Biwottgideon / KANSAS_HOUSING_MARKET_ANALYSIS

Code Issues Pull requests Actions Projects Wiki Security Insights

main

KANSAS_HOUSING_MARKET_ANALYSIS / Kansas_Market_Analysis_CSV.ipynb

Go to file t

Biwottgideon DSF-PT05_Phase0 2 minutes ago

5438 lines (5438 loc) · 1.42 MB

Phase 4 Project - Time Series Modeling

Group 12 Members:

1. Gideon Biwott - Group Lead
2. Silah Rugut
3. James Nyamu
4. Caroline Kisaulu
5. Eunita Nyengo
6. Lorrah Ngine

In [6]:
```
%%html
<img src="miamicover.png">
```

Overview:

This project aims to provide insights into the real estate market in Kansas City, focusing on forecasting future house prices, identifying top-performing zip codes in terms of growth, and analyzing the impact of economic shocks on real estate prices. Leveraging historical data from the Zillow dataset spanning from 1996 to 2018, advanced analytical techniques will be employed to address these research questions.

Problem Statement:

With the real estate market's volatility and complexity, investors and buyers need accurate forecasts and insights to make informed decisions. This project seeks to utilize historical real estate data to forecast house prices in Kansas City over the **next 3 years**, identify top-performing zip codes for investment opportunities, and analyze the effects of economic shocks, such as the financial global crisis, on real estate price growth.

Data Understanding:

The dataset consists of information on over 14,000 zip codes in the United States, stored in the 'data/zillow_data.csv' file. Each record includes details such as **RegionID (unique ID), RegionName (zip code), City, State, Metro region, CountyName, and SizeRank (ranking of zip code's size)**. Additionally, the dataset contains 265 columns representing average home values from April 1996 to April 2018 for each zip code.

Methods:

Time Series Analysis: Utilize time series analysis techniques to forecast future house prices in Kansas City for the next 1/3/5 years. **Growth Analysis:** Identify the top 5 zip codes in terms of growth by analyzing historical price trends and growth rates. **Impact Analysis:** Analyze the impact of economic shocks, such as the financial global crisis, on real estate prices in the top 5 zip codes to understand seasonal trends and market dynamics.

Research Questions

a).What will be the house prices outlook in the next 1/3/5 years in Kansas City? (Forecasting Future Prices)

b).What are the top 5 zip codes in terms of growth? / Which areas should investors or buyers look out for?(Regional Comparison and Growth Analysis)

c).How do economic shocks (the financial global crisis) affect the growth in real estate prices in the top 5 zipcodes? (Seasonal Trends in Real Estate Prices)

Import Necessary Libraries

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
import seaborn as sns
```

Load the Data

In [2]:

```
df = pd.read_csv('zillow_data.csv')
df
```

Out[2]:

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996
0	84654	60657	Chicago	IL	Chicago	Cook	1	334200.0	3354
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	2369
2	91982	77494	Katy	TX	Houston	Harris	3	210400.0	2122
3	84616	60614	Chicago	IL	Chicago	Cook	4	498100.0	5009
4	93144	79936	El Paso	TX	El Paso	El Paso	5	77300.0	773
...	...	...	...	...	...	...	...	...	
14718	58333	1338	Ashfield	MA	Greenfield Town	Franklin	14719	94600.0	943
14719	59107	3293	Woodstock	NH	Claremont	Grafton	14720	92700.0	925
14720	75672	40404	Berea	KY	Richmond	Madison	14721	57100.0	573
14721	93733	81225	Mount Crested Butte	CO	NaN	Gunnison	14722	191100.0	1924
14722	95851	89155	Mesquite	NV	Las Vegas	Clark	14723	176400.0	1763

14723 rows × 272 columns

Data Procesing

In [8]:

```
df.columns = pd.to_datetime(df.columns, errors='ignore')

# Melt the dataframe to have dates in a single column and prices as values
melted_data = df.melt(id_vars=['RegionID', 'RegionName', 'City', 'State', 'Metro', 'CountyNa
                      var_name='Date',
                      value_name='Price')
```

```
# Convert 'Date' to datetime format
melted_data['Date'] = pd.to_datetime(melted_data['Date'])

# Fill missing values with forward fill, then backfill if necessary
melted_data_filled = melted_data.fillna(method='ffill').fillna(method='bfill')

# Sort the data by RegionID and Date for easier analysis
melted_data_filled = melted_data_filled.sort_values(by=['RegionID', 'Date'])

# Check for missing values
missing_values = melted_data_filled.isnull().sum()

# Display the first few rows of the processed data and the missing values information
processed_head = melted_data_filled.head()

(missing_values, processed_head)
```

Out[8]:

```
(RegionID      0
 RegionName    0
 City          0
 State         0
 Metro         0
 CountyName    0
 SizeRank      0
 Date          0
 Price         0
 dtype: int64,
        RegionID  RegionName    City State        Metro CountyName  SizeRank  \
 5850      58196        1001  Agawam    MA  Springfield    Hampden      5851
 20573     58196        1001  Agawam    MA  Springfield    Hampden      5851
 35296     58196        1001  Agawam    MA  Springfield    Hampden      5851
 50019     58196        1001  Agawam    MA  Springfield    Hampden      5851
 64742     58196        1001  Agawam    MA  Springfield    Hampden      5851

             Date     Price
 5850  1996-04-01  113100.0
 20573 1996-05-01  112800.0
 35296 1996-06-01  112600.0
 50019 1996-07-01  112300.0
 64742 1996-08-01  112100.0  )
```

In [35]:

```
df.dtypes
```

Out[35]:

```
RegionID       int64
RegionName     int64
City          object
State         object
Metro         object
               ...
2017-12        int64
2018-01        int64
2018-02        int64
2018-03        int64
2018-04        int64
Length: 272, dtype: object
```

In [36]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14723 entries, 0 to 14722
Columns: 272 entries, RegionID to 2018-04
dtypes: float64(219), int64(49), object(4)
memory usage: 30.6+ MB
```

Feature Engineering

In [9]:

```
# Ensure melted_data_filled is sorted by date
melted_data_filled = melted_data_filled.sort_values(by=['Date'])
```

```
# Lag Features: Create a 1-month lagged price feature
melted_data_filled['Price_lag1'] = melted_data_filled.groupby('RegionID')['Price'].shift(1)

# Rolling Window Statistics: Create a rolling mean and standard deviation feature for the la
melted_data_filled['Price_rolling_mean3'] = melted_data_filled.groupby('RegionID')['Price'].
melted_data_filled['Price_rolling_std3'] = melted_data_filled.groupby('RegionID')['Price'].r

# Month and Year Extraction: Extract month and year as separate features
melted_data_filled['Month'] = melted_data_filled['Date'].dt.month
melted_data_filled['Year'] = melted_data_filled['Date'].dt.year

# Display the first few rows of the dataframe to verify the new features
melted_data_filled.head()
```

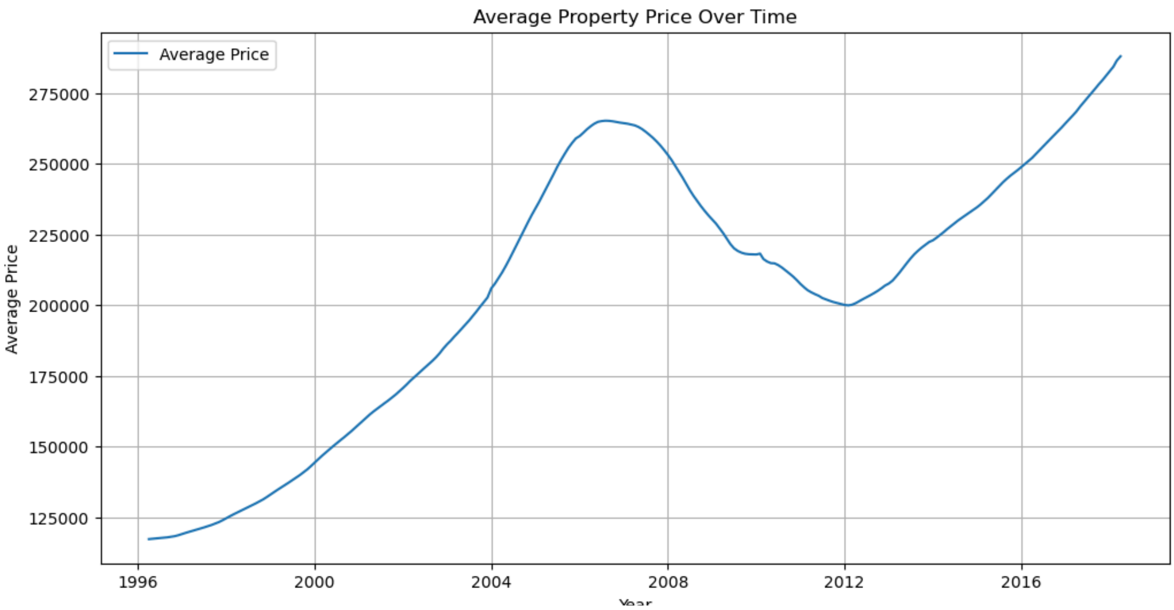
Out[9]:

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	Date	Price
5850	58196	1001	Agawam	MA	Springfield	Hampden	5851	1996-04-01	113100.0
9038	59257	3865	Plaistow	NH	Boston	Rockingham	9039	1996-04-01	124200.0
4898	99058	97058	The Dalles	OR	The Dalles	Wasco	4899	1996-04-01	93500.0
4259	69766	28203	Charlotte	NC	Charlotte	Mecklenburg	4260	1996-04-01	162500.0
12290	88348	69145	Kimball	NE	New York	Kimball	12291	1996-04-01	62200.0

EDA and Visualization

In [10]:

```
# 1. Trend Analysis
# Calculate average price per month
average_price_monthly = melted_data_filled.groupby('Date')['Price'].mean().reset_index()

# Plot the average price trend
plt.figure(figsize=(12, 6))
plt.plot(average_price_monthly['Date'], average_price_monthly['Price'], label='Average Price
plt.title('Average Property Price Over Time')
plt.xlabel('Year')
plt.ylabel('Average Price')
plt.legend()
plt.grid(True)
plt.show()
```



1.Average Property Price Over Time:

- The graph shows a long-term upward trend in average property prices from around 1996 until 2008.
- There is a significant dip observable around the 2008 mark, which aligns with the global financial crisis that impacted the housing market dramatically.
- There's a recovery and prices begin to increase again post-2012, suggesting a period of market correction and possible economic recovery or growth.
- The continued rise after 2012 may indicate a robust recovery and a return to a seller's market, with increasing property values.

In [11]:

```
# 2. Distribution Analysis
# Histogram of property prices
plt.figure(figsize=(12, 6))
sns.histplot(melted_data_filled['Price'], bins=50, kde=True)
plt.title('Distribution of Property Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```

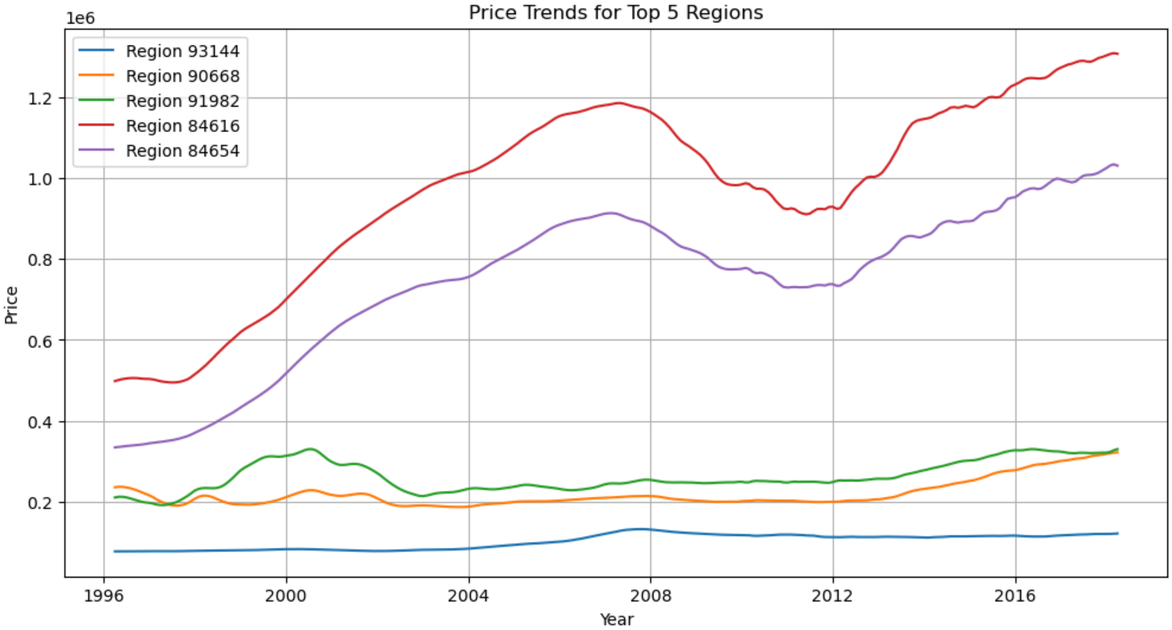
2.Distribution of Property Prices:

- The histogram displays a highly right-skewed distribution of property prices, indicating that most properties are at the lower end of the price range, with a few properties at very high prices.
- The long tail to the right suggests the presence of some extremely high-value properties in the dataset. These could be luxury properties or properties in very high-demand areas.
- The distribution also suggests that the housing market is not evenly distributed but is dominated by properties that are more affordable to the general population.

In [12]:

```
# 3. Comparative Analysis
# Compare price trends of top 5 regions by size rank
top_regions = melted_data_filled[melted_data_filled['SizeRank'] <= 5]['RegionID'].unique()
top_regions_data = melted_data_filled[melted_data_filled['RegionID'].isin(top_regions)]

plt.figure(figsize=(12, 6))
for region in top_regions:
    region_data = top_regions_data[top_regions_data['RegionID'] == region]
    plt.plot(region_data['Date'], region_data['Price'], label=f'Region {region}')

plt.title('Price Trends for Top 5 Regions')
plt.xlabel('Year')
```

```
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```



3.Price Trends for Top 5 Regions:

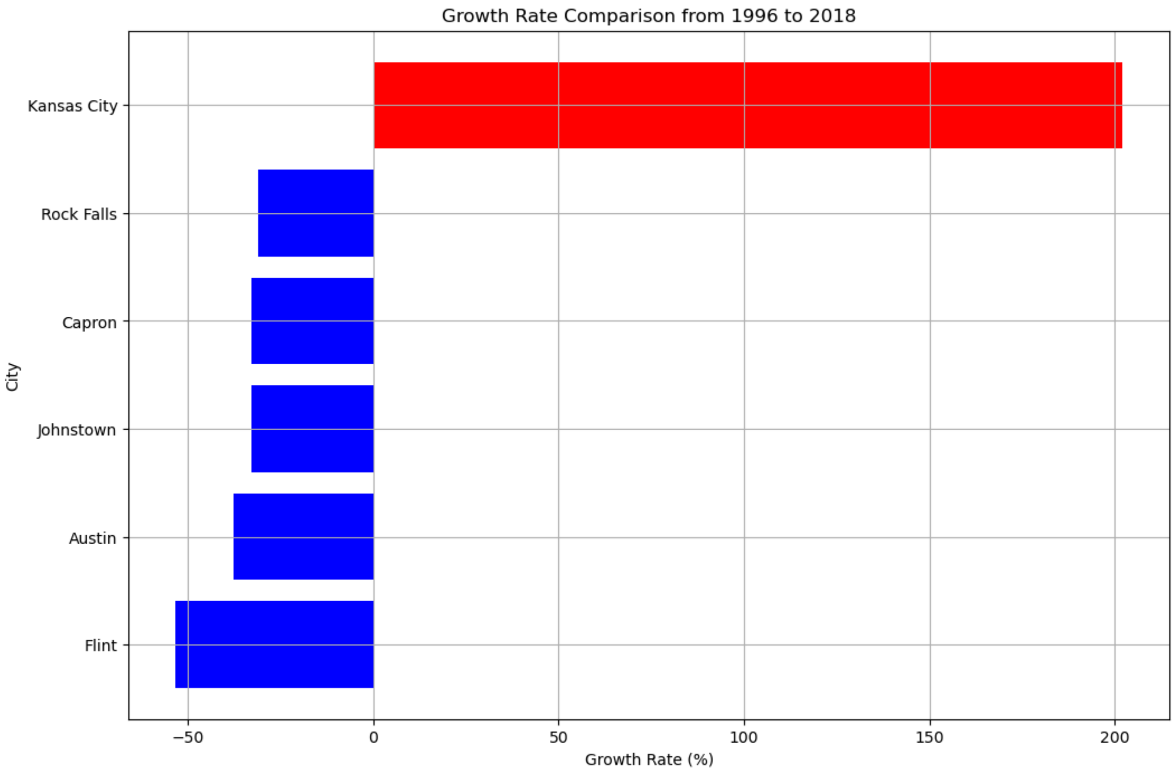
- The multiple line graph compares price trends in the top 5 regions by size rank.
- It shows that different regions have different price trajectories, reflecting regional economic conditions, housing demand, and supply dynamics.
- One region (presumably Region 84616) shows a much steeper increase in property prices than the others, indicating a stronger market or possibly a more affluent area with higher property valuation.
- Other regions also show growth, but it is more modest compared to the steep rise and volatility of Region 84616.
- All regions show the impact of the 2008 financial crisis, with a noticeable dip in prices around that period. However, the recovery trajectories vary, suggesting different rates of economic recovery or differences in regional housing market resilience.

