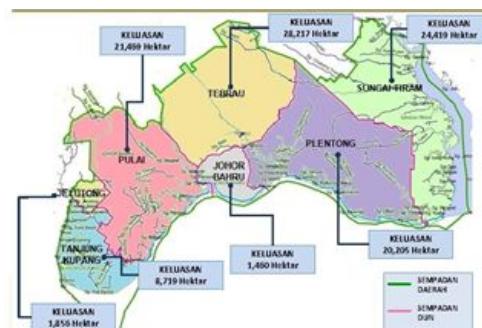
```
12 # Forecast for the next 4 quarters (Q1 2024 - Q4 2024)
13 for t in range(4):
14     # Train ETS (Exponential Smoothing) model using only the last 8 observations
15     model = ExponentialSmoothing(history[-8:], trend='add', seasonal=None)
16     model_fit = model.fit(optimized=True)
17
18     # Forecast the next step (1 quarter ahead)
19     output = model_fit.forecast(1)
20     yhat = output[0]
21     future_predictions.append(yhat) # Store forecasted value
22
23     # Update history with the forecast (sliding window of 8)
24     history = history[1:] + [yhat] # Keep only the last 8 points
25
26 # Create a DataFrame for the forecasted quarters
27 future_quarters = pd.date_range(start='2024-01-01', periods=4, freq='Q').
28     to_period('Q')
29 forecast_summary = pd.DataFrame({
30     'Quarter': future_quarters.astype(str),
31     'Forecasted_Price': future_predictions
32 })
33
34 # Display the forecast summary
35 print(forecast_summary)
```



Comparison Model Of Machine learning

Model Type	LSTM (Long Short-Term Memory)	ARIMA (AutoRegressive Integrated Moving Average)	ETS (Exponential Smoothing)
Model Type Definition	STM 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

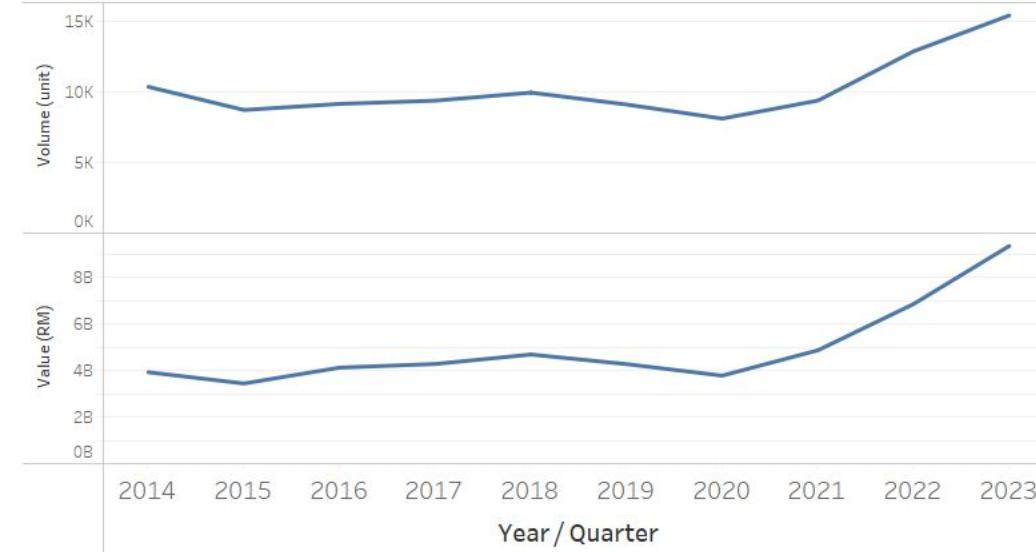


Johor Bahru Property Market Overview: Boundary Map & Market Activity Trend

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



Johor Bahru All House : Transaction Volume Trend by Price Category



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

2016

2017

2018

2019

2020

2021

2022

2023

0%

20%

40%

60%

80%

100%

20.0%

40.0%

60.0%

80.0%

100.0%

Volume (unit)

Value (RM)

