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

```
#!pip install ydata-profiling
#!pip install tensorflow

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import kagglehub
import math
import os
import warnings

#from ydata_profiling import ProfileReport
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error, mean_abs

from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
#from tensorflow.keras.models import Sequential
#from tensorflow.keras.layers import Dense
from IPython.display import clear_output, display, HTML

warnings.filterwarnings("ignore")
clear_output()
```

Adding Housing Data

```python
# Download housing data
path = kagglehub.dataset_download("paultimothymooney/zillow-house-price-data")

print("Files in the dataset:")
for root, dirs, files in os.walk(path):
    for file in files:
        print(os.path.join(root, file))
```

```
Files in the dataset:
/kaggle/input/zillow-house-price-data/Sale_Prices_State.csv
/kaggle/input/zillow-house-price-data/State_Zhvi_2bedroom.csv
/kaggle/input/zillow-house-price-data/DaysOnZillow_State.csv
/kaggle/input/zillow-house-price-data/State_MedianRentalPrice_Sfr.csv
/kaggle/input/zillow-house-price-data/City_Zhvi_SingleFamilyResidence.csv
/kaggle/input/zillow-house-price-data/State_Zri_SingleFamilyResidenlceRental.csv
/kaggle/input/zillow-house-price-data/City_Zri_SingleFamilyResidenceRental.csv
/kaggle/input/zillow-house-price-data/City_Zri_AllHomesPlusMultifamily.csv
/kaggle/input/zillow-house-price-data/State_Zhvi_3bedroom.csv
/kaggle/input/zillow-house-price-data/City_MedianRentalPrice_1Bedroom.csv
/kaggle/input/zillow-house-price-data/State_Zhvi_5BedroomOrMore.csv
/kaggle/input/zillow-house-price-data/State_MedianRentalPrice_Studio.csv
/kaggle/input/zillow-house-price-data/State_MedianRentalPrice_3Bedroom.csv
/kaggle/input/zillow-house-price-data/City_MedianRentalPrice_4Bedroom.csv
/kaggle/input/zillow-house-price-data/City_Zhvi_5BedroomOrMore.csv
/kaggle/input/zillow-house-price-data/City_MedianRentalPrice_AllHomes.csv
/kaggle/input/zillow-house-price-data/City_Zhvi_2bedroom.csv
/kaggle/input/zillow-house-price-data/City_Zhvi_Condominum.csv
/kaggle/input/zillow-house-price-data/State_Zhvi_AllHomes.csv
/kaggle/input/zillow-house-price-data/City_Zhvi_4bedroom.csv
/kaggle/input/zillow-house-price-data/City_Zhvi_3bedroom.csv
/kaggle/input/zillow-house-price-data/City_Zhvi_1bedroom.csv
/kaggle/input/zillow-house-price-data/Sale_Prices_City.csv
/kaggle/input/zillow-house-price-data/DaysOnZillow_City.csv
/kaggle/input/zillow-house-price-data/City_MedianRentalPrice_3Bedroom.csv
/kaggle/input/zillow-house-price-data/City_MedianRentalPrice_Sfr.csv
/kaggle/input/zillow-house-price-data/State_MedianRentalPrice_AllHomes.csv
/kaggle/input/zillow-house-price-data/State_Zhvi_SingleFamilyResidence.csv
/kaggle/input/zillow-house-price-data/State_Zri_AllHomesPlusMultifamily.csv
/kaggle/input/zillow-house-price-data/City_MedianRentalPrice_2Bedroom.csv
/kaggle/input/zillow-house-price-data/City_MedianRentalPrice_5BedroomOrMore.csv
/kaggle/input/zillow-house-price-data/State_Zhvi_Condominum.csv
/kaggle/input/zillow-house-price-data/State_MedianRentalPrice_4Bedroom.csv
/kaggle/input/zillow-house-price-data/City_MedianRentalPrice_Studio.csv
/kaggle/input/zillow-house-price-data/State_Zhvi_4bedroom.csv
/kaggle/input/zillow-house-price-data/State_Zhvi_1bedroom.csv
/kaggle/input/zillow-house-price-data/State_MedianRentalPrice_5BedroomOrMore.csv
/kaggle/input/zillow-house-price-data/State_MedianRentalPrice_2Bedroom.csv
/kaggle/input/zillow-house-price-data/City_Zhvi_AllHomes.csv
/kaggle/input/zillow-house-price-data/State_MedianRentalPrice_1Bedroom.csv
```

```
csv_path = os.path.join(path, "City_Zhvi_AllHomes.csv")
df = pd.read_csv(csv_path)
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]
```

```
# remove rows with NaN
df_cleaned = df.dropna()
print("DataFrame after removing rows with any NaN values:")
print(df_cleaned.head())
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  ...
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]
```

```
# Remove location identifier since only one city has data for each month/year
data.drop('State',axis=1,inplace=True)
data.drop('CountyName',axis=1,inplace=True)
data.drop('SizeRank',axis=1,inplace=True)
data.drop('Metro',axis=1,inplace=True)
data.drop('Unnamed: 0',axis=1,inplace=True)
data.drop('RegionID',axis=1,inplace=True)
data.drop('RegionType',axis=1,inplace=True)
data.drop('StateName',axis=1,inplace=True)
data = data.reset_index(drop=True)

# Select single city (New York)
data = data[data['RegionName']=='New York']
data.drop('RegionName',axis=1,inplace=True)
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

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

```

```
print("Files in the dataset:")
for root, dirs, files in os.walk(path):
    for file in files:
        print(os.path.join(root, file))

xlsx_path = os.path.join(path, "Us-Interest Rate-Weekly.xlsx")
ir_df = pd.read_excel(xlsx_path)
ir_df.drop('Unnamed: 0',axis=1,inplace=True)
print(ir_df.head())
print(ir_df.tail())
```

```
Files in the dataset:
/kaggle/input/us-interest-rate-weekly/Us-Interest Rate-Weekly.xlsx
         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
```

```
# convert date format
ir_df['Date'] = pd.to_datetime(ir_df['Date'])

# Filter to include only rows between January 1996 and March 2020 to match housing data
start_date = pd.to_datetime('1996-01-01')
end_date = pd.to_datetime('2020-03-31')
filtered_ir_df = ir_df[(ir_df['Date'] >= start_date) & (ir_df['Date'] <= end_date)]

# Resample the data to get the monthly average
ir_df = filtered_ir_df.resample('M', on='Date').mean().reset_index()

# create time index
ir_df['Year'] = ir_df['Date'].dt.year
ir_df['Month'] = ir_df['Date'].dt.month
ir_df['TimeIndex'] = (ir_df['Year'] - ir_df['Year'].min()) * 12 + (ir_df['Month'] - i
ir_df.drop('Date',axis=1,inplace=True)

print(ir_df.head())
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

```
path = kagglehub.dataset_download("varpit94/us-inflation-data-updated-till-may-2021")

print("Files in the dataset:")
for root, dirs, files in os.walk(path):
    for file in files:
        print(os.path.join(root, file))

csv_path = os.path.join(path, "US CPI.csv")
cpi_df = pd.read_csv(csv_path)

print(cpi_df.head())
print(cpi_df.tail())
```

```
Files in the dataset:
/kaggle/input/us-inflation-data-updated-till-may-2021/US CPI.csv
      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
```

```
cpi_df['Yearmon'] = pd.to_datetime(cpi_df['Yearmon'], format='%d-%m-%Y')

start_date = pd.to_datetime('1996-01-01')
end_date = pd.to_datetime('2020-03-31')
filtered_cpi_df = cpi_df[(cpi_df['Yearmon'] >= start_date) & (cpi_df['Yearmon'] <= end_c
filtered_cpi_df = filtered_cpi_df.reset_index(drop=True)

filtered_cpi_df['Year'] = filtered_cpi_df['Yearmon'].dt.year
filtered_cpi_df['Month'] = filtered_cpi_df['Yearmon'].dt.month
filtered_cpi_df['TimeIndex'] = (filtered_cpi_df['Year'] - filtered_cpi_df['Year'].min())
filtered_cpi_df = filtered_cpi_df.reset_index(drop=True)

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

```
# download unemployment rate data
path = kagglehub.dataset_download("axeltorbenson/unemployment-data-19482021")

print("Files in the dataset:")
for root, dirs, files in os.walk(path):
    for file in files:
        print(os.path.join(root, file))

# Load CSV file
csv_path = os.path.join(path, "unemployment_rate_data.csv")
un_df = pd.read_csv(csv_path)

print(un_df.head())
print(un_df.tail())
```

```
Files in the dataset:
/kaggle/input/unemployment-data-19482021/unemployment_rate_data.csv
       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
```

```
# select same range of dates of housing data and only the overall unemployment rate
un_df = un_df.iloc[576:576+291][['unrate','date']]
un_df = un_df.reset_index(drop=True)

# Convert the date column to get specific year and month feature
un_df['date'] = pd.to_datetime(un_df['date'])
un_df['Year'] = un_df['date'].dt.year
un_df['Month'] = un_df['date'].dt.month
un_df['TimeIndex'] = (un_df['Year'] - un_df['Year'].min()) * 12 + (un_df['Month'] - un_df
un_df.drop('date',axis=1,inplace=True)
```

Adding GDP Growth %

```
# Download data
path = kagglehub.dataset_download("rajkumarpandey02/economy-of-the-united-states")

print("Path to dataset files:", path)

print("Files in the dataset:")
for root, dirs, files in os.walk(path):
    for file in files:
        print(os.path.join(root, file))

csv_path = os.path.join(path, "Economy of the United States.csv")
gdp_df = pd.read_csv(csv_path)

print(gdp_df.head())
print(gdp_df.tail())
```

```
Path to dataset files: /kaggle/input/economy-of-the-united-states
Files in the dataset:
/kaggle/input/economy-of-the-united-states/Economy of the United States.csv
   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
46                   29165.5                          86521.2
47                   30281.5                          89546.4

   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%
```

```
gdp_df = gdp_df[gdp_df['Year'] >= 1996]
gdp_df = gdp_df[gdp_df['Year'] <= 2020]
gdp_df = gdp_df.reset_index(drop=True)
gdp_df = gdp_df[['Year','GDP growth (real)']]

gdp_df['GDP growth (real)'] = gdp_df['GDP growth (real)'].str.replace('%', '')
gdp_df['GDP Growth'] = pd.to_numeric(gdp_df['GDP growth (real)'])
gdp_df.drop('GDP growth (real)',axis=1,inplace=True)

# add instance for each month
gdp_df = gdp_df.loc[gdp_df.index.repeat(12)].reset_index(drop=True)
gdp_df['Month'] = (gdp_df.groupby('Year').cumcount() % 12) + 1
gdp_df = gdp_df.iloc[:-9]

print(gdp_df.head())
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
```

```
# reshape data to have rows correspond to each time, with features being the time, price
reshaped_data = []

# Loop through each column to get feature dates
for column in data.columns:
  year, month,day = map(int, column.split('-'))

  # Loop through each row to get price for the current date
  for index, row in data.iterrows():
   zhvi = row[column]

   reshaped_data.append({
      'ZHVI': zhvi,
      'Year': year,
      'Month': month,
      'Year-Month': f'{year}-{month}'
      })

```

```
reshaped_df = pd.DataFrame(reshaped_data)

# Add a time index
reshaped_df['TimeIndex'] = (reshaped_df['Year'] - reshaped_df['Year'].min()) * 12 + (res

# Sort data by month/year
full_df = reshaped_df.sort_values(by=['Year', 'Month']).reset_index(drop=True)
full_df['Unemployment Rate'] = un_df['unrate']
full_df['CPI'] = filtered_cpi_df['CPI']
full_df['Interest Rate'] = ir_df['Value']
full_df['GDP Growth'] = gdp_df['GDP Growth']
print("Reshaped DataFrame:")
print(full_df)
```