Growth Rate

In [36]:

```
# Calculate growth rates for all cities from April 1996 to April 2018
df['GrowthRate'] = ((df['2018-04'] - df['1996-04']) / df['1996-04']) * 100

# Filter the dataset
kansas_city_growth_rate = df[df['City'] == 'Kansas City']['GrowthRate'].mean()

# cities with growth rates lower than Kansas City's
lower_growth_cities = df[df['GrowthRate'] < kansas_city_growth_rate]

# Sort these cities by growth rate and select six cities with the lowest growth rate
lowest_growth_cities = lower_growth_cities.sort_values('GrowthRate').head(6)

# Display the results
kansas_city_growth_rate, lowest_growth_cities[['City', 'GrowthRate']]

# Prepare the data for visualization by combining the lowest growth cities with the highest
cities_to_visualize = pd.concat([
    lowest_growth_cities,
    df[df['City'] == 'Kansas City'].sort_values('GrowthRate', ascending=False).head(1)
])

# Plotting
plt.figure(figsize=(12, 8))
```

```
plt.barh(cities_to_visualize['City'], cities_to_visualize['GrowthRate'],
         color=['red' if city == 'Kansas City' else 'blue' for city in cities_to_visualize['
plt.xlabel('Growth Rate (%)')
plt.ylabel('City')
plt.title('Growth Rate Comparison from 1996 to 2018')
plt.grid(True)
plt.show()
```



Reshape from Wide to Long Format

In [45]:

```
id_vars = df.columns[:7]
date_columns = df.columns[7:]

# Reshape the dataset from wide to long format
long_format = pd.melt(df, id_vars=id_vars, value_vars=date_columns, var_name='Date', value_n

# Display the first few rows of the reshaped data
print(long_format.head())
```

```
   RegionID  RegionName      City State             Metro CountyName  \
0     84654       60657   Chicago    IL           Chicago       Cook
1     90668       75070  McKinney    TX  Dallas-Fort Worth     Collin
2     91982       77494      Katy    TX           Houston     Harris
3     84616       60614   Chicago    IL           Chicago       Cook
4     93144       79936   El Paso    TX           El Paso    El Paso

   SizeRank     Date     Price
0         1  1996-04  334200.0
1         2  1996-04  235700.0
2         3  1996-04  210400.0
3         4  1996-04  498100.0
4         5  1996-04   77300.0
```

Filter Dataframe To focus on Kansas City

In [6]:

```
df.drop(['RegionID', 'SizeRank'], axis=1, inplace=True)
```

In [7]:

```
df[df['City'] == 'Kansas City'].shape
```

Out[7]: (37, 270)

In [8]:
```
def melt_data(df):
    '''
    function to reshape dataframe into a pandas datetime from wide to long format
    '''
    melted = pd.melt(df, id_vars=['RegionName', 'City', 'State', 'Metro', 'CountyName'], var
    melted['time'] = pd.to_datetime(melted['time'], infer_datetime_format=True)
    melted = melted.dropna(subset=['value'])
    return melted.groupby('time').aggregate({'value':'mean'})


def evaluate_arima_model(X, arima_order):
    '''
    evaluate an ARIMA model for a given order (p,d,q)
    '''
    # prepare training dataset
    train_size = int(len(X) * 0.66)
    train, test = X[0:train_size], X[train_size:]
    history = [x for x in train]
    # make predictions
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=arima_order)
        model_fit = model.fit(disp=0)
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(test[t])
    # calculate out of sample error
    error = mean_squared_error(test, predictions)
    return error

def evaluate_models(dataset, p_values, d_values, q_values):
    '''
     evaluate combinations of p, d and q values for an ARIMA model
    '''
    dataset = dataset.astype('float32')
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    mse = evaluate_arima_model(dataset, order)
                    if mse < best_score:
                        best_score, best_cfg = mse, order
                    print('ARIMA%s MSE=%.3f' % (order,mse))
                except:
                    continue
    print('Best ARIMA%s MSE=%.3f' % (best_cfg, best_score))
    return best_cfg
```

In [9]:
```
df_melted = melt_data(df)
df_melted.head()
```

Out[9]:

	value
time	
1996-04-01	118299.123063
1996-05-01	118419.044139
1996-06-01	118537.423268
1996-07-01	118653.069278
1996-08-01	118780.254312

ARIMA MODELING FITTING

In [11]:

```
roi_list = {}
for index, x in df[df['City'] == 'Kansas City'].iterrows():
    '''
    function that loop through each single zipcode, apply and evaluate the best ARIMA model
    through the smallest MSE based on several values and
    '''
    print(x['RegionName'])
    series = melt_data(df.loc[[index]])
    # evaluate parameters
    p_values = [0, 1, 2]
    d_values = range(0, 2)
    q_values = range(0, 2)
    warnings.filterwarnings("ignore")

    order = evaluate_models(series.values, p_values, d_values, q_values)

    model= ARIMA(series, order=order)
    model_fit= model.fit()
    thirty_six_months = model_fit.forecast(steps=36)[0][-1]
    today = series.iloc[-1].value
    roi = (thirty_six_months - today)/today
    roi_list[x['RegionName']] = roi
```

```
64119
ARIMA(0, 0, 0) MSE=93611590.076
ARIMA(0, 0, 1) MSE=24344135.774
ARIMA(0, 1, 0) MSE=875810.127
ARIMA(0, 1, 1) MSE=351653.230
ARIMA(1, 0, 0) MSE=921897.011
ARIMA(1, 1, 0) MSE=337473.443
ARIMA(1, 1, 1) MSE=256744.799
ARIMA(2, 0, 0) MSE=338327.407
ARIMA(2, 0, 1) MSE=259557.192
ARIMA(2, 1, 0) MSE=291309.276
ARIMA(2, 1, 1) MSE=259864.982
Best ARIMA(1, 1, 1) MSE=256744.799
64114
ARIMA(0, 0, 0) MSE=559210333.009
ARIMA(0, 0, 1) MSE=144359078.033
ARIMA(0, 1, 0) MSE=881044.928
ARIMA(0, 1, 1) MSE=322385.536
ARIMA(1, 0, 0) MSE=1130423.428
ARIMA(1, 1, 0) MSE=182842.663
ARIMA(1, 1, 1) MSE=157165.738
ARIMA(2, 0, 0) MSE=185511.413
ARIMA(2, 1, 0) MSE=170707.770
ARIMA(2, 1, 1) MSE=159283.549
Best ARIMA(1, 1, 1) MSE=157165.738
64151
ARIMA(0, 0, 0) MSE=348965340.874
ARIMA(0, 0, 1) MSE=90068757.065
ARIMA(0, 1, 0) MSE=1476214.284
ARIMA(0, 1, 1) MSE=555361.512
ARIMA(1, 0, 0) MSE=1611215.523
ARIMA(1, 1, 0) MSE=589199.533
ARIMA(1, 1, 1) MSE=372531.470
ARIMA(2, 0, 0) MSE=593654.932
ARIMA(2, 0, 1) MSE=379476.445
ARIMA(2, 1, 0) MSE=422441.139
ARIMA(2, 1, 1) MSE=357723.083
Best ARIMA(2, 1, 1) MSE=357723.083
64111
ARIMA(0, 0, 0) MSE=1046478451.082
ARIMA(0, 0, 1) MSE=268239852.715
ARIMA(0, 1, 0) MSE=1320317.965
ARIMA(0, 1, 1) MSE=442952.078
ARIMA(1, 0, 0) MSE=1681559.936
ARIMA(1, 1, 0) MSE=319922.896
ARIMA(1, 1, 1) MSE=230671.390
ARIMA(2, 0, 0) MSE=325903.166
```

```
ARIMA(2, 1, 0) MSE=291122.050
ARIMA(2, 1, 1) MSE=234076.938
Best ARIMA(1, 1, 1) MSE=230671.390
66102
ARIMA(0, 0, 0) MSE=309201073.471
ARIMA(0, 1, 0) MSE=730242.214
ARIMA(0, 1, 1) MSE=302242.199
ARIMA(1, 1, 0) MSE=208462.255
ARIMA(1, 1, 1) MSE=191855.952
ARIMA(2, 0, 1) MSE=192234.200
ARIMA(2, 1, 0) MSE=204267.921
ARIMA(2, 1, 1) MSE=189578.085
Best ARIMA(2, 1, 1) MSE=189578.085
64131
ARIMA(0, 0, 0) MSE=232139298.370
ARIMA(0, 0, 1) MSE=59730963.958
ARIMA(0, 1, 0) MSE=904157.486
ARIMA(0, 1, 1) MSE=348353.288
ARIMA(1, 0, 0) MSE=976655.899
ARIMA(1, 1, 0) MSE=242356.072
ARIMA(1, 1, 1) MSE=178909.643
ARIMA(2, 0, 0) MSE=242725.226
ARIMA(2, 0, 1) MSE=180343.328
ARIMA(2, 1, 0) MSE=205178.700
ARIMA(2, 1, 1) MSE=178953.122
Best ARIMA(1, 1, 1) MSE=178909.643
66104
ARIMA(0, 0, 0) MSE=402167993.965
ARIMA(0, 1, 0) MSE=707218.505
ARIMA(0, 1, 1) MSE=261610.613
ARIMA(1, 1, 0) MSE=245209.506
ARIMA(2, 0, 1) MSE=176771.260
ARIMA(2, 1, 0) MSE=209465.978
ARIMA(2, 1, 1) MSE=181514.236
Best ARIMA(2, 0, 1) MSE=176771.260
64155
ARIMA(0, 0, 0) MSE=350738953.197
ARIMA(0, 0, 1) MSE=90381034.697
ARIMA(0, 1, 0) MSE=880585.902
ARIMA(0, 1, 1) MSE=321075.701
ARIMA(1, 0, 0) MSE=1027629.161
ARIMA(1, 1, 0) MSE=403036.680
ARIMA(1, 1, 1) MSE=255431.359
ARIMA(2, 0, 0) MSE=410202.983
ARIMA(2, 0, 1) MSE=262988.060
ARIMA(2, 1, 0) MSE=343982.004
ARIMA(2, 1, 1) MSE=257561.217
Best ARIMA(1, 1, 1) MSE=255431.359
66109
ARIMA(0, 0, 0) MSE=437244681.925
ARIMA(0, 0, 1) MSE=112558202.706
ARIMA(0, 1, 0) MSE=1450302.123
ARIMA(0, 1, 1) MSE=546690.772
ARIMA(1, 0, 0) MSE=1631513.648
ARIMA(1, 1, 0) MSE=474070.524
ARIMA(1, 1, 1) MSE=355338.319
ARIMA(2, 0, 0) MSE=478354.407
ARIMA(2, 1, 0) MSE=456894.424
ARIMA(2, 1, 1) MSE=358810.257
Best ARIMA(1, 1, 1) MSE=355338.319
64134
ARIMA(0, 0, 0) MSE=255146346.458
ARIMA(0, 1, 0) MSE=700681.155
ARIMA(0, 1, 1) MSE=278811.170
ARIMA(1, 1, 0) MSE=280967.115
ARIMA(1, 1, 1) MSE=214217.993
ARIMA(2, 1, 0) MSE=252179.031
ARIMA(2, 1, 1) MSE=216196.534
Best ARIMA(1, 1, 1) MSE=214217.993
64116
ARIMA(0, 0, 0) MSE=138843629.088
ARIMA(0, 0, 1) MSE=36432457.068
ARIMA(0, 1, 0) MSE=904780.627
ARIMA(0, 1, 1) MSE=339553.586
```

```
ARIMA(1, 0, 0) MSE=1010416.082
ARIMA(1, 1, 0) MSE=339119.725
ARIMA(1, 1, 1) MSE=241521.751
ARIMA(2, 0, 0) MSE=341456.542
ARIMA(2, 0, 1) MSE=244732.991
ARIMA(2, 1, 0) MSE=268419.561
ARIMA(2, 1, 1) MSE=243622.719
Best ARIMA(1, 1, 1) MSE=241521.751
66106
ARIMA(0, 0, 0) MSE=247231778.541
ARIMA(0, 0, 1) MSE=62379165.572
ARIMA(0, 1, 0) MSE=1481413.136
ARIMA(0, 1, 1) MSE=527868.379
ARIMA(1, 0, 0) MSE=1545032.610
ARIMA(1, 1, 0) MSE=427083.818
ARIMA(1, 1, 1) MSE=304219.282
ARIMA(2, 0, 1) MSE=305942.661
ARIMA(2, 1, 0) MSE=359695.807
ARIMA(2, 1, 1) MSE=311069.269
Best ARIMA(1, 1, 1) MSE=304219.282
64157
ARIMA(0, 0, 0) MSE=1085569622.600
ARIMA(0, 0, 1) MSE=277591612.261
ARIMA(0, 1, 0) MSE=1978434.756
ARIMA(0, 1, 1) MSE=953431.830
ARIMA(1, 0, 0) MSE=2300447.010
ARIMA(1, 1, 0) MSE=860686.761
ARIMA(1, 1, 1) MSE=742147.112
ARIMA(2, 0, 1) MSE=759729.718
ARIMA(2, 1, 0) MSE=797181.245
ARIMA(2, 1, 1) MSE=750168.945
Best ARIMA(1, 1, 1) MSE=742147.112
64110
ARIMA(0, 0, 0) MSE=984075017.408
ARIMA(0, 0, 1) MSE=248534682.327
ARIMA(0, 1, 0) MSE=4549252.019
ARIMA(0, 1, 1) MSE=1569944.757
ARIMA(1, 0, 0) MSE=4647083.143
ARIMA(1, 1, 0) MSE=1491652.228
ARIMA(2, 0, 0) MSE=1488429.552
ARIMA(2, 1, 0) MSE=1078731.058
ARIMA(2, 1, 1) MSE=887691.723
Best ARIMA(2, 1, 1) MSE=887691.723
64117
ARIMA(0, 0, 0) MSE=59823991.760
ARIMA(0, 0, 1) MSE=15442607.750
ARIMA(0, 1, 0) MSE=973520.515
ARIMA(0, 1, 1) MSE=363420.069
ARIMA(1, 0, 0) MSE=993169.888
ARIMA(1, 1, 0) MSE=389622.452
ARIMA(1, 1, 1) MSE=281020.372
ARIMA(2, 0, 0) MSE=390467.961
ARIMA(2, 0, 1) MSE=282056.524
ARIMA(2, 1, 0) MSE=364573.977
ARIMA(2, 1, 1) MSE=293626.571
Best ARIMA(1, 1, 1) MSE=281020.372
66103
ARIMA(0, 0, 0) MSE=343376948.190
ARIMA(0, 0, 1) MSE=87690408.302
ARIMA(0, 1, 0) MSE=2098837.662
ARIMA(0, 1, 1) MSE=806242.831
ARIMA(1, 0, 0) MSE=2217390.475
ARIMA(1, 1, 0) MSE=758674.626
ARIMA(1, 1, 1) MSE=543321.274
ARIMA(2, 0, 0) MSE=764257.734
ARIMA(2, 0, 1) MSE=548192.959
ARIMA(2, 1, 0) MSE=670996.997
ARIMA(2, 1, 1) MSE=559347.307
Best ARIMA(1, 1, 1) MSE=543321.274
64112
ARIMA(0, 0, 0) MSE=3102533072.448
ARIMA(0, 0, 1) MSE=790913339.464
ARIMA(0, 1, 0) MSE=6056556.781
ARIMA(0, 1, 1) MSE=2549875.094
```

```
ARIMA(1, 0, 0) MSE=6515770.433
ARIMA(1, 1, 0) MSE=2872199.441
ARIMA(1, 1, 1) MSE=1904775.019
ARIMA(2, 0, 0) MSE=2936610.063
ARIMA(2, 0, 1) MSE=1976315.188
ARIMA(2, 1, 0) MSE=2263884.010
ARIMA(2, 1, 1) MSE=1877795.124
Best ARIMA(2, 1, 1) MSE=1877795.124
66112
ARIMA(0, 0, 0) MSE=213645290.889
ARIMA(0, 0, 1) MSE=55089523.326
ARIMA(0, 1, 0) MSE=1295061.397
ARIMA(0, 1, 1) MSE=554812.713
ARIMA(1, 0, 0) MSE=1412608.918
ARIMA(1, 1, 0) MSE=438699.160
ARIMA(1, 1, 1) MSE=367293.675
ARIMA(2, 0, 0) MSE=441231.188
ARIMA(2, 0, 1) MSE=370320.533
ARIMA(2, 1, 0) MSE=415043.094
ARIMA(2, 1, 1) MSE=376985.125
Best ARIMA(1, 1, 1) MSE=367293.675
64154
ARIMA(0, 0, 0) MSE=698619975.364
ARIMA(0, 0, 1) MSE=177591663.464
ARIMA(0, 1, 0) MSE=2455815.101
ARIMA(0, 1, 1) MSE=808450.471
ARIMA(1, 0, 0) MSE=2595022.332
ARIMA(1, 1, 0) MSE=905958.551
ARIMA(2, 0, 0) MSE=911340.392
ARIMA(2, 1, 0) MSE=674685.064
ARIMA(2, 1, 1) MSE=487282.692
Best ARIMA(2, 1, 1) MSE=487282.692
64108
ARIMA(0, 0, 0) MSE=1729712739.844
ARIMA(0, 0, 1) MSE=439333985.336
ARIMA(0, 1, 0) MSE=4009170.393
ARIMA(0, 1, 1) MSE=1509389.170
ARIMA(1, 0, 0) MSE=4414299.775
ARIMA(1, 1, 0) MSE=1372491.474
ARIMA(1, 1, 1) MSE=986971.272
ARIMA(2, 0, 0) MSE=1374109.014
ARIMA(2, 0, 1) MSE=998631.183
ARIMA(2, 1, 0) MSE=1143313.976
ARIMA(2, 1, 1) MSE=1020623.520
Best ARIMA(1, 1, 1) MSE=986971.272
64113
ARIMA(0, 0, 0) MSE=6315600546.710
ARIMA(0, 0, 1) MSE=1606894556.240
ARIMA(0, 1, 0) MSE=7808308.974
ARIMA(0, 1, 1) MSE=2860230.853
ARIMA(1, 0, 0) MSE=9282897.184
ARIMA(1, 1, 0) MSE=2501230.035
ARIMA(1, 1, 1) MSE=1727285.014
ARIMA(2, 0, 0) MSE=2543366.022
ARIMA(2, 1, 0) MSE=2043239.104
Best ARIMA(1, 1, 1) MSE=1727285.014
64106
ARIMA(0, 0, 0) MSE=925684761.540
ARIMA(0, 0, 1) MSE=235622708.291
ARIMA(0, 1, 0) MSE=2880733.788
ARIMA(0, 1, 1) MSE=1253214.131
ARIMA(1, 0, 0) MSE=3043104.256
ARIMA(1, 1, 0) MSE=1335622.848
ARIMA(1, 1, 1) MSE=1025797.115
ARIMA(2, 0, 1) MSE=1047934.932
ARIMA(2, 1, 0) MSE=1244628.182
ARIMA(2, 1, 1) MSE=1051288.496
Best ARIMA(1, 1, 1) MSE=1025797.115
64124
ARIMA(0, 0, 0) MSE=32342507.850
ARIMA(0, 0, 1) MSE=8132788.940
ARIMA(0, 1, 0) MSE=898324.751
ARIMA(0, 1, 1) MSE=389453.148
ARIMA(1, 0, 0) MSE=889290.759
```

```
ARIMA(1, 1, 0) MSE=458830.874
ARIMA(1, 1, 1) MSE=352059.833
ARIMA(2, 0, 0) MSE=457405.631
ARIMA(2, 0, 1) MSE=352024.494
ARIMA(2, 1, 0) MSE=418637.109
ARIMA(2, 1, 1) MSE=364514.845
Best ARIMA(2, 0, 1) MSE=352024.494
64137
ARIMA(0, 0, 0) MSE=156443588.099
ARIMA(0, 0, 1) MSE=39966608.100
ARIMA(0, 1, 0) MSE=1002359.131
ARIMA(0, 1, 1) MSE=363517.372
ARIMA(1, 0, 0) MSE=1058134.628
ARIMA(1, 1, 0) MSE=315308.656
ARIMA(1, 1, 1) MSE=225963.580
ARIMA(2, 0, 0) MSE=315311.063
ARIMA(2, 0, 1) MSE=227134.652
ARIMA(2, 1, 0) MSE=281724.451
ARIMA(2, 1, 1) MSE=231416.459
Best ARIMA(1, 1, 1) MSE=225963.580
64105
ARIMA(0, 0, 0) MSE=233439718.259
ARIMA(0, 0, 1) MSE=59342038.307
ARIMA(0, 1, 0) MSE=1685229.670
ARIMA(0, 1, 1) MSE=734414.826
ARIMA(1, 0, 0) MSE=1585561.399
ARIMA(1, 1, 0) MSE=1016577.926
ARIMA(1, 1, 1) MSE=682609.006
ARIMA(2, 0, 0) MSE=1018268.790
ARIMA(2, 0, 1) MSE=686752.896
ARIMA(2, 1, 0) MSE=856858.046
ARIMA(2, 1, 1) MSE=676616.762
Best ARIMA(2, 1, 1) MSE=676616.762
64129
ARIMA(0, 0, 0) MSE=116257594.074
ARIMA(0, 0, 1) MSE=28874146.911
ARIMA(0, 1, 0) MSE=727670.750
ARIMA(0, 1, 1) MSE=294036.101
ARIMA(1, 0, 0) MSE=705104.698
ARIMA(1, 1, 0) MSE=357629.389
ARIMA(1, 1, 1) MSE=251863.181
ARIMA(2, 0, 1) MSE=250819.286
ARIMA(2, 1, 0) MSE=323110.471
ARIMA(2, 1, 1) MSE=256519.700
Best ARIMA(2, 0, 1) MSE=250819.286
64123
ARIMA(0, 0, 0) MSE=23137869.162
ARIMA(0, 0, 1) MSE=5809465.250
ARIMA(0, 1, 0) MSE=680250.676
ARIMA(0, 1, 1) MSE=340062.338
ARIMA(1, 0, 0) MSE=679806.239
ARIMA(1, 1, 0) MSE=464566.705
ARIMA(1, 1, 1) MSE=335517.778
ARIMA(2, 0, 0) MSE=464087.269
ARIMA(2, 0, 1) MSE=336980.487
ARIMA(2, 1, 0) MSE=408912.597
ARIMA(2, 1, 1) MSE=330272.639
Best ARIMA(2, 1, 1) MSE=330272.639
66111
ARIMA(0, 0, 0) MSE=270700073.764
ARIMA(0, 0, 1) MSE=68902081.186
ARIMA(0, 1, 0) MSE=1522300.128
ARIMA(0, 1, 1) MSE=583145.639
ARIMA(1, 0, 0) MSE=1642704.646
ARIMA(1, 1, 0) MSE=522273.996
ARIMA(1, 1, 1) MSE=362812.034
ARIMA(2, 0, 0) MSE=523006.138
ARIMA(2, 0, 1) MSE=364750.120
ARIMA(2, 1, 0) MSE=424415.423
ARIMA(2, 1, 1) MSE=359336.055
Best ARIMA(2, 1, 1) MSE=359336.055
64126
ARIMA(0, 0, 0) MSE=9302438.448
ARIMA(0, 0, 1) MSE=2572034.202
```

```
ARIMA(0, 1, 0) MSE=1011228.625
ARIMA(0, 1, 1) MSE=484736.721
ARIMA(1, 0, 0) MSE=982513.641
ARIMA(1, 1, 0) MSE=751156.561
ARIMA(2, 0, 0) MSE=749276.894
ARIMA(2, 1, 0) MSE=649879.036
ARIMA(2, 1, 1) MSE=494761.931
Best ARIMA(0, 1, 1) MSE=484736.721
64156
ARIMA(0, 0, 0) MSE=1987314352.207
ARIMA(0, 0, 1) MSE=509258509.272
ARIMA(0, 1, 0) MSE=5396842.954
ARIMA(0, 1, 1) MSE=3192384.003
ARIMA(1, 0, 0) MSE=10266588.810
ARIMA(1, 1, 0) MSE=2560744.697
ARIMA(1, 1, 1) MSE=2610064.022
ARIMA(2, 0, 0) MSE=4567042.631
ARIMA(2, 0, 1) MSE=4574804.121
ARIMA(2, 1, 0) MSE=2904981.488
ARIMA(2, 1, 1) MSE=2826921.785
Best ARIMA(1, 1, 0) MSE=2560744.697
64153
ARIMA(0, 0, 0) MSE=405425066.508
ARIMA(0, 0, 1) MSE=104406824.447
ARIMA(0, 1, 0) MSE=1632733.615
ARIMA(0, 1, 1) MSE=683477.576
ARIMA(1, 0, 0) MSE=1767352.797
ARIMA(1, 1, 0) MSE=713363.680
ARIMA(1, 1, 1) MSE=501070.502
ARIMA(2, 1, 0) MSE=556128.582
ARIMA(2, 1, 1) MSE=487084.001
Best ARIMA(2, 1, 1) MSE=487084.001
64158
ARIMA(0, 0, 0) MSE=535971325.519
ARIMA(0, 0, 1) MSE=138847679.179
ARIMA(0, 1, 0) MSE=1801796.522
ARIMA(0, 1, 1) MSE=697578.679
ARIMA(1, 0, 0) MSE=2061480.728
ARIMA(1, 1, 0) MSE=795279.486
ARIMA(1, 1, 1) MSE=581202.629
ARIMA(2, 0, 0) MSE=810237.160
ARIMA(2, 1, 0) MSE=808510.991
ARIMA(2, 1, 1) MSE=620524.468
Best ARIMA(1, 1, 1) MSE=581202.629
64145
ARIMA(0, 0, 0) MSE=698416407.048
ARIMA(0, 0, 1) MSE=181022940.882
ARIMA(0, 1, 0) MSE=3728323.303
ARIMA(0, 1, 1) MSE=1391958.910
ARIMA(1, 0, 0) MSE=4119127.706
ARIMA(1, 1, 0) MSE=1048360.639
ARIMA(2, 0, 0) MSE=1059684.748
ARIMA(2, 1, 0) MSE=983785.155
ARIMA(2, 1, 1) MSE=800960.941
Best ARIMA(2, 1, 1) MSE=800960.941
64136
ARIMA(0, 0, 0) MSE=89624739.896
ARIMA(0, 0, 1) MSE=23065188.384
ARIMA(0, 1, 0) MSE=1302823.292
ARIMA(0, 1, 1) MSE=516083.836
ARIMA(1, 0, 0) MSE=1330663.866
ARIMA(1, 1, 0) MSE=524851.031
ARIMA(1, 1, 1) MSE=377390.264
ARIMA(2, 0, 0) MSE=524826.288
ARIMA(2, 0, 1) MSE=378563.310
ARIMA(2, 1, 0) MSE=415251.678
ARIMA(2, 1, 1) MSE=372776.898
Best ARIMA(2, 1, 1) MSE=372776.898
64125
ARIMA(0, 0, 0) MSE=14948908.061
ARIMA(0, 0, 1) MSE=3741717.912
ARIMA(0, 1, 0) MSE=748214.552
ARIMA(0, 1, 1) MSE=351609.228
ARIMA(1, 0, 0) MSE=721007.484
```

```
ARIMA(1, 1, 0) MSE=511892.501
ARIMA(1, 1, 1) MSE=345514.503
ARIMA(2, 0, 0) MSE=511035.828
ARIMA(2, 1, 0) MSE=424887.638
ARIMA(2, 1, 1) MSE=345409.487
Best ARIMA(2, 1, 1) MSE=345409.487
64139
ARIMA(0, 0, 0) MSE=773593906.519
ARIMA(0, 0, 1) MSE=199269420.426
ARIMA(0, 1, 0) MSE=2885219.471
ARIMA(0, 1, 1) MSE=1131444.680
ARIMA(1, 0, 0) MSE=3189163.259
ARIMA(1, 1, 0) MSE=1003659.597
ARIMA(1, 1, 1) MSE=734503.934
ARIMA(2, 0, 0) MSE=1016866.053
ARIMA(2, 0, 1) MSE=745325.025
ARIMA(2, 1, 0) MSE=849916.836
ARIMA(2, 1, 1) MSE=752767.193
Best ARIMA(1, 1, 1) MSE=734503.934
64146
ARIMA(0, 0, 0) MSE=138493454.959
ARIMA(0, 0, 1) MSE=35488321.406
ARIMA(0, 1, 0) MSE=1854495.736
ARIMA(0, 1, 1) MSE=747863.172
ARIMA(1, 0, 0) MSE=1860432.815
ARIMA(1, 1, 0) MSE=599299.334
ARIMA(1, 1, 1) MSE=456711.515
ARIMA(2, 0, 0) MSE=599686.698
ARIMA(2, 0, 1) MSE=457338.153
ARIMA(2, 1, 0) MSE=511273.913
ARIMA(2, 1, 1) MSE=463110.939
Best ARIMA(1, 1, 1) MSE=456711.515
```

