

ZHVI dataset comes from https://www.kaggle.com/datasets/paultimothymooney/zillow-house-price-data?select=Sale_Prices_City.csv

Unemployment rate dataset comes from <https://www.kaggle.com/datasets/axeltorbenson/unemployment-data-19482021>

Inflation Rate(CPI) Dataset <https://www.kaggle.com/datasets/varpit94/us-inflation-data-updated-till-may-2021>

Interest rate dataset <https://www.kaggle.com/datasets/raoofiali/us-interest-rate-weekly>

GDP Growth Rate dataset <https://www.kaggle.com/datasets/rajkumarpandey02/economy-of-the-united-states>

```
1 #!pip install ydata-profiling
2 #!pip install tensorflow
3
4 import pandas as pd
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import statsmodels.api as sm
8 import kagglehub
9 import math
10 import os
11 import warnings
12
13 #from ydata_profiling import ProfileReport
14 from sklearn.model_selection import train_test_split
15 from sklearn.linear_model import Ridge
16 from sklearn.linear_model import Lasso
17 from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error, mean_absc
18
19 from sklearn.ensemble import RandomForestRegressor
20 from sklearn.preprocessing import PolynomialFeatures
21 from sklearn.preprocessing import StandardScaler
22 #from tensorflow.keras.models import Sequential
23 #from tensorflow.keras.layers import Dense
24 from IPython.display import clear_output, display, HTML
25
26 warnings.filterwarnings("ignore")
27 clear_output()
```



Adding Housing Data

```

1 # Download housing data
2 path = kagglehub.dataset_download("paultimothymooney/zillow-house-price-data")
3
4 print("Files in the dataset:")
5 for root, dirs, files in os.walk(path):
6     for file in files:
7         print(os.path.join(root, file))

```



Downloading from <https://www.kaggle.com/api/v1/datasets/download/paultimothymooney/zillow-house-price-data>
 100%|██████████| 124M/124M [00:01<00:00, 66.1MB/s]Extracting files...

Files in the dataset:

```

/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Da
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Sa
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Da
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/1
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/1
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/Ci
/root/.cache/kagglehub/datasets/paultimothymooney/zillow-house-price-data/versions/14/St

```

```

1 csv_path = os.path.join(path, "City_Zhvi_AllHomes.csv")
2 df = pd.read_csv(csv_path)
3 print(df.head())

```

```

→ Unnamed: 0  RegionID  SizeRank  RegionName  RegionType  StateName  State  \
0           0        6181         0    New York      City      NY      NY
1           1       12447         1  Los Angeles      City      CA      CA
2           2       39051         2    Houston      City      TX      TX
3           3       17426         3    Chicago      City      IL      IL
4           4        6915         4  San Antonio      City      TX      TX

           Metro      CountyName  1996-01-31  ...  \
0  New York-Newark-Jersey City  Queens County  196258.0  ...
1  Los Angeles-Long Beach-Anaheim  Los Angeles County  185649.0  ...
2  Houston-The Woodlands-Sugar Land  Harris County  93518.0  ...
3  Chicago-Naperville-Elgin  Cook County  130920.0  ...
4  San Antonio-New Braunfels  Bexar County  94041.0  ...

2019-06-30  2019-07-31  2019-08-31  2019-09-30  2019-10-31  2019-11-30  \
0  659421.0  659007.0  658239.0  656925.0  655613.0  654394.0
1  712660.0  713807.0  715688.0  718245.0  721896.0  725180.0
2  186844.0  187464.0  188070.0  188496.0  189125.0  189612.0
3  248372.0  248646.0  248725.0  248483.0  248278.0  248090.0
4  182732.0  183350.0  183930.0  184846.0  185490.0  186244.0

2019-12-31  2020-01-31  2020-02-29  2020-03-31
0  653930.0  653901.0  653565.0  652307.0
1  730358.0  735910.0  744137.0  752508.0
2  190179.0  190395.0  190938.0  191907.0
3  248029.0  248220.0  248599.0  249152.0
4  186420.0  186962.0  187129.0  187718.0

```

[5 rows x 300 columns]

```

1 # remove rows with NaN
2 df_cleaned = df.dropna()
3 print("DataFrame after removing rows with any NaN values:")
4 print(df_cleaned.head())
5 data = df_cleaned

```

```

→ DataFrame after removing rows with any NaN values:
   Unnamed: 0  RegionID  SizeRank  RegionName  RegionType  StateName  State  \
0           0        6181         0    New York      City      NY      NY
1           1       12447         1  Los Angeles      City      CA      CA
2           2       39051         2    Houston      City      TX      TX
3           3       17426         3    Chicago      City      IL      IL
4           4        6915         4  San Antonio      City      TX      TX

           Metro      CountyName  1996-01-31  ...  \
0  New York-Newark-Jersey City  Queens County  196258.0  ...
1  Los Angeles-Long Beach-Anaheim  Los Angeles County  185649.0  ...
2  Houston-The Woodlands-Sugar Land  Harris County  93518.0  ...
3  Chicago-Naperville-Elgin  Cook County  130920.0  ...

```

	San Antonio-New Braunfels			Bexar County			94041.0 ...
	2019-06-30	2019-07-31	2019-08-31	2019-09-30	2019-10-31	2019-11-30	\
0	659421.0	659007.0	658239.0	656925.0	655613.0	654394.0	
1	712660.0	713807.0	715688.0	718245.0	721896.0	725180.0	
2	186844.0	187464.0	188070.0	188496.0	189125.0	189612.0	
3	248372.0	248646.0	248725.0	248483.0	248278.0	248090.0	
4	182732.0	183350.0	183930.0	184846.0	185490.0	186244.0	
	2019-12-31	2020-01-31	2020-02-29	2020-03-31			
0	653930.0	653901.0	653565.0	652307.0			
1	730358.0	735910.0	744137.0	752508.0			
2	190179.0	190395.0	190938.0	191907.0			
3	248029.0	248220.0	248599.0	249152.0			
4	186420.0	186962.0	187129.0	187718.0			

[5 rows x 300 columns]

```

1 # Remove location identifier since only one city has data for each month/year
2 data.drop('State',axis=1,inplace=True)
3 data.drop('CountyName',axis=1,inplace=True)
4 data.drop('SizeRank',axis=1,inplace=True)
5 data.drop('Metro',axis=1,inplace=True)
6 data.drop('Unnamed: 0',axis=1,inplace=True)
7 data.drop('RegionID',axis=1,inplace=True)
8 data.drop('RegionType',axis=1,inplace=True)
9 data.drop('StateName',axis=1,inplace=True)
10 data = data.reset_index(drop=True)
11
12 # Select single city (New York)
13 data = data[data['RegionName']=='New York']
14 data.drop('RegionName',axis=1,inplace=True)
15 print(data)