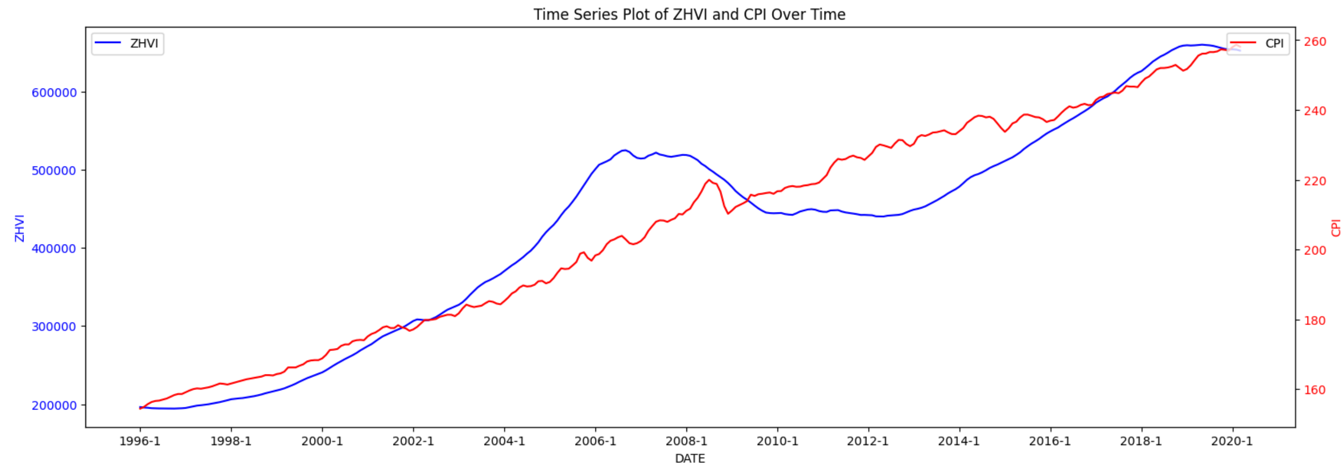
```
Reshaped DataFrame:
         ZHVI  Year  Month Year-Month  TimeIndex  Unemployment Rate      CPI  \
0    196258.0  1996      1     1996-1          0                6.3  154.400
1    195693.0  1996      2     1996-2          1                6.0  154.900
2    195383.0  1996      3     1996-3          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]
```

```
# Create figure and primary axis
fig, ax1 = plt.subplots(figsize=(18, 6))

# Plot the first ZHVI dataset
ax1.plot(full_df['Year-Month'], full_df['ZHVI'], color='blue', label='ZHVI')
ax1.set_xlabel('DATE')
ax1.set_ylabel('ZHVI', color='blue')
ax1.tick_params(axis='y', labelcolor='blue')

# Create a second axis sharing the same x-axis
```

```python
ax2 = ax1.twinx()

# Plot the Unemployment Rate data
ax2.plot(un_df['TimeIndex'], un_df['unrate'], color='red', label='Unemployment Rate')
ax2.set_ylabel('Unemployment Rate', color='red')
ax2.tick_params(axis='y', labelcolor='red')

x_ticks = np.arange(0, 290, 24)
ax1.set_xticks(x_ticks)

plt.title('Time Series Plot of ZHVI and Unemployment Rate Over Time')

# legend
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

plt.show()
```

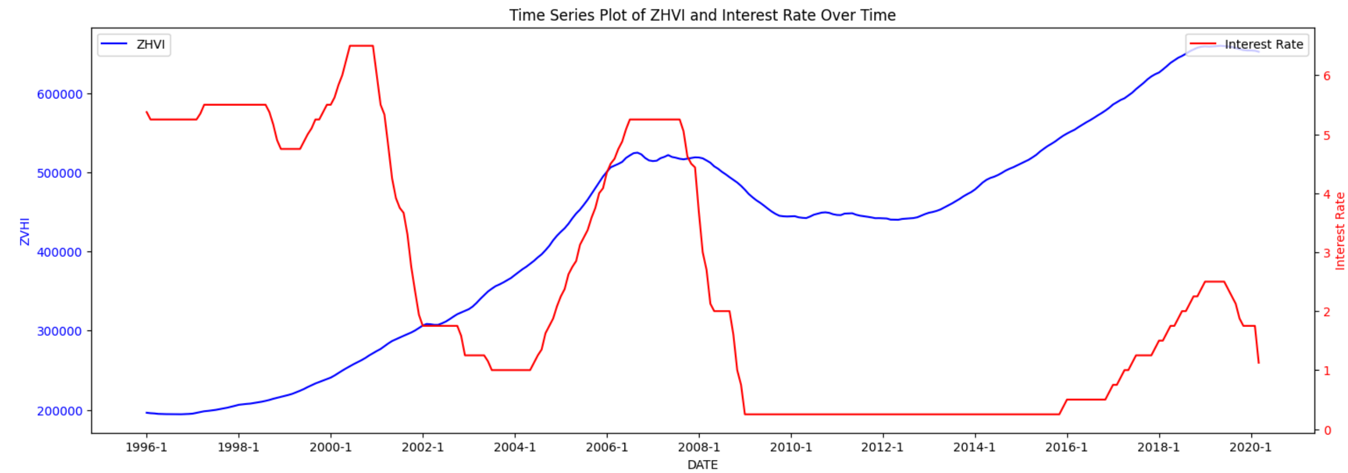

```python
# Create figure and primary axis
fig, ax1 = plt.subplots(figsize=(18, 6))

```

```
# Plot the first ZHVI dataset
ax1.plot(full_df['Year-Month'], full_df['ZHVI'], color='blue', label='ZHVI')
ax1.set_xlabel('DATE')
ax1.set_ylabel('ZHVI', color='blue')
ax1.tick_params(axis='y', labelcolor='blue')

# Create a second axis sharing the same x-axis
ax2 = ax1.twinx()

# Plot the Unemployment Rate data
ax2.plot(filtered_cpi_df['TimeIndex'], filtered_cpi_df['CPI'], color='red', label='CPI')
ax2.set_ylabel('CPI', color='red')
ax2.tick_params(axis='y', labelcolor='red')

x_ticks = np.arange(0, 290, 24)
ax1.set_xticks(x_ticks)

plt.title('Time Series Plot of ZHVI and CPI Over Time')

# legend
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

plt.show()
```



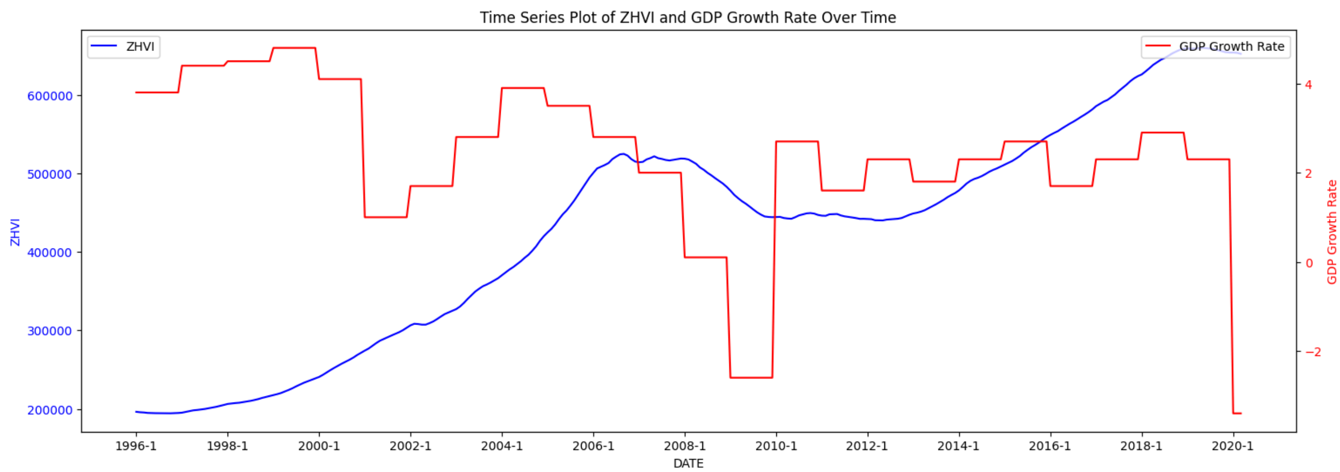

```
# Create figure and primary axis
fig, ax1 = plt.subplots(figsize=(18, 6))

# Plot the first ZHVI dataset
ax1.plot(full_df['Year-Month'], full_df['ZHVI'], color='blue', label='ZHVI')
ax1.set_xlabel('DATE')
ax1.set_ylabel('ZVHI', color='blue')
ax1.tick_params(axis='y', labelcolor='blue')

# Create a second axis sharing the same x-axis
ax2 = ax1.twinx()

# Plot the Unemployment Rate data
ax2.plot(ir_df['TimeIndex'], ir_df['Value'], color='red', label='Interest Rate')
ax2.set_ylabel('Interest Rate', color='red')
ax2.tick_params(axis='y', labelcolor='red')

x_ticks = np.arange(0, 290, 24)
ax1.set_xticks(x_ticks)

```

```
plt.title('Time Series Plot of ZHVI and Interest Rate Over Time')

# legend
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

plt.show()
```




```
# Create figure and primary axis
fig, ax1 = plt.subplots(figsize=(18, 6))

# Plot the first ZHVI dataset
ax1.plot(full_df['Year-Month'], full_df['ZHVI'], color='blue', label='ZHVI')
ax1.set_xlabel('DATE')
ax1.set_ylabel('ZHVI', color='blue')
ax1.tick_params(axis='y', labelcolor='blue')

# Create a second axis sharing the same x-axis
ax2 = ax1.twinx()

# Plot the Unemployment Rate data
```

```
ax2.plot(ir_df['TimeIndex'], gdp_df['GDP Growth'], color='red', label='GDP Growth Rate')
ax2.set_ylabel('GDP Growth Rate', color='red')
ax2.tick_params(axis='y', labelcolor='red')

x_ticks = np.arange(0, 290, 24)
ax1.set_xticks(x_ticks)

plt.title('Time Series Plot of ZHVI and GDP Growth Rate Over Time')

# legend
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

plt.show()
```



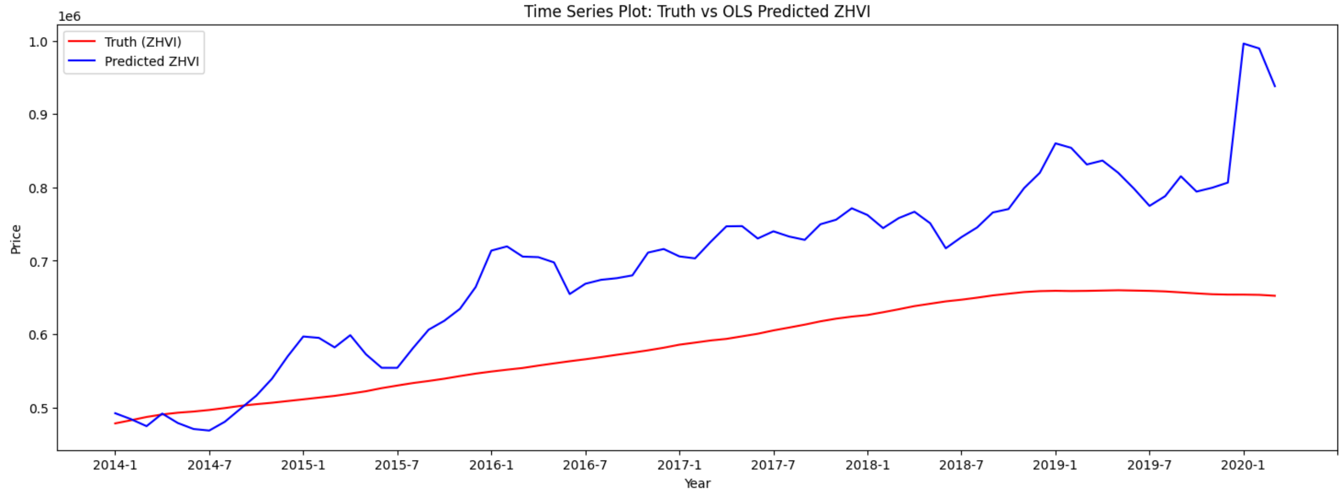
OLS Model

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

```

# Define features and target
X_train = train[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI', 'Interest Rat
y_train = train['ZHVI']

# Prediction test
X_test = test[['Year', 'Month', 'TimeIndex','Unemployment Rate', 'CPI','Interest Rate',

# add polynomial features and scale
scaler = StandardScaler()
poly = PolynomialFeatures(degree=2)

X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

X_train_scaled = scaler.fit_transform(X_train_poly)
X_test_scaled = scaler.transform(X_test_poly)

# add constant
X_train_scaled = sm.add_constant(X_train_scaled)
X_test_scaled = sm.add_constant(X_test_scaled)

# Fit OLS model
model = sm.OLS(y_train, X_train_scaled)
results = model.fit()

predictions = results.predict(X_test_scaled)
test['Predicted_ZHVI'] = predictions

y_test = test['ZHVI']
y_pred = test['Predicted_ZHVI']
OLS_pred = test['Predicted_ZHVI']

# model evaluation
rmse = math.sqrt(mean_squared_error(y_test, y_pred))
print(f"OLS Root Mean Squared Error (RMSE): {rmse}")
mape = mean_absolute_percentage_error(y_test, y_pred)
print("OLS Mean Absolute Percentage Error(MAPE):", mape)
MAE = mean_absolute_error(y_test, y_pred)
print("OLS Mean Absolute Error(MAE):", MAE)
r2 = r2_score(y_pred,y_test)
print(f"OLS R-squared(R^2): {r2}")

# Plot truth vs prediction
plt.figure(figsize=(18, 6))
plt.plot(test['Year-Month'], test['ZHVI'], color='red', label='Truth (ZHVI)')
plt.plot(test['Year-Month'], test['Predicted_ZHVI'], color='blue', label='Predicted ZHVI
plt.xlabel('Year')
plt.ylabel('Price')
x_ticks = np.arange(0, 80, 6)
plt.xticks(x_ticks)
```

```
plt.title('Time Series Plot: Truth vs OLS Predicted ZHVI')
plt.legend(loc='upper left')
plt.show()
```

