London_Housing

February 22, 2024

0.1 Housing in London

Historical prices, sales, crimes, resident satisfaction and salaries by borough

About: The datasets is primarily centered around the housing market of London. However, it contains a lot of additional relevant data:

- Monthly average house prices
- Yearly number of houses
- Yearly median salary of the residents of the area
- Yearly mean salary of the residents of the area
- Monthly number of crimes committed
- Yearly number of jobs

[ ]:
```
# Data processing and cleaning
import pandas as pd
import numpy as np

# Visualisation
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from statsmodels.regression.linear_model import OLS
```

0.2 What Do We want to Find Out?

- How much has the average salary increased by to afford a home in london?
- What are the most expensive areas to live in London?

[ ]:
```
df = pd.read_csv('housing_in_london_monthly_variables.csv')
df.head()
```

[ ]:

	date	area	average_price	code	houses_sold	\
0	1995-01-01	city of london	91449	E09000001	17.0	
1	1995-02-01	city of london	82203	E09000001	7.0	

```
2  1995-03-01  city of london          79121  E09000001         14.0
3  1995-04-01  city of london          77101  E09000001          7.0
4  1995-05-01  city of london          84409  E09000001         10.0

   no_of_crimes  borough_flag
0           NaN             1
1           NaN             1
2           NaN             1
3           NaN             1
4           NaN             1
```

[ ]:
```
# Convert 'date' column to datetime format
df['date'] = pd.to_datetime(df['date'])
```

[ ]:
```
df.info
```

[ ]:
```
<bound method DataFrame.info of             date            area  average_price
code  houses_sold  \
0     1995-01-01  city of london          91449  E09000001         17.0
1     1995-02-01  city of london          82203  E09000001          7.0
2     1995-03-01  city of london          79121  E09000001         14.0
3     1995-04-01  city of london          77101  E09000001          7.0
4     1995-05-01  city of london          84409  E09000001         10.0
…            …               …             …          …           …
13544 2019-09-01         england         249942  E92000001      64605.0
13545 2019-10-01         england         249376  E92000001      68677.0
13546 2019-11-01         england         248515  E92000001      67814.0
13547 2019-12-01         england         250410  E92000001          NaN
13548 2020-01-01         england         247355  E92000001          NaN

       no_of_crimes  borough_flag
0               NaN             1
1               NaN             1
2               NaN             1
3               NaN             1
4               NaN             1
…               …               …
13544           NaN             0
13545           NaN             0
13546           NaN             0
13547           NaN             0
13548           NaN             0

[13549 rows x 7 columns]>
```

[ ]:
```
# Calculate the percentage of null values in each column of the DataFrame
null_pct = df.apply(pd.isnull).sum() / df.shape[0]
```

```
null_pct
```

[ ]:
```
date              0.000000
area              0.000000
average_price     0.000000
code              0.000000
houses_sold       0.006938
no_of_crimes      0.450956
borough_flag      0.000000
dtype: float64
```

[ ]:
```
# Select columns from the DataFrame where the null percentage is less than 5%
valid_columns = df.columns[null_pct < 0.05]
```

[ ]:
```
# Create a new DataFrame by selecting only the columns specified in␣
↪'valid_columns'
df = df[valid_columns].copy()
```

[ ]:
```
# Forward fill (ffill) to replace missing values with the last valid␣
↪observation in the DataFrame
df = df.ffill()

# Check for any remaining null values in each column after forward filling
null_counts_after_ffill = df.apply(pd.isnull).sum()
```

[ ]:
```
df.head()
```

[ ]:
```
        date            area  average_price       code  houses_sold  \
0 1995-01-01  city of london          91449  E09000001         17.0
1 1995-02-01  city of london          82203  E09000001          7.0
2 1995-03-01  city of london          79121  E09000001         14.0
3 1995-04-01  city of london          77101  E09000001          7.0
4 1995-05-01  city of london          84409  E09000001         10.0

   borough_flag
0             1
1             1
2             1
3             1
4             1
```

[ ]:
```
df.groupby('date')
```

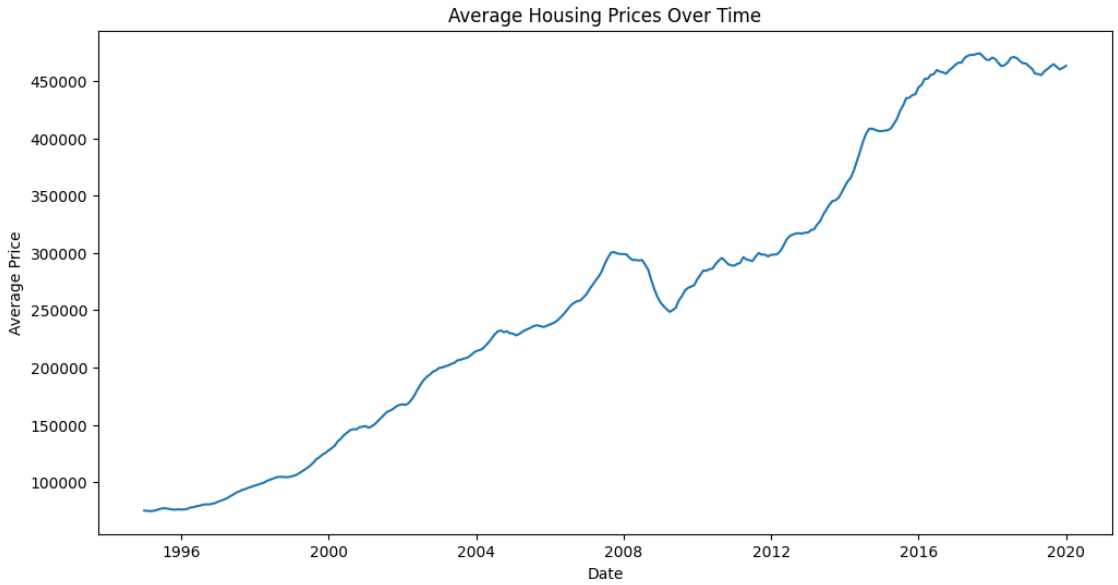
[ ]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x17ddf8500>
```

[ ]:
```
# Check the data types of columns
print(df.dtypes)
```

```
# Convert 'mean_salary' column to numeric (if needed)
df['average_price'] = pd.to_numeric(df['average_price'], errors='coerce')

# Group by date and calculate the mean of mean_salary for each date
df_grouped = df.groupby('date')['average_price'].mean().reset_index()

# Print the grouped and averaged DataFrame
print(df_grouped)
```

```
date             datetime64[ns]
area                     object
average_price             int64
code                     object
houses_sold             float64
borough_flag              int64
dtype: object
          date  average_price
0   1995-01-01   75157.733333
1   1995-02-01   74804.555556
2   1995-03-01   74702.888889
3   1995-04-01   74851.066667
4   1995-05-01   75564.911111
..         …            …
296 2019-09-01  464585.022222
297 2019-10-01  462245.733333
298 2019-11-01  460050.488889
299 2019-12-01  461400.755556
300 2020-01-01  463329.977778

[301 rows x 2 columns]
```

[ ]:
```
# Visualise the data
plt.figure(figsize=(12, 6))
sns.lineplot(data=df_grouped, x='date', y='average_price')
plt.title('Average Housing Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Average Price')
plt.show()
```



[ ]:
```
# Split the data
X = df_grouped[['date']]
y = df_grouped['average_price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[ ]:
```
# Create dattime parameter
df['date'] = pd.to_datetime(df['date'])
df['date_num'] = (df['date'] - df['date'].min()) / np.timedelta64(1,'D')

# Define X (independent variables) and y (dependent variable)
X = df[['date_num']]
y = df['average_price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Create a linear regression model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)
```

```
# Calculate the mean squared error and R-squared score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print('Mean Squared Error:', mse)
print('R-Squared:', r2)
```

```
Mean Squared Error: 17749433127.149418
R-Squared: 0.45078212018662756
```