In [12]:

```
roi_list
```

Out[12]:

```
{64119: 0.05679732209422842,
 64114: 0.1192970729762698,
 64151: 0.05905026321941397,
 64111: 0.09693136094924668,
 66102: 0.01407573960837967,
 64131: 0.04546284263701976,
 66104: 0.0017268933293670607,
 64155: 0.06257263719843865,
 66109: 0.07817196811403077,
 64134: 0.04056394230247112,
 64116: 0.07975137309361202,
 66106: 0.1373985358861824,
 64157: 0.06528310078178046,
 64110: 0.08353030324922359,
 64117: 0.06822308914977174,
 66103: 0.10155508045775981,
 64112: 0.10592691843828653,
 66112: 0.08488559860951322,
 64154: 0.04150743227539524,
 64108: 0.03203111596063606,
 64113: 0.08080219700287117,
 64106: 0.14737096179258374,
 64124: 0.0067724159021344005,
 64137: 0.06481306716880608,
 64105: 0.06870997231295099,
 64129: -0.01684479110929328,
 64123: 0.05745865294676608,
 66111: 0.06020684041017004,
 64126: 0.06094194217722193,
 64156: 0.06920193050296697,
 64153: 0.05623209319876416,
 64158: 0.06907822253949673,
 64145: 0.09720449413548049,
 64136: 0.04962982541605204,
 64125: 0.05666884598152787,
 64139: 0.09670521678169679,
 64146: 0.044572768695209576}
```

In [13]:

```
s = [(k, roi_list[k]) for k in sorted(roi_list, key=roi_list.get, reverse=True)]
for k, v in s:
    print(k, v)
```

```
64106 0.14737096179258374
66106 0.1373985358861824
64114 0.1192970729762698
64112 0.10592691843828653
66103 0.10155508045775981
64145 0.09720449413548049
64111 0.09693136094924668
64139 0.09670521678169679
66112 0.08488559860951322
64110 0.08353030324922359
64113 0.08080219700287117
64116 0.07975137309361202
66109 0.07817196811403077
64156 0.06920193050296697
64158 0.06907822253949673
64105 0.06870997231295099
64117 0.06822308914977174
64157 0.06528310078178046
64137 0.06481306716880608
64155 0.06257263719843865
64126 0.06094194217722193
66111 0.06020684041017004
64151 0.05905026321941397
64123 0.05745865294676608
64119 0.05679732209422842
64125 0.05666884598152787
64153 0.05623209319876416
64136 0.04962982541605204
64131 0.04546284263701976
64146 0.044572768695209576
64154 0.04150743227539524
64134 0.04056394230247112
64108 0.03203111596063606
66102 0.01407573960837967
64124 0.0067724159021344005
66104 0.0017268933293670607
64129 -0.01684479110929328
```

Here we have our 5 best zipcodes.

In [14]:

```
top_5 = s[:5]
top_5
```

Out[14]:

```
[(64106, 0.14737096179258374),
 (66106, 0.1373985358861824),
 (64114, 0.1192970729762698),
 (64112, 0.10592691843828653),
 (66103, 0.10155508045775981)]
```

In [15]:

```
df_top_5 = pd.DataFrame(top_5)
df_top_5
df_top_5.columns = ['ZipCode', 'ROI']
df_top_5
```

Out[15]:

	ZipCode	ROI
0	64106	0.147371
1	66106	0.137399
2	64114	0.119297
3	64112	0.105927
4	66103	0.101555

In [16]:

```
# let's set this option so that the results are not defined in scientific terms
pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

Let's check the relation between zipcode, city and ROI

In [17]:

```
#Get Location Names
best5_zipcodes = top_5
best_5 = {}
for i in top_5:
    city = df[df['RegionName']==i[0]].City.values[0]
    state = df[df['RegionName']==i[0]].State.values[0]
    print(f'Zipcode : {i[0]} \nLocation: {city}, {state}\nROI : {i[1]}\n')
```

```
Zipcode : 64106
Location: Kansas City, MO
ROI : 0.14737096179258374

Zipcode : 66106
Location: Kansas City, KS
ROI : 0.1373985358861824

Zipcode : 64114
Location: Kansas City, MO
ROI : 0.1192970729762698

Zipcode : 64112
Location: Kansas City, MO
ROI : 0.10592691843828653

Zipcode : 66103
Location: Kansas City, KS
ROI : 0.10155508045775981
```

In [18]:

```
df_top_5['Value In 3 Years 1000000 Invested'] = (df_top_5['ROI']+1)*1000000
df_top_5
```

Out[18]:

	ZipCode	ROI	Value In 3 Years 1000000 Invested
0	64106	0.15	1147370.96
1	66106	0.14	1137398.54
2	64114	0.12	1119297.07
3	64112	0.11	1105926.92
4	66103	0.10	1101555.08

TIME SERIES ANALYSIS

Zipcode : 64106

Location: Kansas City, MO

In [19]:

```
zc_64106 = df[df['RegionName'] == 64106]
zc_64106 = melt_data(zc_64106)
zc_64106.head()
zc_64106.tail()
```

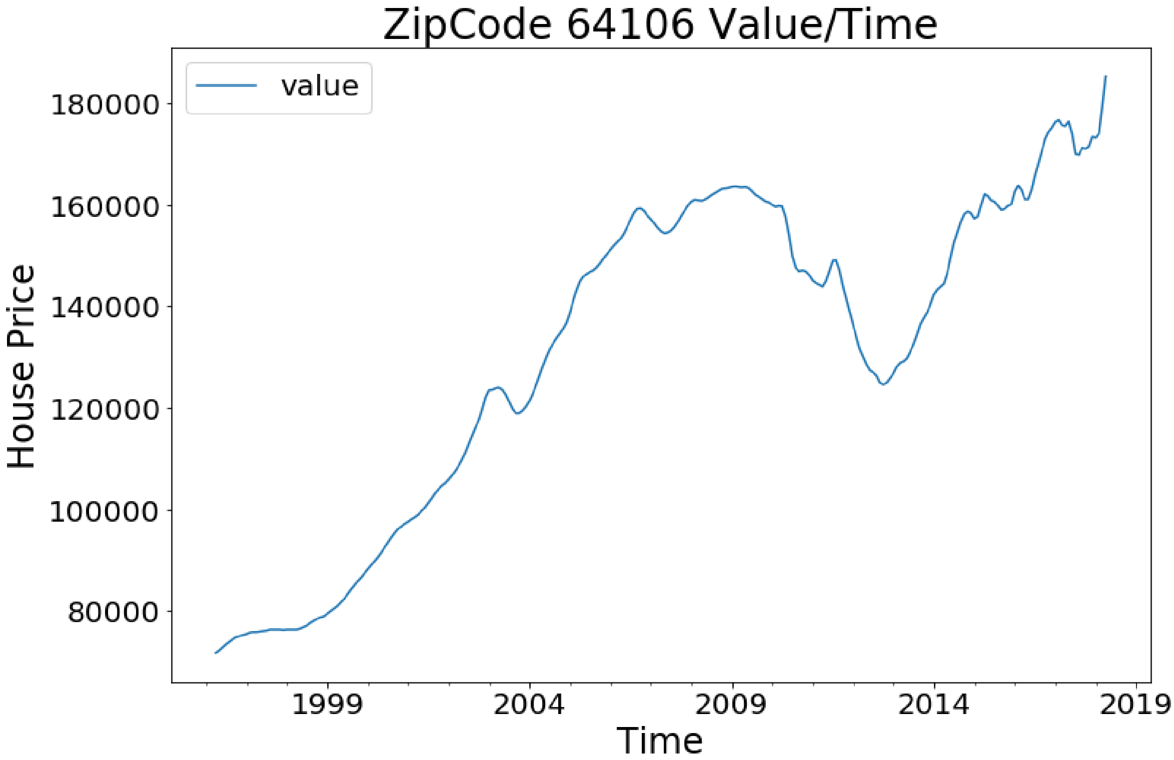
Out[19]:

	value
time	

2017-12-01	173400.00
2018-01-01	173200.00
2018-02-01	174000.00
2018-03-01	179300.00
2018-04-01	185300.00

In [20]:

```
zc_64106.plot(figsize=(12,8))
plt.title('ZipCode 64106 Value/Time', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```



In [21]:

```
model = ARIMA(zc_64106, order = (1,1,1))

model_fit = model.fit(disp=0)

print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                D.value   No. Observations:                  264
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -2079.527
Method:                       css-mle   S.D. of innovations            635.630
Date:                Thu, 07 Nov 2019   AIC                           4167.054
Time:                        14:33:57   BIC                           4181.358
Sample:                    05-01-1996   HQIC                          4172.801
                         - 04-01-2018
=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
const           469.4010    179.878      2.610      0.010     116.846     821.956
ar.L1.D.value     0.6409      0.052     12.245      0.000       0.538       0.744
ma.L1.D.value     0.6632      0.043     15.603      0.000       0.580       0.746
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
```

```
AR.1            1.5602          +0.0000j            1.5602            0.0000
MA.1           -1.5079          +0.0000j            1.5079            0.5000
-----------------------------------------------------------------------------
```

The model returns a lot of information, but we'll focus only on the table of coefficients.