- (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: Market Type

2014

2015

2016

2017

2018

2019

2020

2021

2022

2023

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

100%

20%

40%

60%

80%

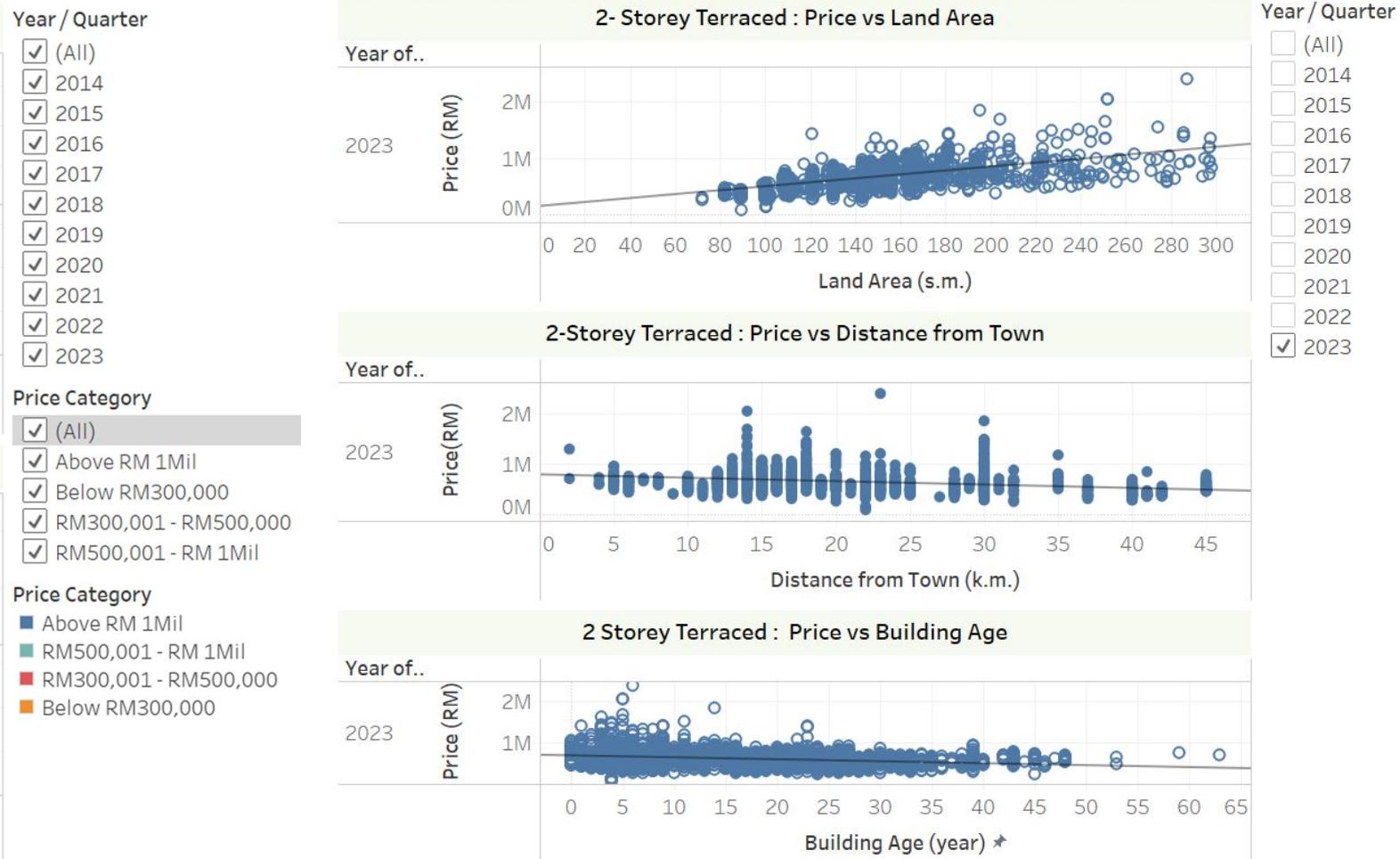
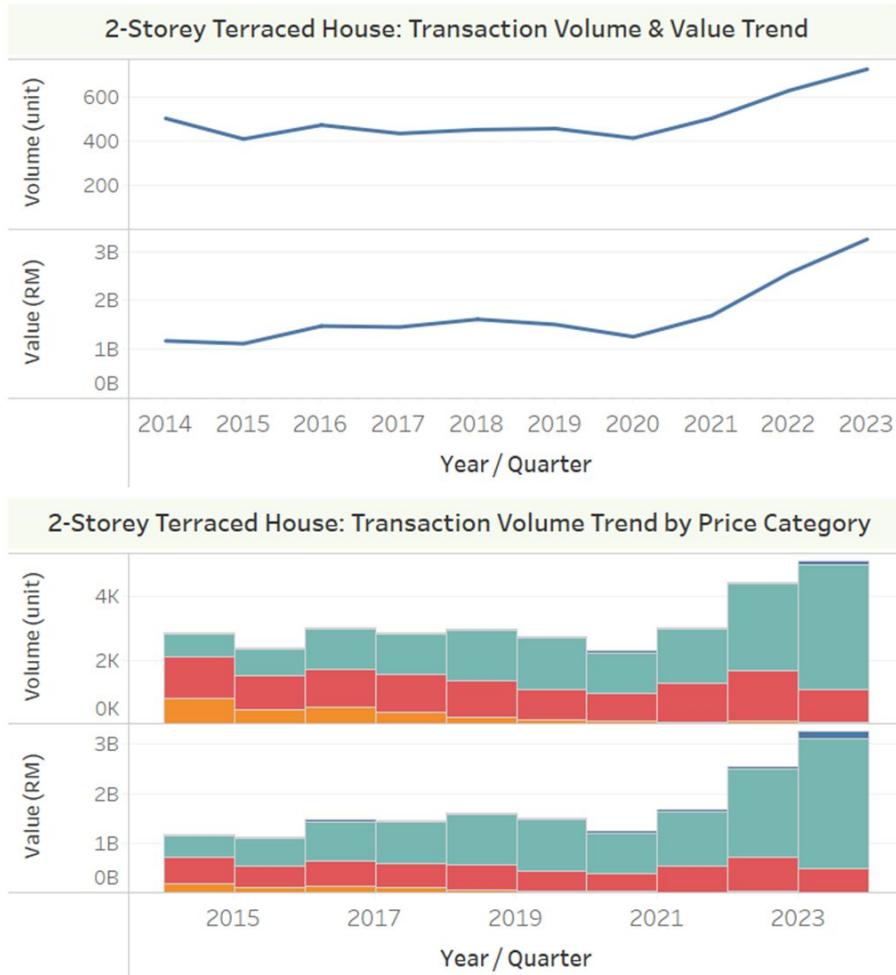
100%