```

	1996-01-31	1996-02-29	1996-03-31	1996-04-30	1996-05-31	1996-06-30	\
0	196258.0	195693.0	195383.0	194836.0	194652.0	194520.0	
	1996-07-31	1996-08-31	1996-09-30	1996-10-31	... 2019-06-30	\	
0	194447.0	194313.0	194271.0	194341.0	... 659421.0		
	2019-07-31	2019-08-31	2019-09-30	2019-10-31	2019-11-30	2019-12-31	\
0	659007.0	658239.0	656925.0	655613.0	654394.0	653930.0	
	2020-01-31	2020-02-29	2020-03-31				
0	653901.0	653565.0	652307.0				

[1 rows x 291 columns]

Adding Interest Rate Data

```

1 path = kagglehub.dataset_download("raoofiali/us-interest-rate-weekly")
2

```

```

3 print("Files in the dataset:")
4 for root, dirs, files in os.walk(path):
5     for file in files:
6         print(os.path.join(root, file))
7
8 xlsx_path = os.path.join(path, "Us-Interest Rate-Weekly.xlsx")
9 ir_df = pd.read_excel(xlsx_path)
10 ir_df.drop('Unnamed: 0',axis=1,inplace=True)
11 print(ir_df.head())
12 print(ir_df.tail())

```



Downloading from <https://www.kaggle.com/api/v1/datasets/download/raoofiali/us-interest-r>
 100%[██████████] 31.5k/31.5k [00:00<00:00, 29.3MB/s]Extracting files...

Files in the dataset:

/root/.cache/kagglehub/datasets/raoofiali/us-interest-rate-weekly/versions/1/Us-Interest

	Date	Value
0	1971-08-04	5.50
1	1971-08-15	5.50
2	1971-08-16	5.75
3	1971-08-31	5.75
4	1971-09-01	5.13
	Date	Value
1678	2024-02-29	5.5
1679	2024-03-19	5.5
1680	2024-03-20	5.5
1681	2024-04-30	5.5
1682	2024-05-01	5.5



```

1 # convert date format
2 ir_df['Date'] = pd.to_datetime(ir_df['Date'])
3
4 # Filter to include only rows between January 1996 and March 2020 to match housing data
5 start_date = pd.to_datetime('1996-01-01')
6 end_date = pd.to_datetime('2020-03-31')
7 filtered_ir_df = ir_df[(ir_df['Date'] >= start_date) & (ir_df['Date'] <= end_date)]
8
9 # Resample the data to get the monthly average
10 ir_df = filtered_ir_df.resample('M', on='Date').mean().reset_index()
11
12 # create time index
13 ir_df['Year'] = ir_df['Date'].dt.year
14 ir_df['Month'] = ir_df['Date'].dt.month
15 ir_df['TimeIndex'] = (ir_df['Year'] - ir_df['Year'].min()) * 12 + (ir_df['Month'] - ir_c
16 ir_df.drop('Date',axis=1,inplace=True)
17
18 print(ir_df.head())
19 print(ir_df.tail())

```



	Value	Year	Month	TimeIndex
0	5.375	1996	1	0

1	5.250	1996	2	1
2	5.250	1996	3	2
3	5.250	1996	4	3
4	5.250	1996	5	4
	Value	Year	Month	TimeIndex
286	1.750	2019	11	286
287	1.750	2019	12	287
288	1.750	2020	1	288
289	1.750	2020	2	289
290	1.125	2020	3	290

Adding Inflation Rate Data

```

1 path = kagglehub.dataset_download("varpit94/us-inflation-data-updated-till-may-2021")
2
3 print("Files in the dataset:")
4 for root, dirs, files in os.walk(path):
5     for file in files:
6         print(os.path.join(root, file))
7
8 csv_path = os.path.join(path, "US CPI.csv")
9 cpi_df = pd.read_csv(csv_path)
10
11 print(cpi_df.head())
12 print(cpi_df.tail())

```



Downloading from <https://www.kaggle.com/api/v1/datasets/download/varpit94/us-inflation-c>
 100%|██████████| 4.53k/4.53k [00:00<00:00, 6.90MB/s]Extracting files...

Files in the dataset:

/root/.cache/kagglehub/datasets/varpit94/us-inflation-data-updated-till-may-2021/version

	Yearmon	CPI
0	01-01-1913	9.8
1	01-02-1913	9.8
2	01-03-1913	9.8
3	01-04-1913	9.8
4	01-05-1913	9.7
	Yearmon	CPI
1298	01-03-2021	264.877
1299	01-04-2021	267.054
1300	01-05-2021	269.195
1301	01-06-2021	271.696
1302	01-07-2021	273.003



```

1 cpi_df['Yearmon'] = pd.to_datetime(cpi_df['Yearmon'], format='%d-%m-%Y')
2
3 start_date = pd.to_datetime('1996-01-01')
4 end_date = pd.to_datetime('2020-03-31')
5 filtered_cpi_df = cpi_df[(cpi_df['Yearmon'] >= start_date) & (cpi_df['Yearmon'] <= end_c
6 filtered_cpi_df = filtered_cpi_df.reset_index(drop=True)

```

```

7
8 filtered_cpi_df['Year'] = filtered_cpi_df['Yearmon'].dt.year
9 filtered_cpi_df['Month'] = filtered_cpi_df['Yearmon'].dt.month
10 filtered_cpi_df['TimeIndex'] = (filtered_cpi_df['Year'] - filtered_cpi_df['Year'].min())
11 filtered_cpi_df = filtered_cpi_df.reset_index(drop=True)
12
13 print(filtered_cpi_df)

```

```

↗
   Yearmon    CPI  Year  Month  TimeIndex
0  1996-01-01  154.400  1996     1         0
1  1996-02-01  154.900  1996     2         1
2  1996-03-01  155.700  1996     3         2
3  1996-04-01  156.300  1996     4         3
4  1996-05-01  156.600  1996     5         4
..      ...     ...     ...     ...     ...
286 2019-11-01  257.208  2019    11        286
287 2019-12-01  256.974  2019    12        287
288 2020-01-01  257.971  2020     1        288
289 2020-02-01  258.678  2020     2        289
290 2020-03-01  258.115  2020     3        290

```

[291 rows x 5 columns]

Adding Unemployment rate data

```

1 # download unemployment rate data
2 path = kagglehub.dataset_download("axelorbenson/unemployment-data-19482021")
3
4 print("Files in the dataset:")
5 for root, dirs, files in os.walk(path):
6     for file in files:
7         print(os.path.join(root, file))
8
9 # Load CSV file
10 csv_path = os.path.join(path, "unemployment_rate_data.csv")
11 un_df = pd.read_csv(csv_path)
12
13 print(un_df.head())
14 print(un_df.tail())

```

```

↗ Downloading from https://www.kaggle.com/api/v1/datasets/download/axelorbenson/unemp
100%|██████████| 13.5k/13.5k [00:00<00:00, 14.4MB/s]Extracting files...
Files in the dataset:
/root/.cache/kagglehub/datasets/axelorbenson/unemployment-data-19482021/versions/1/unemp
   date  unrate  unrate_men  unrate_women  unrate_16_to_17 \
0  1/1/1948    4.0         4.2          3.5         10.8
1  2/1/1948    4.7         4.7          4.8         15.0
2  3/1/1948    4.5         4.5          4.4         13.2
3  4/1/1948    4.0         4.0          4.1          9.9
4  5/1/1948    3.4         3.3          3.4          6.4

   unrate_18_to_19  unrate_20_to_24  unrate_25_to_34  unrate_35_to_44 \