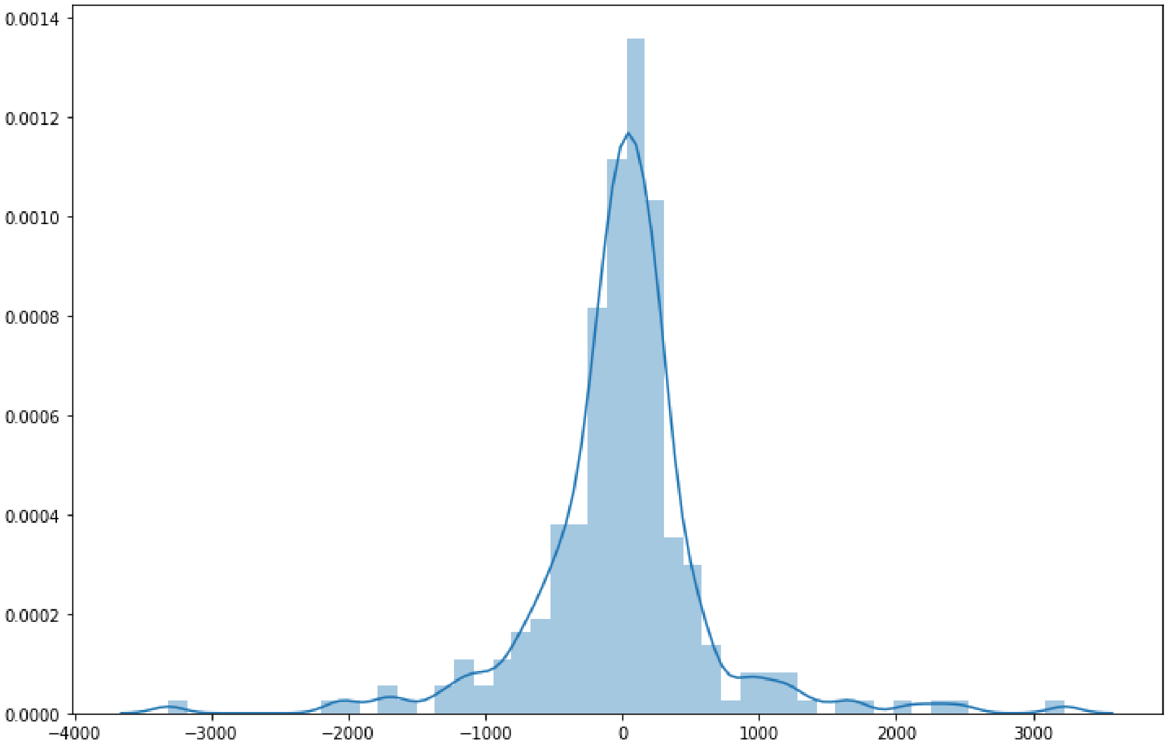
The coef column above shows the importance of each feature and how each one impacts the time series patterns.

The P>|z| provides the significance of each feature weight.

For our time-series, we see that each weight has a p-value lower than 0.05, so it is reasonable to retain all of them in our model.

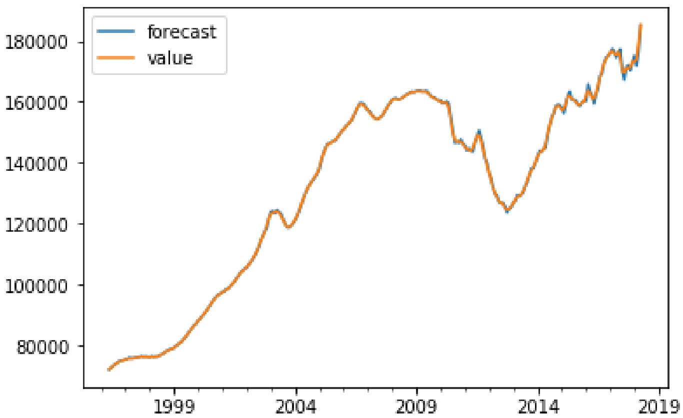
In [22]:

```
residuals = pd.DataFrame(model_fit.resid)
plt.figure(figsize=(12,8))
sns.distplot(residuals);
```



In [23]:

```
model_fit.plot_predict()
plt.show()
```



As we can see our residuals are relatively normal.

Next step is to create a graphic with the actual value for the properties that we already have and the forecast. To do so we are first going to create a new DateTime Index starting 1 month after our actual

DateTime and ending 36 months after.

In [24]:
```
new_per = pd.date_range(start='2018/04/01', periods=36, freq='MS')
new_per[:5]
```

Out[24]:
```
DatetimeIndex(['2018-04-01', '2018-05-01', '2018-06-01', '2018-07-01',
               '2018-08-01'],
              dtype='datetime64[ns]', freq='MS')
```

Here we can see all our values forecasted for the next 36 months.

In [25]:
```
model_fit.forecast(36)[0]
```

Out[25]:
```
array([189506.93847454, 192371.89933686, 194376.72496013, 195830.25030631,
       196930.42192191, 197804.11286245, 198532.64191072, 199168.13001869,
       199743.98390421, 200281.6154709 , 200794.7485936 , 201292.1795356 ,
       201779.54622702, 202260.46227688, 202737.24381355, 203211.37535036,
       203683.80838025, 204155.15275884, 204625.79937071, 205095.99875174,
       205565.91148195, 206035.64048444, 206505.25172738, 206974.78749281,
       207444.27488122, 207913.73126257, 208383.16777007, 208852.5915395 ,
       209322.00714452, 209791.41751659, 210260.82453463, 210730.22940291,
       211199.63289331, 211669.03550056, 212138.43754177, 212607.83922017])
```

In [26]:
```
df_forecast = pd.DataFrame(model_fit.forecast(36)[0])
df_forecast.columns = ['Value']
df_forecast.head()
```

Out[26]:

	Value
0	189506.94
1	192371.90
2	194376.72
3	195830.25
4	196930.42

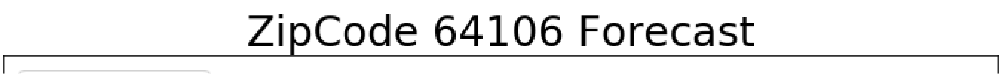
In [27]:
```
df_forecast = df_forecast.set_index(new_per)
df_forecast.head()
```

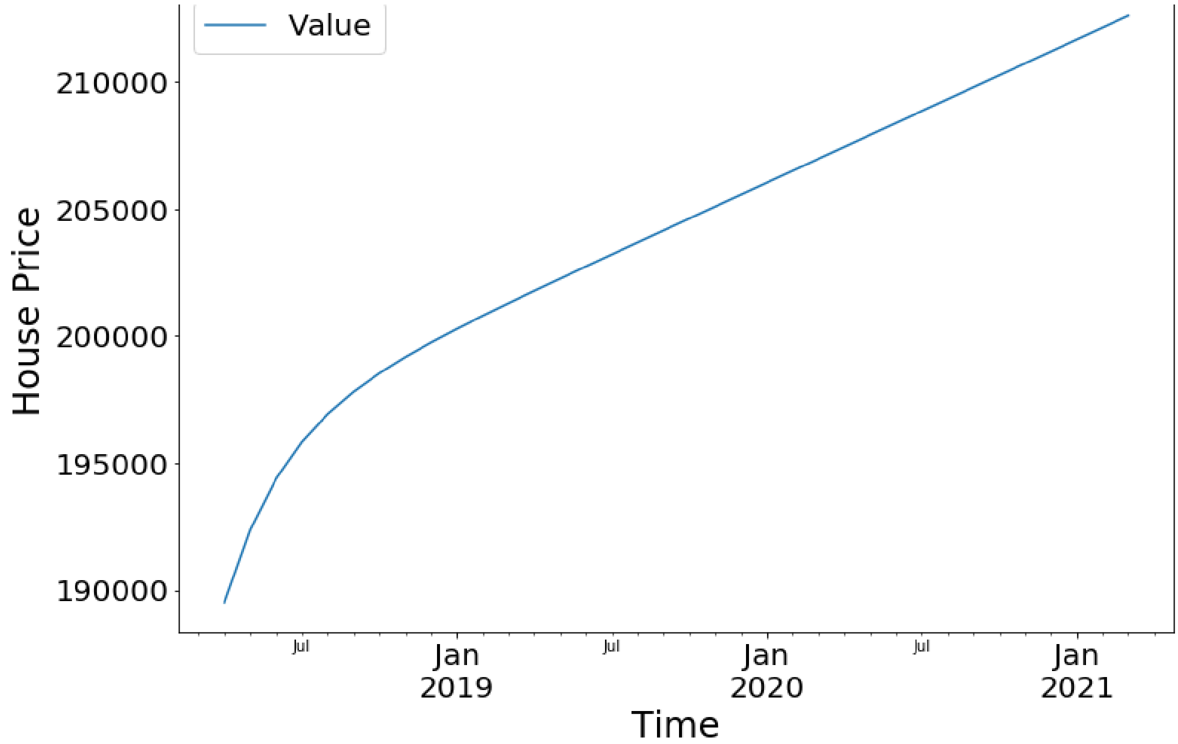
Out[27]:

	Value
2018-04-01	189506.94
2018-05-01	192371.90
2018-06-01	194376.72
2018-07-01	195830.25
2018-08-01	196930.42

We are going to plot our forecast.

In [28]:
```
df_forecast.plot(figsize=(12,8))
plt.title('ZipCode 64106 Forecast', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```





At this point we concatenate the actual values and the forecast.

In [29]:
```
zc_64106_forecast = pd.concat([zc_64106, df_forecast])
print(zc_64106_forecast.head())
print(zc_64106_forecast.tail())
```

```
              Value     value
1996-04-01      nan  71800.00
1996-05-01      nan  72300.00
1996-06-01      nan  72900.00
1996-07-01      nan  73500.00
1996-08-01      nan  74000.00
                Value   value
2020-11-01  210730.23     nan
2020-12-01  211199.63     nan
2021-01-01  211669.04     nan
2021-02-01  212138.44     nan
2021-03-01  212607.84     nan
```

In [30]:
```
zc_64106_forecast.rename(columns={"Value": "Forecast Value", "value": "Current Value"}, inpl
zc_64106_forecast.head()
```

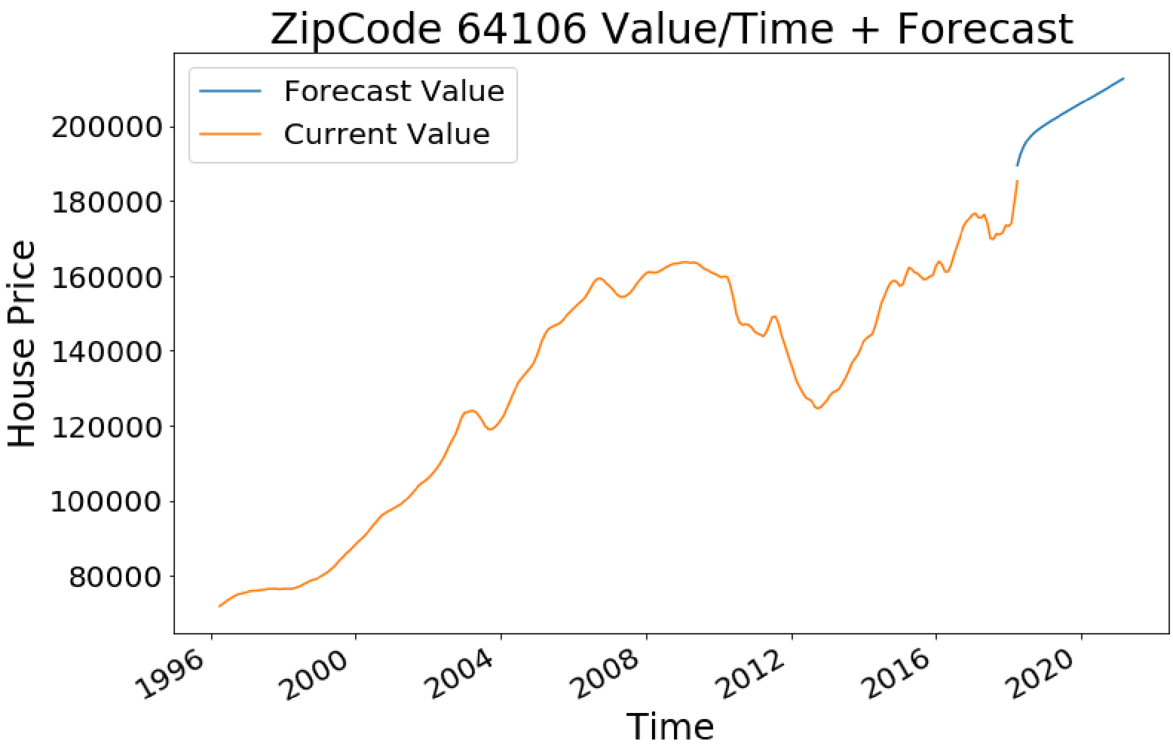
Out[30]:

	Forecast Value	Current Value
1996-04-01	nan	71800.00
1996-05-01	nan	72300.00
1996-06-01	nan	72900.00
1996-07-01	nan	73500.00
1996-08-01	nan	74000.00

Now we are going to plot our Value plus the forecast on the same graphic.

In [31]:
```
zc_64106_forecast.plot(figsize=(12,8))
plt.title('ZipCode 64106 Value/Time + Forecast', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```

```
plt.legend(fontsize=20);
```



Zipcode : 66106

Location: Kansas City, KS

In [32]:

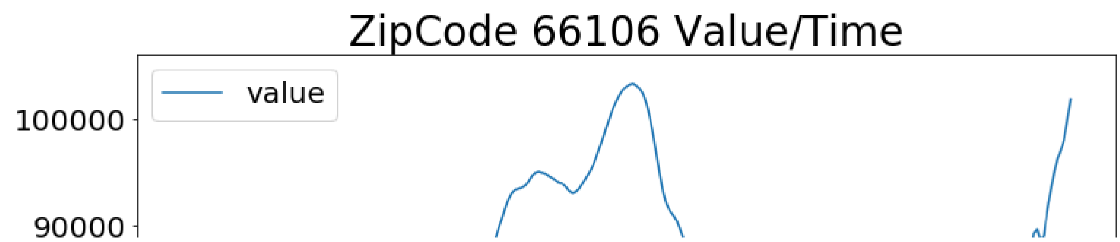
```
zc_66106 = df[df['RegionName'] == 66106]
zc_66106 = melt_data(zc_66106)
zc_66106.head()
```

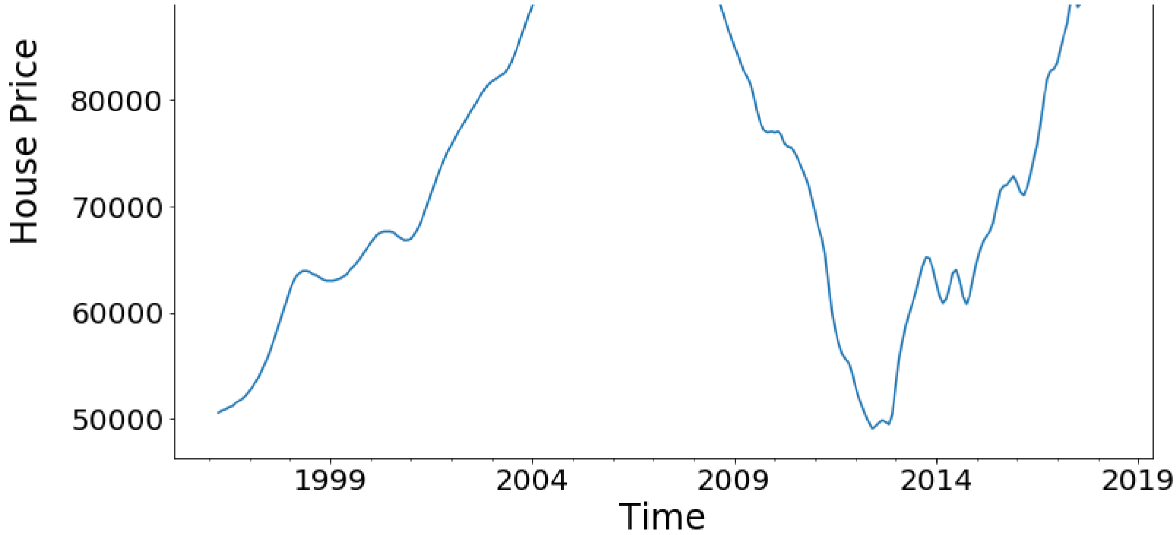
Out[32]:

time	value
1996-04-01	50600.00
1996-05-01	50800.00
1996-06-01	50900.00
1996-07-01	51100.00
1996-08-01	51200.00

In [33]:

```
zc_66106.plot(figsize=(12,8))
plt.title('ZipCode 66106 Value/Time', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```





In [34]:

```
model = ARIMA(zc_66106, order = (1,1,1))

model_fit = model.fit(disp=0)

print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                D.value   No. Observations:                  264
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -1919.076
Method:                       css-mle   S.D. of innovations            345.845
Date:                Thu, 07 Nov 2019   AIC                           3846.152
Time:                        14:33:58   BIC                           3860.456
Sample:                    05-01-1996   HQIC                          3851.900
                         - 04-01-2018
=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
const           219.5043    166.302      1.320      0.188    -106.441     545.449
ar.L1.D.value     0.7970      0.038     20.703      0.000       0.722       0.872
ma.L1.D.value     0.6112      0.042     14.686      0.000       0.530       0.693
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.2548           +0.0000j            1.2548            0.0000
MA.1           -1.6361           +0.0000j            1.6361            0.5000
-----------------------------------------------------------------------------
```

The model returns a lot of information, but we'll focus only on the table of coefficients.

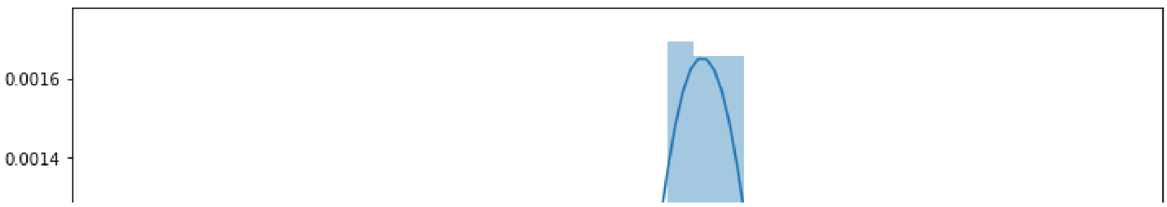
The coef column above shows the importance of each feature and how each one impacts the time series patterns.

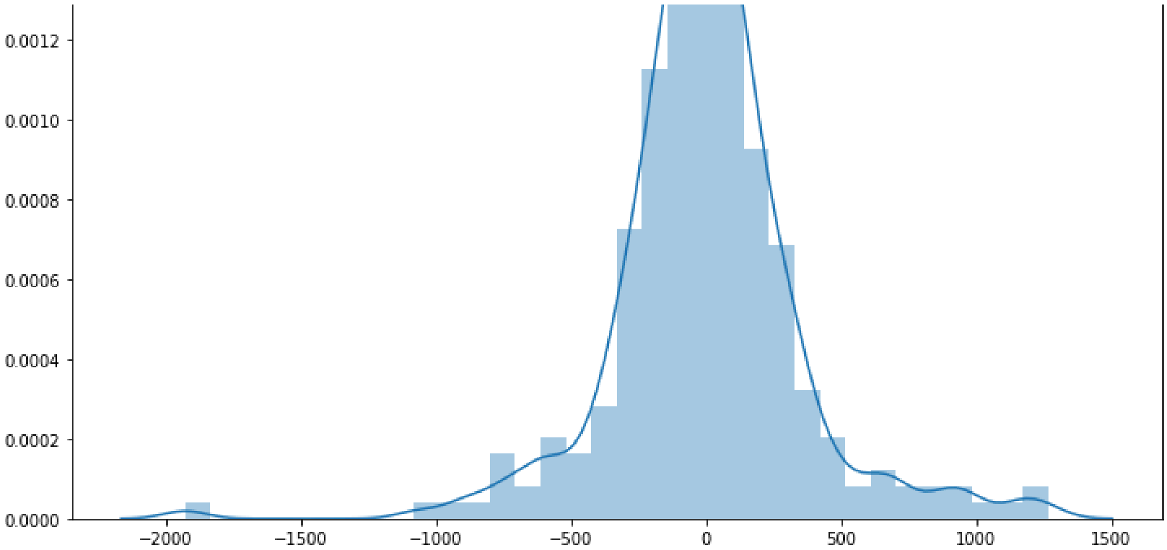
The P>|z| provides the significance of each feature weight.

For our time-series, we see that each weight has a p-value lower than 0.05, so it is reasonable to retain all of them in our model.

In [35]:

```
residuals = pd.DataFrame(model_fit.resid)
plt.figure(figsize=(12,8))
sns.distplot(residuals);
```



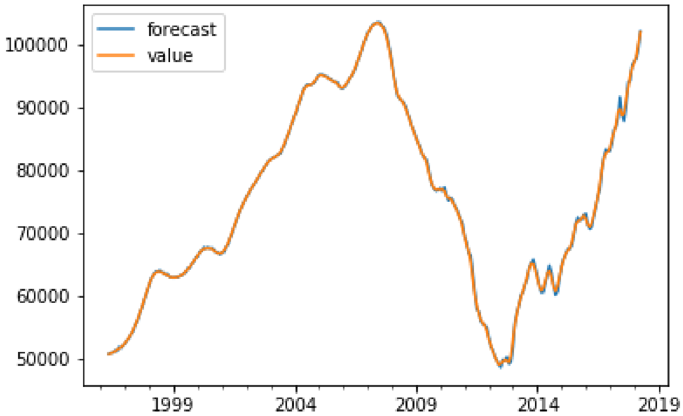


As we can see our residuals are relatively normal.

In [36]:

```
plt.figure(figsize=(12,8))
model_fit.plot_predict();
```

```
<Figure size 864x576 with 0 Axes>
```



In [37]:

```
new_per = pd.date_range(start='2018/04/01', periods=36, freq='MS')
new_per[:5]
```

Out[37]:

```
DatetimeIndex(['2018-04-01', '2018-05-01', '2018-06-01', '2018-07-01',
               '2018-08-01'],
              dtype='datetime64[ns]', freq='MS')
```

Here we can see all our values forecasted for the next 36 months.

In [38]:

```
model_fit.forecast(36)[0]
```

Out[38]:

```
array([103358.17446377, 104564.84354963, 105571.07408599, 106417.56405352,
       107136.74792456, 107754.47452443, 108291.34421207, 108763.77455316,
       109184.84961406, 109564.99681107, 109912.52632914, 110234.06101438,
       110534.87897924, 110819.18664352, 111090.3363359 , 111350.99971254,
       111603.30596403, 111848.95196016, 112089.29003034, 112325.39792067,
       112558.13454667, 112788.1844256 , 113016.09308717, 113242.29529416,
       113467.13753309, 113690.89593825, 113913.7905763 , 114135.99683056,
       114357.6544737 , 114578.87489829, 114799.74687922, 115020.34116612,
       115240.7141433 , 115460.9107465 , 115680.96678754, 115900.9108068 ])
```

In [39]:

```
df_forecast = pd.DataFrame(model_fit.forecast(36)[0])
df_forecast.columns = ['Value']
df_forecast.head()
```

Out[39]: **Value**

0	103358.17
1	104564.84
2	105571.07
3	106417.56
4	107136.75

In [40]:

```
df_forecast = df_forecast.set_index(new_per)
df_forecast.head()
```

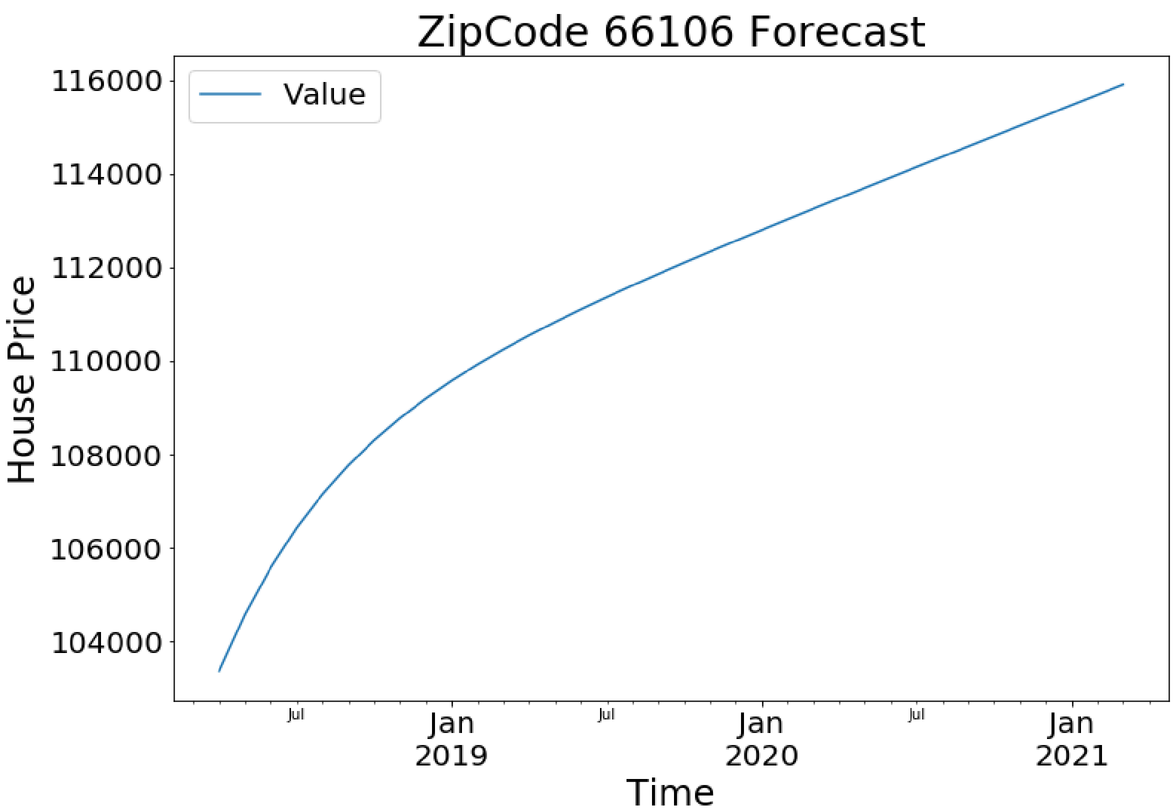
Out[40]:

	Value
2018-04-01	103358.17
2018-05-01	104564.84
2018-06-01	105571.07
2018-07-01	106417.56
2018-08-01	107136.75

We are going to plot our forecast.

In [41]:

```
df_forecast.plot(figsize=(12,8))
plt.title('ZipCode 66106 Forecast', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```



At this point we concatenate the actual values and the forecast.

In [42]:

```
zc_66106_forecast = pd.concat([zc_66106, df_forecast])
print(zc_66106_forecast.head())
```

```
print(zc_66106_forecast.head())
print(zc_66106_forecast.tail())
```

```
              Value     value
1996-04-01      nan  50600.00
1996-05-01      nan  50800.00
1996-06-01      nan  50900.00
1996-07-01      nan  51100.00
1996-08-01      nan  51200.00
                 Value   value
2020-11-01  115020.34     nan
2020-12-01  115240.71     nan
2021-01-01  115460.91     nan
2021-02-01  115680.97     nan
2021-03-01  115900.91     nan
```

In [43]:

```
zc_66106_forecast.rename(columns={"Value": "Forecast Value", "value": "Current Value"}, inpl
zc_66106_forecast.head()
```

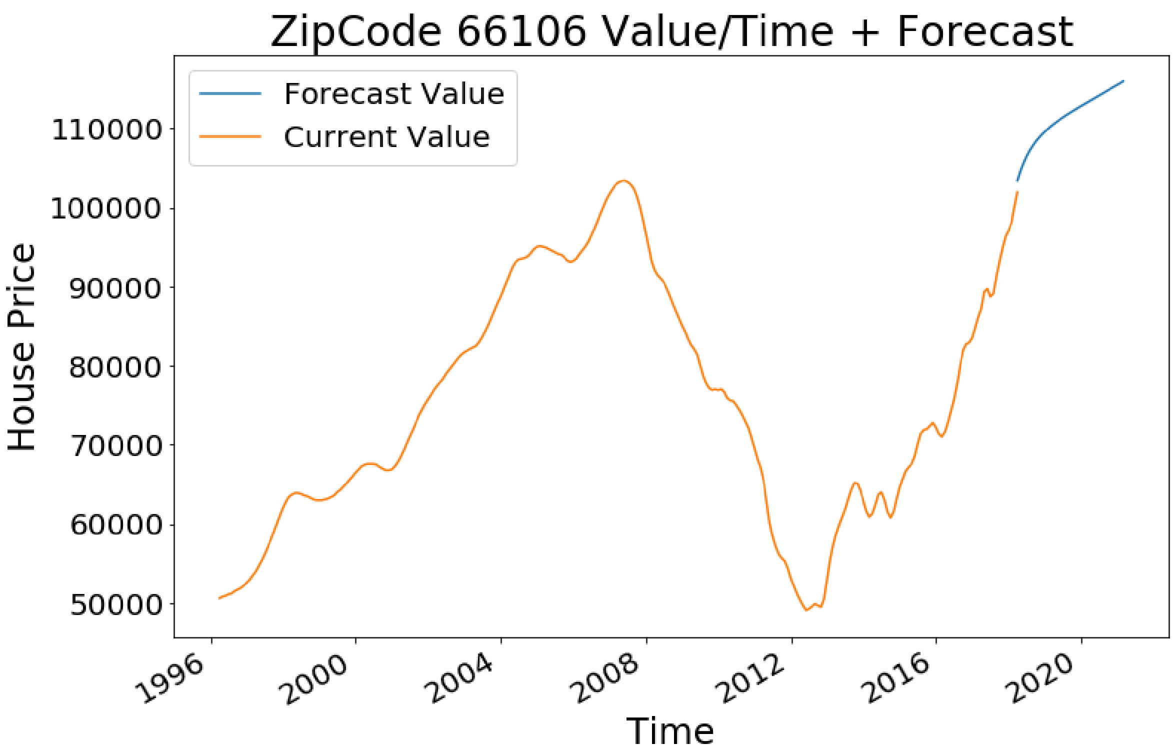
Out[43]:

	Forecast Value	Current Value
1996-04-01	nan	50600.00
1996-05-01	nan	50800.00
1996-06-01	nan	50900.00
1996-07-01	nan	51100.00
1996-08-01	nan	51200.00

Now we are going to plot our Value plus the forecast on the same graphic.

In [44]:

```
zc_66106_forecast.plot(figsize=(12,8))
plt.title('ZipCode 66106 Value/Time + Forecast', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```



Zipcode : 64114

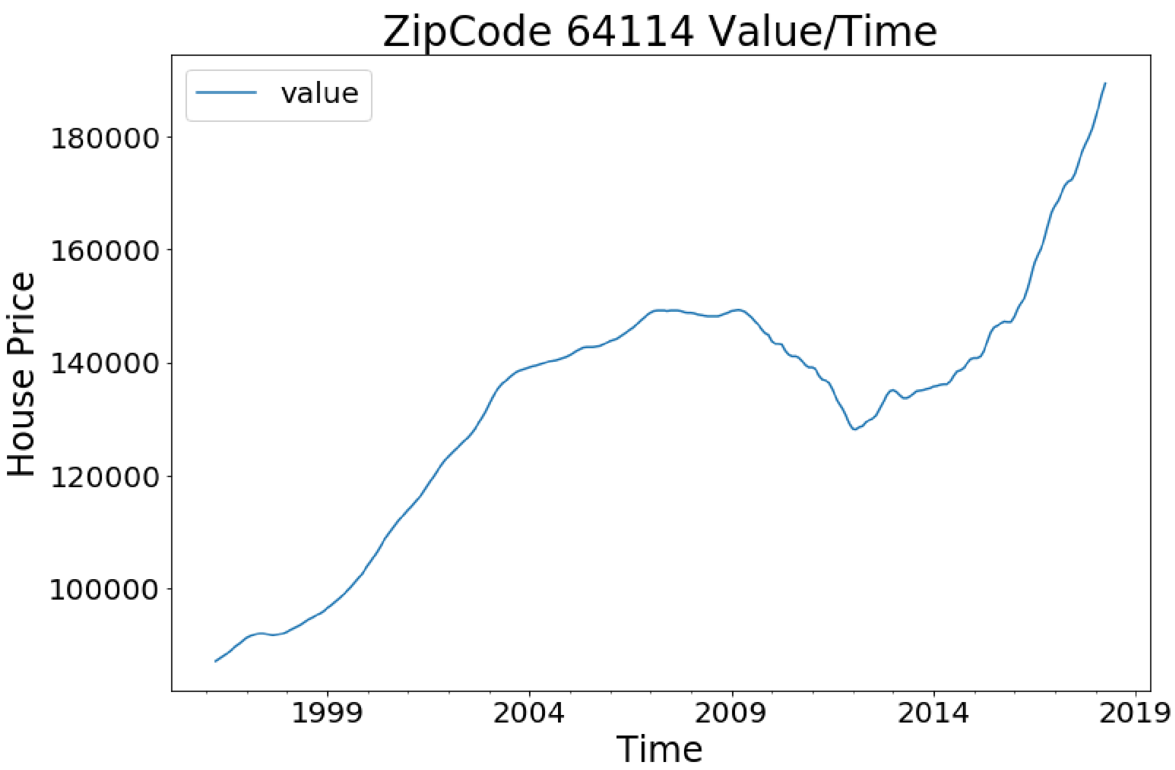
Location: Kansas City, MO

In [45]:
```
zc_64114 = df[df['RegionName'] == 64114]
zc_64114 = melt_data(zc_64114)
zc_64114.head()
```

Out[45]:

time	value
1996-04-01	87000.00
1996-05-01	87400.00
1996-06-01	87800.00
1996-07-01	88200.00
1996-08-01	88700.00

In [46]:
```
zc_64114.plot(figsize=(12,8))
plt.title('ZipCode 64114 Value/Time', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```



In [47]:
```
model = ARIMA(zc_64114, order = (1,1,1))

model_fit = model.fit(disp=0)

print(model_fit.summary())
```

```
                              ARIMA Model Results
==============================================================================
Dep. Variable:                D.value   No. Observations:                  264
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -1844.584
Method:                       css-mle   S.D. of innovations            260.939
```

```
Date:                Thu, 07 Nov 2019   AIC                          3697.168
Time:                        14:33:59   BIC                          3711.472
Sample:                    05-01-1996   HQIC                         3702.915
                         - 04-01-2018
==================================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
const            416.5372    135.790      3.068      0.002     150.394     682.680
ar.L1.D.value      0.8325      0.037     22.730      0.000       0.761       0.904
ma.L1.D.value      0.4421      0.053      8.319      0.000       0.338       0.546
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.2011           +0.0000j            1.2011            0.0000
MA.1           -2.2618           +0.0000j            2.2618            0.5000
-----------------------------------------------------------------------------
```

The model returns a lot of information, but we'll focus only on the table of coefficients.