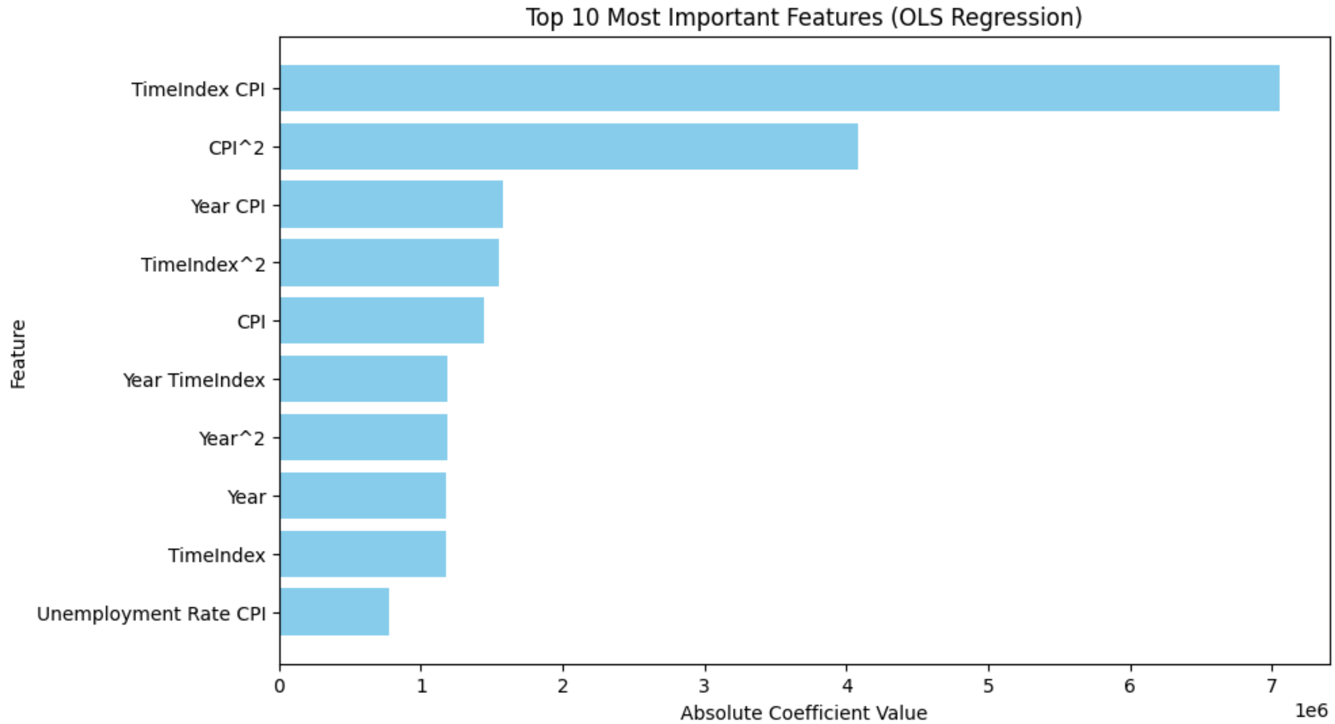
```
OLS Root Mean Squared Error (RMSE): 130532.09645357582
OLS Mean Absolute Percentage Error(MAPE): 0.1853761687014548
OLS Mean Absolute Error(MAE): 112483.27602848076
OLS R-squared(R^2): -0.14393633055157418
```




```
# Get feature names
feature_names = ['const'] + list(poly.get_feature_names_out(X_train.columns))

# Create dataframe to store coefficients and feature names
coefficients_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': results.params
})

# sort features by coeff magnitude
coefficients_df['Absolute_Coefficient'] = np.abs(coefficients_df['Coefficient'])
coefficients_df = coefficients_df.sort_values(by='Absolute_Coefficient', ascending=False

```

```
print("Sorted Top 10 OLS Regression Coefficients:")
print(coefficients_df[0:10])

# plot coeff
plt.figure(figsize=(10, 6))
plt.barh(coefficients_df['Feature'][:10], coefficients_df['Absolute_Coefficient'][:10],
plt.xlabel('Absolute Coefficient Value')
plt.ylabel('Feature')
plt.title('Top 10 Most Important Features (OLS Regression)')
plt.gca().invert_yaxis()
plt.show()
```

```
Sorted Top 10 OLS Regression Coefficients:
                  Feature   Coefficient  Absolute_Coefficient
x24          TimeIndex CPI -7.053306e+06          7.053306e+06
x31                  CPI^2  4.083916e+06          4.083916e+06
x13               Year CPI -1.580288e+06          1.580288e+06
x22            TimeIndex^2  1.550302e+06          1.550302e+06
x6                     CPI -1.441312e+06          1.441312e+06
x11         Year TimeIndex  1.185220e+06          1.185220e+06
x9                  Year^2  1.183376e+06          1.183376e+06
x2                    Year  1.181150e+06          1.181150e+06
x4               TimeIndex  1.181129e+06          1.181129e+06
x28  Unemployment Rate CPI  7.756565e+05          7.756565e+05
```



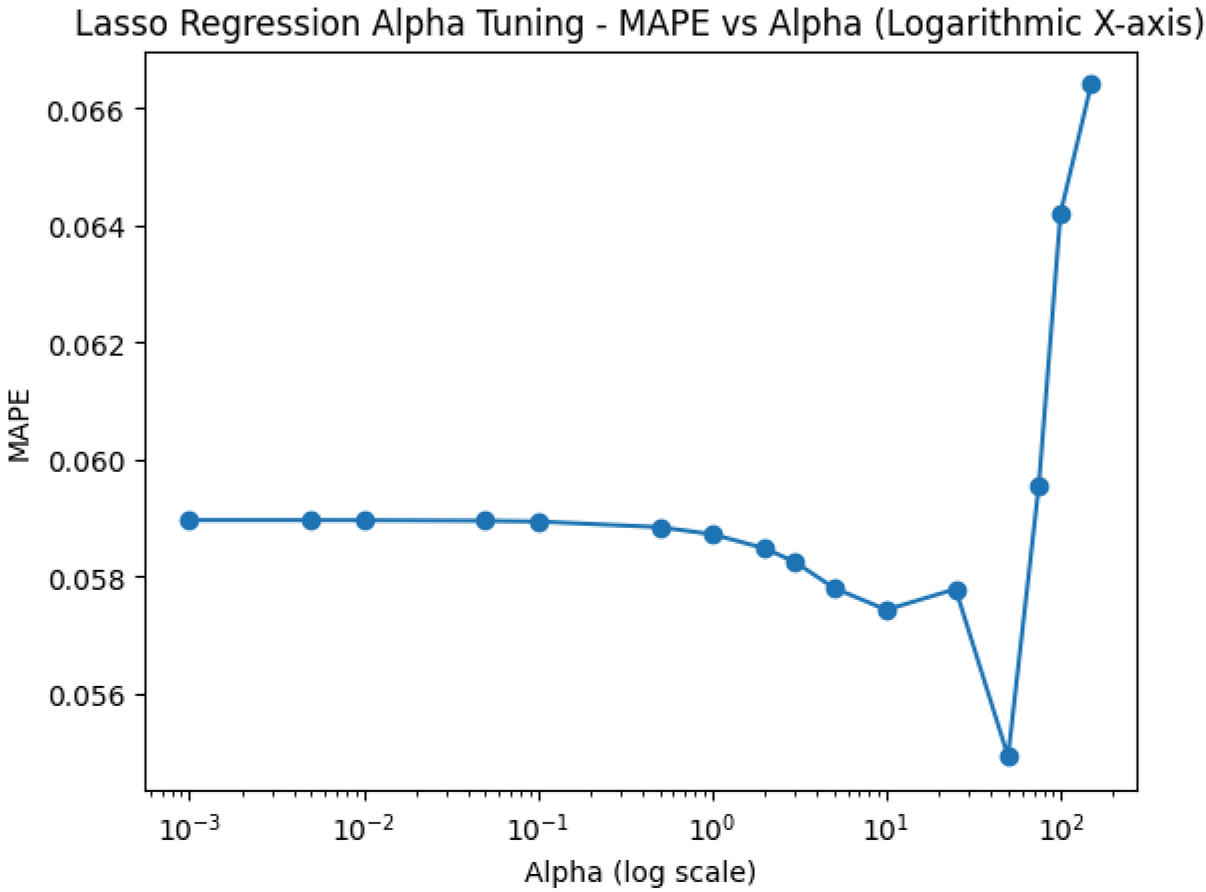
Lasso Regression Model

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

# Define features and target
X_train = train[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI','Interest Rate
y_train = train['ZHVI']
```

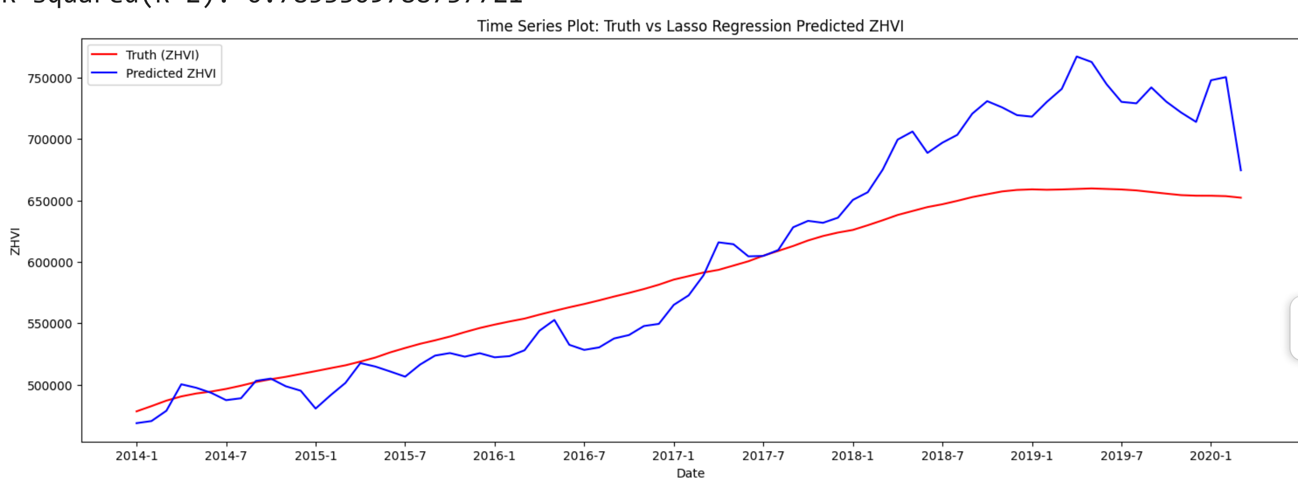
```

# Prepare test data for prediction
X_test = test[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI','Interest Rate',

# add polynomial features and scale
scaler = StandardScaler()
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

X_train_scaled = scaler.fit_transform(X_train_poly)
X_test_scaled = scaler.transform(X_test_poly)

# Fit Lasso regression model
alphas = [0.001,0.005,0.01, 0.05, 0.1, 0.5, 1, 2, 3, 5, 10,25,50,75,100,150]
results = []
lowest_alpha = alphas[0]
lowest_mape = float('inf')

for alpha in alphas:
  lasso_model = Lasso(alpha=alpha)
  lasso_model.fit(X_train_scaled, y_train)

  # Predict
  predictions = lasso_model.predict(X_test_scaled)
  test['Predicted_ZHVI'] = predictions

  # Model evaluation
  y_test = test['ZHVI']
  y_pred = test['Predicted_ZHVI']
  lasso_pred = test['Predicted_ZHVI']

  mape = mean_absolute_percentage_error(y_test, y_pred)
  results.append(mape)
  if mape < lowest_mape:
    lowest_mape = mape
    lowest_alpha = alpha

print("Lowest MAPE:", lowest_mape)
print("Lowest Alpha:", lowest_alpha)

# Plot hyperparameter tuning
plt.plot(alphas, results, marker='o')
plt.xscale('log')

plt.xlabel('Alpha (log scale)')
plt.ylabel('MAPE')
plt.title('Lasso Regression Alpha Tuning - MAPE vs Alpha (Logarithmic X-axis)')

plt.show()

```

```
alpha = lowest_alpha

lasso_model = Lasso(alpha=alpha)
lasso_model.fit(X_train_scaled, y_train)

# Predict
predictions = lasso_model.predict(X_test_scaled)
test['Predicted_ZHVI'] = predictions

# Model evaluation
y_test = test['ZHVI']
y_pred = test['Predicted_ZHVI']
lasso_pred = test['Predicted_ZHVI']

lasso_rmse = math.sqrt(mean_squared_error(y_test, y_pred))
print(f"\n\nLasso Root Mean Squared Error (RMSE): {lasso_rmse}")
lasso_mape = mean_absolute_percentage_error(y_test, y_pred)
print("Lasso Mean Absolute Percentage Error (MAPE):", lasso_mape)
lasso_MAE = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", lasso_MAE)
lasso_r2 = r2_score(y_pred,y_test)
print(f"R-squared(R^2): {lasso_r2}")

# Plot truth vs prediction
plt.figure(figsize=(18, 6))
plt.plot(test['Year-Month'], test['ZHVI'], color='red', label='Truth (ZHVI)')
plt.plot(test['Year-Month'], test['Predicted_ZHVI'], color='blue', label='Predicted ZHVI
plt.xlabel('Date')
plt.ylabel('ZHVI')
x_ticks = np.arange(0, 80, 6)
plt.xticks(x_ticks)
plt.title('Time Series Plot: Truth vs Lasso Regression Predicted ZHVI')
plt.legend(loc='upper left')
plt.show()
```

```
Lowest MAPE: 0.05492470799176251
Lowest Alpha: 50
```

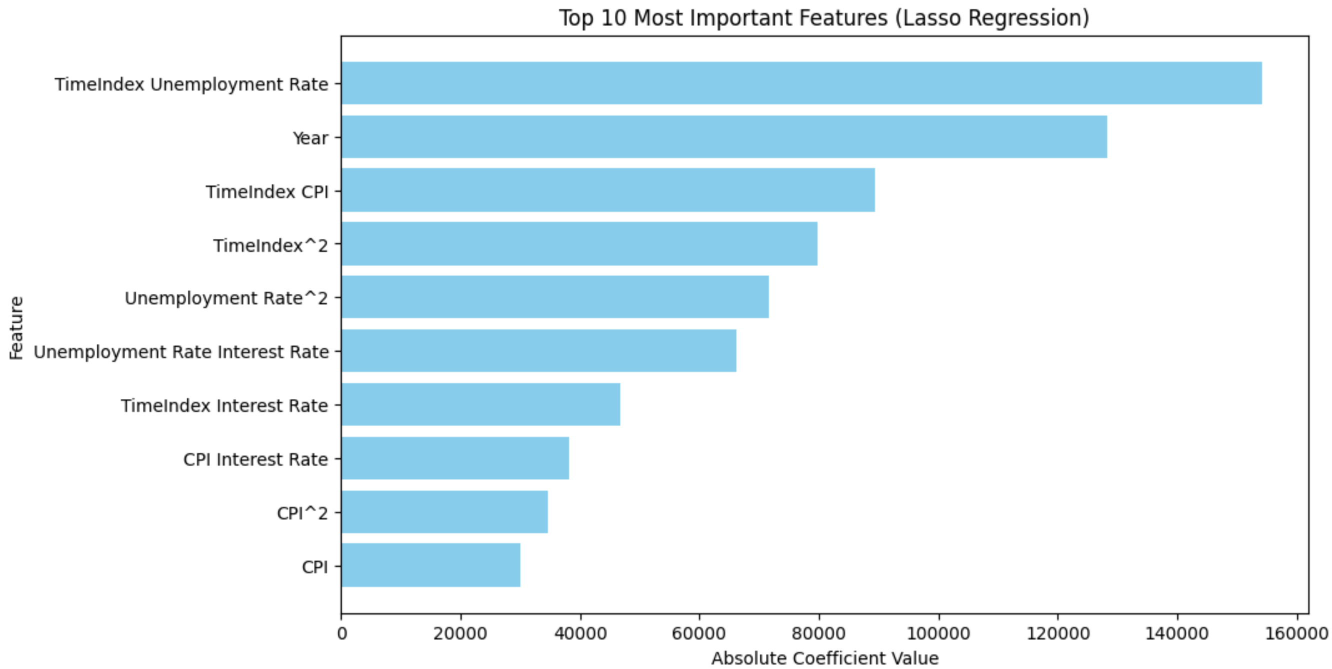



```
Lasso Root Mean Squared Error (RMSE): 44618.016149679155
Lasso Mean Absolute Percentage Error (MAPE): 0.05492470799176251
Mean Absolute Error (MAE): 34030.374729689334
R-squared(R^2): 0.7853309788737721
```



```
# Get feature names
feature_names = poly.get_feature_names_out(X_train.columns)

# Create dataframe to store coefficients and feature names
coefficients_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': lasso_model.coef_
})

# sort features by coeff magnitude
coefficients_df['Absolute_Coefficient'] = np.abs(coefficients_df['Coefficient'])
coefficients_df = coefficients_df.sort_values(by='Absolute_Coefficient', ascending=False

print("Sorted Top 10 Lasso Regression Coefficients:")
print(coefficients_df[0:10])

# plot coeff
plt.figure(figsize=(10, 6))
plt.barh(coefficients_df['Feature'][:10], coefficients_df['Absolute_Coefficient'][:10],
plt.xlabel('Absolute Coefficient Value')
plt.ylabel('Feature')
plt.title('Top 10 Most Important Features (Lasso Regression)')
plt.gca().invert_yaxis()
plt.show()
```

```
Sorted Top 10 Lasso Regression Coefficients:
                               Feature    Coefficient  Absolute_Coefficient
22      TimeIndex Unemployment Rate -154296.036934         154296.036934
1                              Year  128222.423511         128222.423511
23                    TimeIndex CPI   89468.745570          89468.745570
21                      TimeIndex^2  -79800.859453          79800.859453
26               Unemployment Rate^2   71651.217699          71651.217699
28  Unemployment Rate Interest Rate   66275.131194          66275.131194
24           TimeIndex Interest Rate   46722.278427          46722.278427
31                 CPI Interest Rate  -38094.302067          38094.302067
30                            CPI^2   34600.125584          34600.125584
5                               CPI   29987.267681          29987.267681
```



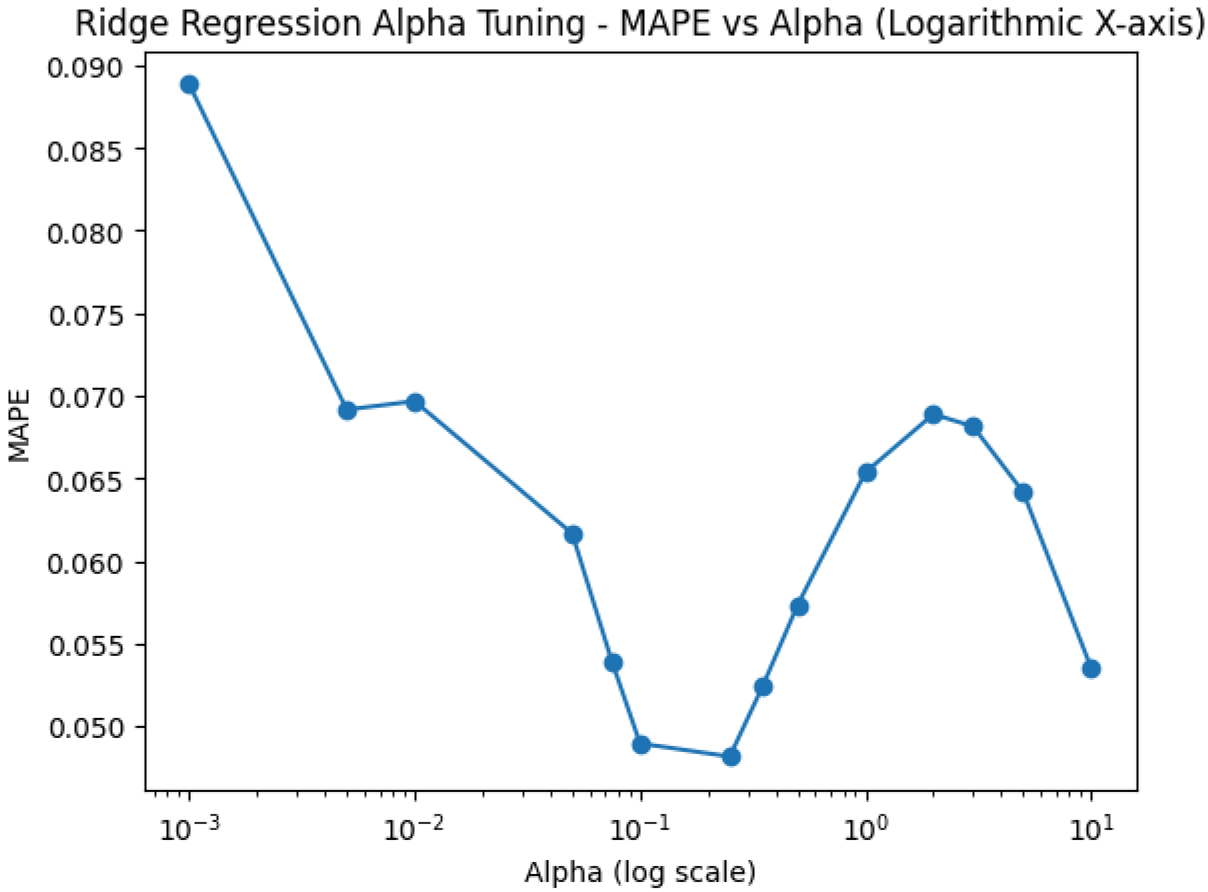
Ridge Regression Model

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

# Define features and target
X_train = train[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI','Interest Rate
y_train = train['ZHVI']
```

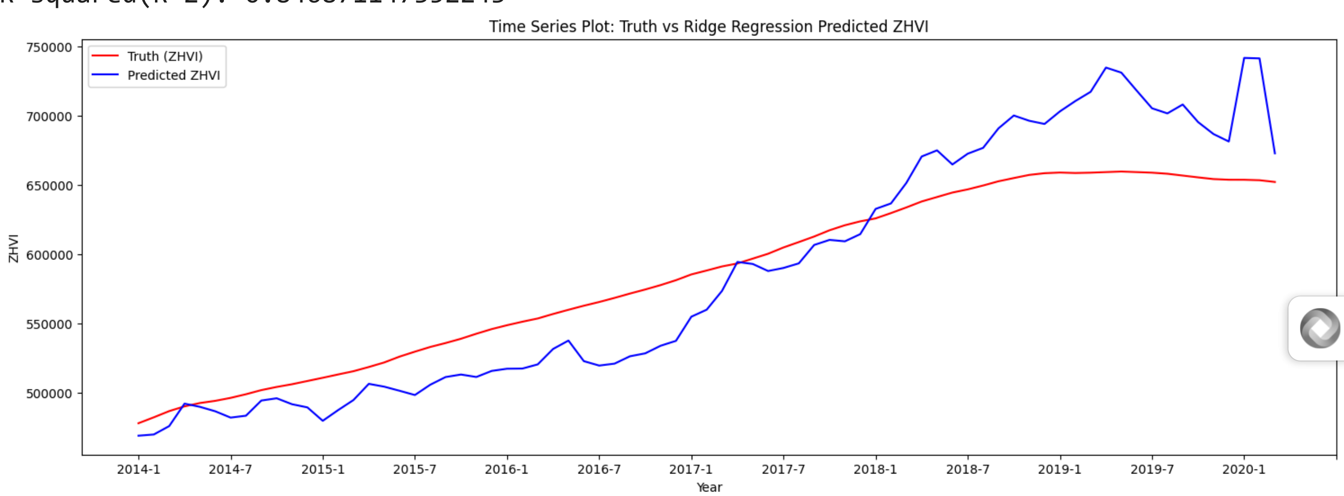
```

X_test = test[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI','Interest Rate',

# add polynomial features and scale
scaler = StandardScaler()
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

X_train_scaled = scaler.fit_transform(X_train_poly)
X_test_scaled = scaler.transform(X_test_poly)

# Fit Ridge regression model
alphas = [0.001,0.005,0.01, 0.05, 0.075, 0.1, 0.25, 0.35, 0.5, 1, 2, 3, 5, 10]

lowest_alpha = alphas[0]
lowest_mape = float('inf')
results = []
for alpha in alphas:
  ridge_model = Ridge(alpha=alpha)
  ridge_model.fit(X_train_scaled, y_train)

  # Predict
  predictions = ridge_model.predict(X_test_scaled)
  test['Predicted_ZHVI'] = predictions

  # Model evaluation
  y_test = test['ZHVI']
  y_pred = test['Predicted_ZHVI']
  ridge_pred = test['Predicted_ZHVI']

  mape = mean_absolute_percentage_error(y_test, y_pred)
  results.append(mape)
  if mape < lowest_mape:
    lowest_mape = mape
    lowest_alpha = alpha

print("Lowest MAPE:", lowest_mape)
print("Lowest Alpha:", lowest_alpha)

# Plot hyperparameter tuning
plt.plot(alphas, results, marker='o')
plt.xscale('log')

plt.xlabel('Alpha (log scale)')
plt.ylabel('MAPE')
plt.title('Ridge Regression Alpha Tuning - MAPE vs Alpha (Logarithmic X-axis)')

plt.show()

alpha = lowest_alpha
```

```
ridge_model = Ridge(alpha=alpha)
ridge_model.fit(X_train_scaled, y_train)

# Predict
predictions = ridge_model.predict(X_test_scaled)
test['Predicted_ZHVI'] = predictions

# Model evaluation
y_test = test['ZHVI']
y_pred = test['Predicted_ZHVI']
ridge_pred = test['Predicted_ZHVI']

ridge_rmse = math.sqrt(mean_squared_error(y_test, y_pred))
print(f"\n\nRR Root Mean Squared Error (RMSE): {ridge_rmse}")
ridge_mape = mean_absolute_percentage_error(y_test, y_pred)
print("RR Mean Absolute Percentage Error(MAPE):", ridge_mape)
ridge_MAE = mean_absolute_error(y_test, y_pred)
print("RR Mean Absolute Error(MAE):", ridge_MAE)
ridge_r2 = r2_score(y_pred,y_test)
print(f"R-squared(R^2): {ridge_r2}")

# Plot truth vs prediction
plt.figure(figsize=(18, 6))
plt.plot(test['Year-Month'], test['ZHVI'], color='red', label='Truth (ZHVI)')
plt.plot(test['Year-Month'], test['Predicted_ZHVI'], color='blue', label='Predicted ZHVI
plt.xlabel('Year')
plt.ylabel('ZHVI')
x_ticks = np.arange(0, 80, 6)
plt.xticks(x_ticks)
plt.title('Time Series Plot: Truth vs Ridge Regression Predicted ZHVI')
plt.legend(loc='upper left')
plt.show()
```

```
Lowest MAPE: 0.04815805234530082
Lowest Alpha: 0.25
```



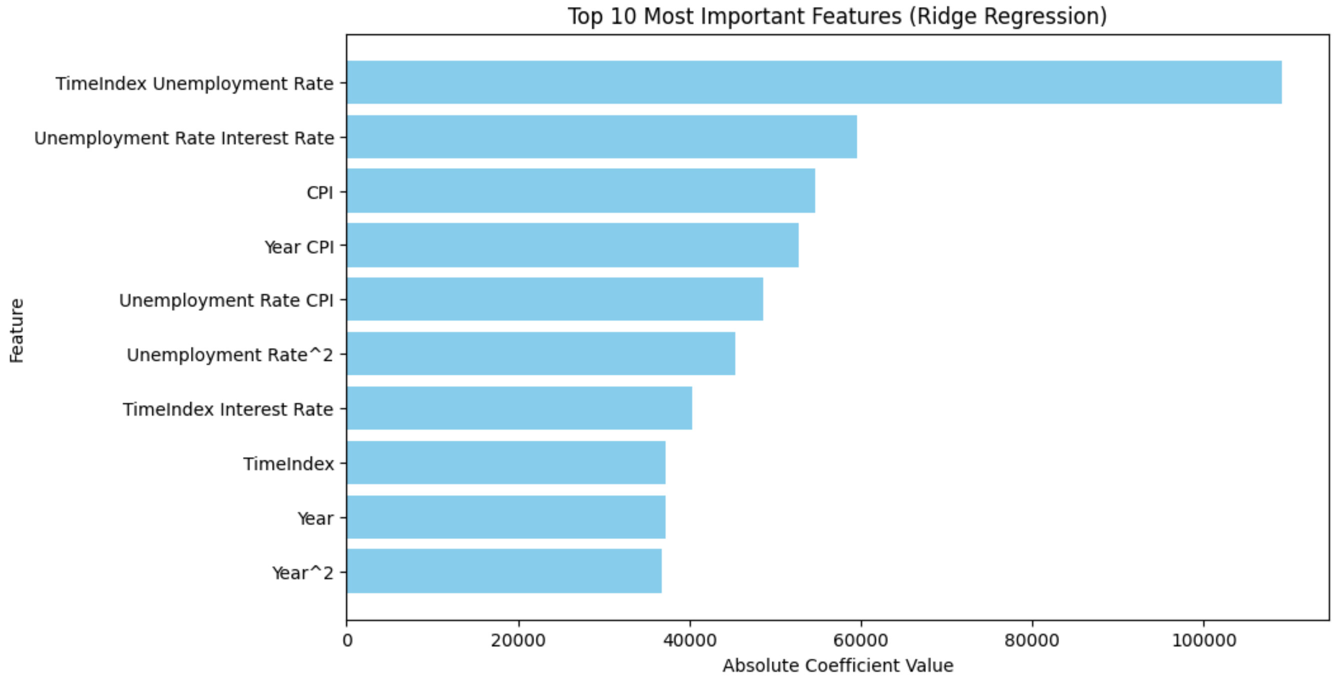

```
RR Root Mean Squared Error (RMSE): 34586.997710405834
RR Mean Absolute Percentage Error(MAPE): 0.04815805234530082
RR Mean Absolute Error(MAE): 28891.172716798137
R-squared(R^2): 0.846871147352243
```



```
# Get feature names
feature_names = poly.get_feature_names_out(X_train.columns)

# Create dataframe to store coefficients and feature names
coefficients_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': ridge_model.coef_
})

# sort features by coeff magnitude
coefficients_df['Absolute_Coefficient'] = np.abs(coefficients_df['Coefficient'])
coefficients_df = coefficients_df.sort_values(by='Absolute_Coefficient', ascending=False)

print("Sorted Top 10 Ridge Regression Coefficients:")
print(coefficients_df[0:10])

# plot coeff
plt.figure(figsize=(10, 6))
plt.barh(coefficients_df['Feature'][:10], coefficients_df['Absolute_Coefficient'][:10], c
plt.xlabel('Absolute Coefficient Value')
plt.ylabel('Feature')
plt.title('Top 10 Most Important Features (Ridge Regression)')
plt.gca().invert_yaxis()
plt.show()
```

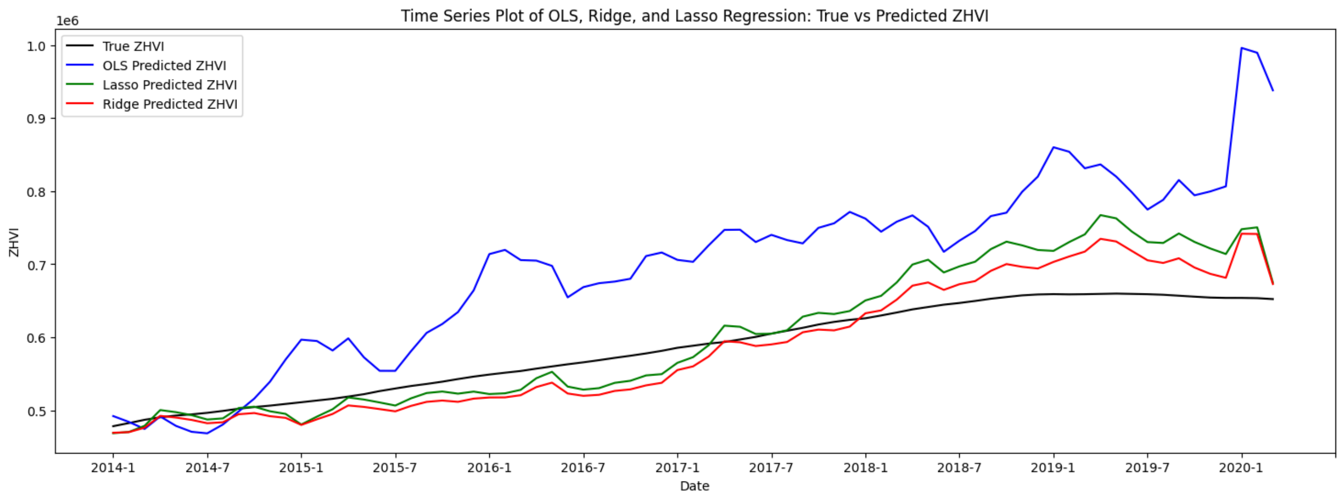
Sorted Top 10 Ridge Regression Coefficients:

	Feature	Coefficient	Absolute_Coefficient
22	TimeIndex Unemployment Rate	-109178.576829	109178.576829
28	Unemployment Rate Interest Rate	59548.087563	59548.087563
5	CPI	54636.320560	54636.320560
12	Year CPI	52733.852309	52733.852309
27	Unemployment Rate CPI	-48590.347941	48590.347941
26	Unemployment Rate^2	45311.048503	45311.048503
24	TimeIndex Interest Rate	40266.286996	40266.286996
3	TimeIndex	37172.836936	37172.836936
1	Year	37166.437453	37166.437453
8	Year^2	36844.519700	36844.519700




```
plt.figure(figsize=(18, 6))
plt.plot(test['Year-Month'], test['ZHVI'], color='black', label='True ZHVI')
plt.plot(test['Year-Month'], OLS_pred, color='blue', label='OLS Predicted ZHVI')
plt.plot(test['Year-Month'], lasso_pred, color='green', label='Lasso Predicted ZHVI')
plt.plot(test['Year-Month'], ridge_pred, color='red', label='Ridge Predicted ZHVI')

plt.xlabel('Date')
plt.ylabel('ZHVI')
x_ticks = np.arange(0, 80, 6)
plt.xticks(x_ticks)
```

```
plt.title('Time Series Plot of OLS, Ridge, and Lasso Regression: True vs Predicted ZHVI')

plt.legend(loc='upper left')

plt.show()
```



milestone 2

```
!pip install keras-tuner
!pip install -q streamlit
!npm install localtunnel

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout, Activation
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error, mean_abs
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
```

```
from sklearn.model_selection import RandomizedSearchCV, TimeSeriesSplit
from sklearn.linear_model import LinearRegression
from tabulate import tabulate

import tensorflow as tf
import keras_tuner as kt
import math
import matplotlib.pyplot as plt
import numpy as np
import random
import xgboost as xgb
import statsmodels.api as sm
import joblib
import xgboost as xgb

clear_output()
```

DNN Model

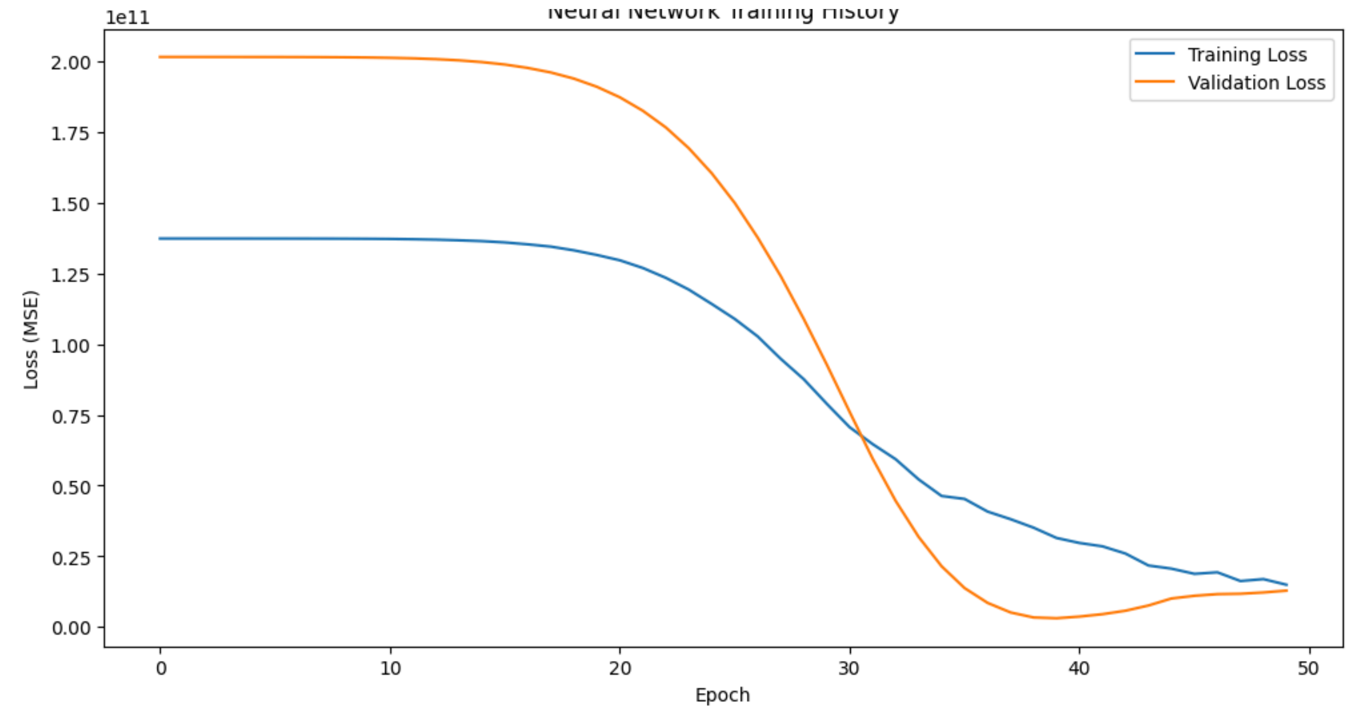
```
# Hyperparameter tuning

random.seed(158)

# Preprocess data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the model building function for Keras Tuner
def build_model(hp):

    model = Sequential()

    # Tune the number of units in the first Dense layer
    model.add(Dense(
        units=hp.Int('units_1', min_value=64, max_value=256, step=64),
        activation='relu',
        input_shape=(X_train_scaled.shape[1],)
    ))
    model.add(Dropout(
        rate=hp.Float('dropout_1', min_value=0.0, max_value=0.5, step=0.1)
    ))

    # Tune the number of hidden layers (1-3)
    for i in range(hp.Int('num_layers', 1, 3)):
        model.add(Dense(
            units=hp.Int(f'units_{i+2}', min_value=32, max_value=128, step=32),
            activation='relu')
        )
        model.add(Dropout(
```

```
            rate=hp.Float(f'dropout_{i+2}', min_value=0.0, max_value=0.5, step=0.1)
        ))

    model.add(Dense(1))

    # Tune the learning rate
    learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mse', metrics=['mae'])

    return model

# Set up the tuner
tuner = kt.RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=20,
    executions_per_trial=2,
    directory='keras_tuner',
    project_name='zhvi_prediction'
)

# Early stopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

# Perform hyperparameter search
tuner.search(
    X_train_scaled, y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stopping],
    verbose=1
)

# Get the best hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

print(f"""
The hyperparameter search is complete. The optimal hyperparameters are:
- Units in first layer: {best_hps.get('units_1')}
- Number of hidden layers: {best_hps.get('num_layers')}
- Learning rate: {best_hps.get('learning_rate')}
- Dropout rate (first layer): {best_hps.get('dropout_1')}
""")

```

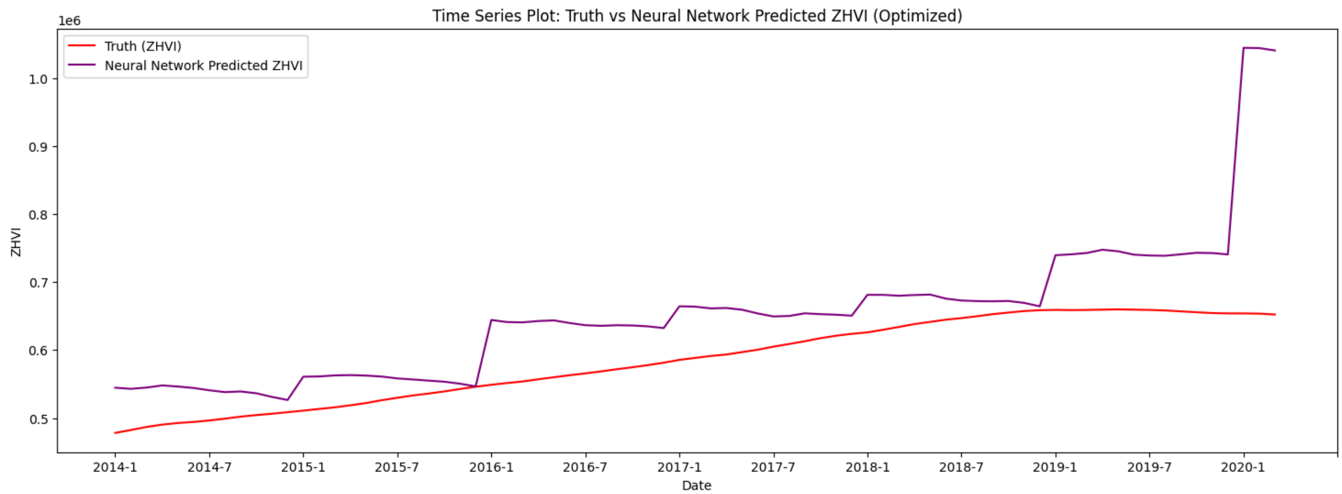
```
# Build the model with the best hyperparameters
best_model = tuner.hypermodel.build(best_hps)

# Train the best model
history = best_model.fit(
    X_train_scaled, y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stopping],
    verbose=1
)

# Plot training history
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.title('Neural Network Training History')
plt.legend()
plt.show()

# Predictions
nn_predictions = best_model.predict(X_test_scaled).flatten()
test['NN_Predicted_ZHVI'] = nn_predictions

# Model evaluation
nn_rmse = math.sqrt(mean_squared_error(y_test, nn_predictions))
print(f"Neural Network Root Mean Squared Error (RMSE): {nn_rmse}")
nn_mape = mean_absolute_percentage_error(y_test, nn_predictions)
print("Neural Network Mean Absolute Percentage Error (MAPE):", nn_mape)
nn_mae = mean_absolute_error(y_test, nn_predictions)
print("Neural Network Mean Absolute Error (MAE):", nn_mae)
nn_r2 = r2_score(y_test, nn_predictions)
print(f"Neural Network R-squared (R^2): {nn_r2}")

# Plot truth vs prediction
plt.figure(figsize=(18, 6))
plt.plot(test['Year-Month'], test['ZHVI'], color='red', label='Truth (ZHVI)')
plt.plot(test['Year-Month'], test['NN_Predicted_ZHVI'], color='purple', label='Neura
plt.xlabel('Date')
plt.ylabel('ZHVI')
x_ticks = np.arange(0, 80, 6)
plt.xticks(x_ticks)
plt.title('Time Series Plot: Truth vs Neural Network Predicted ZHVI (Optimized)')
plt.legend(loc='upper left')
plt.show()
```

```
Trial 20 Complete [00h 00m 14s]
val_loss: 3773630720.0

Best val_loss So Far: 2747978496.0
Total elapsed time: 00h 11m 01s

The hyperparameter search is complete. The optimal hyperparameters are:
- Units in first layer: 192
- Number of hidden layers: 3
- Learning rate: 0.001
- Dropout rate (first layer): 0.4

Epoch 1/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - loss: 142135230464.0000 - mae: 357076.5938 - val
Epoch 2/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 137808904192.0000 - mae: 351127.2188 - val
Epoch 3/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 137643327488.0000 - mae: 350684.6562 - val
Epoch 4/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 143591079936.0000 - mae: 359643.5312 - val
Epoch 5/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 132160069632.0000 - mae: 343093.5000 - val
Epoch 6/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 137535651840.0000 - mae: 351015.3125 - val
Epoch 7/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 136340348928.0000 - mae: 349860.6562 - val
Epoch 8/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 136219549696.0000 - mae: 350163.6875 - val
Epoch 9/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 134784786432.0000 - mae: 346724.8438 - val
Epoch 10/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 140556959744.0000 - mae: 354887.2188 - val
Epoch 11/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 137576038400.0000 - mae: 350448.6250 - val
Epoch 12/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 137234776064.0000 - mae: 350552.9375 - val
Epoch 13/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 139266768896.0000 - mae: 353964.1875 - val
Epoch 14/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 131869065216.0000 - mae: 341946.0938 - val
Epoch 15/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 137546448896.0000 - mae: 351033.6250 - val
Epoch 16/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 131390136320.0000 - mae: 342966.9375 -
Epoch 17/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 142992736256.0000 - mae: 359089.5000 - val
Epoch 18/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 135150436352.0000 - mae: 345449.7500 - val
Epoch 19/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 128360128512.0000 - mae: 337479.7500 - val
Epoch 20/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 132962910208.0000 - mae: 343252.2188 - val
Epoch 21/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 135287021568.0000 - mae: 346663.2500 - val
Epoch 22/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 121984532480.0000 - mae: 327748.7188 - val
```

```
Epoch 23/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 116732141568.0000 - mae: 316579.8750 - val
Epoch 24/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 112045711360.0000 - mae: 310469.1250 - val
Epoch 25/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 113209040896.0000 - mae: 310353.0938 - val
Epoch 26/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 110794670080.0000 - mae: 303401.6250 - val
Epoch 27/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 101304696832.0000 - mae: 285922.2188 - val
Epoch 28/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 91559772160.0000 - mae: 265344.7500 - val_
Epoch 29/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 84936499200.0000 - mae: 248896.7031 - val_
Epoch 30/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 76313583616.0000 - mae: 228155.7656 - val_
Epoch 31/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 69197414400.0000 - mae: 220884.8906 - val_
Epoch 32/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 71095009280.0000 - mae: 228538.0312 - val_
Epoch 33/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 62709309440.0000 - mae: 215566.1250 - val_
Epoch 34/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 50817744896.0000 - mae: 194482.9688 - val_
Epoch 35/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 47198117888.0000 - mae: 192297.9062 - val_
Epoch 36/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 45227626496.0000 - mae: 191217.2031 - val_
Epoch 37/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 40039145472.0000 - mae: 180178.0469 - val_
Epoch 38/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 38438428672.0000 - mae: 176614.7188 - val_
Epoch 39/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 35704451072.0000 - mae: 169358.5781 - val_
Epoch 40/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - loss: 33121267712.0000 - mae: 165117.3594 - val_
Epoch 41/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 31072069632.0000 - mae: 159930.7500 - val_
Epoch 42/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - loss: 29801713664.0000 - mae: 152902.3125 - val_
Epoch 43/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 26341781504.0000 - mae: 145040.1875 - val_
Epoch 44/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 21740562432.0000 - mae: 130391.8125 -
Epoch 45/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 22103099392.0000 - mae: 129428.6250 - val_
Epoch 46/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 17934903296.0000 - mae: 117617.8984 - val_
Epoch 47/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - loss: 18871126016.0000 - mae: 119005.3906 - val_
Epoch 48/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - loss: 15961183232.0000 - mae: 107405.6875 - val_
Epoch 49/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 16721813504.0000 - mae: 110311.5703 - val_
Epoch 50/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 14423368704.0000 - mae: 101693.5312 - val_
```



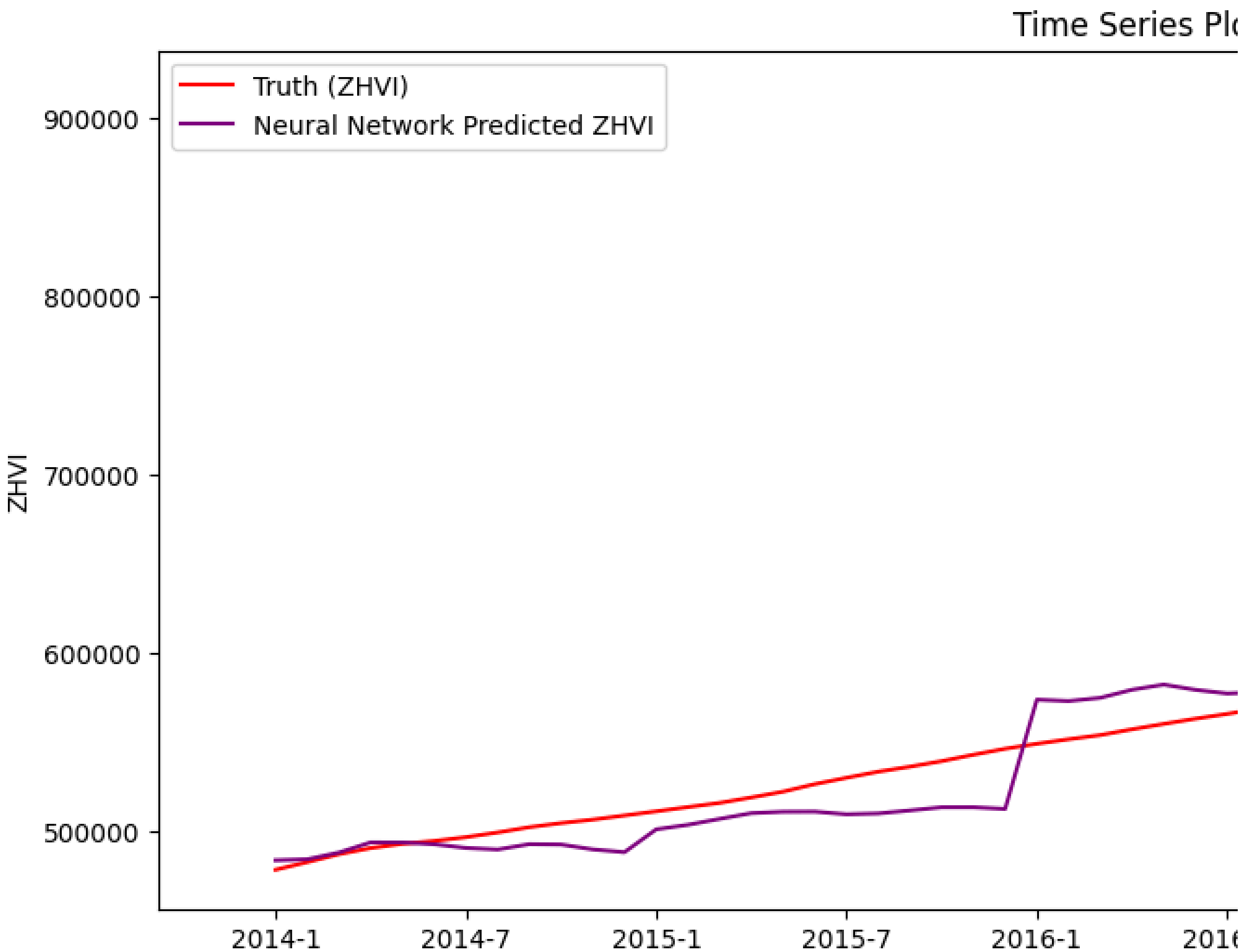

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
Neural Network Root Mean Squared Error (RMSE): 96556.13473193542
Neural Network Mean Absolute Percentage Error (MAPE): 0.11076739748861884
Neural Network Mean Absolute Error (MAE): 66117.98833333333
Neural Network R-squared (R^2): -1.5689345558410244
```



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

# Define features and target
X_train = train[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI','Interest Rate
y_train = train['ZHVI']

# Prepare test data for prediction
X_test = test[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI','Interest Rate',

# set seed for easy reproducibility of different configurations
random.seed(158)

# preprocess data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build neural network
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1)
])

optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mse', metrics=['mae'])

# Train the model
history = model.fit(
    X_train_scaled, y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    verbose=1
)

clear_output()

# Predictions
nn_predictions = model.predict(X_test_scaled).flatten()
test['NN_Predicted_ZHVI'] = nn_predictions

# Model evaluation
nn_rmse = math.sqrt(mean_squared_error(y_test, nn_predictions))
print(f"Neural Network Root Mean Squared Error (RMSE): {nn_rmse}")
```

```
nn_mape = mean_absolute_percentage_error(y_test, nn_predictions)
print("Neural Network Mean Absolute Percentage Error (MAPE):", nn_mape)
nn_mae = mean_absolute_error(y_test, nn_predictions)
print("Neural Network Mean Absolute Error (MAE):", nn_mae)
nn_r2 = r2_score(y_test, nn_predictions)
print(f"Neural Network R-squared (R^2): {nn_r2}")

# Plot truth vs prediction
plt.figure(figsize=(18, 6))
plt.plot(test['Year-Month'], test['ZHVI'], color='red', label='Truth (ZHVI)')
plt.plot(test['Year-Month'], test['NN_Predicted_ZHVI'], color='purple', label='Neural Ne
plt.xlabel('Date')
plt.ylabel('ZHVI')
x_ticks = np.arange(0, 80, 6)
plt.xticks(x_ticks)
plt.title('Time Series Plot: Truth vs Neural Network Predicted ZHVI')
plt.legend(loc='upper left')
plt.show()
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
Neural Network Root Mean Squared Error (RMSE): 55288.09761836565
Neural Network Mean Absolute Percentage Error (MAPE): 0.041211447799650795
Neural Network Mean Absolute Error (MAE): 25382.914583333335
Neural Network R-squared (R^2): 0.15771980293931132
```



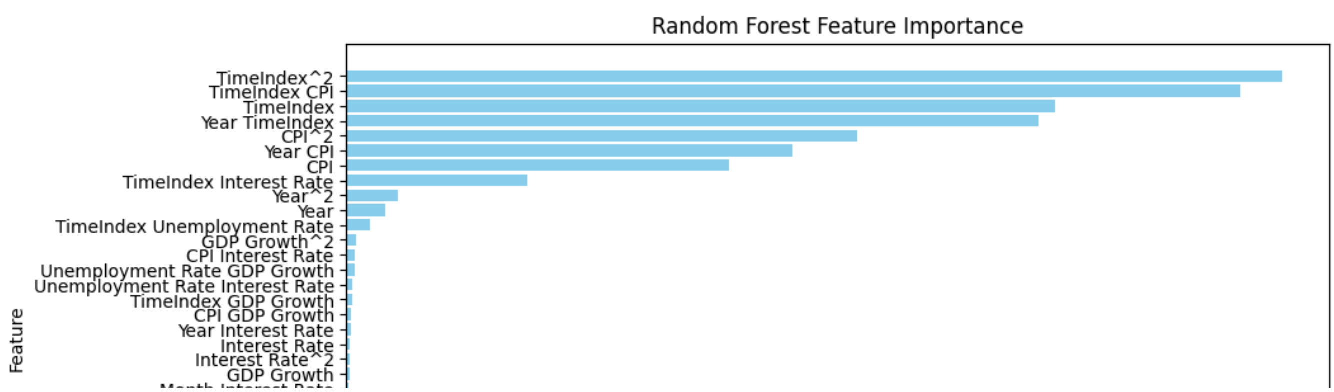
Random Forest Model

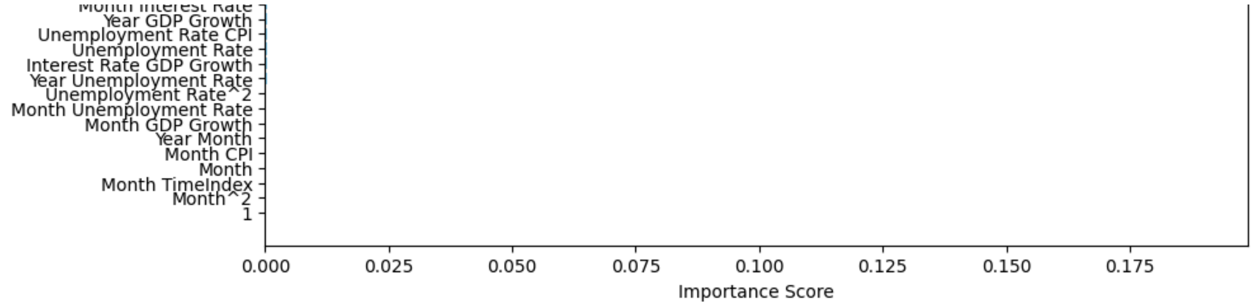
```
# Split data into training and test (same as before)
train = full_df[(full_df['Year'] < 2014) | ((full_df['Year'] == 2013) & (full_df['Mon
test = full_df[(full_df['Year'] > 2013) | ((full_df['Year'] == 2014) & (full_df['Mont

# Define features and target
X_train = train[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI', 'Interest Rat
y_train = train['ZHVI']
X_test = test[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI', 'Interest Rate'
y_test = test['ZHVI']

# add polynomial features and scale
scaler = StandardScaler()
poly = PolynomialFeatures(degree=2)
```

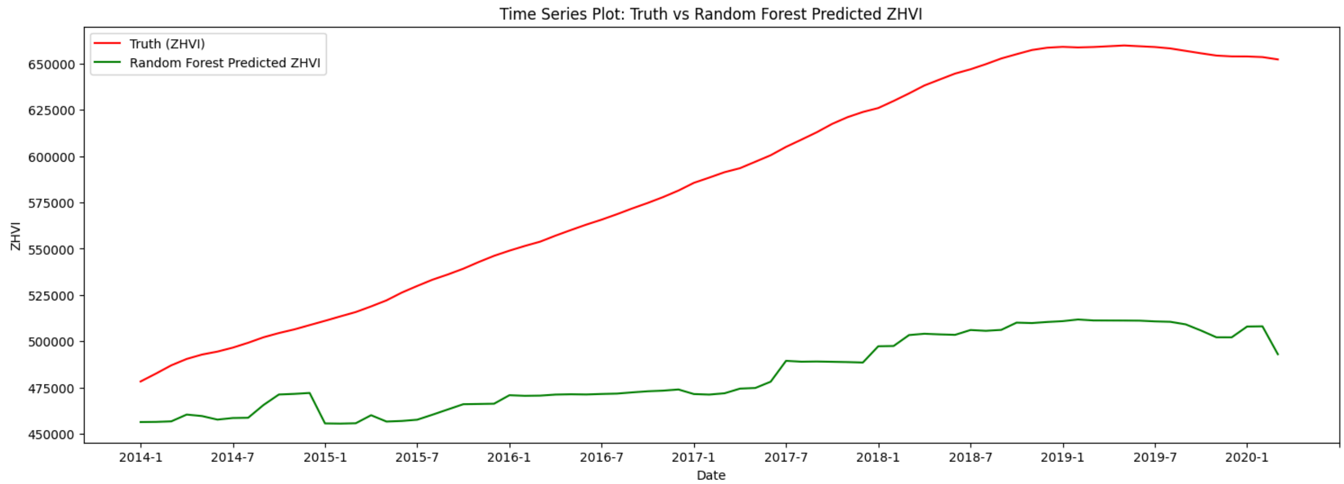
```
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Get feature names from polynomial features
feature_names = poly.get_feature_names_out(input_features=X_train.columns)

X_train_scaled = scaler.fit_transform(X_train_poly)
X_test_scaled = scaler.transform(X_test_poly)

# Hyperparameter tuning with GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')
grid_search.fit(X_train_scaled, y_train)

# Best model
best_rf = grid_search.best_estimator_
print(f"Best Random Forest parameters: {grid_search.best_params_}")

# Predictions
rf_predictions = best_rf.predict(X_test_scaled)
test['RF_Predicted_ZHVI'] = rf_predictions

# Model evaluation
rf_rmse = math.sqrt(mean_squared_error(y_test, rf_predictions))
print(f"Random Forest Root Mean Squared Error (RMSE): {rf_rmse}")
rf_mape = mean_absolute_percentage_error(y_test, rf_predictions)
print("Random Forest Mean Absolute Percentage Error (MAPE):", rf_mape)
rf_mae = mean_absolute_error(y_test, rf_predictions)
print("Random Forest Mean Absolute Error (MAE):", rf_mae)
rf_r2 = r2_score(y_test, rf_predictions)
print(f"Random Forest R-squared (R^2): {rf_r2}")

# Feature importance
feature_importance = pd.DataFrame({
    'Feature': feature_names,
    'Importance': best_rf.feature_importances_
}).sort_values('Importance', ascending=False)

print("\nRandom Forest Feature Importance:")
print(feature_importance)

# Plot feature importance
plt.figure(figsize=(10, 6))
```

```
plt.barh(feature_importance['Feature'], feature_importance['Importance'], color='skyblue
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.title('Random Forest Feature Importance')
plt.gca().invert_yaxis()
plt.show()

# Plot truth vs prediction
plt.figure(figsize=(18, 6))
plt.plot(test['Year-Month'], test['ZHVI'], color='red', label='Truth (ZHVI)')
plt.plot(test['Year-Month'], test['RF_Predicted_ZHVI'], color='green', label='Random For
plt.xlabel('Date')
plt.ylabel('ZHVI')
x_ticks = np.arange(0, 80, 6)
plt.xticks(x_ticks)
plt.title('Time Series Plot: Truth vs Random Forest Predicted ZHVI')
plt.legend(loc='upper left')
plt.show()

```

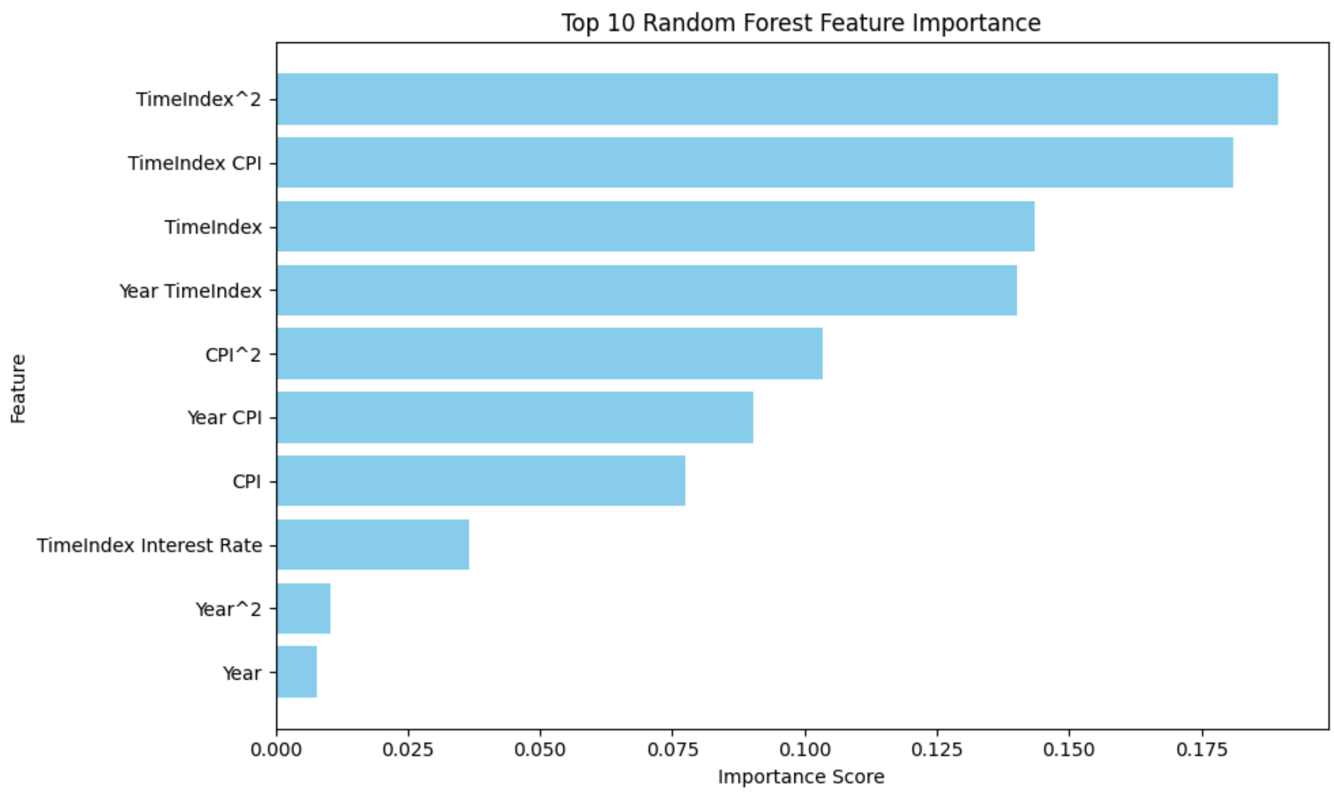
```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best Random Forest parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_s
Random Forest Root Mean Squared Error (RMSE): 110934.39674566274
Random Forest Mean Absolute Percentage Error (MAPE): 0.17001389114529938
Random Forest Mean Absolute Error (MAE): 102800.76702222222
Random Forest R-squared (R^2): -2.3909840626295815

Random Forest Feature Importance:
                          Feature  Importance
21                    TimeIndex^2    0.189539
23                  TimeIndex CPI    0.181033
3                       TimeIndex    0.143556
10                 Year TimeIndex    0.140208
30                          CPI^2    0.103491
12                       Year CPI    0.090341
5                             CPI    0.077447
24        TimeIndex Interest Rate    0.036600
8                          Year^2    0.010326
1                            Year    0.007912
22    TimeIndex Unemployment Rate    0.004745
35                  GDP Growth^2    0.001947
31              CPI Interest Rate    0.001729
29   Unemployment Rate GDP Growth    0.001691
28 Unemployment Rate Interest Rate    0.001140
25           TimeIndex GDP Growth    0.001053
32                 CPI GDP Growth    0.000835
13             Year Interest Rate    0.000798
6                   Interest Rate    0.000763
33                Interest Rate^2    0.000742
7                      GDP Growth    0.000715
19            Month Interest Rate    0.000534
14                Year GDP Growth    0.000463
27          Unemployment Rate CPI    0.000446
4               Unemployment Rate    0.000351
34       Interest Rate GDP Growth    0.000339
11         Year Unemployment Rate    0.000333
26            Unemployment Rate^2    0.000277
17        Month Unemployment Rate    0.000152
20               Month GDP Growth    0.000141
9                      Year Month    0.000074
18                      Month CPI    0.000072
2                           Month    0.000072
16                Month TimeIndex    0.000070
15                        Month^2    0.000066
0                               1    0.000000
```




Year GDP Growth
Unemployment Rate CPI
Unemployment Rate
Interest Rate GDP Growth
Year Unemployment Rate
Unemployment Rate^2
Month Unemployment Rate
Month GDP Growth
Year Month
Month CPI
Month
Month TimeIndex
Month^2
1
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
Importance Score


Time Series Plot: Truth vs Random Forest Predicted ZHVI
Truth (ZHVI)
Random Forest Predicted ZHVI
ZHVI
650000
625000
600000
575000
550000
525000
500000
475000
450000
2014-1
2014-7
2015-1
2015-7
2016-1
2016-7
2017-1
2017-7
2018-1
2018-7
2019-1
2019-7
2020-1
Date

```python
# Get top 10 features
top_10_features = feature_importance.head(10)

# Plot top 10 feature importance
plt.figure(figsize=(10, 6))
plt.barh(top_10_features['Feature'], top_10_features['Importance'], color='skyblue')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.title('Top 10 Random Forest Feature Importance')
plt.gca().invert_yaxis()  # Most important at top
plt.tight_layout()  # Prevent label cutoff
plt.show()
```



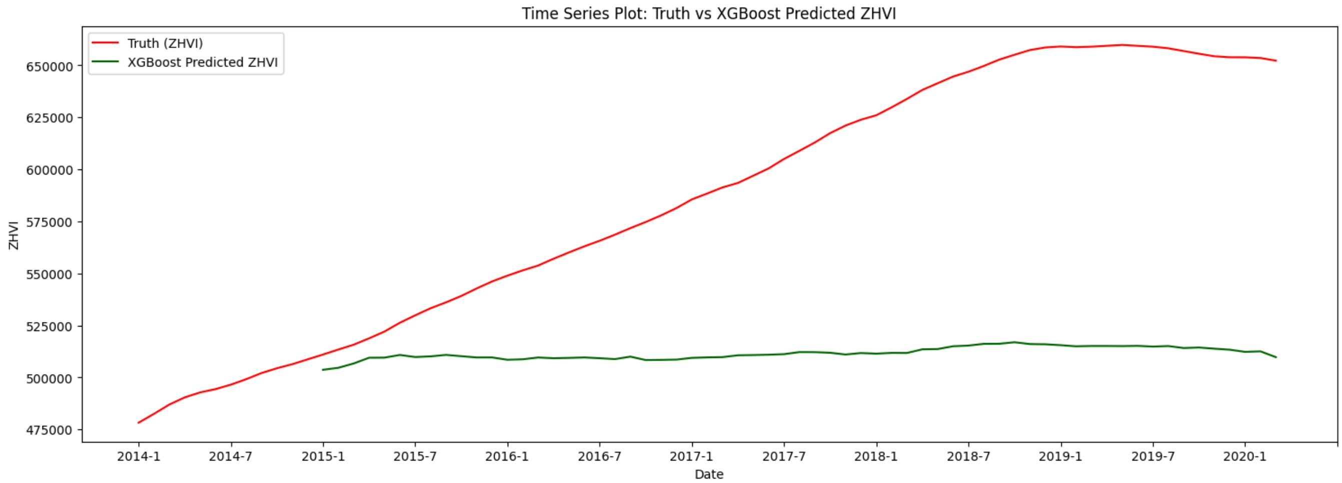
XGBoost Model

```
# Split data into training and test (same as before)
train = full_df[(full_df['Year'] < 2014) | ((full_df['Year'] == 2013) & (full_df['Montl
test = full_df[(full_df['Year'] > 2013) | ((full_df['Year'] == 2014) & (full_df['Month'

# Define features and target
X_train = train[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI', 'Interest Ra
y_train = train['ZHVI']
X_test = test[['Year', 'Month', 'TimeIndex', 'Unemployment Rate', 'CPI', 'Interest Rate
y_test = test['ZHVI']

# Feature Engineering Functions
def create_features(df, target_col='ZHVI'):
    # Create lag features
    for lag in [1, 2, 3, 6, 12]:  # Multiple time horizons
        df[f'{target_col}_lag_{lag}'] = df[target_col].shift(lag)

    # Create rolling statistics
    for window in [3, 6, 12]:
        df[f'{target_col}_rolling_avg_{window}'] = df[target_col].rolling(window).mean(
        df[f'{target_col}_rolling_std_{window}'] = df[target_col].rolling(window).std()

    # Month/year indicators
    df['month_sin'] = np.sin(2 * np.pi * df['Month']/12)
    df['month_cos'] = np.cos(2 * np.pi * df['Month']/12)

    return df

# Data Preparation
train = create_features(train.copy())
test = create_features(test.copy())

# Define features
base_features = ['Year', 'Month', 'TimeIndex', 'Unemployment Rate',
                 'CPI', 'Interest Rate', 'GDP Growth', 'month_sin', 'month_cos']
lag_features = [col for col in train.columns if 'lag_' in col or 'rolling_' in col]
features = base_features + lag_features

# Handle missing values from lag features
X_train = train[features].dropna()
y_train = train.loc[X_train.index, 'ZHVI']
X_test = test[features].dropna()
y_test = test.loc[X_test.index, 'ZHVI']

# add polynomial features and scale
scaler = StandardScaler()
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
X_train_scaled = scaler.fit_transform(X_train_poly)
X_test_scaled = scaler.transform(X_test_poly)

```

```
# XGBoost with Native API for MAPE Optimization
def xgboost_mape_train(X_train, y_train, X_test, y_test, params):
    # Convert to DMatrix format
    dtrain = xgb.DMatrix(X_train, label=y_train)
    dtest = xgb.DMatrix(X_test, label=y_test)

    # Custom MAPE evaluation metric
    def mape_eval(preds, dmatrix):
        labels = dmatrix.get_label()
        return 'mape', np.mean(np.abs((labels - preds) / (labels + 1e-6))) * 100

    # Train model
    model = xgb.train(
        params,
        dtrain,
        num_boost_round=1000,
        evals=[(dtrain, 'train'), (dtest, 'test')],
        early_stopping_rounds=50,
        feval=mape_eval,
        verbose_eval=50
    )
    return model

# Parameter Tuning with scikit-learn API
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0],
    'gamma': [0, 0.1, 0.2],
    'min_child_weight': [1, 3, 5]
}

xgb_sklearn = xgb.XGBRegressor(
    objective='reg:squarederror',
    n_estimators=100,
    random_state=42,
    n_jobs=-1
)

tscv = TimeSeriesSplit(n_splits=3)
search = RandomizedSearchCV(
    estimator=xgb_sklearn,
    param_distributions=param_grid,
    n_iter=20,
    cv=tscv,
    scoring='neg_mean_absolute_percentage_error',
    verbose=1,
    n_jobs=-1,
    random_state=42
)
```

```python

search.fit(X_train_scaled, y_train)
best_params = search.best_params_

# Final Model Training with MAPE Focus
final_params = {
    **best_params,
    'objective': 'reg:squarederror',
    'seed': 158
}

# Train with native API
xgb_model = xgboost_mape_train(
    X_train_scaled, y_train,
    X_test_scaled, y_test,
    final_params
)

# Evaluation
xgb_predictions = xgb_model.predict(xgb.DMatrix(X_test_scaled))
test.loc[X_test.index, 'XGBoost_Predicted_ZHVI'] = xgb_predictions

xgb_rmse = math.sqrt(mean_squared_error(y_test, xgb_predictions))
xgb_mape = mean_absolute_percentage_error(y_test, xgb_predictions)
xgb_mae = mean_absolute_error(y_test, xgb_predictions)
xgb_r2 = r2_score(y_test, xgb_predictions)

metrics = {
    'RMSE': xgb_rmse,
    'MAPE': xgb_mape,
    'MAE': xgb_mae,
    'R2': xgb_r2
}

print("\nModel Performance:")
for name, value in metrics.items():
    print(f"{name}: {value:.4f}")

# Plot truth vs prediction
plt.figure(figsize=(18, 6))
plt.plot(test['Year-Month'], test['ZHVI'], color='red', label='Truth (ZHVI)')
plt.plot(test['Year-Month'], test['XGBoost_Predicted_ZHVI'], color='darkgreen', label='
plt.xlabel('Date')
plt.ylabel('ZHVI')
x_ticks = np.arange(0, 80, 6)
plt.xticks(x_ticks)
plt.title('Time Series Plot: Truth vs XGBoost Predicted ZHVI')
plt.legend(loc='upper left')
plt.show()

```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[0]	train-rmse:97728.66420	train-mape:27.91250	test-rmse:214233.40336	test-ma
[50]	train-rmse:1000.27290	train-mape:0.22388	test-rmse:103571.44737	test-ma
[100]	train-rmse:172.37367	train-mape:0.03824	test-rmse:102197.62927	test-ma
[146]	train-rmse:75.17564	train-mape:0.01729	test-rmse:102249.79864	test-ma

Model Performance:
RMSE: 102249.7987
MAPE: 0.1449
MAE: 90674.5417
R2: -3.2301
```




```
# Display best parameters from hyperparameter tuning
print("\nBest Hyperparameters from RandomizedSearchCV:")
for param, value in search.best_params_.items():
    print(f"{param}: {value}")
```

```
Best Hyperparameters from RandomizedSearchCV:
subsample: 0.8
min_child_weight: 1
max_depth: 5
learning_rate: 0.1
```

```
gamma: 0.2
colsample_bytree: 0.9
```

Comparison of Models

```
# Combined Model Comparison Plot
plt.figure(figsize=(18, 6))
plt.plot(test['Year-Month'], test['ZHVI'], color='black', label='True ZHVI')
plt.plot(test['Year-Month'], OLS_pred, color='pink', label='OLS Predicted ZHVI')
plt.plot(test['Year-Month'], lasso_pred, color='green', label='Lasso Predicted ZHVI')
plt.plot(test['Year-Month'], ridge_pred, color='red', label='Ridge Predicted ZHVI')
plt.plot(test['Year-Month'], nn_predictions, color='purple', label='Neural Network Predic
plt.plot(test['Year-Month'], rf_predictions, color='orange', label='Random Forest Predict
plt.plot(test['Year-Month'], test['XGBoost_Predicted_ZHVI'], color='blue', label='XGBoost


plt.xlabel('Date')
plt.ylabel('ZHVI')
x_ticks = np.arange(0, 80, 6)
plt.xticks(x_ticks)
plt.title('Time Series Plot: True vs All Model Predictions')
plt.legend(loc='upper left')
plt.show()

# Model Performance Comparison Table
model_comparison = pd.DataFrame({
    'Model': ['OLS', 'Lasso', 'Ridge', 'Neural Network', 'Random Forest', 'XGBoost'],
    'RMSE': [rmse, lasso_rmse, ridge_rmse, nn_rmse, rf_rmse, xgb_rmse],
    'MAPE': [mape, lasso_mape, ridge_mape, nn_mape, rf_mape, xgb_mape],
    'MAE': [MAE, lasso_MAE, ridge_MAE, nn_mae, rf_mae, xgb_mae],
    'R2': [r2, lasso_r2, ridge_r2, nn_r2, rf_r2, xgb_r2]
})

print("\nModel Performance Comparison:")
print(model_comparison.sort_values('RMSE'))
```