```

0	9.6	6.6	3.6	2.6
1	9.5	8.0	4.0	3.2
2	9.3	8.6	3.5	3.2
3	8.1	6.8	3.5	3.1
4	7.2	6.3	2.8	2.5

	unrate_45_to_54	unrate_55_over
0	2.7	3.6
1	3.4	4.0
2	2.9	3.5
3	2.9	3.2
4	2.3	2.9

	date	unrate	unrate_men	unrate_women	unrate_16_to_17 \
882	7/1/2021	5.7	5.5	5.8	12.8
883	8/1/2021	5.3	5.1	5.5	10.7
884	9/1/2021	4.6	4.6	4.5	9.2
885	10/1/2021	4.3	4.2	4.4	8.6
886	11/1/2021	3.9	3.9	3.9	9.7

	unrate_18_to_19	unrate_20_to_24	unrate_25_to_34	unrate_35_to_44 \
882	9.9	9.5	6.3	4.8
883	11.0	9.1	5.8	4.4
884	12.6	7.7	5.0	3.8
885	12.7	6.8	4.5	3.6
886	11.0	6.6	3.8	3.6

	unrate_45_to_54	unrate_55_over
882	4.0	4.6
883	4.2	4.1
884	3.7	3.3
885	3.5	3.3
886	2.8	3.1

```

1 # select same range of dates of housing data and only the overall unemployment rate
2 un_df = un_df.iloc[576:576+291][['unrate','date']]
3 un_df = un_df.reset_index(drop=True)
4
5 # Convert the date column to get specific year and month feature
6 un_df['date'] = pd.to_datetime(un_df['date'])
7 un_df['Year'] = un_df['date'].dt.year
8 un_df['Month'] = un_df['date'].dt.month
9 un_df['TimeIndex'] = (un_df['Year'] - un_df['Year'].min()) * 12 + (un_df['Month'] - un_c
10 un_df.drop('date',axis=1,inplace=True)

```

Adding GDP Growth %

```

1 # Download data
2 path = kagglehub.dataset_download("rajkumarpandey02/economy-of-the-united-states")
3
4 print("Path to dataset files:", path)

```



```

5
6 print("Files in the dataset:")
7 for root, dirs, files in os.walk(path):
8     for file in files:
9         print(os.path.join(root, file))
10
11 csv_path = os.path.join(path, "Economy of the United States.csv")
12 gdp_df = pd.read_csv(csv_path)
13
14 print(gdp_df.head())
15 print(gdp_df.tail())

```



Path to dataset files: /root/.cache/kagglehub/datasets/rajkumarpandey02/economy-of-th

Files in the dataset:

/root/.cache/kagglehub/datasets/rajkumarpandey02/economy-of-the-united-states/version

Unnamed: 0	Year	GDP (in Bil. US\$PPP)	GDP per capita (in US\$ PPP)	\
0	0 1980	2857.3	12552.9	
1	1 1981	3207.0	13948.7	
2	2 1982	3343.8	14405.0	
3	3 1983	3634.0	15513.7	
4	4 1984	4037.7	17086.4	

	GDP (in Bil. US\$nominal)	GDP per capita (in US\$ nominal)	\
0	2857.3	12552.9	
1	3207.0	13948.7	
2	3343.8	14405.0	
3	3634.0	15513.7	
4	4037.7	17086.4	

	GDP growth (real)	Inflation rate (in Percent)	Unemployment (in Percent)	\
0	-0.30%	13.50%	7.20%	
1	2.50%	10.40%	7.60%	
2	-1.80%	6.20%	9.70%	
3	4.60%	3.20%	9.60%	
4	7.20%	4.40%	7.50%	

	Government debt (in % of GDP)
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN

Unnamed: 0	Year	GDP (in Bil. US\$PPP)	GDP per capita (in US\$ PPP)	\
43	43 2023	26185.2	78421.9	
44	44 2024	27057.2	80779.3	
45	45 2025	28045.3	83463.2	
46	46 2026	29165.5	86521.2	
47	47 2027	30281.5	89546.4	

	GDP (in Bil. US\$nominal)	GDP per capita (in US\$ nominal)	\
43	26185.2	78421.9	
44	27057.2	80779.3	
45	28045.3	83463.2	

	GDP growth (real)	Inflation rate (in Percent)	Unemployment (in Percent)	\
43	1.00%	3.50%	4.60%	
44	1.20%	2.20%	5.40%	
45	1.80%	2.00%	5.40%	
46	2.10%	2.00%	4.90%	
47	1.90%	2.00%	4.70%	

	Government debt (in % of GDP)
43	122.90%
44	126.00%
45	129.40%
46	132.20%
47	134.90%

```

1 gdp_df = gdp_df[gdp_df['Year'] >= 1996]
2 gdp_df = gdp_df[gdp_df['Year'] <= 2020]
3 gdp_df = gdp_df.reset_index(drop=True)
4 gdp_df = gdp_df[['Year', 'GDP growth (real)']]
5
6 gdp_df['GDP growth (real)'] = gdp_df['GDP growth (real)'].str.replace('%', '')
7 gdp_df['GDP Growth'] = pd.to_numeric(gdp_df['GDP growth (real)'])
8 gdp_df.drop('GDP growth (real)', axis=1, inplace=True)
9
10 # add instance for each month
11 gdp_df = gdp_df.loc[gdp_df.index.repeat(12)].reset_index(drop=True)
12 gdp_df['Month'] = (gdp_df.groupby('Year').cumcount() % 12) + 1
13 gdp_df = gdp_df.iloc[:-9]
14
15 print(gdp_df.head())
16 print(gdp_df.tail())

```

```

➡
   Year  GDP Growth  Month
0  1996         3.8      1
1  1996         3.8      2
2  1996         3.8      3
3  1996         3.8      4
4  1996         3.8      5
   Year  GDP Growth  Month
286  2019         2.3     11
287  2019         2.3     12
288  2020        -3.4      1
289  2020        -3.4      2
290  2020        -3.4      3

```

```

1 # reshape data to have rows correspond to each time, with features being the time, price
2 reshaped_data = []
3
4 # Loop through each column to get feature dates
5 for column in data.columns:
6     year, month, day = map(int, column.split('-'))
7

```

```

8 # Loop through each row to get price for the current date
9 for index, row in data.iterrows():
10     zhvi = row[column]
11
12     reshaped_data.append({
13         'ZHVI': zhvi,
14         'Year': year,
15         'Month': month,
16         'Year-Month': f'{year}-{month}'
17     })
18
19 reshaped_df = pd.DataFrame(reshaped_data)
20
21 # Add a time index
22 reshaped_df['TimeIndex'] = (reshaped_df['Year'] - reshaped_df['Year'].min()) * 12 + (res
23
24 # Sort data by month/year
25 full_df = reshaped_df.sort_values(by=['Year', 'Month']).reset_index(drop=True)
26 full_df['Unemployment Rate'] = un_df['unrate']
27 full_df['CPI'] = filtered_cpi_df['CPI']
28 full_df['Interest Rate'] = ir_df['Value']
29 full_df['GDP Growth'] = gdp_df['GDP Growth']
30 print("Reshaped DataFrame:")
31 print(full_df)