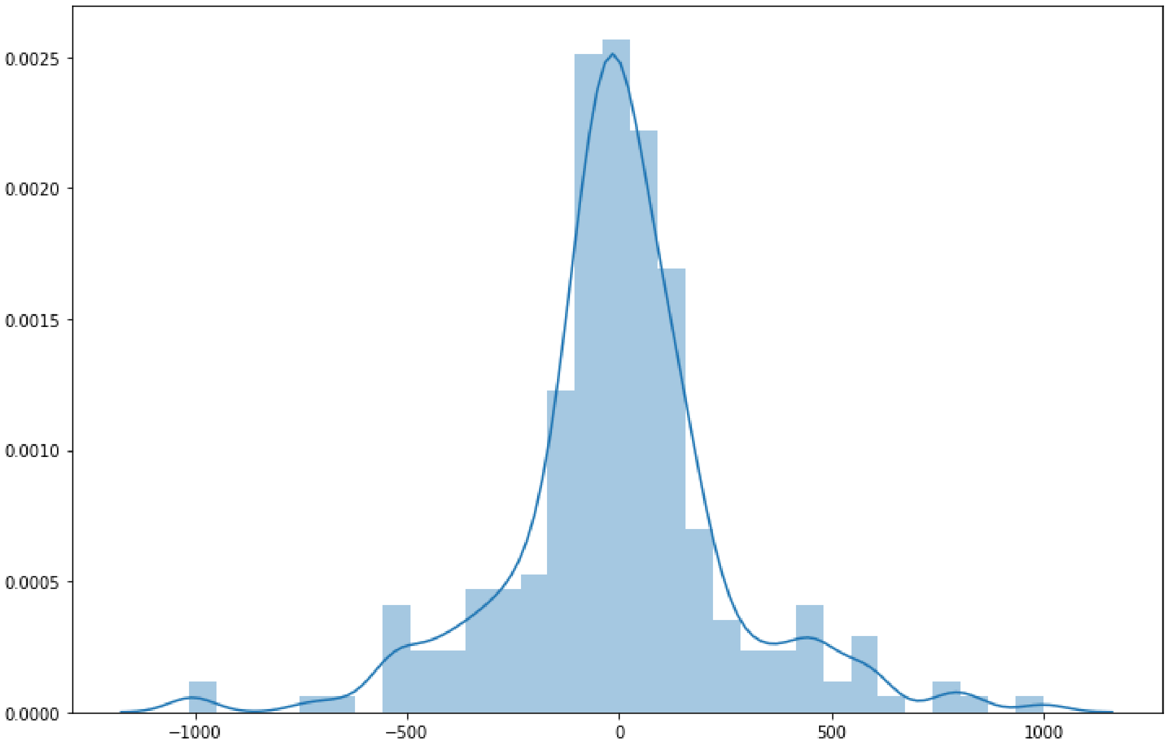
The coef column above shows the importance of each feature and how each one impacts the time series patterns.

The P>|z| provides the significance of each feature weight.

For our time-series, we see that each weight has a p-value lower than 0.05, so it is reasonable to retain all of them in our model.

In [48]:

```
residuals = pd.DataFrame(model_fit.resid)
plt.figure(figsize=(12,8))
sns.distplot(residuals);
```

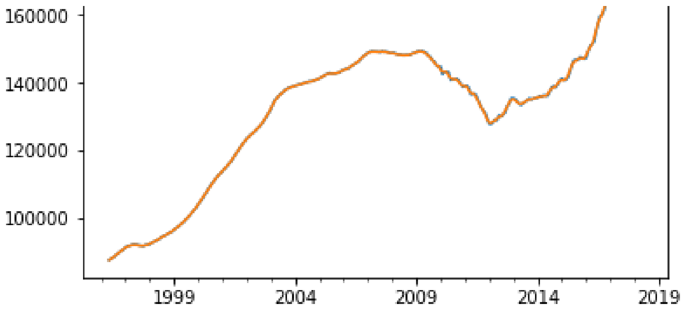


As we can see our residuals are relatively normal.

In [49]:

```
plt.figure(figsize=(12,8))
model_fit.plot_predict();
```

```
<Figure size 864x576 with 0 Axes>
```



Next step is to create a graphic with the actual value for the properties that we already have and the forecast. To do so we are first going to create a new DateTime Index starting 1 month after our actual DateTime and ending 36 months after.

In [50]:
```
new_per = pd.date_range(start='2018/04/01', periods=36, freq='MS')
new_per[:5]
```

Out[50]:
```
DatetimeIndex(['2018-04-01', '2018-05-01', '2018-06-01', '2018-07-01',
               '2018-08-01'],
              dtype='datetime64[ns]', freq='MS')
```

Here we can see all our values forecasted for the next 36 months.

In [51]:
```
model_fit.forecast(36)[0]
```

Out[51]:
```
array([191192.90444041, 192672.06624522, 193973.27912155, 195126.34270112,
       196156.06631841, 197083.10469432, 197924.65367491, 198695.02946043,
       199406.1508358 , 200067.94064582, 200688.66003874, 201275.18673683,
       201833.24670719, 202367.60703672, 202882.23650778, 203380.43928366,
       203864.9662068 , 204338.10745888, 204801.76970385, 205257.54031243,
       205706.74083155, 206150.47149961, 206589.64830722, 207025.03385164,
       207457.26302429, 207886.86439657, 208314.27802442, 208739.87027128,
       209163.9461487 , 209586.75959052, 210008.52200635, 210429.4094028 ,
       210849.56831211, 211269.12072799, 211688.16821486, 212106.795329  ])
```

In [52]:
```
df_forecast = pd.DataFrame(model_fit.forecast(36)[0])
df_forecast.columns = ['Value']
df_forecast.head()
```

Out[52]:

	Value
0	191192.90
1	192672.07
2	193973.28
3	195126.34
4	196156.07

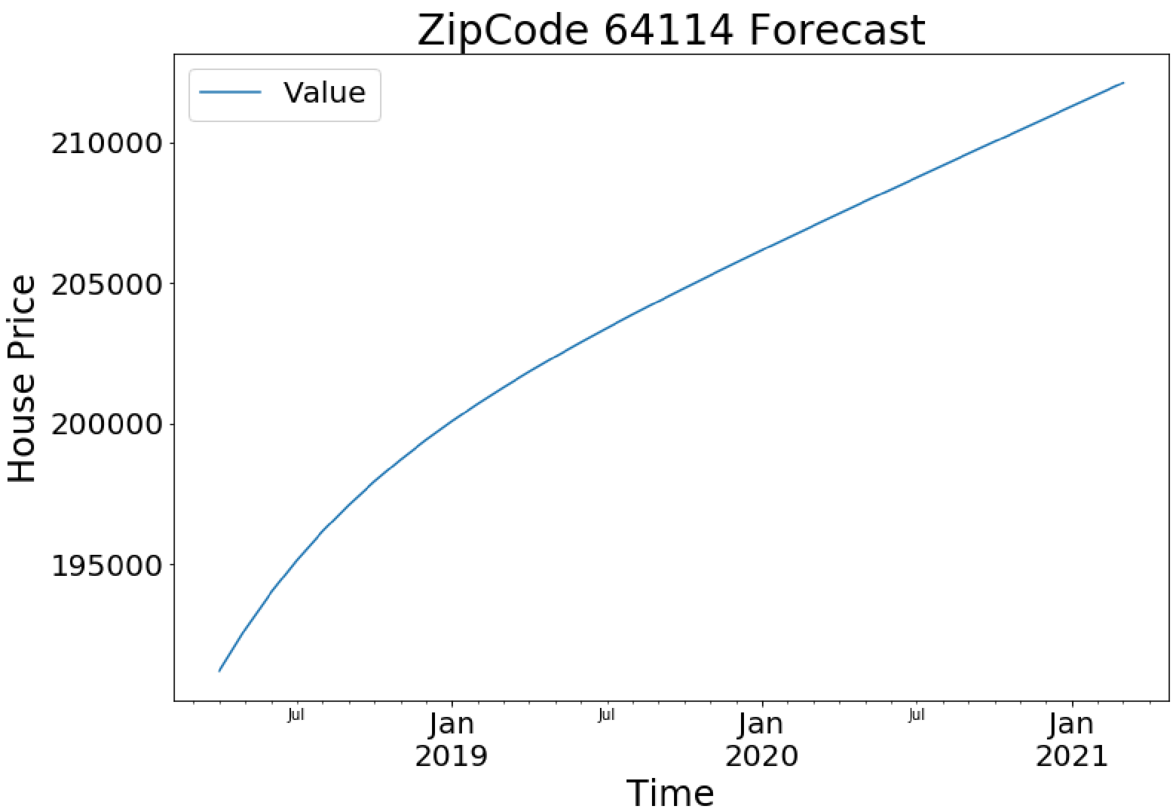
In [53]:
```
df_forecast = df_forecast.set_index(new_per)
df_forecast.head()
```

Out[53]:

	Value
2018-04-01	191192.90
2018-05-01	192672.07
2018-06-01	193973.28
2018-07-01	195126.34
2018-08-01	196156.07

We are going to plot our forecast.

In [54]:
```
df_forecast.plot(figsize=(12,8))
plt.title('ZipCode 64114 Forecast', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```



At this point we concatenate the actual values and the forecast.

In [55]:
```
zc_64114_forecast = pd.concat([zc_64114, df_forecast])
print(zc_64114_forecast.head())
print(zc_64114_forecast.tail())
```

```
              Value     value
1996-04-01      nan  87000.00
1996-05-01      nan  87400.00
1996-06-01      nan  87800.00
1996-07-01      nan  88200.00
1996-08-01      nan  88700.00
                 Value   value
2020-11-01  210429.41     nan
2020-12-01  210849.57     nan
2021-01-01  211269.12     nan
2021-02-01  211688.17     nan
2021-03-01  212106.80     nan
```

In [56]:
```
zc_64114_forecast.rename(columns={"Value": "Forecast Value", "value": "Current Value"}, inpl
zc_64114_forecast.head()
```

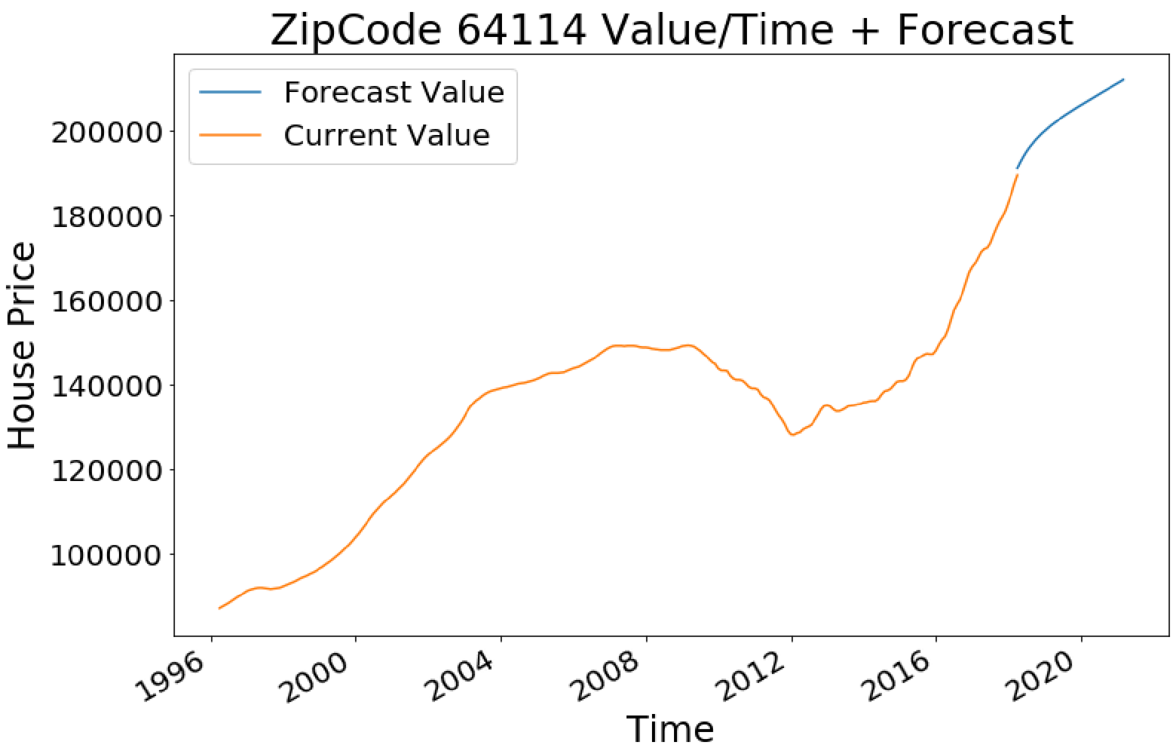
Out[56]:

	Forecast Value	Current Value
1996-04-01	nan	87000.00
1996-05-01	nan	87400.00
1996-06-01	nan	87800.00

1996-07-01	nan	88200.00
1996-08-01	nan	88700.00

Now we are going to plot our Value plus the forecast on the same graphic.

In [57]:
```
zc_64114_forecast.plot(figsize=(12,8))
plt.title('ZipCode 64114 Value/Time + Forecast', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```



Zipcode : 64112

Location: Kansas City, MO

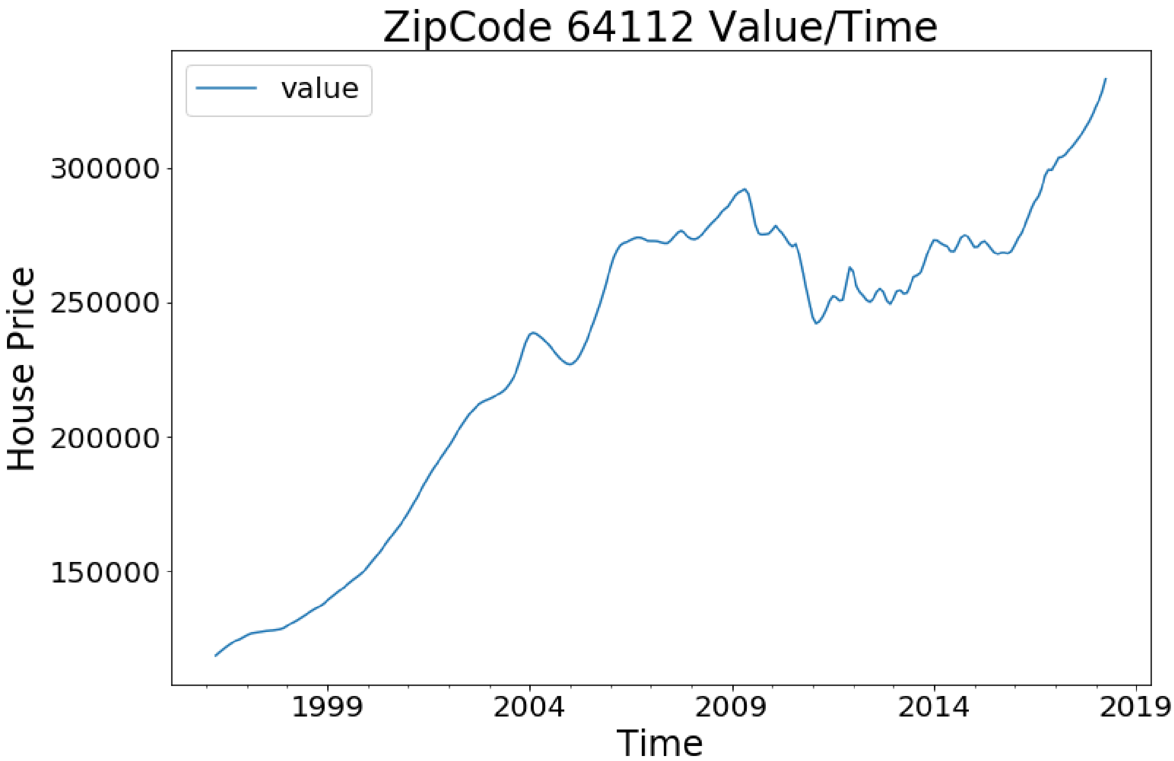
In [58]:
```
zc_64112 = df[df['RegionName'] == 64112]
zc_64112 = melt_data(zc_64112)
zc_64112.head()
```

Out[58]:

	value
time	
1996-04-01	118500.00
1996-05-01	119600.00
1996-06-01	120600.00
1996-07-01	121600.00
1996-08-01	122500.00

In [59]:
```
zc_64112.plot(figsize=(12,8))
plt.title('ZipCode 64112 Value/Time', fontsize=28)
```

```
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```



In [60]:

```
model = ARIMA(zc_64112, order = (2,1,1))

model_fit = model.fit(disp=0)

print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                D.value   No. Observations:                  264
Model:                 ARIMA(2, 1, 1)   Log Likelihood               -2185.783
Method:                       css-mle   S.D. of innovations            950.484
Date:                Thu, 07 Nov 2019   AIC                           4381.565
Time:                        14:34:00   BIC                           4399.445
Sample:                    05-01-1996   HQIC                          4388.750
                         - 04-01-2018
=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
const           830.6250    230.141      3.609      0.000     379.557    1281.693
ar.L1.D.value     0.8407      0.076     11.050      0.000       0.692       0.990
ar.L2.D.value    -0.2270      0.073     -3.101      0.002      -0.371      -0.084
ma.L1.D.value     0.5273      0.055      9.645      0.000       0.420       0.634
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.8517           -0.9882j            2.0988           -0.0780
AR.2            1.8517           +0.9882j            2.0988            0.0780
MA.1           -1.8965           +0.0000j            1.8965            0.5000
-----------------------------------------------------------------------------
```

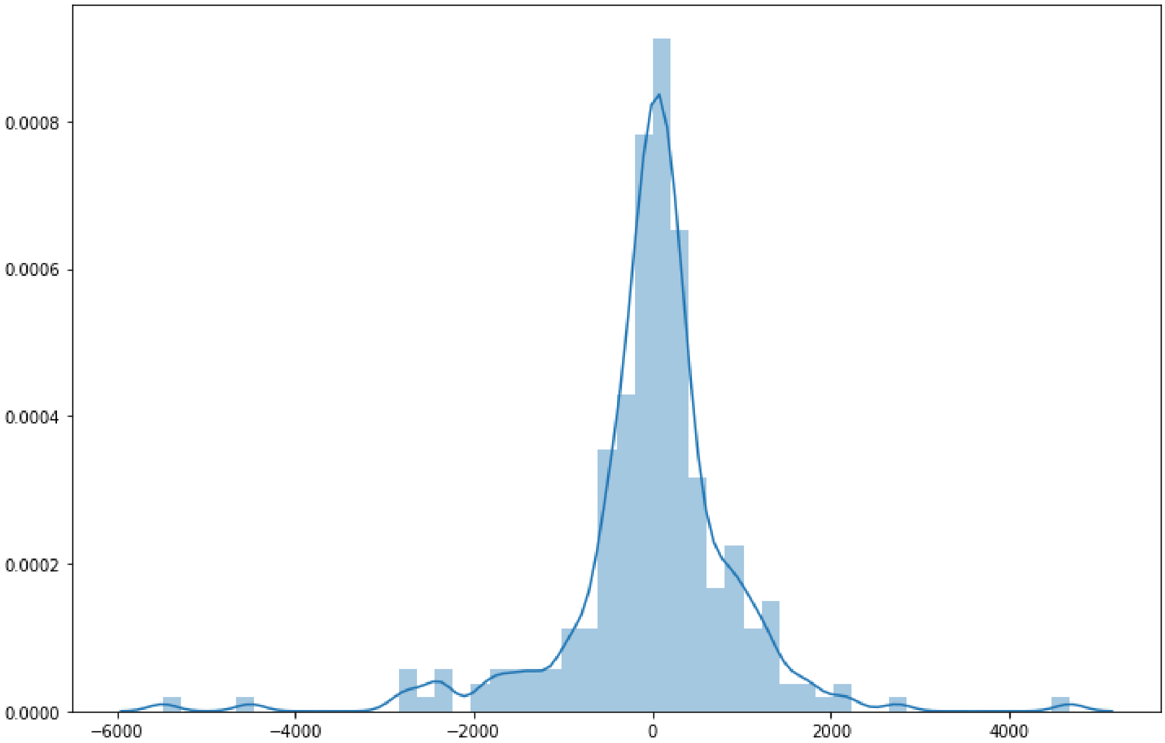
The model returns a lot of information, but we'll focus only on the table of coefficients.

The coef column above shows the importance of each feature and how each one impacts the time series patterns.

The P>|z| provides the significance of each feature weight.

For our time-series, we see that each weight has a p-value lower than 0.05, so it is reasonable to retain all of them in our model.

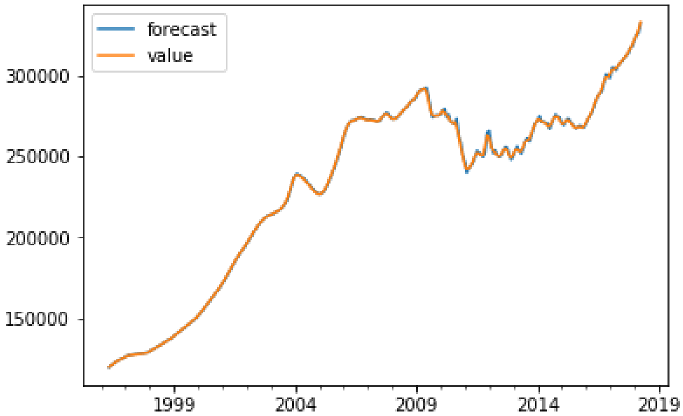
In [61]:
```
residuals = pd.DataFrame(model_fit.resid)
plt.figure(figsize=(12,8))
sns.distplot(residuals);
```



As we can see our residuals are relatively normal.

In [62]:
```
plt.figure(figsize=(12,8))
model_fit.plot_predict();
```

```
<Figure size 864x576 with 0 Axes>
```



Next step is to create a graphic with the actual value for the properties that we already have and the forecast. To do so we are first going to create a new DateTime Index starting 1 month after our actual DateTime and ending 36 months after.

In [63]:
```
new_per = pd.date_range(start='2018/04/01', periods=36, freq='MS')
new_per[:5]
```

Out[63]:
```
DatetimeIndex(['2018-04-01', '2018-05-01', '2018-06-01', '2018-07-01',
               '2018-08-01'],
              dtype='datetime64[ns]', freq='MS')
```

Here we can see all our values forecasted for the next 36 months.

In [64]:
```
model_fit.forecast(36)[0]
```

Out[64]:
```
array([336761.2940615 , 339200.01888399, 340717.26793375, 341760.07733417,
       342613.21368165, 343414.59597308, 344215.52665864, 345027.82631264,
       345849.78625211, 346677.28663114, 347507.25182662, 348338.03143743,
       349168.93618314, 349999.7612489 , 350830.49092187, 351661.15848722,
       352491.79549439, 353322.4209106 , 354153.04351941, 354983.66639931,
       355814.29014442, 356644.91455536, 357475.53932965, 358306.16425826,
       359136.7892341 , 359967.41421464, 360798.03918839, 361628.66415538,
       362459.28911822, 363289.9140791 , 364120.53903929, 364951.16399933,
       365781.78895941, 366612.41391955, 367443.03887974, 368273.66383995])
```

In [65]:
```
df_forecast = pd.DataFrame(model_fit.forecast(36)[0])
df_forecast.columns = ['Value']
df_forecast.head()
```

Out[65]:

	Value
0	336761.29
1	339200.02
2	340717.27
3	341760.08
4	342613.21

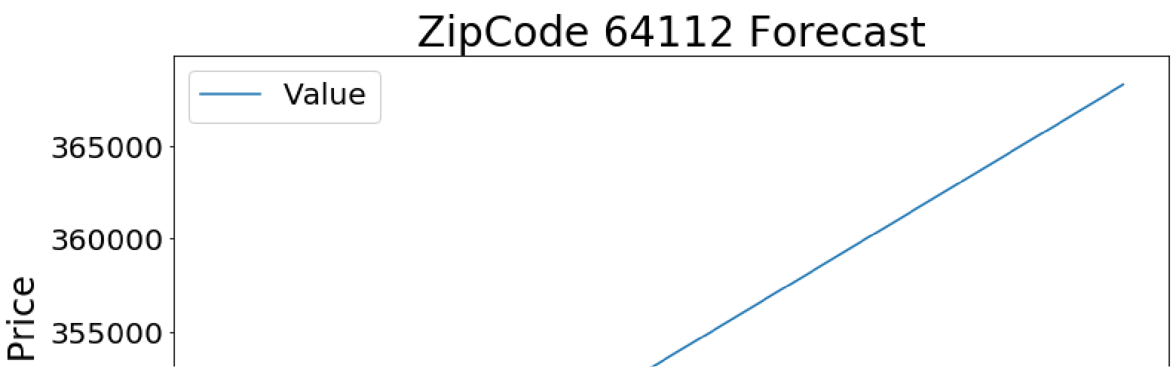
In [66]:
```
df_forecast = df_forecast.set_index(new_per)
df_forecast.head()
```

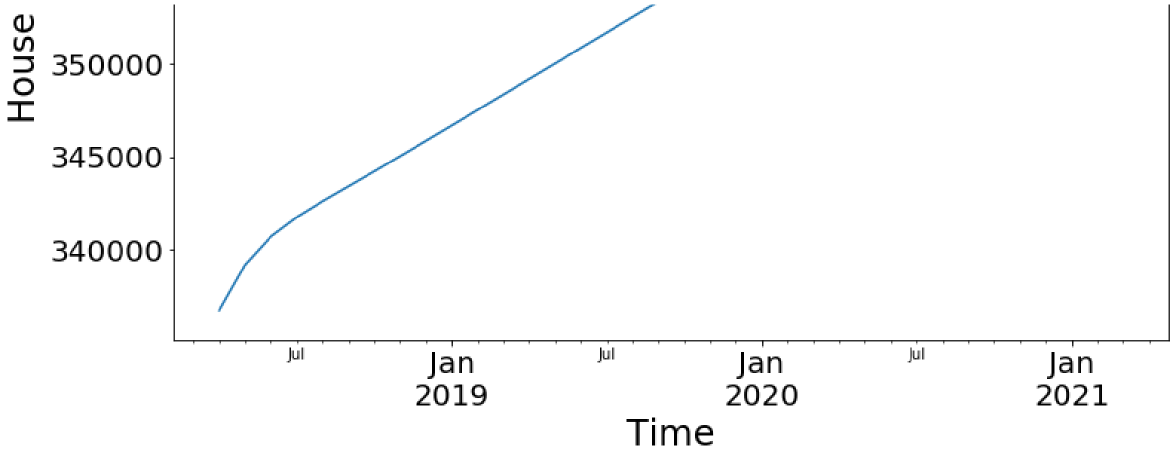
Out[66]:

	Value
2018-04-01	336761.29
2018-05-01	339200.02
2018-06-01	340717.27
2018-07-01	341760.08
2018-08-01	342613.21

We are going to plot our forecast.

In [67]:
```
df_forecast.plot(figsize=(12,8))
plt.title('ZipCode 64112 Forecast', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```





At this point we concatenate the actual values and the forecast.

In [68]:

```
zc_64112_forecast = pd.concat([zc_64112, df_forecast])
print(zc_64112_forecast.head())
print(zc_64112_forecast.tail())
```

```
              Value      value
1996-04-01      nan  118500.00
1996-05-01      nan  119600.00
1996-06-01      nan  120600.00
1996-07-01      nan  121600.00
1996-08-01      nan  122500.00
                 Value  value
2020-11-01  364951.16    nan
2020-12-01  365781.79    nan
2021-01-01  366612.41    nan
2021-02-01  367443.04    nan
2021-03-01  368273.66    nan
```

In [69]:

```
zc_64112_forecast.rename(columns={"Value": "Forecast Value", "value": "Current Value"}, inpl
zc_64112_forecast.head()
```

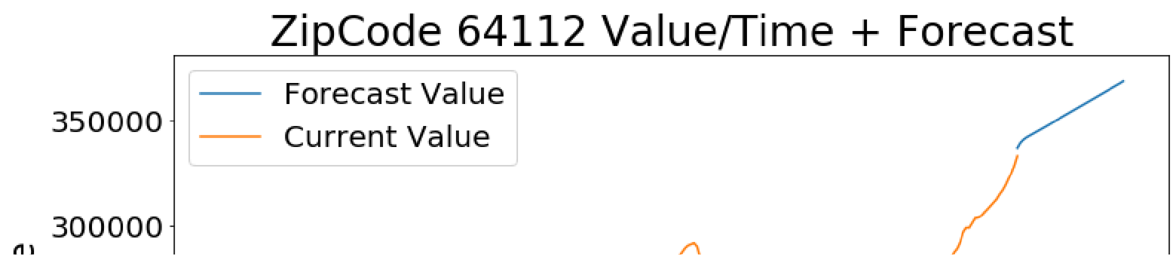
Out[69]:

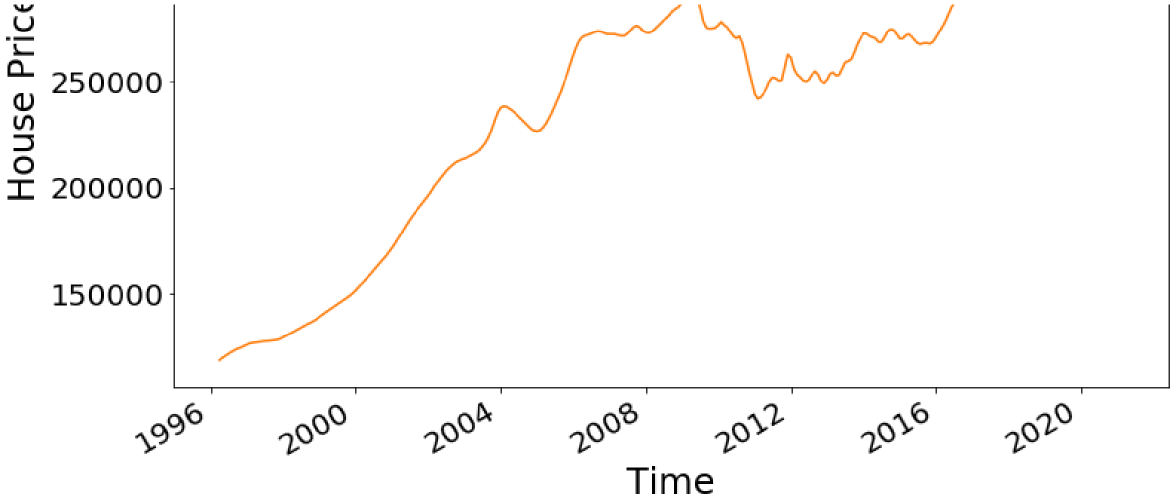
	Forecast Value	Current Value
1996-04-01	nan	118500.00
1996-05-01	nan	119600.00
1996-06-01	nan	120600.00
1996-07-01	nan	121600.00
1996-08-01	nan	122500.00

Now we are going to plot our Value plus the forecast on the same graphic.

In [70]:

```
zc_64112_forecast.plot(figsize=(12,8))
plt.title('ZipCode 64112 Value/Time + Forecast', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```





Zipcode : 66103

Location: Kansas City, KS

In [71]:

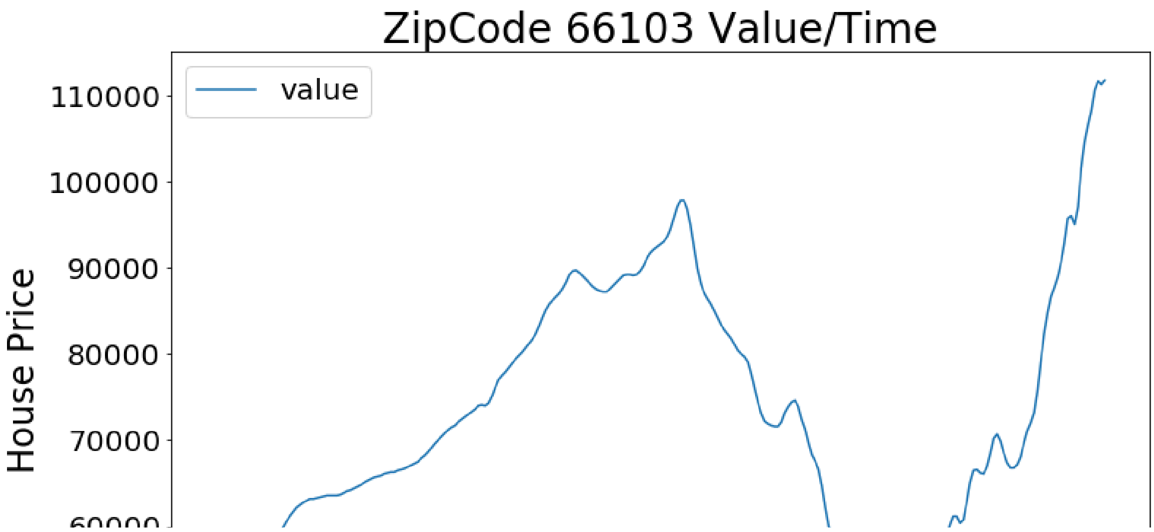
```
zc_66103 = df[df['RegionName'] == 66103]
zc_66103 = melt_data(zc_66103)
zc_66103.head()
```

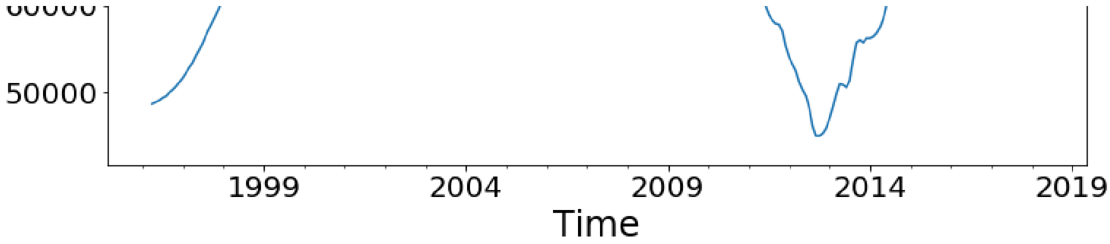
Out[71]:

time	value
1996-04-01	48600.00
1996-05-01	48800.00
1996-06-01	49000.00
1996-07-01	49300.00
1996-08-01	49500.00

In [72]:

```
zc_66103.plot(figsize=(12,8))
plt.title('ZipCode 66103 Value/Time', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```





In [73]:

```
model = ARIMA(zc_66103, order = (1,1,1))

model_fit = model.fit(disp=0)

print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                D.value   No. Observations:                  264
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -1992.198
Method:                       css-mle   S.D. of innovations            456.459
Date:                Thu, 07 Nov 2019   AIC                           3992.396
Time:                        14:34:01   BIC                           4006.700
Sample:                    05-01-1996   HQIC                          3998.144
                         - 04-01-2018
=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
const           248.7080    155.259      1.602      0.110     -55.593     553.009
ar.L1.D.value     0.7045      0.045     15.504      0.000       0.615       0.794
ma.L1.D.value     0.6498      0.040     16.287      0.000       0.572       0.728
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.4194           +0.0000j            1.4194            0.0000
MA.1           -1.5389           +0.0000j            1.5389            0.5000
-----------------------------------------------------------------------------
```

The model returns a lot of information, but we'll focus only on the table of coefficients.