20%

40%

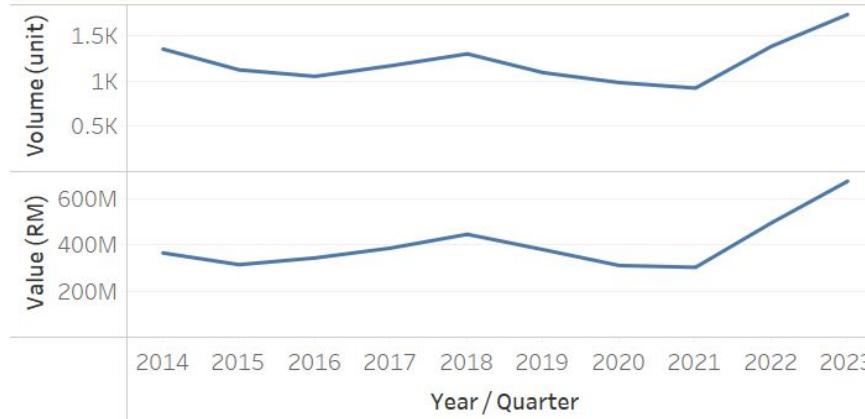
60%

2- STOREY TERRACED HOUSE

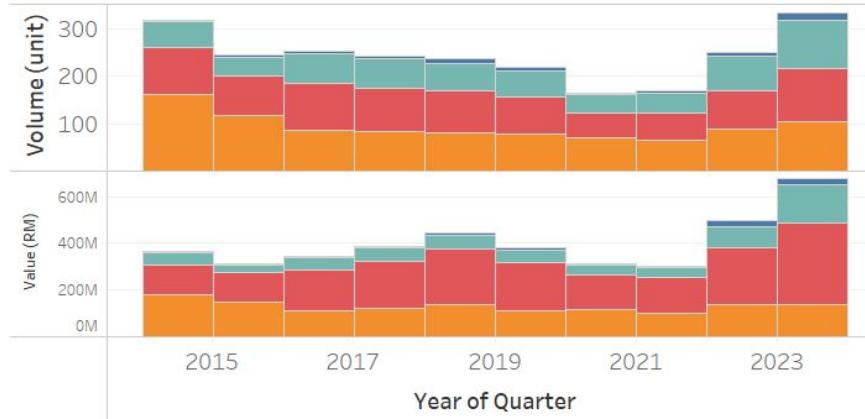


HIGHRISE UNIT

Highrise: Transaction Volume & Value Trend



Highrise : Transaction Volume & Vaule Trend by Price Category



Year / Quarter

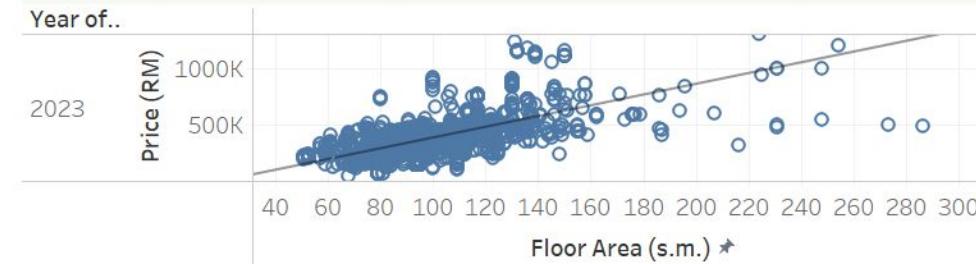
2014 Q1 2023 Q4

Price Category

- (All)
- Above RM 1Mil
- Below RM300,000
- RM300,001 - RM500,000
- RM500,001 - RM 1Mil

- Above RM 1Mil
- RM500,001 - RM 1Mil
- RM300,001 - RM500,000
- Below RM300,000

Highrise : Relationship Between House Price and Built-up Area



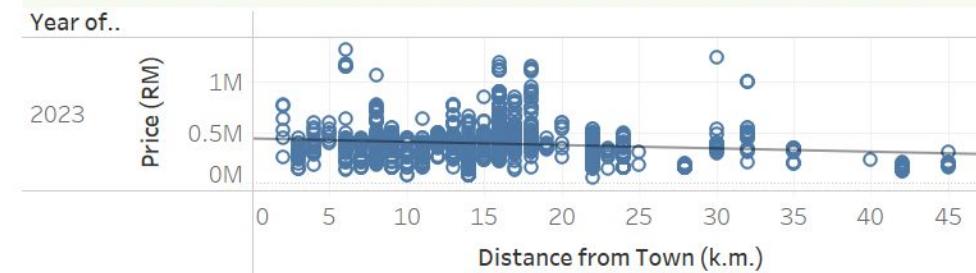
Price Category

- (All)
- Above RM 1Mil
- Below RM300,000
- RM300,001 - RM500,000
- RM500,001 - RM 1Mil

Year / Quarter

- (All)
- 2014
- 2015
- 2016
- 2017
- 2018
- 2019
- 2020
- 2021
- 2022
- 2023

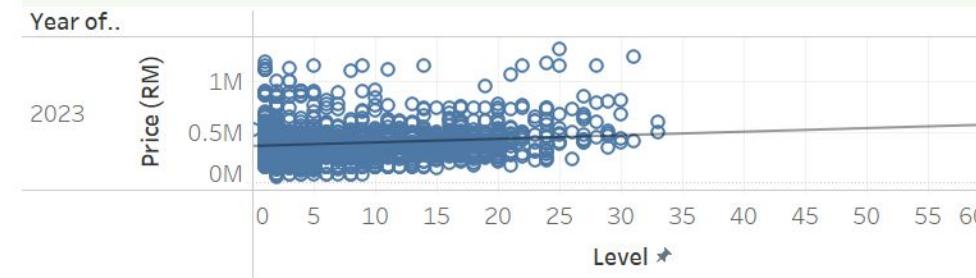
Highrise: Relationship Between House Price and Distance from Town



Year of..

- 2023
- 2014
- 2015
- 2016
- 2017
- 2018
- 2019
- 2020
- 2021
- 2022
- 2023

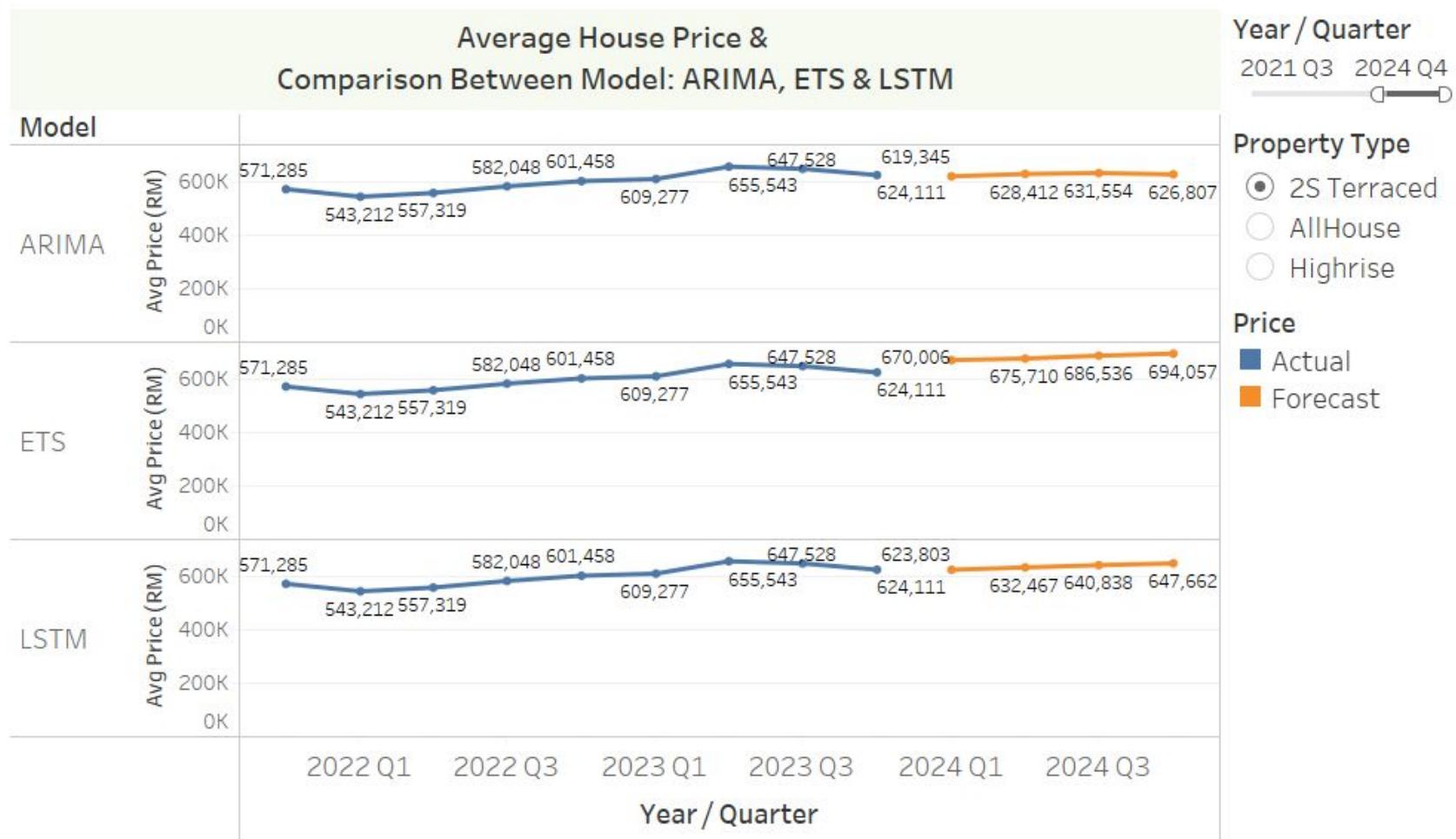
Highrise : Relationship Between House Price and Unit Level



Year of..

- 2023
- 2014
- 2015
- 2016
- 2017
- 2018
- 2019
- 2020
- 2021
- 2022
- 2023

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

DYTTHON
DATA SCIENCE

THANK YOU!



DATA
SCIENCE