```



Reshaped DataFrame:

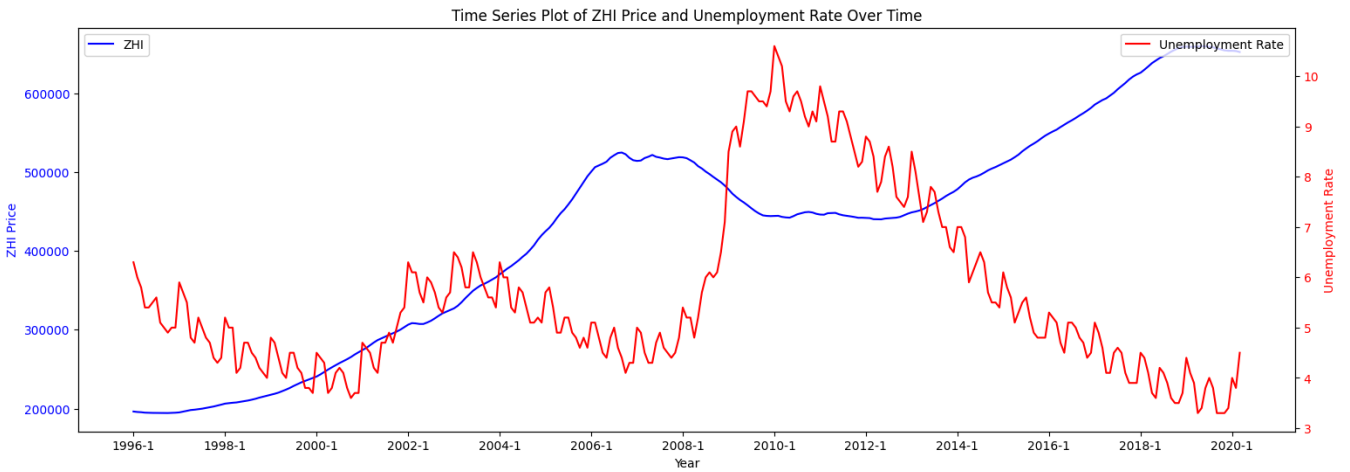
	ZHVI	Year	Month	Year-Month	TimeIndex	Unemployment Rate	CPI \
0	196258.0	1996	1	1996-1	0	6.3	154.400
1	195693.0	1996	2	1996-2	1	6.0	154.900
2	195383.0	1996	3	1996-3	2	5.8	155.700
3	194836.0	1996	4	1996-4	3	5.4	156.300
4	194652.0	1996	5	1996-5	4	5.4	156.600
..
286	654394.0	2019	11	2019-11	286	3.3	257.208
287	653930.0	2019	12	2019-12	287	3.4	256.974
288	653901.0	2020	1	2020-1	288	4.0	257.971
289	653565.0	2020	2	2020-2	289	3.8	258.678
290	652307.0	2020	3	2020-3	290	4.5	258.115

	Interest Rate	GDP Growth
0	5.375	3.8
1	5.250	3.8
2	5.250	3.8
3	5.250	3.8
4	5.250	3.8
..
286	1.750	2.3
287	1.750	2.3
288	1.750	-3.4
289	1.750	-3.4
290	1.125	-3.4

[291 rows x 9 columns]

```
1 # Create figure and primary axis
2 fig, ax1 = plt.subplots(figsize=(18, 6))
3
4 # Plot the first ZHVI dataset
5 ax1.plot(full_df['Year-Month'], full_df['ZHVI'], color='blue', label='ZHI')
6 ax1.set_xlabel('Year')
7 ax1.set_ylabel('ZHI Price', color='blue')
8 ax1.tick_params(axis='y', labelcolor='blue')
9
10 # Create a second axis sharing the same x-axis
11 ax2 = ax1.twinx()
12
13 # Plot the Unemployment Rate data
14 ax2.plot(un_df['TimeIndex'], un_df['unrate'], color='red', label='Unemployment Rate')
15 ax2.set_ylabel('Unemployment Rate', color='red')
16 ax2.tick_params(axis='y', labelcolor='red')
17
18 x_ticks = np.arange(0, 290, 24)
19 ax1.set_xticks(x_ticks)
20
21 plt.title('Time Series Plot of ZHI Price and Unemployment Rate Over Time')
22
23 # legend
24 ax1.legend(loc='upper left')
25 ax2.legend(loc='upper right')
26
27 plt.show()
```





```

1 # Create figure and primary axis
2 fig, ax1 = plt.subplots(figsize=(18, 6))
3
4 # Plot the first ZHVI dataset
5 ax1.plot(full_df['Year-Month'], full_df['ZHVI'], color='blue', label='ZHI')
6 ax1.set_xlabel('Year')
7 ax1.set_ylabel('ZHI Price', color='blue')
8 ax1.tick_params(axis='y', labelcolor='blue')
9
10 # Create a second axis sharing the same x-axis
11 ax2 = ax1.twinx()
12
13 # Plot the Unemployment Rate data
14 ax2.plot(filtered_cpi_df['TimeIndex'], filtered_cpi_df['CPI'], color='red', label='CPI')
15 ax2.set_ylabel('CPI', color='red')
16 ax2.tick_params(axis='y', labelcolor='red')
17
18 x_ticks = np.arange(0, 290, 24)
19 ax1.set_xticks(x_ticks)
20
21 plt.title('Time Series Plot of ZHI Price and CPI Over Time')

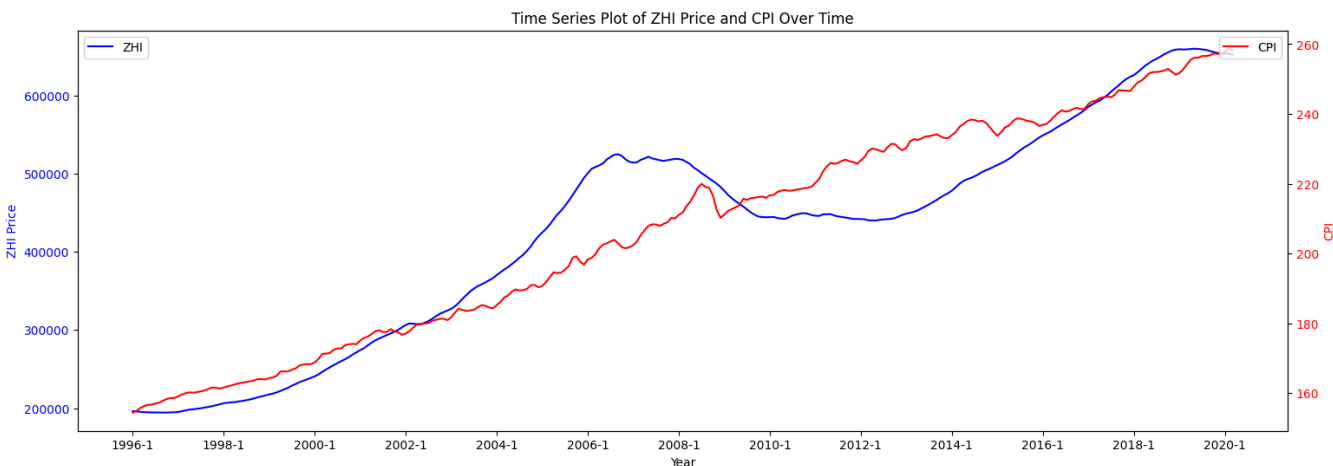
```



```

22
23 # legend
24 ax1.legend(loc='upper left')
25 ax2.legend(loc='upper right')
26
27 plt.show()

```



```

1 # Create figure and primary axis
2 fig, ax1 = plt.subplots(figsize=(18, 6))
3
4 # Plot the first ZHVI dataset
5 ax1.plot(full_df['Year-Month'], full_df['ZHVI'], color='blue', label='ZHI')
6 ax1.set_xlabel('Year')
7 ax1.set_ylabel('ZHI Price', color='blue')
8 ax1.tick_params(axis='y', labelcolor='blue')
9
10 # Create a second axis sharing the same x-axis
11 ax2 = ax1.twinx()
12
13 # Plot the Unemployment Rate data
14 ax2.plot(ir_df['TimeIndex'], ir_df['Value'], color='red', label='CPI')
15 ax2.set_ylabel('Interest Rate', color='red')

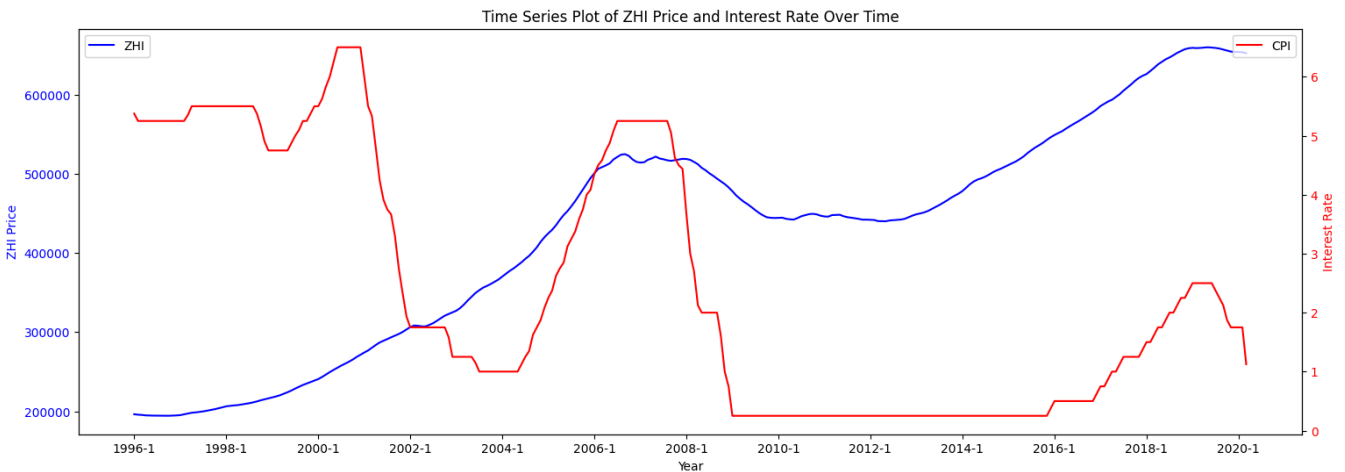
```



```

16 ax2.tick_params(axis='y', labelcolor='red')
17
18 x_ticks = np.arange(0, 290, 24)
19 ax1.set_xticks(x_ticks)
20
21 plt.title('Time Series Plot of ZHI Price and Interest Rate Over Time')
22
23 # legend
24 ax1.legend(loc='upper left')
25 ax2.legend(loc='upper right')
26
27 plt.show()

```



```

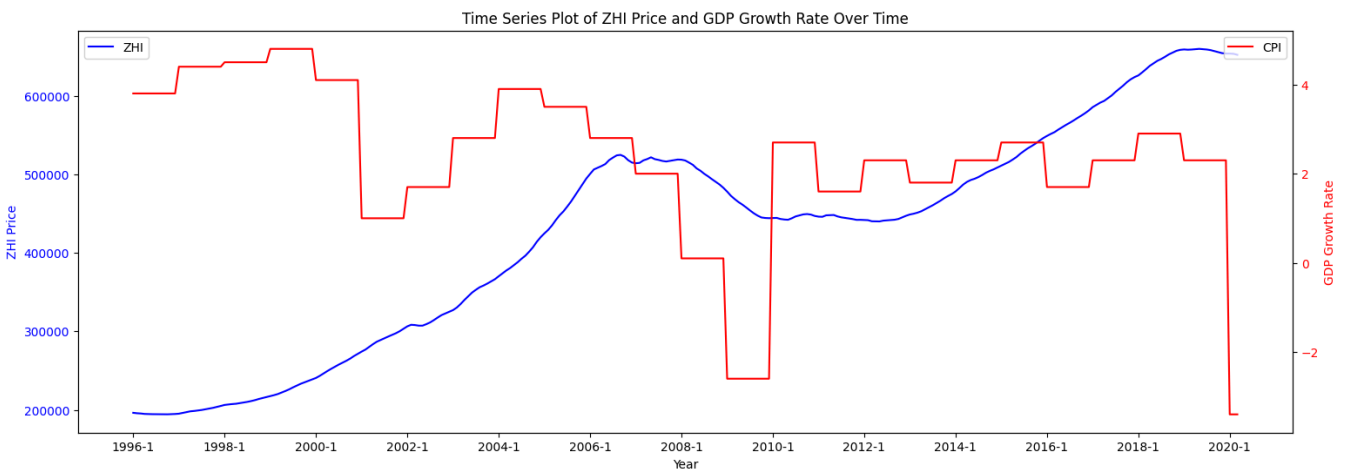
1 # Create figure and primary axis
2 fig, ax1 = plt.subplots(figsize=(18, 6))
3
4 # Plot the first ZHVI dataset
5 ax1.plot(full_df['Year-Month'], full_df['ZHVI'], color='blue', label='ZHI')
6 ax1.set_xlabel('Year')
7 ax1.set_ylabel('ZHI Price', color='blue')
8 ax1.tick_params(axis='y', labelcolor='blue')

```

```

9
10 # Create a second axis sharing the same x-axis
11 ax2 = ax1.twinx()
12
13 # Plot the Unemployment Rate data
14 ax2.plot(ir_df['TimeIndex'], gdp_df['GDP Growth'], color='red', label='CPI')
15 ax2.set_ylabel('GDP Growth Rate', color='red')
16 ax2.tick_params(axis='y', labelcolor='red')
17
18 x_ticks = np.arange(0, 290, 24)
19 ax1.set_xticks(x_ticks)
20
21 plt.title('Time Series Plot of ZHI Price and GDP Growth Rate Over Time')
22
23 # legend
24 ax1.legend(loc='upper left')
25 ax2.legend(loc='upper right')
26
27 plt.show()

```



```

1 # Split data into training and test
2 train = full_df[(full_df['Year'] < 2014) | ((full_df['Year'] == 2013) & (full_df['Month']

```



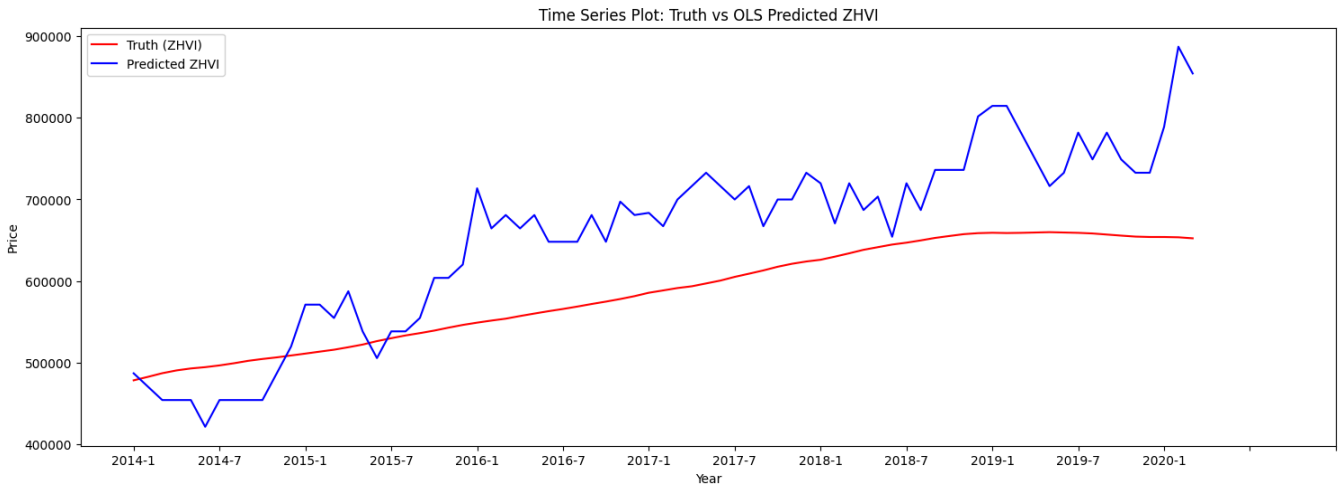
```
3 test = full_df[(full_df['Year'] > 2013) | ((full_df['Year'] == 2014) & (full_df['Month']
4
5 # Define features and target
6 X_train = train[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI', 'Interest Rate
7 y_train = train['ZHVI']
8
9 # add constant
10 X_train = sm.add_constant(X_train)
11
12 # Prediction test
13 X_test = test[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI', 'Interest Rate', '
14 X_test = sm.add_constant(X_test)
15
16 # add polynomial features and scale
17 scaler = StandardScaler()
18 poly = PolynomialFeatures(degree=2)
19 X_train_poly = poly.fit_transform(X_train)
20 X_test_poly = poly.transform(X_test)
21
22 X_train_scaled = scaler.fit_transform(X_train_poly)
23 X_test_scaled = scaler.transform(X_test_poly)
24
25 # Fit OLS model
26 model = sm.OLS(y_train, X_train_scaled)
27 results = model.fit()
28
29 predictions = results.predict(X_test_scaled)
30 test['Predicted_ZHVI'] = predictions
31
32 y_test = test['ZHVI']
33 y_pred = test['Predicted_ZHVI']
34 OLS_pred = test['Predicted_ZHVI']
35
36 # model evaluation
37 rmse = math.sqrt(mean_squared_error(y_test, y_pred))
38 print(f"OLS Root Mean Squared Error (RMSE): {rmse}")
39 mape = mean_absolute_percentage_error(y_test, y_pred)
40 print("OLS Mean Absolute Percentage Error:", mape)
41 MAE = mean_absolute_error(y_test, y_pred)
42 print("OLS Mean Absolute Error:", MAE)
43 r2 = r2_score(y_pred, y_test)
44 print(f"OLS R-squared: {r2}")
45
46 # Plot truth vs prediction
47 plt.figure(figsize=(18, 6))
48 plt.plot(test['Year-Month'], test['ZHVI'], color='red', label='Truth (ZHVI)')
49 plt.plot(test['Year-Month'], test['Predicted_ZHVI'], color='blue', label='Predicted ZHVI')
50 plt.xlabel('Year')
51 plt.ylabel('Price')
52 x_ticks = np.arange(0, 90, 6)
53 plt.xticks(x_ticks)
54 plt.title('Time Series Plot: Truth vs OLS Predicted ZHVI')
```



```
55 plt.legend(loc='upper left')
56 plt.show()
```



OLS Root Mean Squared Error (RMSE): 92178.70622305939
 OLS Mean Absolute Percentage Error: 0.13433149617978665
 OLS Mean Absolute Error: 80354.23611166129
 OLS R-squared: 0.30169319042236065



```
1 # Split data into training and test
2 train = full_df[(full_df['Year'] < 2014) | ((full_df['Year'] == 2013) & (full_df['Month'
3 test = full_df[(full_df['Year'] > 2013) | ((full_df['Year'] == 2014) & (full_df['Mont
4
5 # Define features and target
6 X_train = train[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI', 'Interest Rate
7 y_train = train['ZHVI']
8
9 # Prepare test data for prediction
10 X_test = test[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI', 'Interest Rate',
11
12 # add polynomial features and scale
13 scaler = StandardScaler()
14 poly = PolynomialFeatures(degree=2)
```

```
15 X_train_poly = poly.fit_transform(X_train)
16 X_test_poly = poly.transform(X_test)
17
18 X_train_scaled = scaler.fit_transform(X_train_poly)
19 X_test_scaled = scaler.transform(X_test_poly)
20
21 # Fit Lasso regression model
22 alphas = [0.001,0.005,0.01, 0.05, 0.1, 0.5, 1, 2, 3, 5, 10,25,50,75,100,150]
23 results = []
24 lowest_alpha = alphas[0]
25 lowest_mape = float('inf')
26
27 for alpha in alphas:
28     lasso_model = Lasso(alpha=alpha)
29     lasso_model.fit(X_train_scaled, y_train)
30
31     # Predict
32     predictions = lasso_model.predict(X_test_scaled)
33     test['Predicted_ZHVI'] = predictions
34
35     # Model evaluation
36     y_test = test['ZHVI']
37     y_pred = test['Predicted_ZHVI']
38     lasso_pred = test['Predicted_ZHVI']
39
40     mape = mean_absolute_percentage_error(y_test, y_pred)
41     results.append(mape)
42     if mape < lowest_mape:
43         lowest_mape = mape
44         lowest_alpha = alpha
45
46 print("Lowest MAPE:", lowest_mape)
47 print("Lowest Alpha:", lowest_alpha)
48
49 alpha = lowest_alpha
50
51 lasso_model = Lasso(alpha=alpha)
52 lasso_model.fit(X_train_scaled, y_train)
53
54 # Predict
55 predictions = lasso_model.predict(X_test_scaled)
56 test['Predicted_ZHVI'] = predictions
57
58 # Model evaluation
59 y_test = test['ZHVI']
60 y_pred = test['Predicted_ZHVI']
61 lasso_pred = test['Predicted_ZHVI']
62
63 rmse = math.sqrt(mean_squared_error(y_test, y_pred))
64 print(f"\n\nLasso Root Mean Squared Error (RMSE): {rmse}")
65 mape = mean_absolute_percentage_error(y_test, y_pred)
```



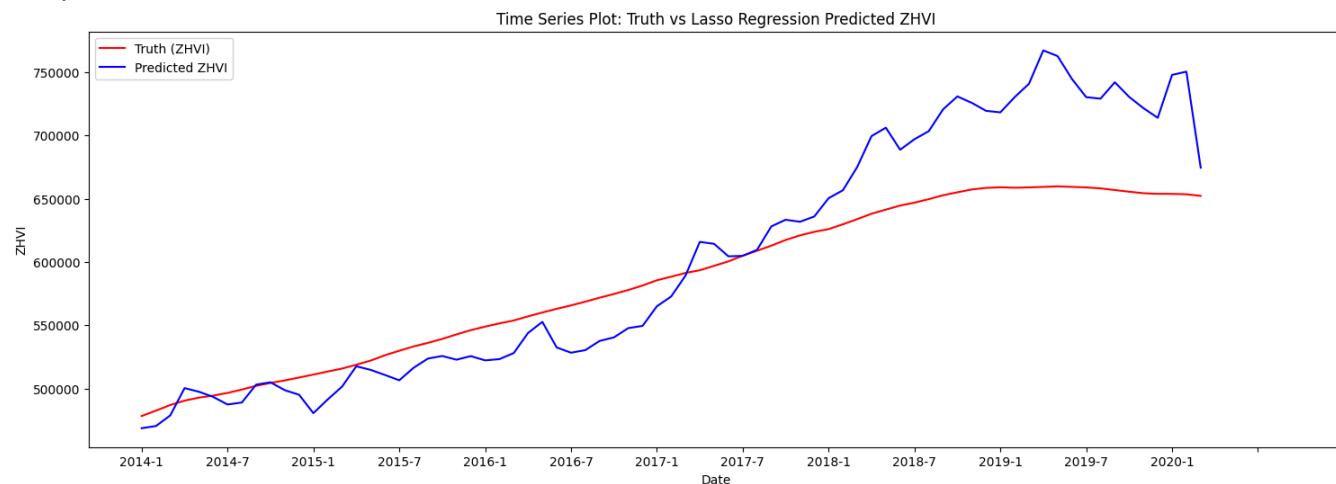
```
66 print("Lasso Mean Absolute Percentage Error:", mape)
67 MAE = mean_absolute_error(y_test, y_pred)
68 print("Mean Absolute Error:", MAE)
69 r2 = r2_score(y_pred, y_test)
70 print(f"R-squared: {r2}")
71
72 # Plot truth vs prediction
73 plt.figure(figsize=(18, 6))
74 plt.plot(test['Year-Month'], test['ZHVI'], color='red', label='Truth (ZHVI)')
75 plt.plot(test['Year-Month'], test['Predicted_ZHVI'], color='blue', label='Predicted ZHVI')
76 plt.xlabel('Date')
77 plt.ylabel('ZHVI')
78 x_ticks = np.arange(0, 86, 6)
79 plt.xticks(x_ticks)
80 plt.title('Time Series Plot: Truth vs Lasso Regression Predicted ZHVI')
81 plt.legend(loc='upper left')
82 plt.show()
```





Lowest MAPE: 0.05492470799176251
Lowest Alpha: 50

Lasso Root Mean Squared Error (RMSE): 44618.016149679155
Lasso Mean Absolute Percentage Error: 0.05492470799176251
Mean Absolute Error: 34030.374729689334
R-squared: 0.7853309788737721

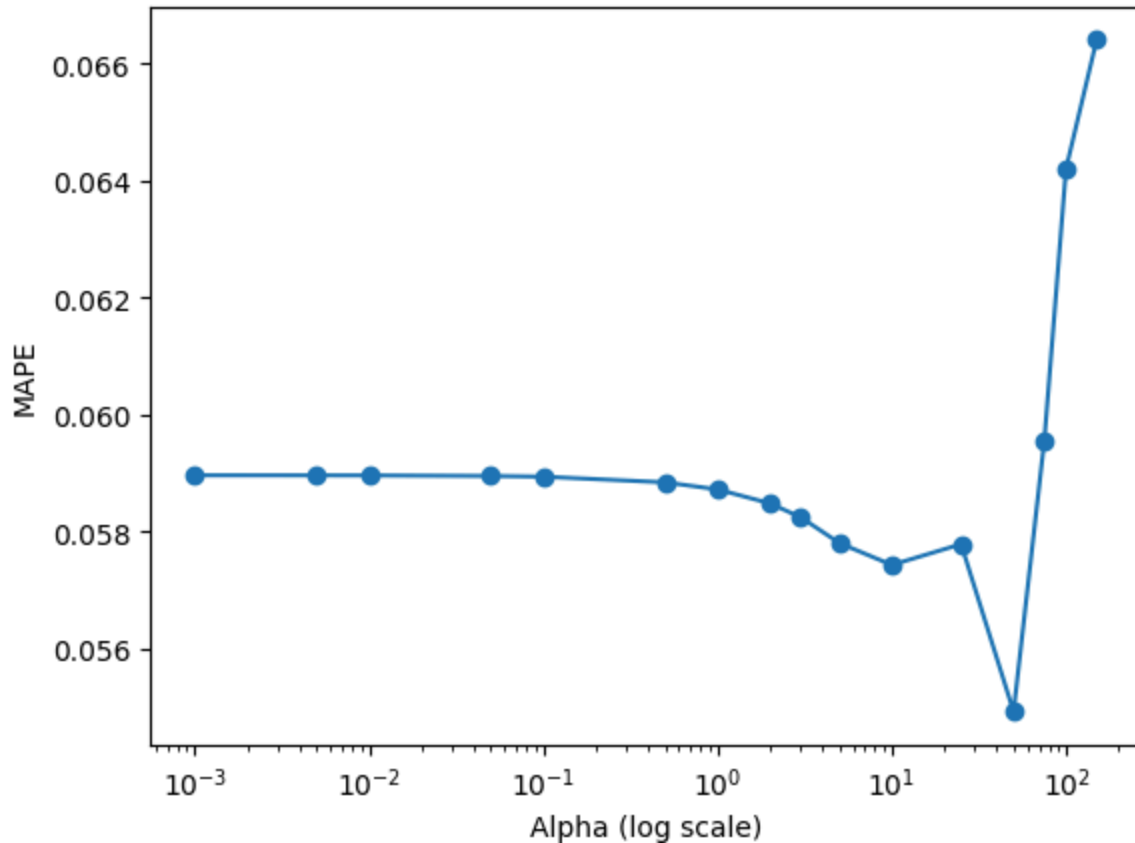


```
1 # Plot hyperparameter tuning
2 plt.plot(alphas, results, marker='o')
3 plt.xscale('log')
4
5 plt.xlabel('Alpha (log scale)')
6 plt.ylabel('MAPE')
7 plt.title('Lasso Regression Alpha Tuning - MAPE vs Alpha (Logarithmic X-axis)')
8
9 plt.show()
```





Lasso Regression Alpha Tuning - MAPE vs Alpha (Logarithmic X-axis)



```

1 # Split data into training and test
2 train = full_df[(full_df['Year'] < 2014) | ((full_df['Year'] == 2013) & (full_df['Month'
3 test = full_df[(full_df['Year'] > 2013) | ((full_df['Year'] == 2014) & (full_df['Month']
4
5 # Define features and target
6 X_train = train[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI', 'Interest Rate
7 y_train = train['ZHVI']
8
9 X_test = test[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI', 'Interest Rate',
10
11 # add polynomial features and scale
12 scaler = StandardScaler()
13 poly = PolynomialFeatures(degree=2)
14 X_train_poly = poly.fit_transform(X_train)
15 X_test_poly = poly.transform(X_test)
16
17 X_train_scaled = scaler.fit_transform(X_train_poly)
18 X_test_scaled = scaler.transform(X_test_poly)
19
20
21 # Fit Ridge regression model
22 alphas = [0.001, 0.005, 0.01, 0.05, 0.075, 0.1, 0.25, 0.35, 0.5, 1, 2, 3, 5, 10]
23
24 lowest_alpha = alphas[0]
25 lowest_mape = float('inf')

```

```
26 results = []
27 for alpha in alphas:
28     ridge_model = Ridge(alpha=alpha)
29     ridge_model.fit(X_train_scaled, y_train)
30
31     # Predict
32     predictions = ridge_model.predict(X_test_scaled)
33     test['Predicted_ZHVI'] = predictions
34
35     # Model evaluation
36     y_test = test['ZHVI']
37     y_pred = test['Predicted_ZHVI']
38     ridge_pred = test['Predicted_ZHVI']
39
40     mape = mean_absolute_percentage_error(y_test, y_pred)
41     results.append(mape)
42     if mape < lowest_mape:
43         lowest_mape = mape
44         lowest_alpha = alpha
45
46 print("Lowest MAPE:", lowest_mape)
47 print("Lowest Alpha:", lowest_alpha)
48
49 alpha = lowest_alpha
50 ridge_model = Ridge(alpha=alpha)
51 ridge_model.fit(X_train_scaled, y_train)
52
53 # Predict
54 predictions = ridge_model.predict(X_test_scaled)
55 test['Predicted_ZHVI'] = predictions
56
57 # Model evaluation
58 y_test = test['ZHVI']
59 y_pred = test['Predicted_ZHVI']
60 ridge_pred = test['Predicted_ZHVI']
61
62 rmse = math.sqrt(mean_squared_error(y_test, y_pred))
63 print(f"\n\nRR Root Mean Squared Error (RMSE): {rmse}")
64 mape = mean_absolute_percentage_error(y_test, y_pred)
65 print("RR Mean Absolute Percentage Error:", mape)
66 MAE = mean_absolute_error(y_test, y_pred)
67 print("RR Mean Absolute Error:", MAE)
68 r2 = r2_score(y_pred, y_test)
69 print(f"R-squared: {r2}")
70
71 # Plot truth vs prediction
72 plt.figure(figsize=(18, 6))
73 plt.plot(test['Year-Month'], test['ZHVI'], color='red', label='Truth (ZHVI)')
74 plt.plot(test['Year-Month'], test['Predicted_ZHVI'], color='blue', label='Predicted ZHVI')
75 plt.xlabel('Year')
76 plt.ylabel('ZHVI')
```



```

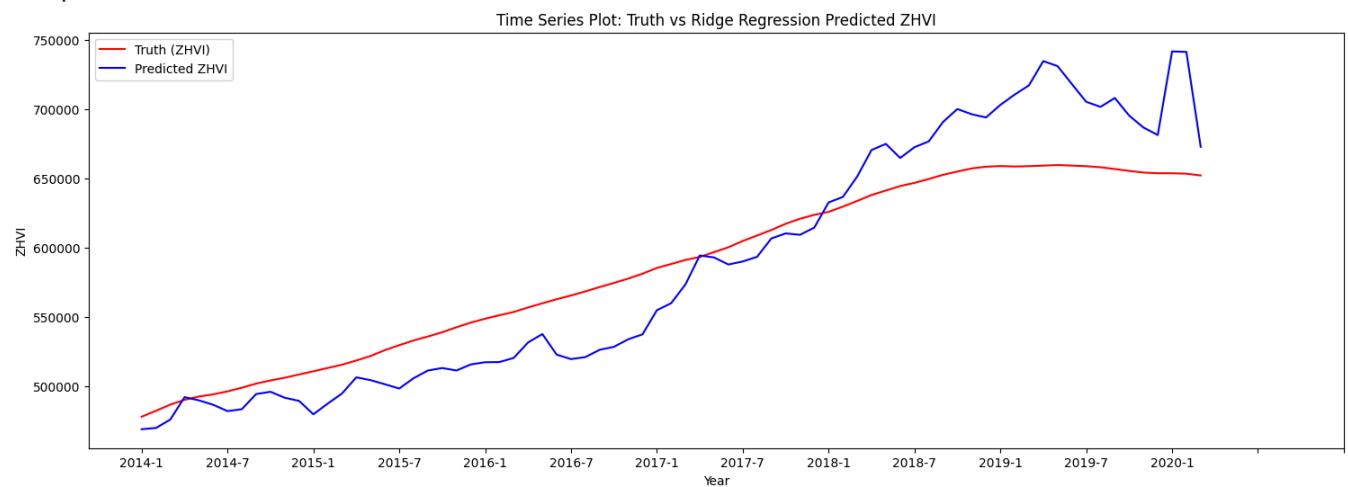
77 x_ticks = np.arange(0, 86, 6)
78 plt.xticks(x_ticks)
79 plt.title('Time Series Plot: Truth vs Ridge Regression Predicted ZHVI')
80 plt.legend(loc='upper left')
81 plt.show()

```



Lowest MAPE: 0.04815805234530082
Lowest Alpha: 0.25

RR Root Mean Squared Error (RMSE): 34586.997710405834
RR Mean Absolute Percentage Error: 0.04815805234530082
RR Mean Absolute Error: 28891.172716798137
R-squared: 0.846871147352243



```

1 # Plot hyperparameter tuning
2 plt.plot(alphas, results, marker='o')
3 plt.xscale('log')
4
5 plt.xlabel('Alpha (log scale)')
6 plt.ylabel('MAPE')
7 plt.title('Ridge Regression Alpha Tuning - MAPE vs Alpha (Logarithmic X-axis)')

```


8