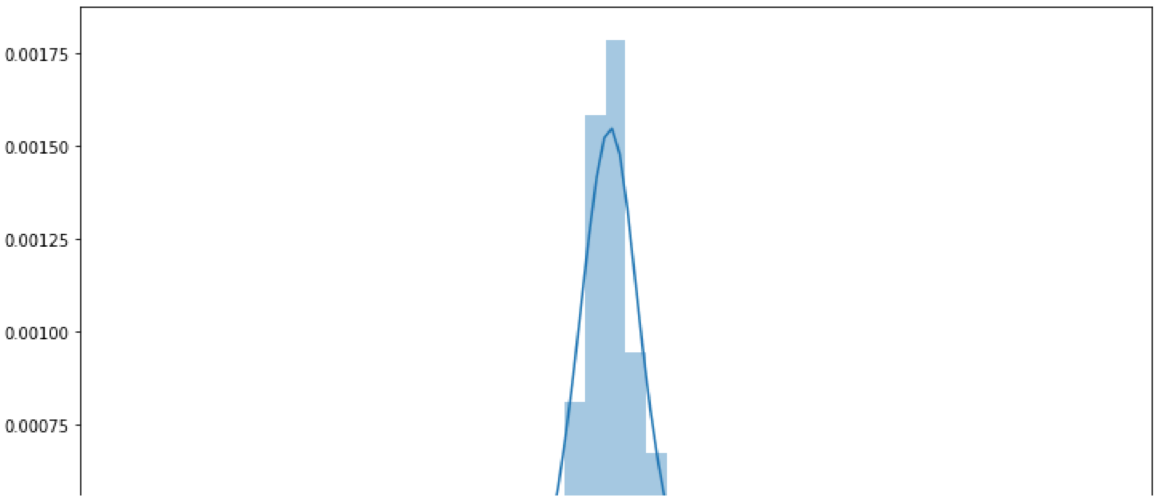
The coef column above shows the importance of each feature and how each one impacts the time series patterns.

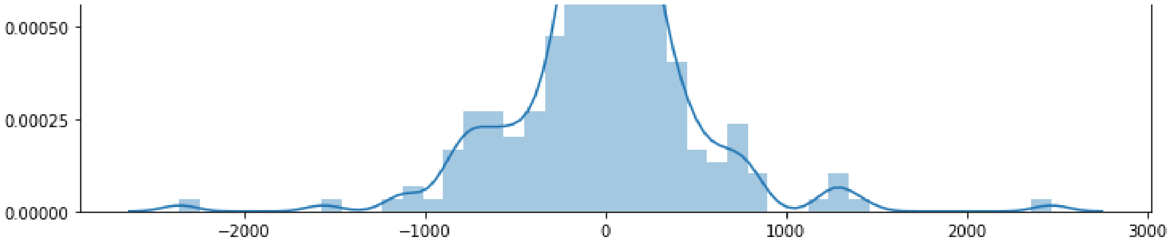
The P>|z| provides the significance of each feature weight.

For our time-series, we see that each weight has a p-value lower than 0.05, so it is reasonable to retain all of them in our model.

In [74]:

```
residuals = pd.DataFrame(model_fit.resid)
plt.figure(figsize=(12,8))
sns.distplot(residuals);
```

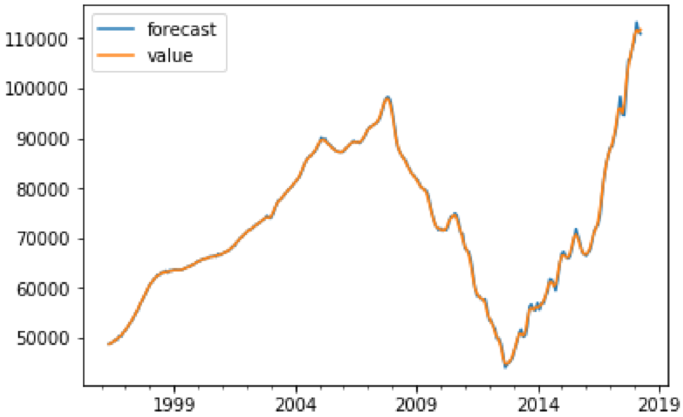




As we can see our residuals are relatively normal.

In [75]:
```
plt.figure(figsize=(12,8))
model_fit.plot_predict();
```

```
<Figure size 864x576 with 0 Axes>
```



Next step is to create a graphic with the actual value for the properties that we already have and the forecast. To do so we are first going to create a new DateTime Index starting 1 month after our actual DateTime and ending 36 months after.

In [76]:
```
new_per = pd.date_range(start='2018/04/01', periods=36, freq='MS')
new_per[:5]
```

Out[76]:
```
DatetimeIndex(['2018-04-01', '2018-05-01', '2018-06-01', '2018-07-01',
               '2018-08-01'],
              dtype='datetime64[ns]', freq='MS')
```

Here we can see all our values forecasted for the next 36 months.

In [77]:
```
model_fit.forecast(36)[0]
```

Out[77]:
```
array([112654.90439252, 113401.16151851, 114000.41666918, 114496.10189965,
       114918.81716935, 115290.12161278, 115625.20467101, 115934.76803151,
       116226.35154479, 116505.26739603, 116775.25827369, 117038.96108073,
       117298.23364137, 117554.38488159, 117808.33700235, 118060.73973829,
       118312.05085855, 118562.59288336, 118812.59304367, 119062.21143452,
       119311.56085054, 119560.72076092, 119809.74715553, 120058.67948189,
       120307.54553266, 120556.36488912, 120805.15134721, 121053.91462681,
       121302.66157607, 121551.39701982, 121800.12435738, 122048.84598375,
       122297.56358632, 122546.27835392, 122794.99112414, 123043.70248713])
```

In [78]:
```
df_forecast = pd.DataFrame(model_fit.forecast(36)[0])
df_forecast.columns = ['Value']
df_forecast.head()
```

Out[78]:

	Value
0	112654.90
1	113401.16
2	114000.42

3	114496.10
4	114918.82

In [79]:

```
df_forecast = df_forecast.set_index(new_per)
df_forecast.head()
```

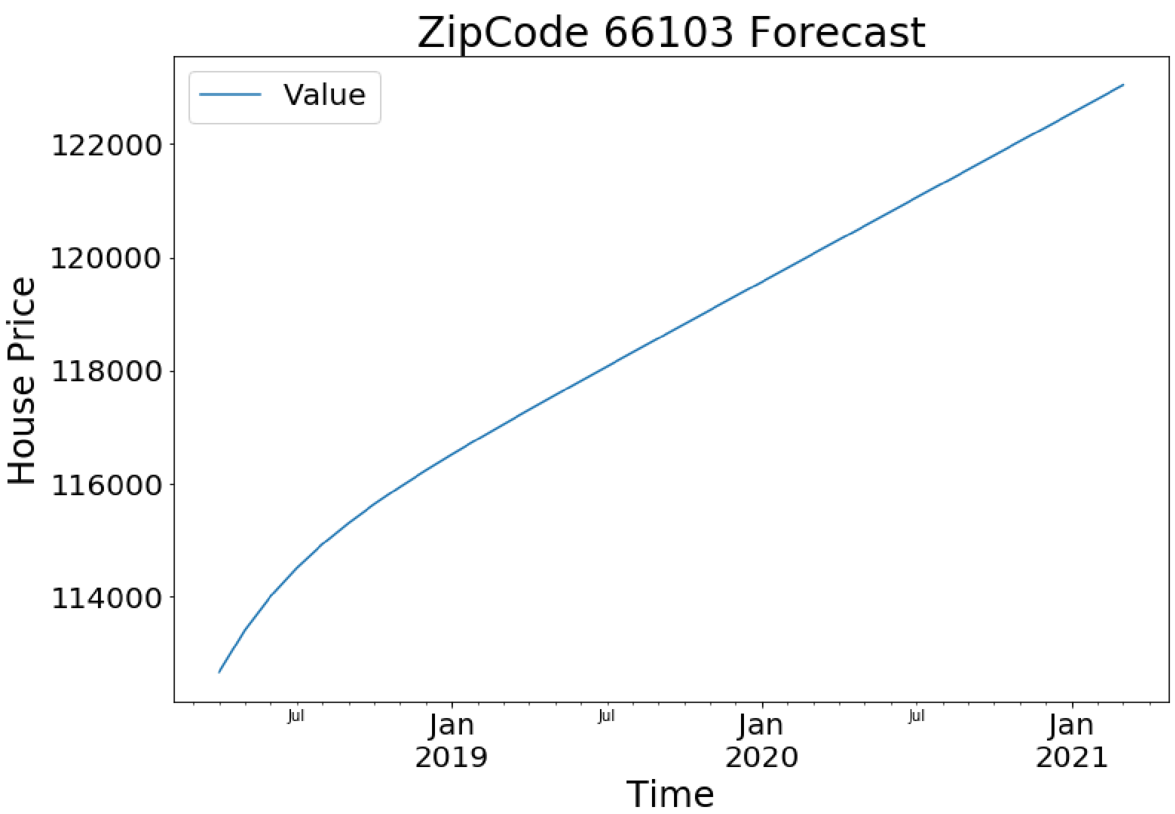
Out[79]:

	Value
2018-04-01	112654.90
2018-05-01	113401.16
2018-06-01	114000.42
2018-07-01	114496.10
2018-08-01	114918.82

We are going to plot our forecast.

In [80]:

```
df_forecast.plot(figsize=(12,8))
plt.title('ZipCode 66103 Forecast', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```



At this point we concatenate the actual values and the forecast.

In [81]:

```
zc_66103_forecast = pd.concat([zc_66103, df_forecast])
print(zc_66103_forecast.head())
print(zc_66103_forecast.tail())
```

```
            Value    value
1996-04-01    nan 48600.00
1996-05-01    nan 48800.00
```

```
1996-06-01    nan 49000.00
1996-07-01    nan 49300.00
1996-08-01    nan 49500.00
               Value   value
2020-11-01 122048.85    nan
2020-12-01 122297.56    nan
2021-01-01 122546.28    nan
2021-02-01 122794.99    nan
2021-03-01 123043.70    nan
```

In [82]:

```
zc_66103_forecast.rename(columns={"Value": "Forecast Value", "value": "Current Value"}, inpl
zc_66103_forecast.head()
```

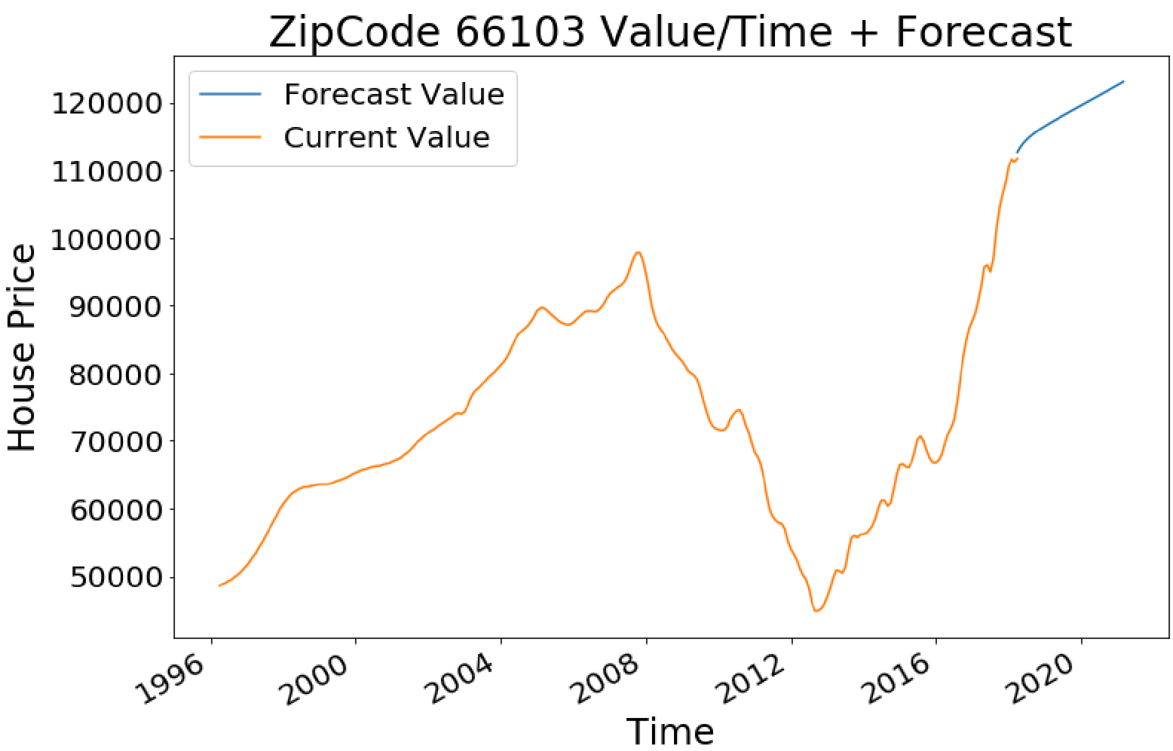
Out[82]:

	Forecast Value	Current Value
1996-04-01	nan	48600.00
1996-05-01	nan	48800.00
1996-06-01	nan	49000.00
1996-07-01	nan	49300.00
1996-08-01	nan	49500.00

Now we are going to plot our Value plus the forecast on the same graphic.

In [83]:

```
zc_66103_forecast.plot(figsize=(12,8))
plt.title('ZipCode 66103 Value/Time + Forecast', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```



Interpreting Results

Here are the 5 best Zip Codes to invest in Kansas City and the relative ROI for 3 years.

In [84]:

```
df_top_5['Value In 3 Years 1000000 Invested'] = (df_top_5['ROI']+1)*1000000
```

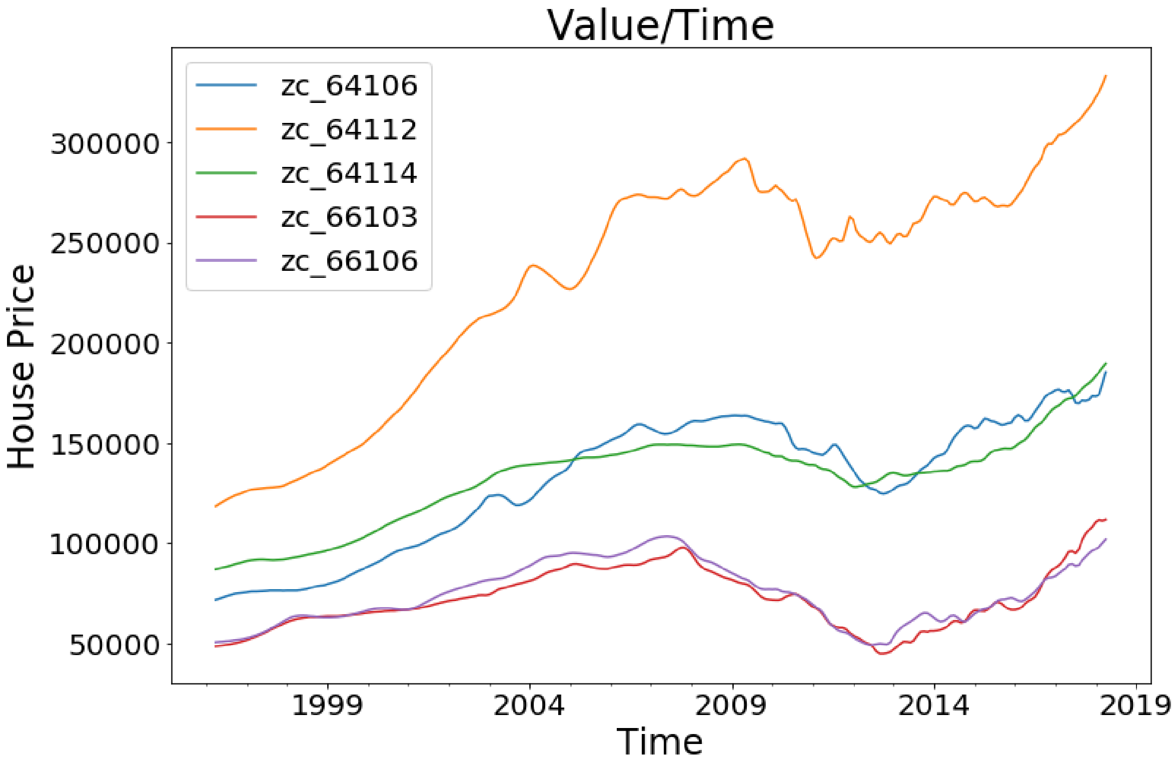
```
df_top_5
```

Out[84]:

	ZipCode	ROI	Value In 3 Years 1000000 Invested
0	64106	0.15	1147370.96
1	66106	0.14	1137398.54
2	64114	0.12	1119297.07
3	64112	0.11	1105926.92
4	66103	0.10	1101555.08

In [85]:

```
zips = pd.concat([zc_64106,zc_64112,zc_64114,zc_66103,zc_66106], axis=1)
zips.columns=['zc_64106','zc_64112','zc_64114','zc_66103','zc_66106']
zips.plot(figsize=(12,8))
plt.title('Value/Time', fontsize=28)
plt.xlabel('Time', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylabel('House Price', fontsize=24)
plt.legend(fontsize=20);
```



Findings

1. Our top 5 zipcodes in returns were Zipcode : 64106,66106, 64114, 64112,66103 in Kansas city.
2. Zipcode 64106 had the highest return on investment.
3. House prices outlook in Kansas City specifically shows an increasing trend, with the first few values being around $189,506.94 and rising over time.

Recommendations

1. Invest in Growth Areas: Focus on regions showing strong, consistent property value increases, such as Zip Code 66103. These areas demonstrate robust market conditions and potential for high return on investment (ROI).
2. Diversify Investments: Consider diversifying investments across multiple zip codes, including 66106

and 64112, to spread risk and capitalize on different market dynamics.
3. Monitor Market Trends: Keep an eye on long-term market trends, particularly the recovery and growth periods post-2012, to make informed investment decisions.
4. Consider Timing: Given the historical impact of economic downturns like in 2008, investors should be cautious about market timing and consider the broader economic indicators.

Conclusions

1. The Kansas City real estate market has experienced a significant recovery since the 2008 financial crisis, with a consistent upward trend in property values, especially post-2012.
2. The market shows a range of investment opportunities, from more affordable properties in areas like Zip Codes 66106 and 66103 to higher-end markets in areas like Zip Code 64112.
3. The analysis suggests a robust market with potential for growth, especially in certain zip codes identified as having strong upward price trends.
4. Investment strategies should be informed by historical market performance, current market conditions, and future market forecasts, considering the long-term upward trend in property prices and regional variances.

In [ ]: