INFO 6105

Ina Tayal NUID : 001494253

Prima Aranha NUID : 001425555

Sindhu Raghavendra NUID : 001490547

Abstract

Introduction

[What is Zillow? (https://www.zillow.com/corp/About.htm)](https://www.zillow.com/corp/About.htm)

Zillow is the leading real estate and rental marketplace dedicated to empowering consumers with data, inspiration and knowledge around the place they call home, and connecting them with the best local professionals who can help.

Zillow serves the full lifecycle of owning and living in a home: buying, selling, renting, financing, remodeling and more. It starts with Zillow's living database of more than 110 million U.S. homes - including homes for sale, homes for rent and homes not currently on the market, as well as Zestimate home values, Rent Zestimates and other home-related information. Zillow operates the most popular suite of mobile real estate apps, with more than two dozen apps across all major platforms.

What are we trying to forecast?

The real estate market has always been through lows and highs in terms of house prices.

In this project, we are forecasting the listing prices per sqft of homes. This forecasting will equip buyers with the knowledge of favorable periods throughout year during which they can get the best deals for houses.

Our Approach

We will be perfoming Time Series Analysis on our dataset using four different types of Machine Learning models - ARIMA, SARIMA, Facebook Prophet and a Hybrid ARIMA - ANN (Artificial Neural Networks) model.

What is Time Series Analysis?

Time Series Analysis is a statistical technique that deals with time series data, or trend analysis. Time series data means that data is in a series of particular time periods or intervals. Time series analysis helps us understand time based patterns of a set of data points which can be used to forecast and make decisions based on the findings.

Dataset

The dataset we will be using in our project is a time series dataset which contains a set of observations with different values that a variable has at different times.

We are using time series dataset pertaining to the real estate market in the fifty states in the USA.

In [1]:
```
from IPython.display import Image
Image(filename='Home-sales-forecast.jpg')
```

...

Analysis

In [2]:
```
# importing libraries
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy import stats
import statsmodels.api as sm
import seaborn as sns
import matplotlib.ticker as tick
import squarify
from random import sample
import folium
from folium import plugins
from folium.plugins import HeatMap
import plotly
import plotly.plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs,init_notebook_mode,plot,iplot
plotly.tools.set_credentials_file(username='ina19', api_key='aVYYQRBtSqq6uuHk
```

Reading the State Time Series dataset

In [3]:
```
df = pd.read_csv("State_time_series.csv")
```

In [4]:
```
# Returns tuple of shape (Rows, columns) of dataframe/series
df.shape
```

Out[4]: (13026, 86)

Checking for null values in the dataset

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
Date                                                        0
RegionName                                                  0
DaysOnZillow_AllHomes                                    8568
HomesSoldAsForeclosuresRatio_AllHomes                    6325
InventorySeasonallyAdjusted_AllHomes                     8334
InventoryRaw_AllHomes                                    8334
MedianListingPricePerSqft_1Bedroom                       9629
MedianListingPricePerSqft_2Bedroom                       8647
MedianListingPricePerSqft_3Bedroom                       8618
MedianListingPricePerSqft_4Bedroom                       8548
MedianListingPricePerSqft_5BedroomOrMore                 8619
MedianListingPricePerSqft_AllHomes                       8583
MedianListingPricePerSqft_CondoCoop                      8922
MedianListingPricePerSqft_DuplexTriplex                  9623
MedianListingPricePerSqft_SingleFamilyResidence          8618
MedianListingPrice_1Bedroom                             10025
MedianListingPrice_2Bedroom                              8749
MedianListingPrice_3Bedroom                              8759
MedianListingPrice_4Bedroom                              8770
MedianListingPrice_5BedroomOrMore                        8963
MedianListingPrice_AllHomes                              9066
MedianListingPrice_CondoCoop                             9094
MedianListingPrice_DuplexTriplex                         9968
MedianListingPrice_SingleFamilyResidence                 9097
MedianPctOfPriceReduction_AllHomes                       8742
MedianPctOfPriceReduction_CondoCoop                      9078
MedianPctOfPriceReduction_SingleFamilyResidence          8742
MedianPriceCutDollar_AllHomes                            8742
MedianPriceCutDollar_CondoCoop                           9078
MedianPriceCutDollar_SingleFamilyResidence               8742
                                                        ...
ZHVIPerSqft_AllHomes                                      784
PctOfHomesDecreasingInValues_AllHomes                    4467
PctOfHomesIncreasingInValues_AllHomes                    4467
PctOfHomesSellingForGain_AllHomes                        4843
PctOfHomesSellingForLoss_AllHomes                        4868
PctOfListingsWithPriceReductionsSeasAdj_AllHomes         8742
PctOfListingsWithPriceReductionsSeasAdj_CondoCoop        8994
PctOfListingsWithPriceReductionsSeasAdj_SingleFamilyResidence       8742
PctOfListingsWithPriceReductions_AllHomes                8742
PctOfListingsWithPriceReductions_CondoCoop               8994
PctOfListingsWithPriceReductions_SingleFamilyResidence   8742
PctTransactionsThatArePreviouslyForeclosuredHomes_AllHomes    6526
PriceToRentRatio_AllHomes                                8999
Turnover_AllHomes                                        4566
ZHVI_1bedroom                                            2496
ZHVI_2bedroom                                            1372
ZHVI_3bedroom                                             603
ZHVI_4bedroom                                             863
ZHVI_5BedroomOrMore                                      1564
ZHVI_AllHomes                                            1276
ZHVI_BottomTier                                          1172
ZHVI_CondoCoop                                           1606
ZHVI_MiddleTier                                          1276
```

```
ZHVI_SingleFamilyResidence                                        1276
ZHVI_TopTier                                                       702
ZRI_AllHomes                                                      8976
ZRI_AllHomesPlusMultifamily                                       8976
ZriPerSqft_AllHomes                                               8894
Zri_MultiFamilyResidenceRental                                    8976
Zri_SingleFamilyResidenceRental                                   8976
Length: 86, dtype: int64
```

In [6]:
```
#print full summary of the dataframe.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13026 entries, 0 to 13025
Data columns (total 86 columns):
Date                                                                  13026 non-
null object
RegionName                                                            13026 non-
null object
DaysOnZillow_AllHomes                                                 4458 non-n
ull float64
HomesSoldAsForeclosuresRatio_AllHomes                                 6701 non-n
ull float64
InventorySeasonallyAdjusted_AllHomes                                  4692 non-n
ull float64
InventoryRaw_AllHomes                                                 4692 non-n
ull float64
MedianListingPricePerSqft_1Bedroom                                    3397 non-n
ull float64
MedianListingPricePerSqft_2Bedroom                                    4379 non-n
ull float64
MedianListingPricePerSqft_3Bedroom                                    4408 non-n
ull float64
MedianListingPricePerSqft_4Bedroom                                    4478 non-n
ull float64
MedianListingPricePerSqft_5BedroomOrMore                              4407 non-n
ull float64
MedianListingPricePerSqft_AllHomes                                    4443 non-n
ull float64
MedianListingPricePerSqft_CondoCoop                                   4104 non-n
ull float64
MedianListingPricePerSqft_DuplexTriplex                               3403 non-n
ull float64
MedianListingPricePerSqft_SingleFamilyResidence                       4408 non-n
ull float64
MedianListingPrice_1Bedroom                                           3001 non-n
ull float64
MedianListingPrice_2Bedroom                                           4277 non-n
ull float64
MedianListingPrice_3Bedroom                                           4267 non-n
ull float64
MedianListingPrice_4Bedroom                                           4256 non-n
ull float64
MedianListingPrice_5BedroomOrMore                                     4063 non-n
ull float64
MedianListingPrice_AllHomes                                           3960 non-n
ull float64
MedianListingPrice_CondoCoop                                          3932 non-n
ull float64
MedianListingPrice_DuplexTriplex                                      3058 non-n
ull float64
MedianListingPrice_SingleFamilyResidence                              3929 non-n
ull float64
MedianPctOfPriceReduction_AllHomes                                    4284 non-n
ull float64
```

```
MedianPctOfPriceReduction_CondoCoop                                     3948 non-n
ull float64
MedianPctOfPriceReduction_SingleFamilyResidence                         4284 non-n
ull float64
MedianPriceCutDollar_AllHomes                                           4284 non-n
ull float64
MedianPriceCutDollar_CondoCoop                                          3948 non-n
ull float64
MedianPriceCutDollar_SingleFamilyResidence                              4284 non-n
ull float64
MedianRentalPricePerSqft_1Bedroom                                       3322 non-n
ull float64
MedianRentalPricePerSqft_2Bedroom                                       3969 non-n
ull float64
MedianRentalPricePerSqft_3Bedroom                                       3971 non-n
ull float64
MedianRentalPricePerSqft_4Bedroom                                       3428 non-n
ull float64
MedianRentalPricePerSqft_5BedroomOrMore                                 1498 non-n
ull float64
MedianRentalPricePerSqft_AllHomes                                       4150 non-n
ull float64
MedianRentalPricePerSqft_CondoCoop                                      3225 non-n
ull float64
MedianRentalPricePerSqft_DuplexTriplex                                  2643 non-n
ull float64
MedianRentalPricePerSqft_MultiFamilyResidence5PlusUnits                 3847 non-n
ull float64
MedianRentalPricePerSqft_SingleFamilyResidence                          4032 non-n
ull float64
MedianRentalPricePerSqft_Studio                                         1002 non-n
ull float64
MedianRentalPrice_1Bedroom                                              3477 non-n
ull float64
MedianRentalPrice_2Bedroom                                              3927 non-n
ull float64
MedianRentalPrice_3Bedroom                                              4022 non-n
ull float64
MedianRentalPrice_4Bedroom                                              3107 non-n
ull float64
MedianRentalPrice_5BedroomOrMore                                        1476 non-n
ull float64
MedianRentalPrice_AllHomes                                              3966 non-n
ull float64
MedianRentalPrice_CondoCoop                                             2643 non-n
ull float64
MedianRentalPrice_DuplexTriplex                                         3315 non-n
ull float64
MedianRentalPrice_MultiFamilyResidence5PlusUnits                        3953 non-n
ull float64
MedianRentalPrice_SingleFamilyResidence                                 3993 non-n
ull float64
MedianRentalPrice_Studio                                                1383 non-n
ull float64
MedianSoldPricePerSqft_AllHomes                                         8372 non-n
ull float64
MedianSoldPricePerSqft_CondoCoop                                        7778 non-n
```

```
ull float64
MedianSoldPricePerSqft_SingleFamilyResidence                     8363 non-n
ull float64
MedianSoldPrice_AllHomes                                         8410 non-n
ull float64
ZHVIPerSqft_AllHomes                                             12242 non-
null float64
PctOfHomesDecreasingInValues_AllHomes                            8559 non-n
ull float64
PctOfHomesIncreasingInValues_AllHomes                            8559 non-n
ull float64
PctOfHomesSellingForGain_AllHomes                                8183 non-n
ull float64
PctOfHomesSellingForLoss_AllHomes                                8158 non-n
ull float64
PctOfListingsWithPriceReductionsSeasAdj_AllHomes                 4284 non-n
ull float64
PctOfListingsWithPriceReductionsSeasAdj_CondoCoop                4032 non-n
ull float64
PctOfListingsWithPriceReductionsSeasAdj_SingleFamilyResidence    4284 non-n
ull float64
PctOfListingsWithPriceReductions_AllHomes                        4284 non-n
ull float64
PctOfListingsWithPriceReductions_CondoCoop                       4032 non-n
ull float64
PctOfListingsWithPriceReductions_SingleFamilyResidence           4284 non-n
ull float64
PctTransactionsThatArePreviouslyForeclosuredHomes_AllHomes       6500 non-n
ull float64
PriceToRentRatio_AllHomes                                        4027 non-n
ull float64
Turnover_AllHomes                                                8460 non-n
ull float64
ZHVI_1bedroom                                                    10530 non-
null float64
ZHVI_2bedroom                                                    11654 non-
null float64
ZHVI_3bedroom                                                    12423 non-
null float64
ZHVI_4bedroom                                                    12163 non-
null float64
ZHVI_5BedroomOrMore                                              11462 non-
null float64
ZHVI_AllHomes                                                    11750 non-
null float64
ZHVI_BottomTier                                                  11854 non-
null float64
ZHVI_CondoCoop                                                   11420 non-
null float64
ZHVI_MiddleTier                                                  11750 non-
null float64
ZHVI_SingleFamilyResidence                                       11750 non-
null float64
ZHVI_TopTier                                                     12324 non-
null float64
ZRI_AllHomes                                                     4050 non-n
ull float64
```

```
ZRI_AllHomesPlusMultifamily                                    4050 non-n
ull float64
ZriPerSqft_AllHomes                                            4132 non-n
ull float64
Zri_MultiFamilyResidenceRental                                 4050 non-n
ull float64
Zri_SingleFamilyResidenceRental                                4050 non-n
ull float64
dtypes: float64(84), object(2)
memory usage: 8.5+ MB
```

Displaying the top five rows

In [7]:

```
df.head()
```

Out[7]:

	Date	RegionName	DaysOnZillow_AllHomes	HomesSoldAsForeclosuresRatio_AllHomes	Inve
0	1996-04-30	Alabama	NaN	NaN	
1	1996-04-30	Arizona	NaN	NaN	
2	1996-04-30	Arkansas	NaN	NaN	
3	1996-04-30	California	NaN	NaN	
4	1996-04-30	Colorado	NaN	NaN	

5 rows × 86 columns

Displaying the bottom five rows

In [8]:
```
df.tail()
```

Out[8]:

	Date	RegionName	DaysOnZillow_AllHomes	HomesSoldAsForeclosuresRatio_AllHomes
13021	2017-08-31	Virginia	NaN	2.6026
13022	2017-08-31	Washington	NaN	2.0805
13023	2017-08-31	WestVirginia	NaN	0.3122
13024	2017-08-31	Wisconsin	NaN	0.9433
13025	2017-08-31	Wyoming	NaN	NaN

5 rows × 86 columns

Displaying the date range in the dataset and number of states in the USA

In [9]:
```
print('Date range:{} to {}'.format(df['Date'].min(),df['Date'].max()))
print('Number of States',df['RegionName'].nunique())
```

```
Date range:1996-04-30 to 2017-08-31
Number of States 52
```

Transforming the date to datetime format

In [10]:
```
df.Date = pd.to_datetime(df.Date)
```

Gathering data for year 2010 and onwards

In [11]:
```
df = df[df['Date'] >= '01-01-2010']
print('Date range:{} to {}'.format(df['Date'].min(),df['Date'].max()))
print('Number of States',df['RegionName'].nunique())
```

```
Date range:2010-01-31 00:00:00 to 2017-08-31 00:00:00
Number of States 52
```

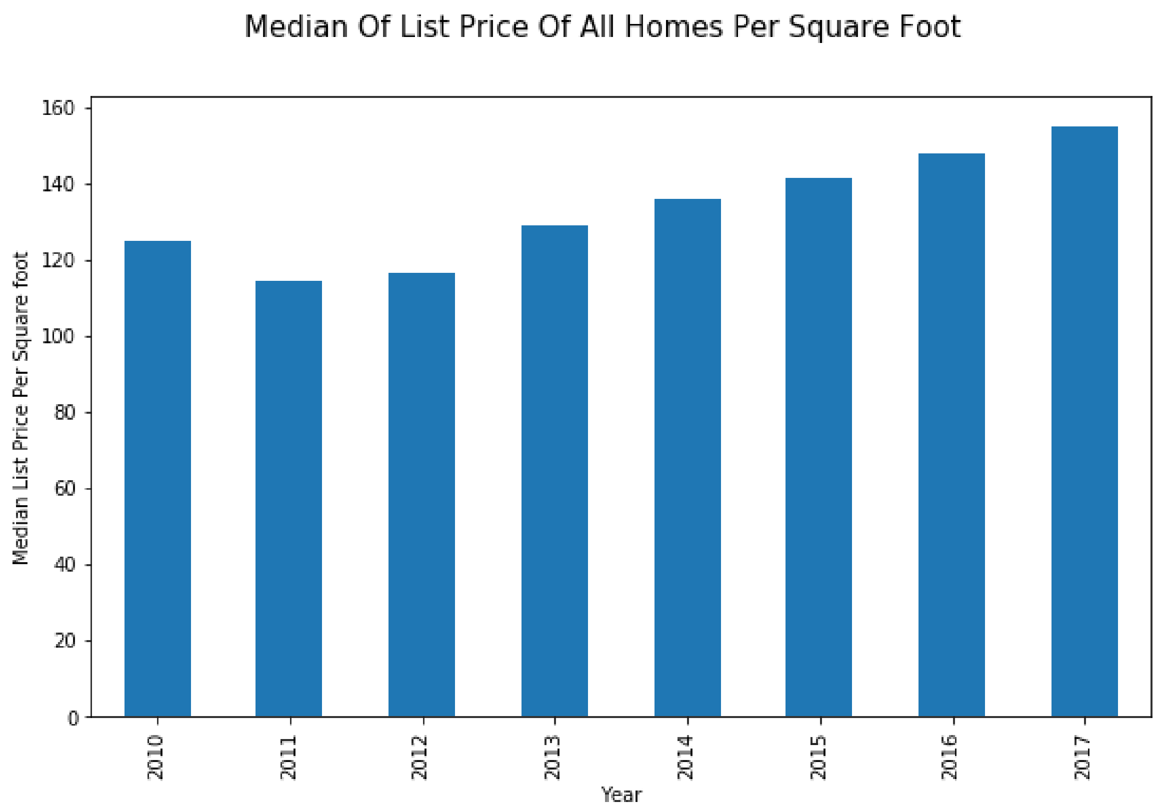
Displaying the geographical location of the data

In [12]:
```
m = folium.Map(location=[38,-10], tiles="Mapbox Bright", zoom_start=1.5)
folium.Circle(
      location=[40, -106],
      popup='North America',
      radius=3000000,
      color='crimson',
      fill=True,
      fill_color='crimson'
   ).add_to(m)
m
```

Out[12]:

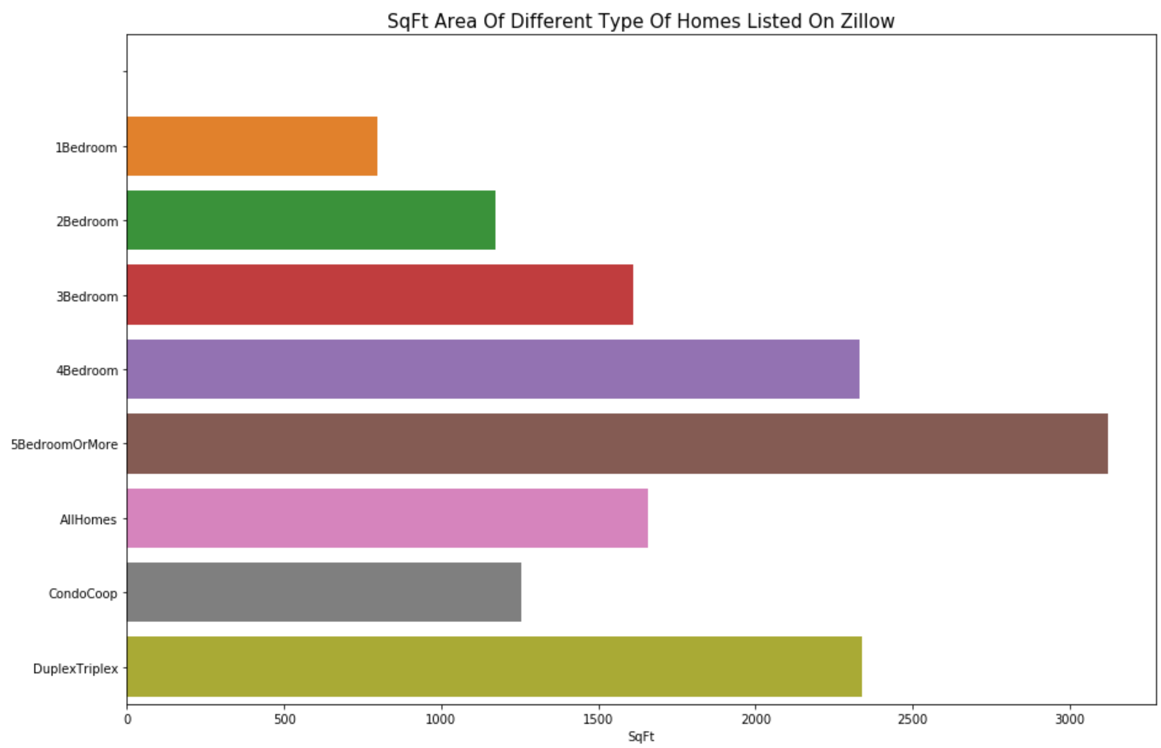
Median Listing Price Per Sqft Of All Homes In the United States Over the Past 8 Years

In [13]:
```
df1 = df.dropna(subset=['MedianListingPricePerSqft_AllHomes'], how='any')
df1.groupby(df1.Date.dt.year)['MedianListingPricePerSqft_AllHomes'].mean().pl
plt.suptitle('Median Of List Price Of All Homes Per Square Foot', fontsize=15
plt.ylabel('Median List Price Per Square foot')
plt.xlabel('Year')
plt.show()
```



SqFt Area of Different Types of Homes Listed on Zillow

In [14]:
```
flat_type = []
flat_area = []
v1_features = df.iloc[:,14:23].columns
v2_features = df.iloc[:,5:14].columns
f, ax = plt.subplots(figsize=(15, 10))
for cn1, cn2 in zip(v1_features,v2_features):
    area = (df[cn1].dropna().mean()/df[cn2].dropna().mean())
    flat_type.append(str(cn2)[26:])
    flat_area.append(area)
g = sns.barplot( y = flat_type,
                x = flat_area)
ax.set_xlabel('SqFt')
plt.title("SqFt Area Of Different Type Of Homes Listed On Zillow", fontsize =
plt.show()
```



Grouping data into monthly time intervals

In [15]:
```
state_month = df.resample('M',on='Date').mean()
state_month = state_month.reset_index()
```

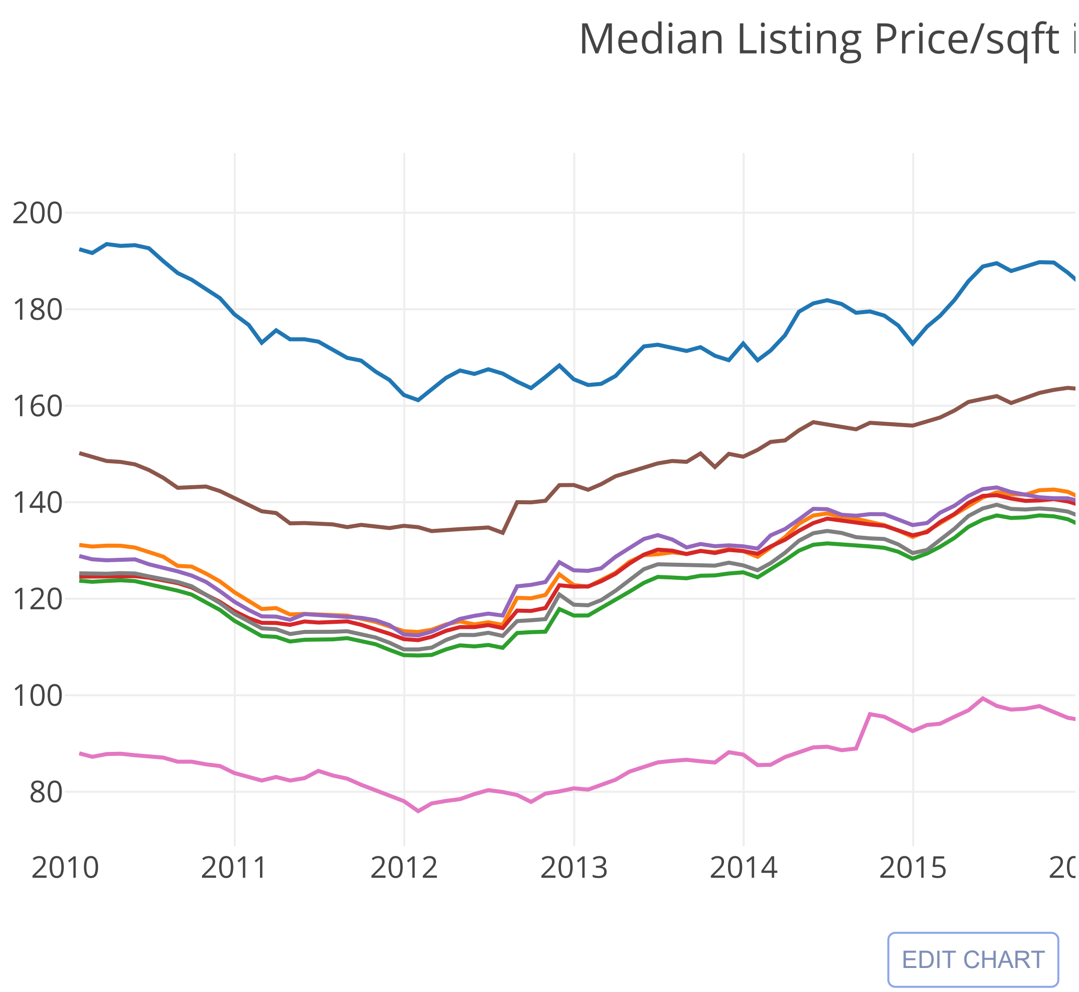
Plotting the Median Listing Price Per Sqft with respect to the monthly time intervals

In [16]:
```
data = [go.Scatter(x = state_month['Date'], y = state_month['MedianListingPri
        go.Scatter(x = state_month['Date'], y = state_month['MedianListingPri
        go.Scatter(x = state_month['Date'], y = state_month['MedianListingPri
        go.Scatter(x = state_month['Date'], y = state_month['MedianListingPri
        go.Scatter(x = state_month['Date'], y = state_month['MedianListingPri
        go.Scatter(x = state_month['Date'], y = state_month['MedianListingPri
        go.Scatter(x = state_month['Date'], y = state_month['MedianListingPri
        go.Scatter(x = state_month['Date'], y = state_month['MedianListingPri
       ]
```

In [17]:
```
layout = {'title': 'Median Listing Price/sqft in $', 'font': dict(size=15),'x
```

In [18]:
```
warnings.filterwarnings("ignore")
fig = dict(data=data, layout=layout)
py.iplot(fig)
```

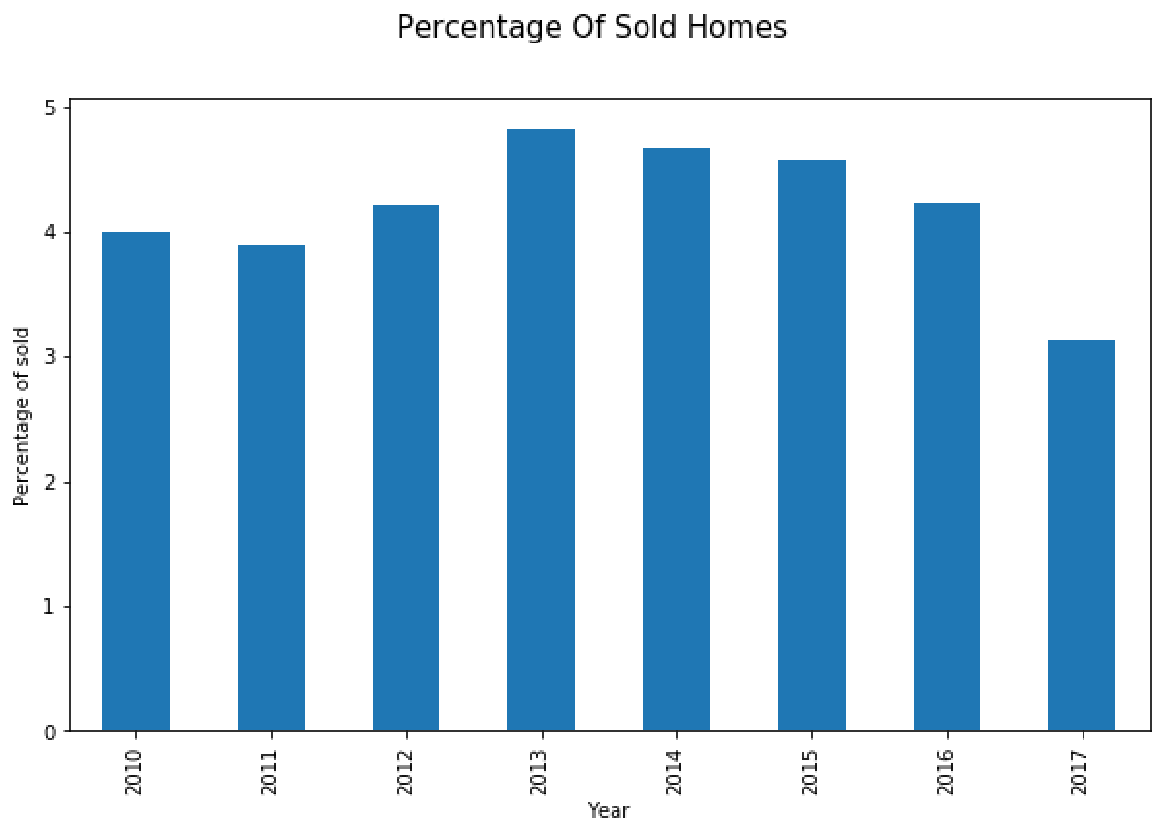
Out[18]:



Percentage of Sold Homes

In [19]:

```
df2 = df.dropna(subset=['Turnover_AllHomes'], how='any')
df2.groupby(df2.Date.dt.year)['Turnover_AllHomes'].mean().plot(kind='bar', fi
plt.suptitle("Percentage Of Sold Homes", fontsize=15)
plt.ylabel('Percentage of sold')
plt.xlabel('Year')
plt.show()
```



Displaying Top Five Homes with Largest Median Listing Price Per SqFt Area

In [20]:

```
dfallhomes = df.groupby('RegionName', as_index=False)['MedianListingPricePerS
    mean().dropna().sort_values('MedianListingPricePerSqft_AllHomes', ascendi
dfallhomes.head()
```

Out[20]:

	RegionName	MedianListingPricePerSqft_AllHomes
8	DistrictofColumbia	458.253909
11	Hawaii	402.616779
4	California	264.384711
21	Massachusetts	206.087124
32	NewYork	174.116544

Displaying Top Five Homes with Lowest Median Listing Price Per SqFt Area

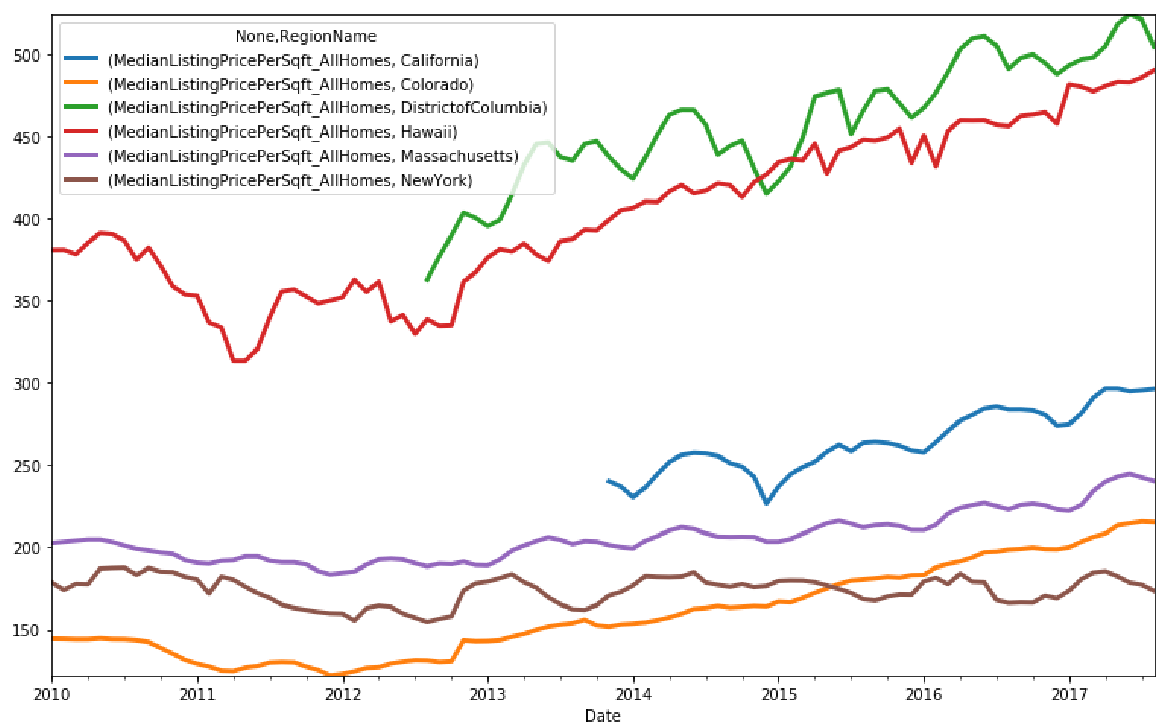
In [21]:
```
dfallhomes.tail(5)
```

Out[21]:

	RegionName	MedianListingPricePerSqft_AllHomes
3	Arkansas	86.605089
36	Oklahoma	86.356722
24	Mississippi	86.343334
35	Ohio	86.035765
14	Indiana	81.160595

Displaying Trends of Median Listing Price Per SqFt Of All Homes Over The Years

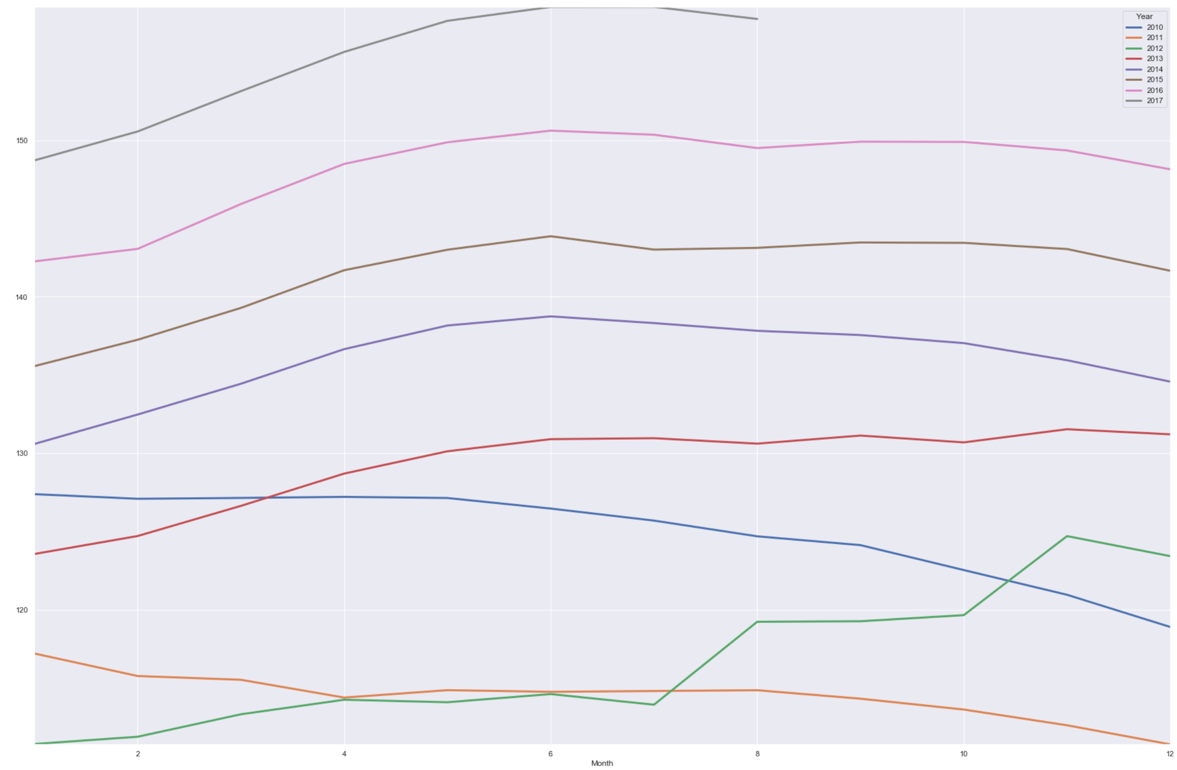
In [22]:
```
%matplotlib inline
import matplotlib.pyplot as plt
statelist = ['Hawaii', 'DistrictofColumbia', 'California', 'Massachusetts', '
stateseries = pd.DataFrame(df[(df['RegionName'].\
    isin(statelist))][['Date','RegionName','MedianListingPricePerSqft_AllHome
    dropna().\
    groupby(['Date', 'RegionName'])['RegionName','MedianListingPricePerSqft_A
stateseries.plot(figsize=(13,8), linewidth=3)
plt.margins(0)
plt.show()
```



In [23]:
```
#converting data to datetime data type
df['Date'] = pd.to_datetime(df['Date'])
```

Seasonal Trends Of Median Listing Price Per SqFt Of All Homes

In [24]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set()
season = df

season['Date'] = df.Date
season['Year'] = df['Date'].dt.year
season['Month'] = df['Date'].dt.month
spivot = pd.pivot_table(season, index='Month', columns = 'Year', values = 'Me
spivot.plot(figsize=(30,20), linewidth=3)
plt.margins(0)
plt.show()
```



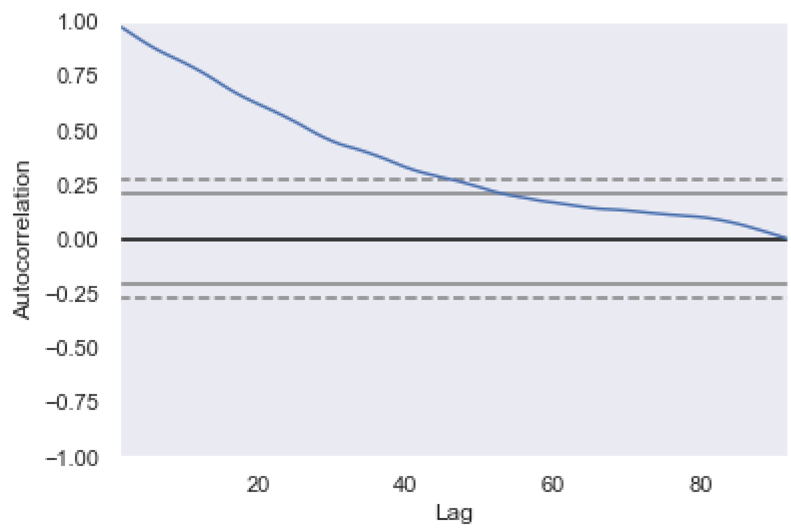
We can interpret the following from the graph -

1. House prices decreased from Year 2010 and further decreased in Year 2011. House prices picked up in Year 2012 past June.
2. The best time to sell houses is from the month of June through October (inclusive). House prices are at a peak during these months across all years.
3. The best time to buy a house is in the months of December and January.

AutoCorrelation

In [25]:

```
#plotting the auto correaltion graph
%matplotlib inline
import matplotlib.pyplot as plt

brtypes = df.groupby('Date')['Date','MedianListingPricePerSqft_1Bedroom', 'Me
    mean().dropna()
pd.plotting.autocorrelation_plot(brtypes);
plt.show()
```



The above graph shows there is a positive correlation for all the bedroom types, but still it is not clear how each bedroom types are correlated. Let us find out.

In [26]:

```
# checking the correlation between Median Listing Price Per Sqft and number o
brtypes.corr()
```

Out[26]:

	MedianListingPricePerSqft_1Bedroom	MedianListing
MedianListingPricePerSqft_1Bedroom	1.000000	
MedianListingPricePerSqft_2Bedroom	0.840396	
MedianListingPricePerSqft_3Bedroom	0.824212	
MedianListingPricePerSqft_4Bedroom	0.767990	
MedianListingPricePerSqft_5BedroomOrMore	0.766047	

The above table confirms the correlation and to be more specific let us remove the seasonality from the data and see. This is called order of correlation.

In [27]:
```
#checking the order difference in correlation
brtypes.diff().corr()
```

Out[27]:

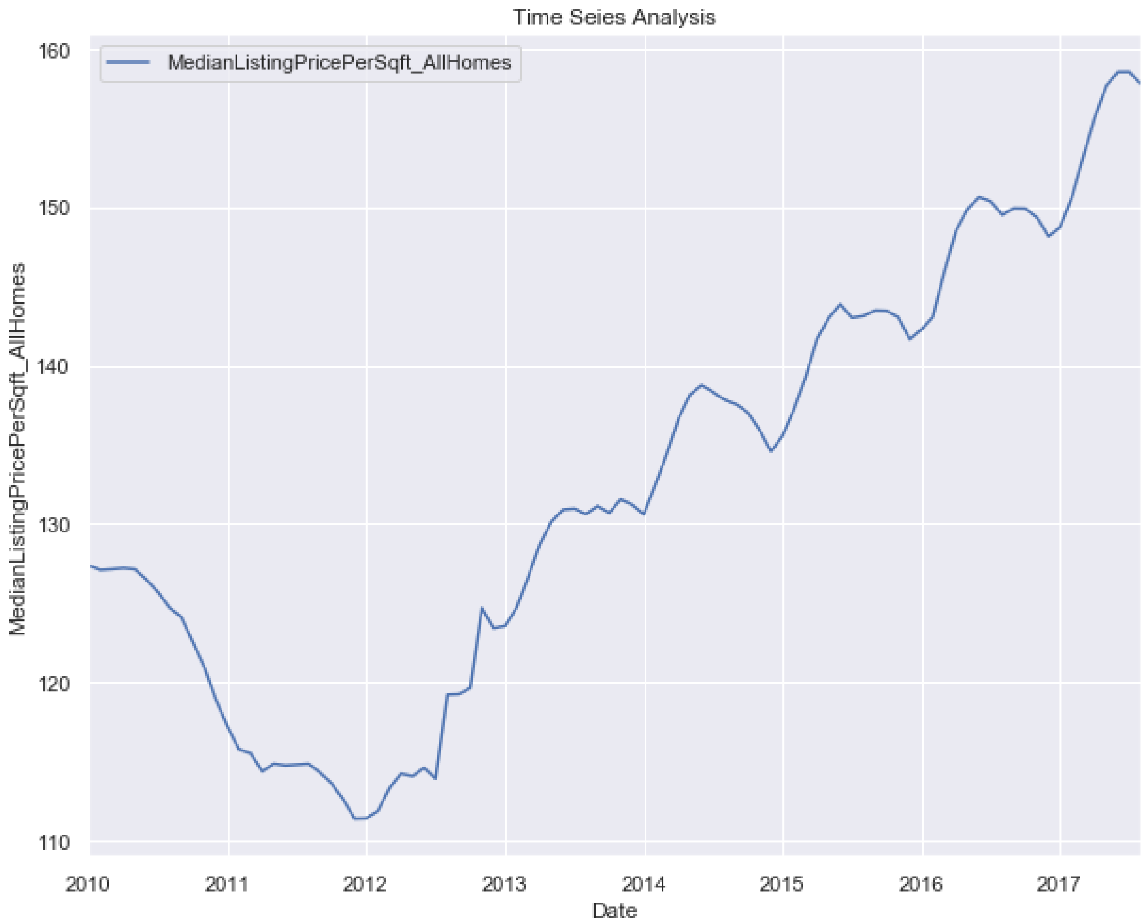
	MedianListingPricePerSqft_1Bedroom	MedianListing
MedianListingPricePerSqft_1Bedroom	1.000000	
MedianListingPricePerSqft_2Bedroom	0.727023	
MedianListingPricePerSqft_3Bedroom	0.761538	
MedianListingPricePerSqft_4Bedroom	0.704150	
MedianListingPricePerSqft_5BedroomOrMore	0.621515	

First order difference in correlation still has better correlation between bedroom types. You can see 1 Bedroom & 2 Bedroom are highly correlated than 1 bedroom & 5 bedroom.

Forecasting using Time Series Analysis

We will be performing Time Series Analysis using the Median Listing Price Per SqFt variable.

In [28]:
```
allhomes = df.groupby('Date')['Date','MedianListingPricePerSqft_AllHomes'].me
allhomes.plot(figsize=(10,8))
plt.title('Time Seies Analysis')
plt.xlabel('Date')
plt.ylabel('MedianListingPricePerSqft_AllHomes')
plt.show()
```



The above graphs shows a clear trend and it also shows seasonality.

ARIMA forecast model

ARIMA is an acronym for AutoRegressive Integrated Moving Average. It is a generalization of an autoregressive moving average (ARMA) model. Both of these models are fitted to time series data to better understand the data or to forecast.

This acronym is descriptive, capturing the key aspects of the model itself. They are -

AR: Autoregression. A model that uses the dependent relationship between an observation and some number of lagged observations. I: Integrated. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.

MA: Moving Average. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

Each of these components are explicitly specified in the model as a parameter. A standard notation is used of ARIMA(p,d,q) where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used.

The parameters of the ARIMA model are defined as follows:

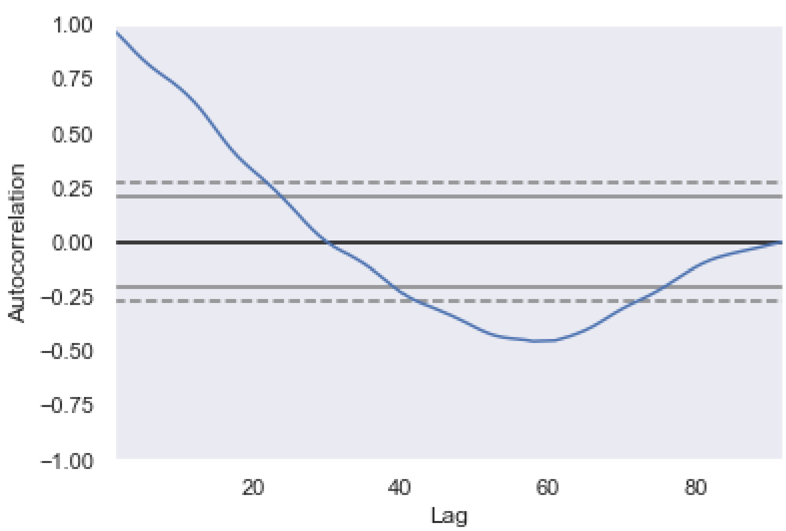
p: The number of lag observations included in the model, also called the lag order. d: The number of times that the raw observations are differenced, also called the degree of differencing. q: The size of the moving average window, also called the order of moving average. A linear regression model is constructed including the specified number and type of terms, and the data is prepared by a degree of differencing in order to make it stationary, i.e. to remove trend and seasonal structures that negatively affect the regression model.

A value of 0 can be used for a parameter, which indicates to not use that element of the model. This way, the ARIMA model can be configured to perform the function of an ARMA model, and even a simple AR, I, or MA model.

Adopting an ARIMA model for a time series assumes that the underlying process that generated the observations is an ARIMA process. This may seem obvious, but helps to motivate the need to confirm the assumptions of the model in the raw observations and in the residual errors of forecasts from the model.

A simple way to identify the p-lag is to draw the autocorrelation plot.

In [29]:
```
pd.plotting.autocorrelation_plot(allhomes);
```



From the above graph, we can select 15 as lag and then start 0, need to run and check the optimimum parameters.

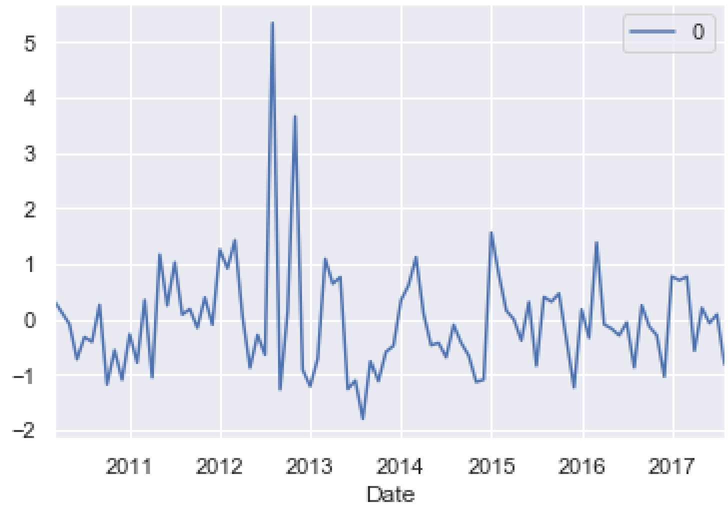
In [30]:
```
import warnings
warnings.filterwarnings("ignore")
```

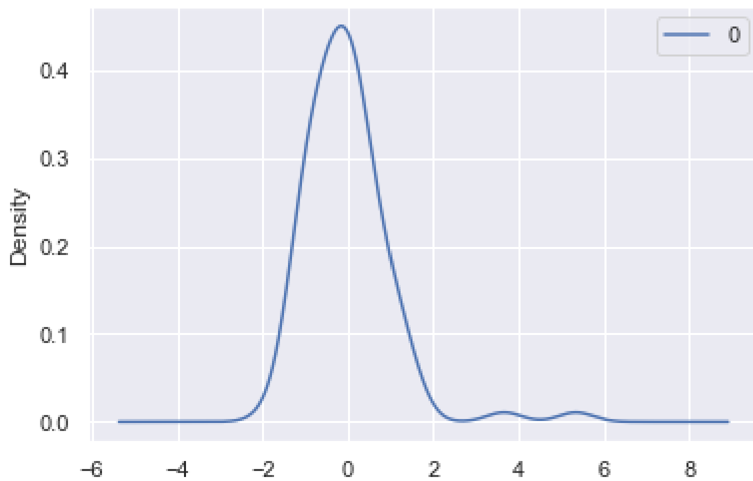
In [33]:

```
#fitting the ARIMA model
from statsmodels.tsa.arima_model import ARIMA
from pandas import DataFrame

model = ARIMA(allhomes, order=(15,2,0))
model_fit = model.fit(disp=0)

# printing the model summary results
print(model_fit.summary())

#plotting the residuals graph for fitted ARIMA model
residuals = DataFrame(model_fit.resid)

residuals.plot()
plt.show()

residuals.plot(kind='kde')
plt.show()
residuals.describe()
```

```
                                       ARIMA Model Results
===========================================================================
======================
Dep. Variable:     D2.MedianListingPricePerSqft_AllHomes   No. Observation
s:                    90
Model:                                    ARIMA(15, 2, 0)   Log Likelihood
-129.646
Method:                                           css-mle   S.D. of innovati
ons               0.993
Date:                                    Fri, 26 Apr 2019   AIC
293.291
Time:                                            22:20:45   BIC
335.788
Sample:                                        03-31-2010   HQIC
310.428
                                             - 08-31-2017
===========================================================================
=====================================
                                                  coef    std err
z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------
-------------------------------------
const                                           0.0161      0.022      0.7
43      0.460      -0.026       0.058
ar.L1.D2.MedianListingPricePerSqft_AllHomes    -0.7425      0.106     -7.0
11      0.000      -0.950      -0.535
ar.L2.D2.MedianListingPricePerSqft_AllHomes    -0.5860      0.133     -4.4
14      0.000      -0.846      -0.326
ar.L3.D2.MedianListingPricePerSqft_AllHomes    -0.2900      0.145     -2.0
04      0.049      -0.574      -0.006
ar.L4.D2.MedianListingPricePerSqft_AllHomes    -0.5232      0.148     -3.5
46      0.001      -0.812      -0.234
ar.L5.D2.MedianListingPricePerSqft_AllHomes    -0.4610      0.159     -2.9
08      0.005      -0.772      -0.150
ar.L6.D2.MedianListingPricePerSqft_AllHomes    -0.4875      0.158     -3.0
78      0.003      -0.798      -0.177
```

```
ar.L7.D2.MedianListingPricePerSqft_AllHomes     -0.2752      0.160     -1.7
19      0.090      -0.589       0.039
ar.L8.D2.MedianListingPricePerSqft_AllHomes     -0.3624      0.160     -2.2
68      0.026      -0.676      -0.049
ar.L9.D2.MedianListingPricePerSqft_AllHomes     -0.4226      0.163     -2.5
96      0.011      -0.742      -0.103
ar.L10.D2.MedianListingPricePerSqft_AllHomes    -0.3951      0.161     -2.4
50      0.017      -0.711      -0.079
ar.L11.D2.MedianListingPricePerSqft_AllHomes    -0.1288      0.160     -0.8
05      0.423      -0.442       0.185
ar.L12.D2.MedianListingPricePerSqft_AllHomes     0.1741      0.150      1.1
62      0.249      -0.120       0.468
ar.L13.D2.MedianListingPricePerSqft_AllHomes     0.2855      0.147      1.9
41      0.056      -0.003       0.574
ar.L14.D2.MedianListingPricePerSqft_AllHomes     0.0896      0.134      0.6
69      0.506      -0.173       0.352
ar.L15.D2.MedianListingPricePerSqft_AllHomes     0.0882      0.106      0.8
35      0.407      -0.119       0.295
                                    Roots
============================================================================
===
                   Real          Imaginary           Modulus          Freque
ncy
----------------------------------------------------------------------------
---
AR.1             0.8828            -0.5116j            1.0204            -0.0
836
AR.2             0.8828            +0.5116j            1.0204             0.0
836
AR.3             1.3506            -0.0000j            1.3506            -0.0
000
AR.4             0.5390            -0.8837j            1.0351            -0.1
628
AR.5             0.5390            +0.8837j            1.0351             0.1
628
AR.6            -1.1035            -0.3066j            1.1453            -0.4
569
AR.7            -1.1035            +0.3066j            1.1453             0.4
569
AR.8            -0.8606            -0.6442j            1.0750            -0.3
977
AR.9            -0.8606            +0.6442j            1.0750             0.3
977
AR.10           -0.5337            -0.9700j            1.1071            -0.3
301
AR.11           -0.5337            +0.9700j            1.1071             0.3
301
AR.12           -0.0786            -1.1219j            1.1246            -0.2
611
AR.13           -0.0786            +1.1219j            1.1246             0.2
611
AR.14           -0.0287            -1.7896j            1.7898            -0.2
526
AR.15           -0.0287            +1.7896j            1.7898             0.2
526
----------------------------------------------------------------------------
---
```





Out[33]:

	0
count	90.000000
mean	-0.022154
std	1.006077
min	-1.796616
25%	-0.676195
50%	-0.096229
75%	0.342568
max	5.349963

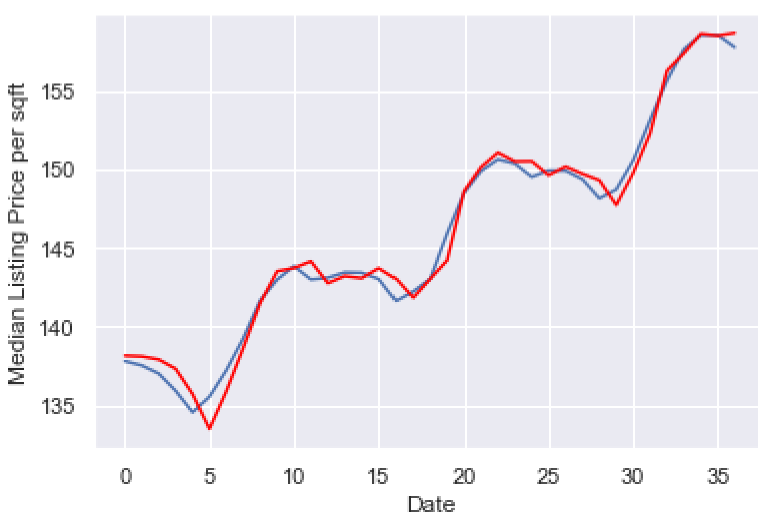
Residuals plot shows there are residual errors in the forecast and the KDE plot clearly shows that the curve is not in the zero. This can be further optimized by changing values based on the ARIMA model results summary.

Let us use forecast() method of ARIMA to forecast the data. Before predicting for the next year, let us validate the model first. We have splitted the dataset into Train & Test with 40% as test data.

In [34]:

```
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt

# split into train and test sets
X = allhomes.values
train_size = int(len(X) * 0.60)
train, test = X[0:train_size], X[train_size:]
history = [x for x in train]
predictions = list()

for t in range(len(test)):
    model = ARIMA(history, order=(15,2,0))
    model_fit = model.fit()
    fcast = model_fit.forecast()[0]
    predictions.append(fcast)
    history.append(test[t])

# evaluate forecasts
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)
# plot forecasts against actual outcomes
plt.plot(test)

plt.xlabel('Date')
plt.ylabel('Median Listing Price per sqft')
plt.plot(predictions, color='red')
plt.show()
```

```
Test RMSE: 0.819
```



In the above graph the blue line is the actual value and the red line is the predicted value. The predictions are almost equivalent to the test data.

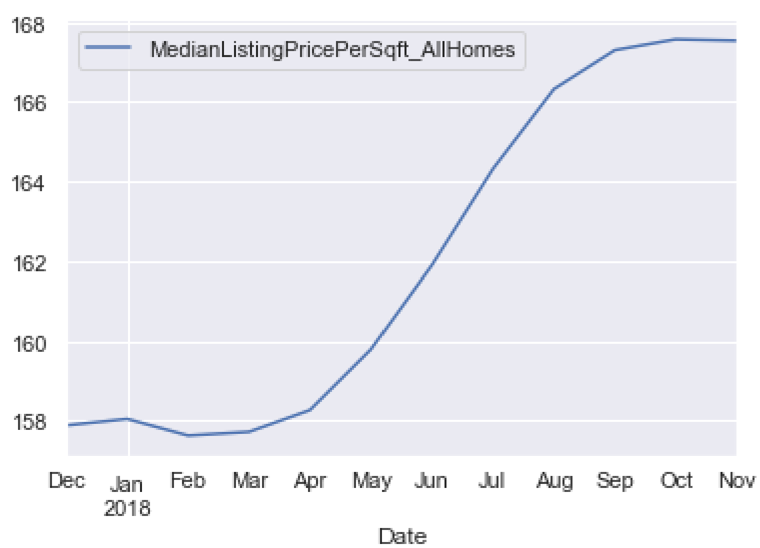
In [36]:

```
#forecasting the ARIMA results
forecast = list()
model = ARIMA(history, order=(15,2,0))
model_fit = model.fit(disp=0)
forecast = model_fit.forecast(steps=12)[0]
for fcast in forecast:
    print(fcast)
```

```
157.8792165146061
158.03903188122467
157.62211774128926
157.71531869921617
158.26139980273706
159.78582606625255
161.90681515799514
164.31867085406952
166.32266539124421
167.30255641630524
167.57433808286973
167.5411656278441
```

In [37]:

```
#plotting the graph which shows the prediction for next 12 months
future = pd.date_range('12/31/2017', periods=12,freq='M')
futdict = {'Date' : future, 'MedianListingPricePerSqft_AllHomes' : forecast}
ts = pd.DataFrame(futdict)
ts.set_index('Date', inplace=True)

ts.plot();
```



The above is the prediction for next 12 months and you can see the graph shows the seasonality as earlier like the price is reducing in Dec 2017 and then it is getting reduced in Oct/Nov 2018

SARIMA forecast model

Seasonal Autoregressive Integrated Moving Average, SARIMA or Seasonal ARIMA, is an extension of ARIMA that explicitly supports univariate time series data with a seasonal component.

It adds three new hyperparameters to specify the autoregression (AR), differencing (I) and moving average (MA) for the seasonal component of the series, as well as an additional parameter for the period of the seasonality.

In [38]:

```
#importing the libraries and determining the parameters
import itertools
#p = [14, 15, 16]
#d = [14, 15, 16]
#q = [14, 15, 16]
p = range(2, 4)
d = range(0, 3)
q = range(0, 3)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d,

#printing parameter combinations for Seasonal ARIMA
print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (2, 0, 1) x (2, 0, 1, 12)
SARIMAX: (2, 0, 1) x (2, 0, 2, 12)
SARIMAX: (2, 0, 2) x (2, 1, 0, 12)
SARIMAX: (2, 0, 2) x (2, 1, 1, 12)
```

In [ ]:

```


import warnings
import statsmodels.api as sm
warnings.filterwarnings("ignore")
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(allhomes,
                                            order = param,
                                            seasonal_order = param_seasona
                                            enforce_stationarity = False,
                                            enforce_invertibility = False)
            results = mod.fit()
            print('ARIMA {}x{}12 - AIC:{}'.format(param, param_seasonal, resu
        except:
            continue
```

SARIMA Model 1

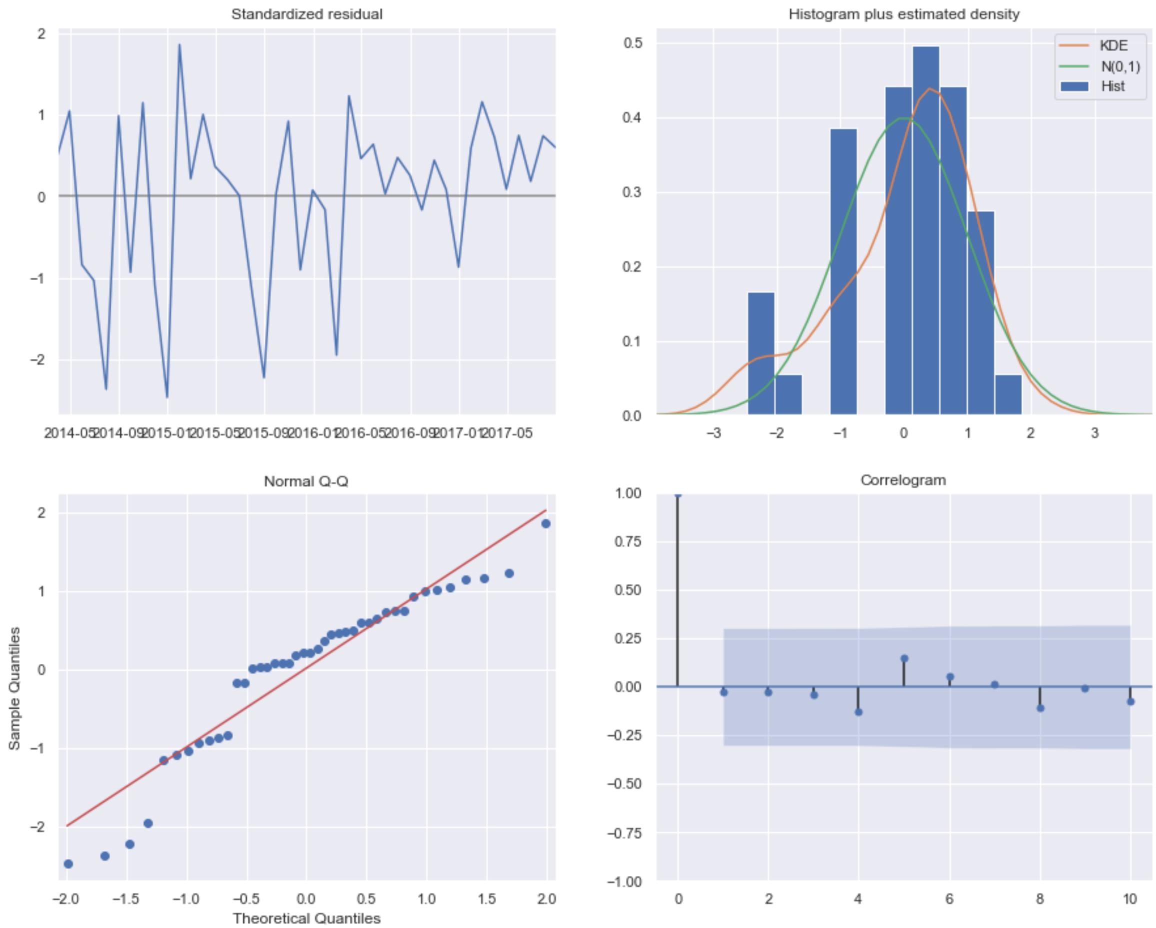
In [39]:

```
import statsmodels.api as sm
#fitting the Seasonal ARIMA model
mod = sm.tsa.statespace.SARIMAX(allhomes,
                                order = (2, 0, 4),
                                seasonal_order = (3, 1, 2, 12),
                                enforce_stationarity = False,
                                enforce_invertibility = False)
results = mod.fit()

# printing the model results
print(results.summary().tables[1])
```

```
===========================================================================
===
                 coef    std err          z      P>|z|      [0.025      0.9
75]
---------------------------------------------------------------------------
---
ar.L1          1.6179      0.080     20.325      0.000       1.462       1.
774
ar.L2         -0.6234      0.077     -8.088      0.000      -0.774      -0.
472
ma.L1         -1.0056      1.287     -0.782      0.434      -3.527       1.
516
ma.L2          0.1820      0.942      0.193      0.847      -1.664       2.
028
ma.L3          0.4998      0.880      0.568      0.570      -1.224       2.
224
ma.L4          0.0164      0.225      0.073      0.942      -0.424       0.
457
ar.S.L12       0.1384      0.048      2.855      0.004       0.043       0.
233
ar.S.L24      -0.0646      0.043     -1.519      0.129      -0.148       0.
019
ar.S.L36       0.2504      0.057      4.430      0.000       0.140       0.
361
ma.S.L12      -0.8471      5.708     -0.148      0.882     -12.034      10.
340
ma.S.L24      -0.2220      0.844     -0.263      0.793      -1.876       1.
432
sigma2         0.1139      0.749      0.152      0.879      -1.353       1.
581
===========================================================================
===
```

The following graphs are used to check if the residuals of our model are uncorrelated and normally distributed with zero-mean.In this case, our model diagnostics suggests that the model residuals are normally distributed

In [40]:
```
#plotting the residual graphs
results.plot_diagnostics(figsize=(15, 12))
plt.show()
```



From the above graph, we can see that

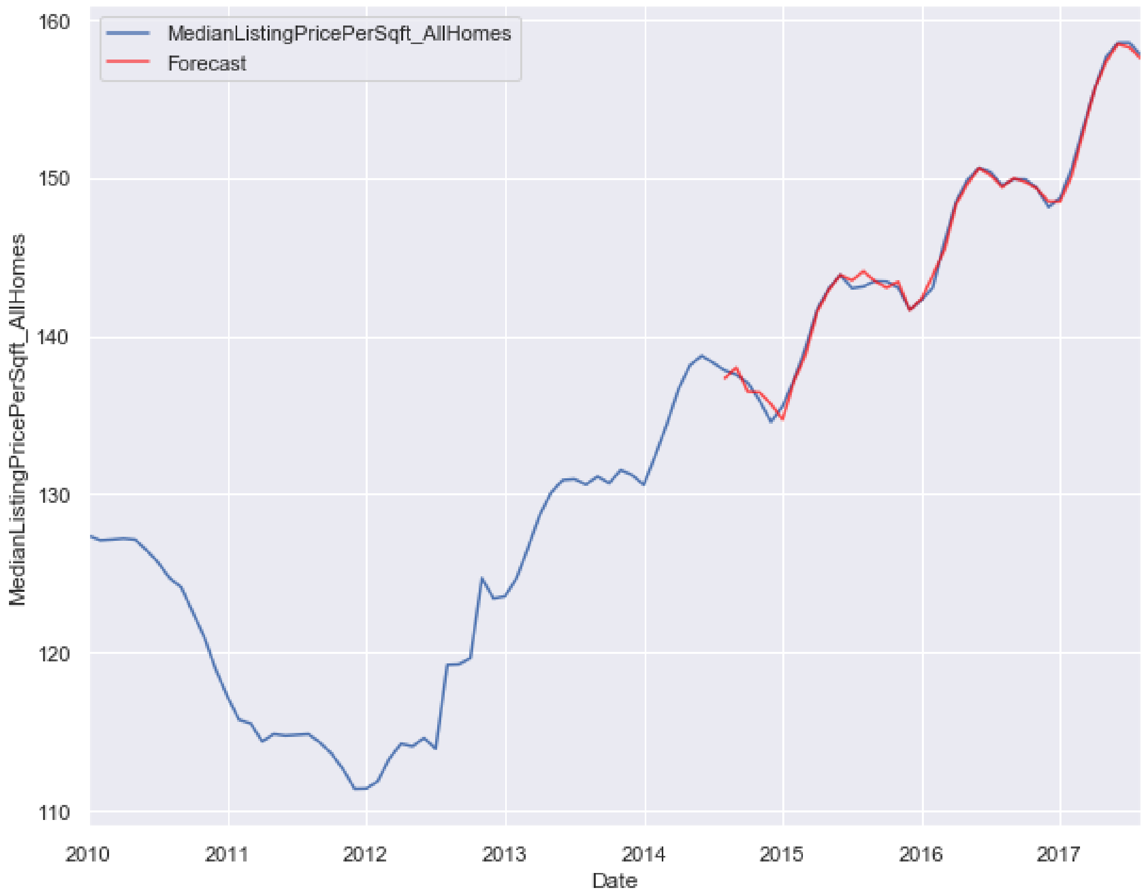
1. The histogram has minor difference with KDE
2. Linear regression can be improved
3. There are still positive correlation, this can be optimized further.

Validate the model

In [41]:
```
# splitting into train and test
train_size = int(len(allhomes) * 0.60)
train, test = allhomes[0:train_size], allhomes[train_size:]
```

In [42]:
```
#predicting the results
pred = results.get_prediction(start = test.iloc[0].name, dynamic = False)
pred_ci = pred.conf_int()
```

In [43]:
```
#plotting the actual vs Forecast graph
ax = allhomes.plot(label='actual', figsize=(10,8))
pred.predicted_mean.plot(ax=ax, label='Forecast', alpha=0.7, color='red')
ax.set_xlabel('Date')
ax.set_ylabel('MedianListingPricePerSqft_AllHomes')
plt.legend()
plt.show()
```



The graph above shows that our predictions are very much sync with the actual data

Calculating the mean squared error for this model

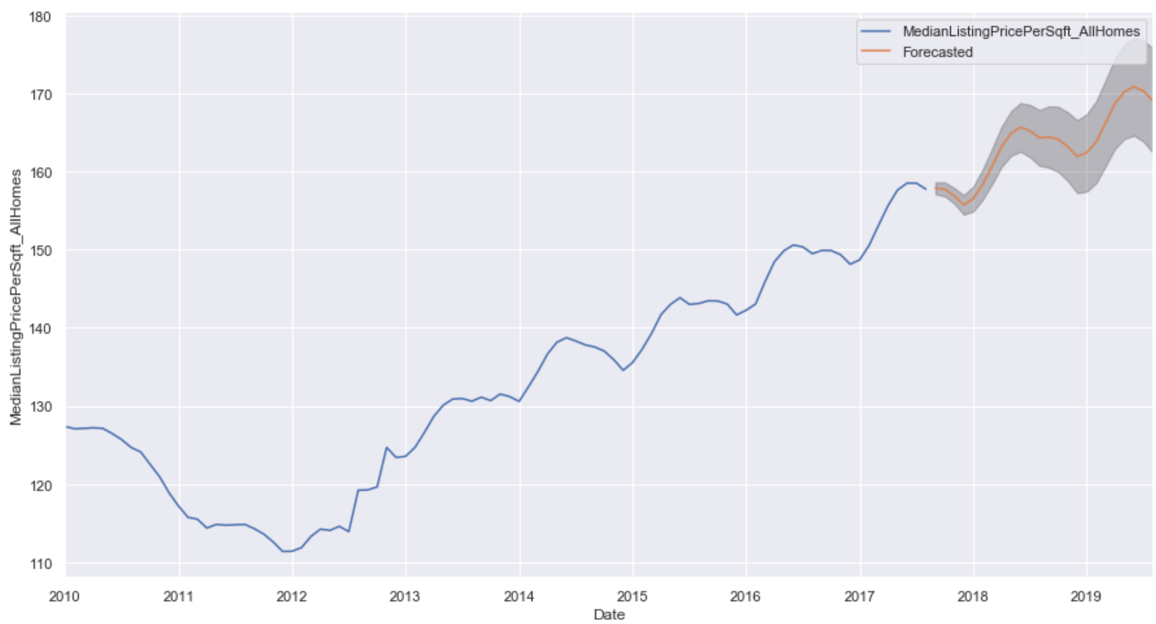
In [44]:
```
y_forecasted = pred.predicted_mean
y_series = pd.Series(allhomes['MedianListingPricePerSqft_AllHomes'].values, i

y_truth = y_series.iloc[49:]

mse = ((y_forecasted - y_truth) ** 2).mean()

print(mse)
```

```
0.17430884277543876
```

The mean squared error is 0.174, so it means the forecast gives us a good value.

Predicting for the future

In [45]:
```
pred_uc = results.get_forecast(steps=24)

pred_ci = pred_uc.conf_int()

ax = allhomes.plot(label = 'Actual', figsize=(15,8))
pred_uc.predicted_mean.plot(ax=ax, label='Forecasted')
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:,0],
                pred_ci.iloc[:,1],
                color='k', alpha=0.25)
ax.set_xlabel('Date')
ax.set_ylabel('MedianListingPricePerSqft_AllHomes')
plt.legend()
plt.show()
```



From the above graph it can be observed that the forecast for next 24 months and the confidence interval are better and not too much variance.

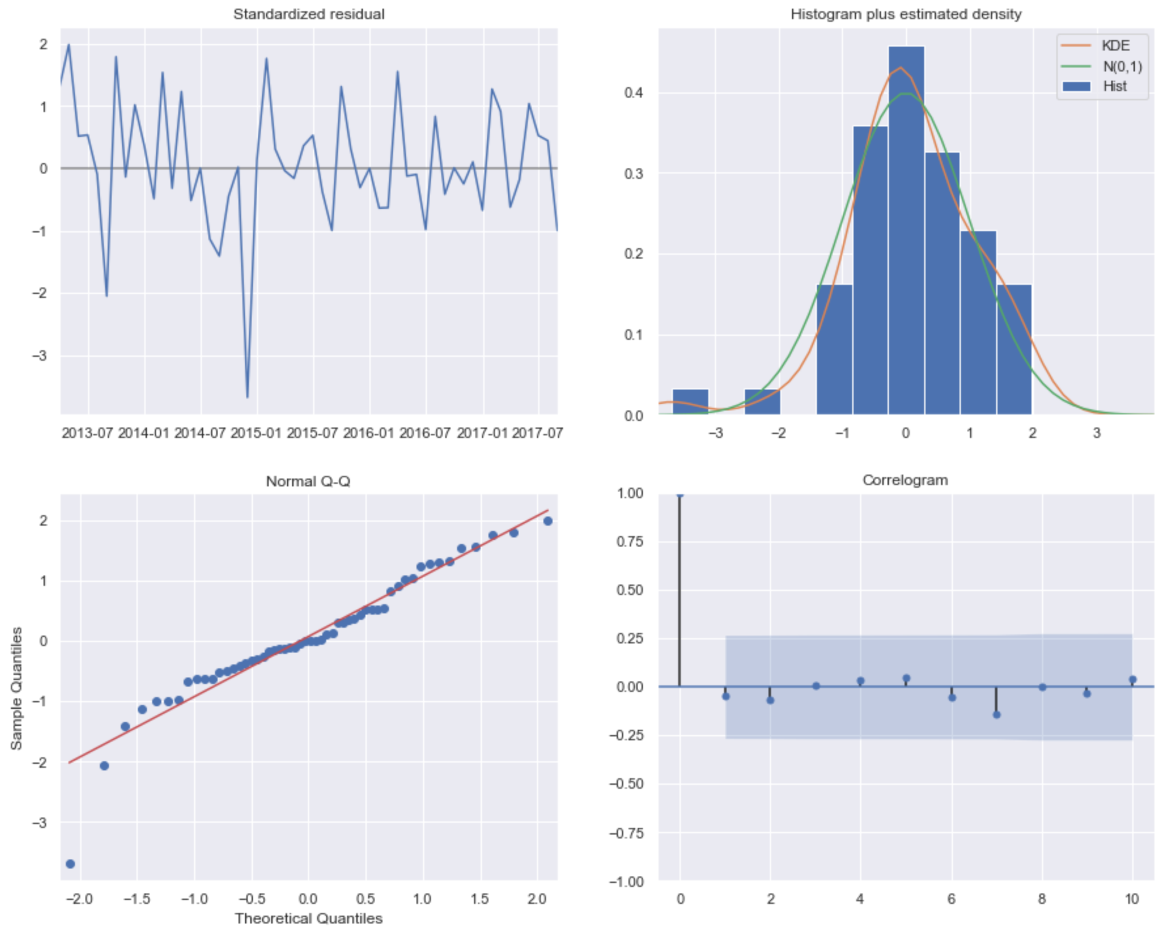
SARIMA Model 2

In [46]:

```
import statsmodels.api as sm
#fitting the Seasonal ARIMA model
mod = sm.tsa.statespace.SARIMAX(allhomes,
                                order = (2, 0, 4),
                                seasonal_order = (2, 1, 1, 12),
                                enforce_stationarity = False,
                                enforce_invertibility = False)
results = mod.fit()

# printing the model results
print(results.summary().tables[1])
```

```
============================================================================
===
                 coef    std err          z      P>|z|      [0.025      0.9
75]
----------------------------------------------------------------------------
---
ar.L1          1.5695      2.063      0.761      0.447      -2.474       5.
613
ar.L2         -0.5701      2.058     -0.277      0.782      -4.604       3.
464
ma.L1         -1.2561      5.225     -0.240      0.810     -11.497       8.
985
ma.L2         -0.6360      2.609     -0.244      0.807      -5.750       4.
478
ma.L3          0.3054      0.572      0.534      0.593      -0.815       1.
426
ma.L4         -0.6225      1.889     -0.330      0.742      -4.324       3.
079
ar.S.L12      -0.7017      0.062    -11.287      0.000      -0.824      -0.
580
ar.S.L24      -0.3679      0.059     -6.187      0.000      -0.484      -0.
251
ma.S.L12       0.9321      1.998      0.467      0.641      -2.984       4.
848
sigma2         0.1161      0.745      0.156      0.876      -1.344       1.
576
============================================================================
===
```

In [47]:
```
#plotting the residual graphs
results.plot_diagnostics(figsize=(15, 12))
plt.show()
```

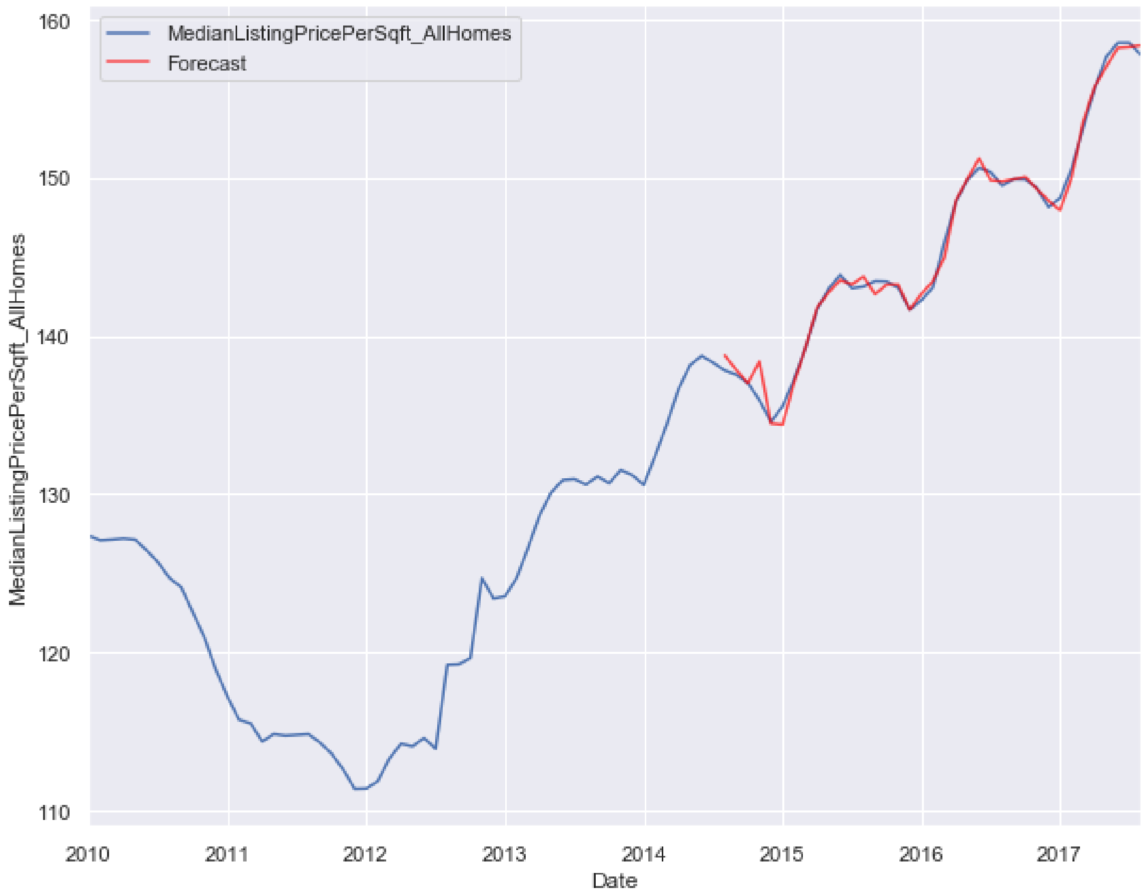


Validate the model

In [48]:
```
# splitting into train and test
train_size = int(len(allhomes) * 0.60)
train, test = allhomes[0:train_size], allhomes[train_size:]
```

In [49]:
```
#predicting the results
pred = results.get_prediction(start = test.iloc[0].name, dynamic = False)
pred_ci = pred.conf_int()
```

In [50]:
```
#plotting the actual vs Forecast graph
ax = allhomes.plot(label='actual', figsize=(10,8))
pred.predicted_mean.plot(ax=ax, label='Forecast', alpha=0.7, color='red')
ax.set_xlabel('Date')
ax.set_ylabel('MedianListingPricePerSqft_AllHomes')
plt.legend()
plt.show()
```



Calculating the mean squared error for this model

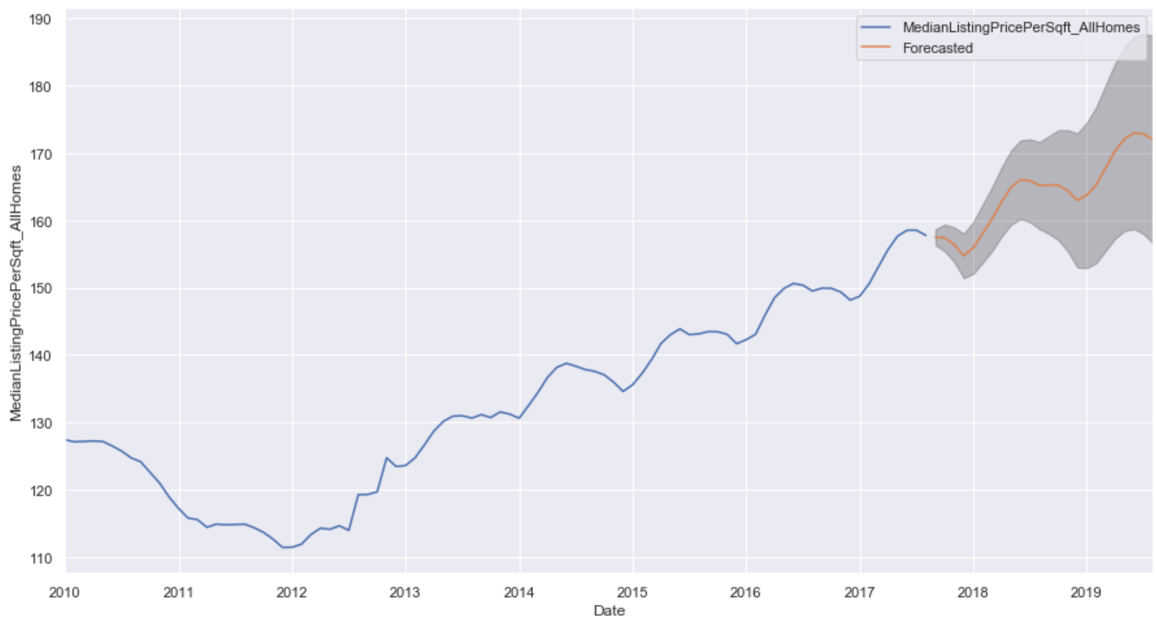
In [51]:
```
y_forecasted = pred.predicted_mean
y_series = pd.Series(allhomes['MedianListingPricePerSqft_AllHomes'].values, i

y_truth = y_series.iloc[49:]

mse = ((y_forecasted - y_truth) ** 2).mean()

print(mse)
```

```
0.3756325643725539
```

Predicting for the future

In [52]:

```
pred_uc = results.get_forecast(steps=24)

pred_ci = pred_uc.conf_int()

ax = allhomes.plot(label = 'Actual', figsize=(15,8))
pred_uc.predicted_mean.plot(ax=ax, label='Forecasted')
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:,0],
                pred_ci.iloc[:,1],
                color='k', alpha=0.25)
ax.set_xlabel('Date')
ax.set_ylabel('MedianListingPricePerSqft_AllHomes')
plt.legend()
plt.show()
```



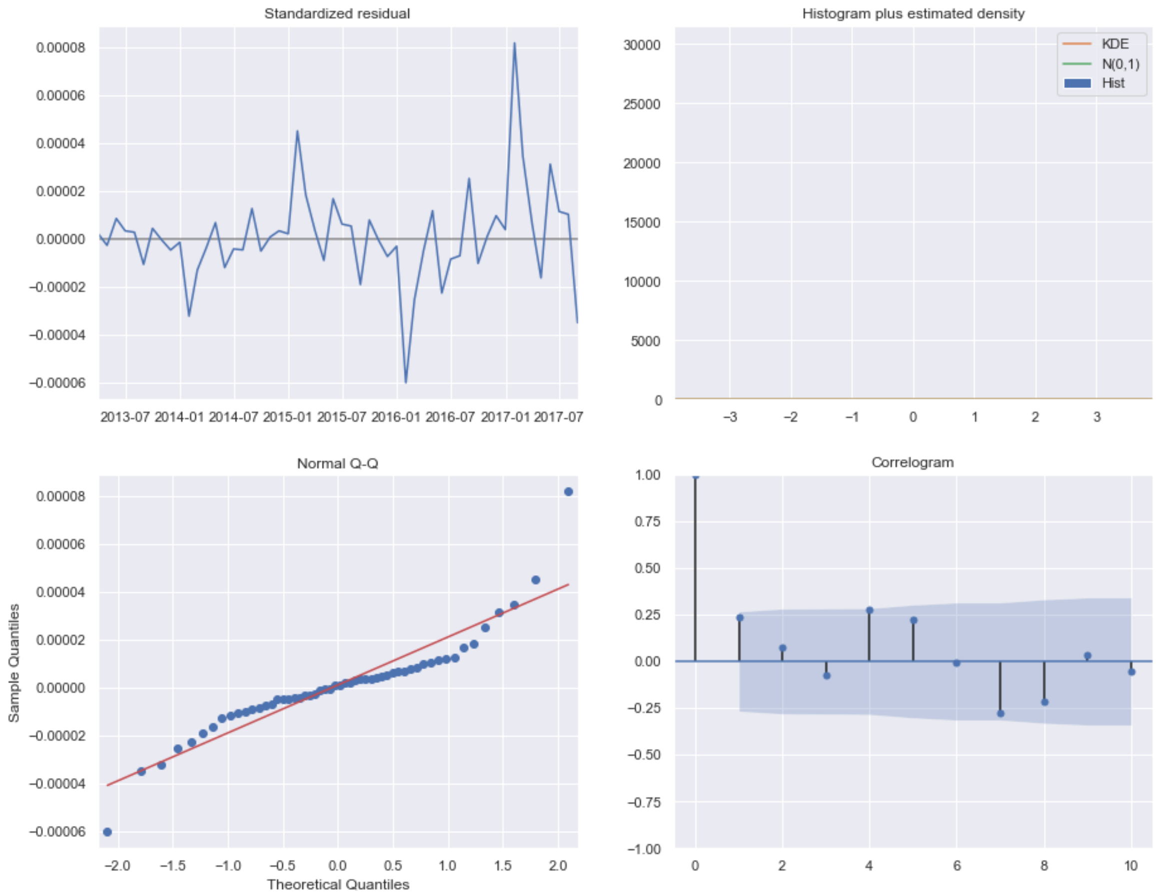
SARIMA Model 3

In [53]:

```
import statsmodels.api as sm
#fitting the Seasonal ARIMA model
mod = sm.tsa.statespace.SARIMAX(allhomes,
                                order = (2, 0, 4),
                                seasonal_order = (3, 0, 2, 12),
                                enforce_stationarity = False,
                                enforce_invertibility = False)
results = mod.fit()

# printing the model results
print(results.summary().tables[1])
```

```
============================================================================
===
                 coef    std err          z      P>|z|      [0.025      0.9
75]
----------------------------------------------------------------------------
---
ar.L1          1.5067   5.35e-27   2.82e+26      0.000       1.507       1.
507
ar.L2         -0.5048         -0        inf      0.000      -0.505      -0.
505
ma.L1         -0.1515   2.12e-24  -7.13e+22      0.000      -0.151      -0.
151
ma.L2          0.0009   1.39e-24   6.31e+20      0.000       0.001       0.
001
ma.L3          0.1566   2.17e-24   7.22e+22      0.000       0.157       0.
157
ma.L4         -0.3574    1.5e-24  -2.38e+23      0.000      -0.357      -0.
357
ar.S.L12       0.1840    2.5e-28   7.35e+26      0.000       0.184       0.
184
ar.S.L24       0.4867   8.71e-28   5.59e+26      0.000       0.487       0.
487
ar.S.L36       0.4605   6.44e-28   7.15e+26      0.000       0.460       0.
460
ma.S.L12    4.017e+11   1.23e-30   3.28e+41      0.000    4.02e+11    4.02e
+11
ma.S.L24    5.447e+11   9.17e-35   5.94e+45      0.000    5.45e+11    5.45e
+11
sigma2      6.769e-12   2.59e-10      0.026      0.979   -5.01e-10    5.14e
-10
============================================================================
===
```

In [54]:
```
#plotting the residual graphs
results.plot_diagnostics(figsize=(15, 12))
plt.show()
```

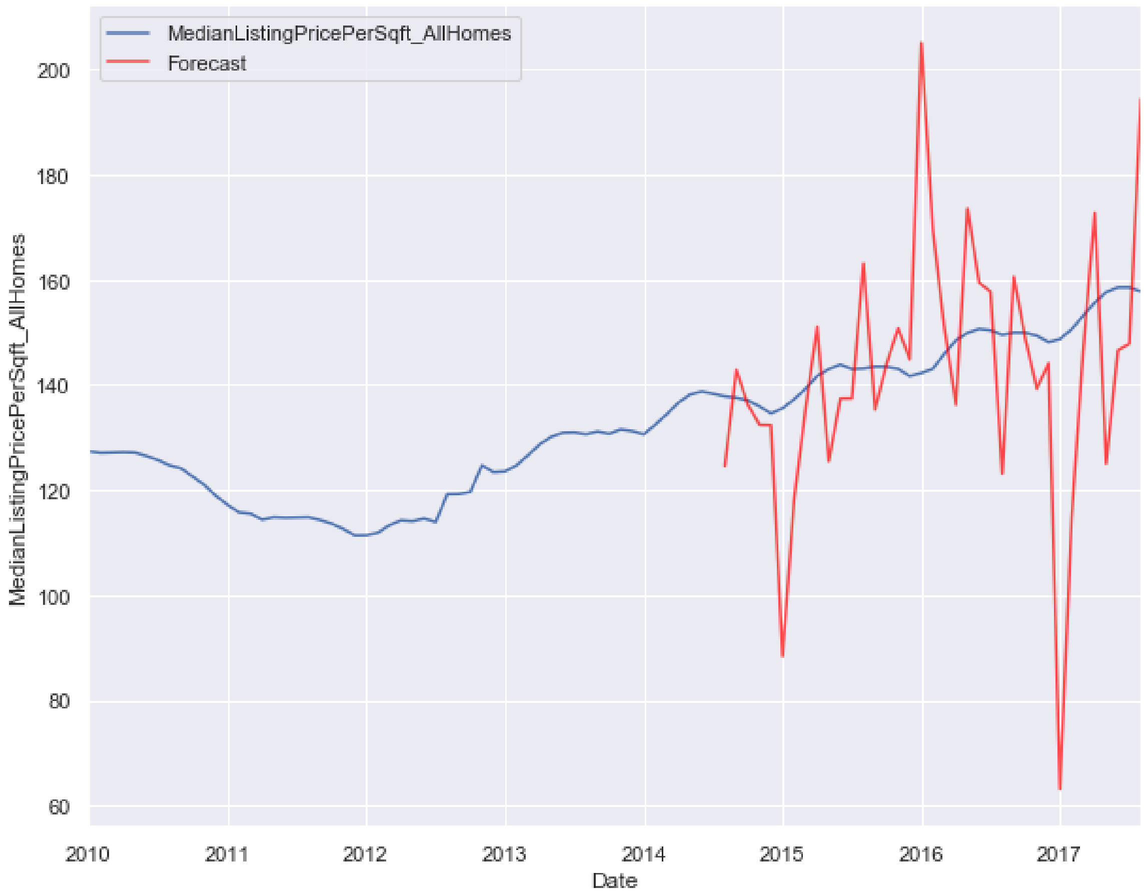


Validate the model

In [55]:
```
# splitting into train and test
train_size = int(len(allhomes) * 0.60)
train, test = allhomes[0:train_size], allhomes[train_size:]
```

In [56]:
```
#predicting the results
pred = results.get_prediction(start = test.iloc[0].name, dynamic = False)
pred_ci = pred.conf_int()
```

In [57]:
```
#plotting the actual vs Forecast graph
ax = allhomes.plot(label='actual', figsize=(10,8))
pred.predicted_mean.plot(ax=ax, label='Forecast', alpha=0.7, color='red')
ax.set_xlabel('Date')
ax.set_ylabel('MedianListingPricePerSqft_AllHomes')
plt.legend()
plt.show()
```



Calculating the mean squared error for this model

In [58]:
```
y_forecasted = pred.predicted_mean
y_series = pd.Series(allhomes['MedianListingPricePerSqft_AllHomes'].values, i

y_truth = y_series.iloc[49:]

mse = ((y_forecasted - y_truth) ** 2).mean()

print(mse)
```

```
590.9739593236156
```

Conclusion from the SARIMA models

Out of the 3 above models for SARIMA,we can observe that the Model 1 with order = (2, 0, 4) and seasonal_order = (3, 1, 2,12)12 gives the lowest Mean squared error value of 0.174 and also the actual values are very much aligned with predicted values. Hence Model1 for SARIMA gives us the best results.

Prophet Forecast Model

Prophet, released by Facebook, is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

In [59]:
```
StateDF=df[['Date','MedianListingPricePerSqft_AllHomes']]
StateDF.dropna(inplace=True)
#prophet expects data in the format as DF('ds','y)
StateDF['ds']=StateDF['Date']
StateDF['y']=np.log(StateDF['MedianListingPricePerSqft_AllHomes'])
```

In [60]:
```
#getting insights of data
del StateDF['Date']
del StateDF['MedianListingPricePerSqft_AllHomes']
StateDF.head()
```

Out[60]:

	ds	y
8243	2010-01-31	4.561315
8244	2010-01-31	5.032316
8246	2010-01-31	4.477337
8248	2010-01-31	4.973699
8249	2010-01-31	5.168514

In [61]:
```
train=StateDF[:len(StateDF)-40]
train.shape
```

Out[61]: (4403, 2)

In [62]:
```
test=StateDF[len(StateDF)-40:]
test.shape
```

Out[62]: (40, 2)

Importing the Prophet Libraries and and fitting the model by instantiating a new Prophet object

In [63]:
```
from fbprophet import Prophet
m = Prophet()
m.fit(StateDF);
```

```
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_season
ality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonal
ity=True to override this.
```

Predictions are then made on a dataframe with a column ds containing the dates for which a prediction is to be made.

In [64]:
```
future = m.make_future_dataframe(periods=40) # days for test split
future.tail()
```

Out[64]:

	ds
4478	2017-10-06
4479	2017-10-07
4480	2017-10-08
4481	2017-10-09
4482	2017-10-10

The predict method will assign each row in future a predicted value which it names yhat. The forecast object here is a new dataframe that includes a column yhat with the forecast, as well as columns for components and uncertainty intervals.

In [65]:
```
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[65]:

	ds	yhat	yhat_lower	yhat_upper
4478	2017-10-06	4.955008	4.511322	5.394336
4479	2017-10-07	4.954340	4.529386	5.389538
4480	2017-10-08	4.954596	4.529838	5.398807
4481	2017-10-09	4.955780	4.506598	5.394272
4482	2017-10-10	4.957868	4.519330	5.430642

We can now plot the forecast by calling the Prophet.plot method and passing in forecast dataframe.

In [66]:
```
m.plot_components(forecast);
```

Cross validate the Prophet model

Using the cross_validation() function on the model and specify the forecast horizon with the horizon parameter. We then call performance_metrics() to get a table with various prediction performance metrics.

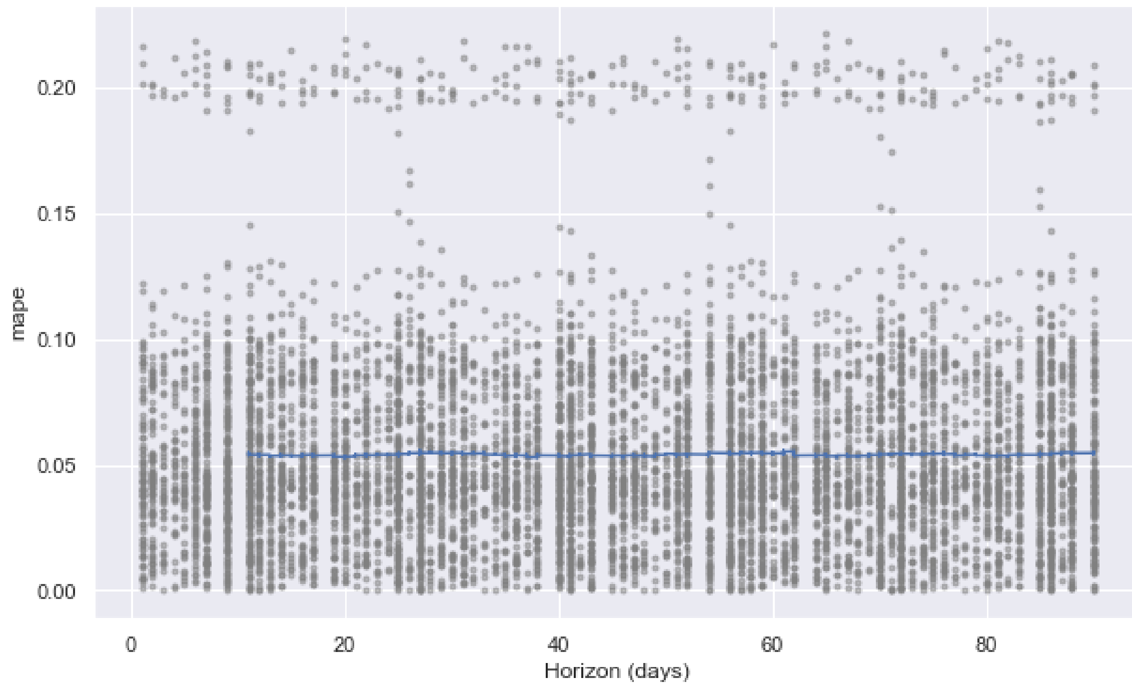
In [67]:
```
from fbprophet.diagnostics import cross_validation, performance_metrics
df_cv = cross_validation(m, horizon='90 days')
df_p = performance_metrics(df_cv)
df_p.head(5)
```

```
INFO:fbprophet:Making 54 forecasts with cutoffs between 2010-11-21 00:00:00
and 2017-06-02 00:00:00
```

Out[67]:

	horizon	mse	rmse	mae	mape	coverage
3502	11 days	0.121294	0.348273	0.263652	0.053576	0.875161
3507	11 days	0.123241	0.351057	0.265007	0.053788	0.873874
3501	11 days	0.123196	0.350994	0.264821	0.053749	0.873874
3500	11 days	0.122961	0.350658	0.264471	0.053681	0.875161
3503	11 days	0.123429	0.351325	0.264998	0.053770	0.873874

In [68]:
```
#plotting the cross validation graph
from fbprophet.plot import plot_cross_validation_metric
fig3 = plot_cross_validation_metric(df_cv, metric='mape')
```



ARIMA- ANN Hybrid Forecast Model

Artificial neural networks (ANN) is a framework for many different machine learning algorithms to work together and process complex data inputs. They have a flexible nonlinear modeling capability.

ARIMA - ANN Hybrid Forecast Model

ARIMA models are good for time series analysis but the disadvantage is the pre-assumed linear form of the model. The major advantage of artificial neural networks (ANNs) is their flexible nonlinear modeling capability.

By combining ARIMA with ANN models, complex autocorrelation structures in the data can be modeled more accurately. In Hybrid – ANN model, the ARIMA model is used to analyze the linearity of data. Then a neural network is created to model the residuals from the ARIMA model. These residuals will contain information about the non linearity that has not been captured by the ARIMA model. The results from the neural network will be used as predictions of the ARIMA model error terms.

In [69]:

```
# importing Keras libraries for implementing neural network
from keras.models import Sequential
from keras.layers import Dense,Activation,Dropout
from sklearn import preprocessing
from keras.wrappers.scikit_learn import KerasRegressor
```

```
Using TensorFlow backend.
```

Filling the null values with mean values of Median Listing Price_All Homes

In [70]:

```
df["MedianListingPrice_AllHomes"] = df.groupby("RegionName")["MedianListingPr
```

In [71]:

```
#printing the Median Listing values
k=df['MedianListingPrice_AllHomes']
print((k))
```

```
8243     167500.000000
8244     238500.000000
8245     243211.228261
8246     139900.000000
8247     443809.652174
8248     239900.000000
8249     273613.200000
8250     229900.000000
8251     478833.016393
8252     239595.608696
8253     189631.071429
8254     479900.000000
8255     211374.719298
8256     189932.065217
8257     132997.821429
8258     132500.000000
8259     149000.000000
8260     148900.000000
8261     170000.000000
8262     205000.000000
8263     239000.000000
8264     273900.000000
8265     140133.000000
8266     164400.000000
8267     150000.000000
8268     139000.000000
8269     249000.000000
8270     159352.745283
8271     235021.065217
8272     239900.000000
                 ...
12996    409900.000000
12997    169000.000000
12998    240500.000000
12999    172900.000000
13000    169000.000000
13001    295000.000000
13002    195000.000000
13003    289900.000000
13004    275000.000000
13005    299000.000000
13006    219900.000000
13007    339900.000000
13008    249000.000000
13009              NaN
13010    154900.000000
13011    176360.000000
13012    349999.000000
13013    189900.000000
13014    299500.000000
13015    230685.500000
13016    213800.000000
```

```
13017    190000.000000
13018    269000.000000
13019    330000.000000
13020    249500.000000
13021    294999.000000
13022    349000.000000
13023    150000.000000
13024    189900.000000
13025    238500.000000
Name: MedianListingPrice_AllHomes, Length: 4783, dtype: float64
```

In [72]:

```
#Checking for null values
k.isnull().sum()
k.dropna(inplace=True)
```

Fitting the ARIMA model as a part of ARIMA-ANN Hybrid Model

In [73]:

```
size = int(len(k) * 0.60)#split into test and train
percentage = 0.6
series = k.tolist()
size = int(len(series) * 0.66)
train, test = series[0:size], series[size:len(series)]
model = ARIMA(train , order = (1,0,0))
model_fit = model.fit()
```

Plotting THE ACF and PACF graphs to display the results

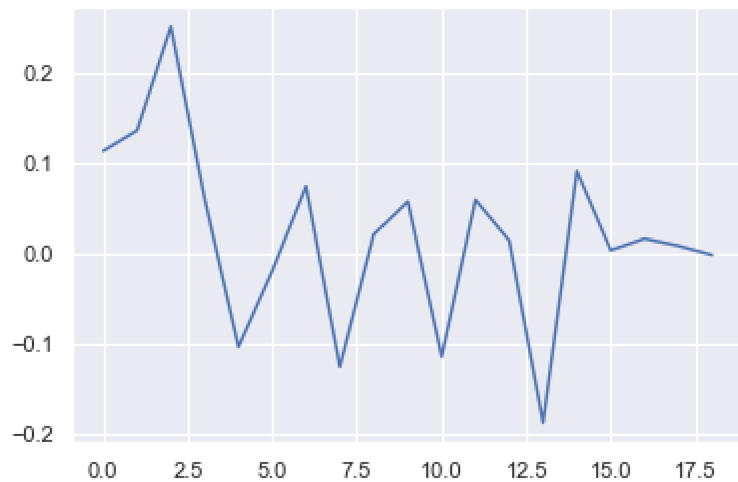
Autocorrelation and partial autocorrelation plots are heavily used in time series analysis and forecasting.

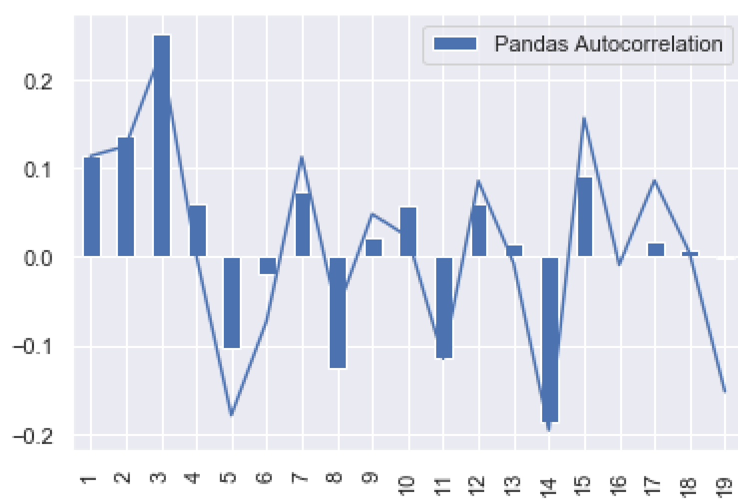
These are plots that graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps.

A partial autocorrelation is a summary of the relationship between an observation in a time series with observations at prior time steps with the relationships of intervening observations removed.

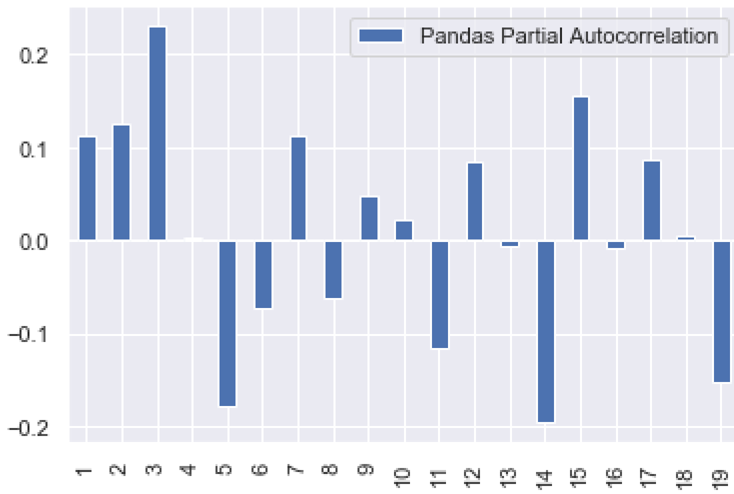
The autocorrelation for an observation and an observation at a prior time step is comprised of both the direct correlation and indirect correlations. These indirect correlations are a linear function of the correlation of the observation, with observations at intervening time steps.

In [74]:
```
from statsmodels.tsa.stattools import acf, pacf
acf_1 = acf(series)[1:20]
plt.plot(acf_1)
test_df = pd.DataFrame([acf_1]).T
test_df.columns = ["Pandas Autocorrelation"]
test_df.index += 1
test_df.plot(kind='bar')
pacf_1 = pacf(series)[1:20]
plt.plot(pacf_1)
plt.show()
test_df = pd.DataFrame([pacf_1]).T
test_df.columns = ['Pandas Partial Autocorrelation']
test_df.index += 1
test_df.plot(kind='bar')
```





Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x28414e166d8>



ARIMA Rolling Forecast

Rolling forecasts are commonly used to compare time series models.

In [75]:

```
"""
Arima Rolling Forecast
"""
predicted1, resid_test1 = [], []
history = train
for t in range(len(test)):
    model1 = ARIMA(history, order=(1,0,0))
    model_fit = model1.fit(disp=0)
    output1 = model_fit.forecast()
    yhat = output1[0]
    resid_test1.append(test[t] - output1[0])
    predicted1.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))
test_resid1 = []
for i in resid_test1:
    test_resid1.append(i[0])
error = mean_squared_error(test, predicted1)
print('Test MSE: %.3f' % error)
plt.plot(test)
plt.plot(predicted1, color='red')
plt.show()
```

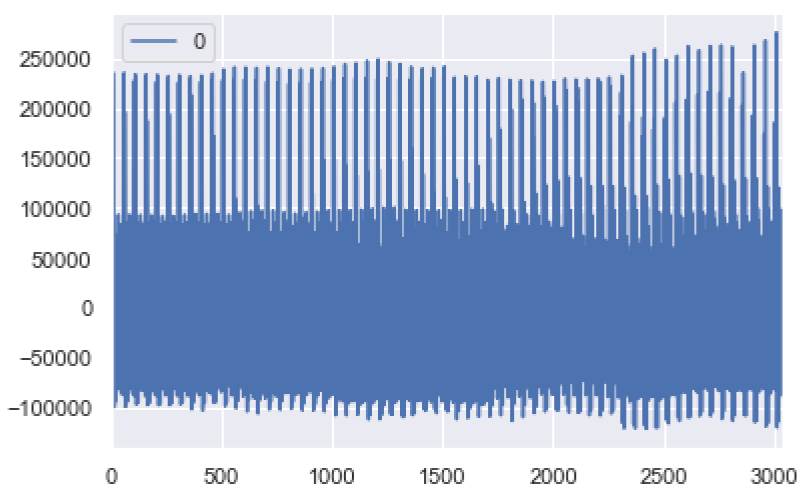
```
predicted=220409.976923, expected=163500.000000
predicted=209204.551287, expected=235000.000000
predicted=215787.165968, expected=249300.000000
predicted=217112.759802, expected=299999.000000
predicted=221811.044828, expected=215000.000000
predicted=213981.962875, expected=329000.000000
predicted=224511.522406, expected=215000.000000
predicted=214013.568917, expected=139000.000000
predicted=206998.510996, expected=159842.500000
predicted=208893.131918, expected=279900.000000
predicted=219968.558829, expected=179900.000000
predicted=210761.421358, expected=275000.000000
predicted=219511.279894, expected=205000.000000
predicted=213083.528048, expected=198000.000000
predicted=212436.825489, expected=170000.000000
predicted=209854.223121, expected=245000.000000
predicted=216744.842126, expected=280000.000000
predicted=219979.072442, expected=236000.000000
predicted=215946.421870, expected=270000.000000
predicted=219088.048322, expected=289900.000000
```

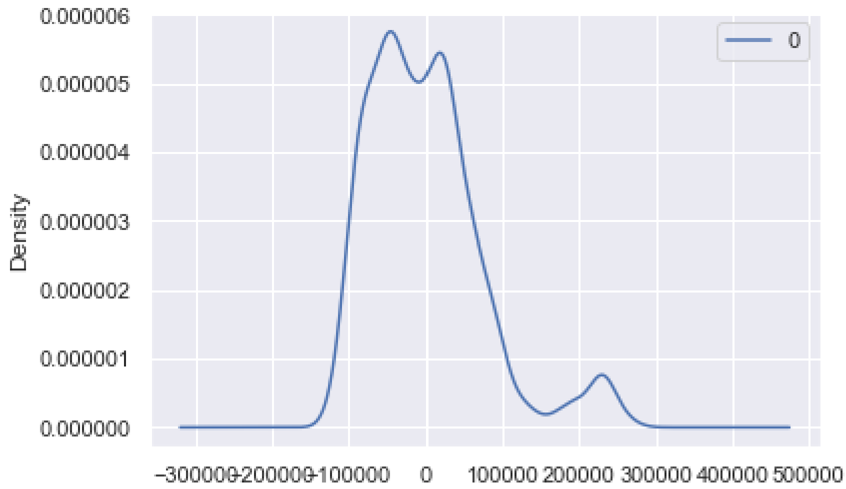
After ARIMA modeling, the residual component is calculated so that it can be input to the ANN model as ANN model will handle the non-linear component of the model.

In [78]:
```
"""
Residual Diagnostics
"""
train, test = series[0:size], series[size:len(series)]
model1 = ARIMA(train, order=(9,0,0))
model_fit = model1.fit(disp=0)
print(model_fit.summary())
# plot residual errors
residuals1 = pd.DataFrame(model_fit.resid)
residuals1.plot()
plt.show()
residuals1.plot(kind='kde')
plt.show()
print(residuals1.describe())
#plot the acf for the residuals
acf_1 = acf(model_fit.resid)[1:20]
plt.plot(acf_1)
test_df = pd.DataFrame([acf_1]).T
test_df.columns = ["Pandas Autocorrelation"]
test_df.index += 1
test_df.plot(kind='bar')
#from the acf obtained from the residuals we concule that
#there is still a nonlinear relationship among the residuals
```

```
                              ARMA Model Results
==============================================================================
===
Dep. Variable:                      y   No. Observations:                    3
036
Model:                     ARMA(9, 0)   Log Likelihood              -38391.
842
Method:                       css-mle   S.D. of innovations          75095.
813
Date:                Fri, 26 Apr 2019   AIC                          76805.
684
Time:                        22:54:47   BIC                          76871.
886
Sample:                             0   HQIC                         76829.
482

==============================================================================
===
                 coef    std err          z      P>|z|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const       2.123e+05   1691.706    125.466      0.000    2.09e+05    2.16e
+05
ar.L1.y        0.0534      0.018      2.942      0.003       0.018       0.
089
ar.L2.y        0.1783      0.018      9.837      0.000       0.143       0.
214
ar.L3.y        0.1919      0.018     10.473      0.000       0.156       0.
228
ar.L4.y       -0.0073      0.019     -0.393      0.695      -0.044       0.
029
```

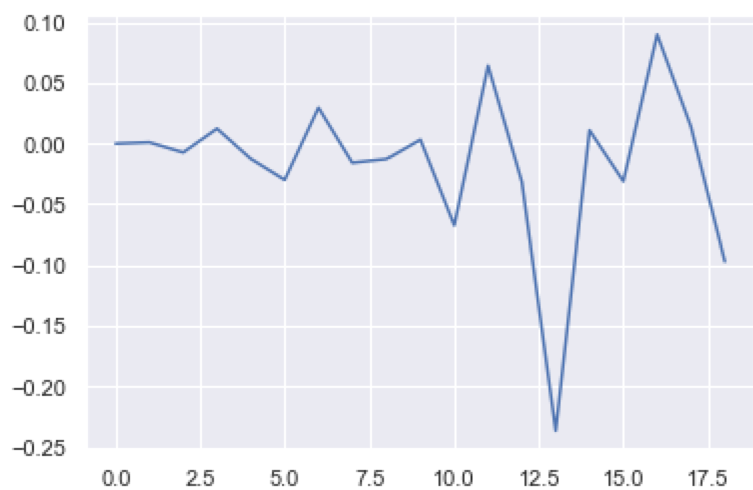
```
ar.L5.y        -0.1726       0.018      -9.408      0.000      -0.209      -0.
137
ar.L6.y        -0.0868       0.019      -4.663      0.000      -0.123      -0.
050
ar.L7.y         0.0995       0.018       5.422      0.000       0.064       0.
135
ar.L8.y        -0.0738       0.018      -4.065      0.000      -0.109      -0.
038
ar.L9.y         0.0117       0.018       0.644      0.520      -0.024       0.
047
                                    Roots
============================================================================
==
                  Real          Imaginary           Modulus         Frequen
cy
----------------------------------------------------------------------------
--
AR.1           -1.1512           -0.4925j            1.2521           -0.43
57
AR.2           -1.1512           +0.4925j            1.2521            0.43
57
AR.3           -0.3917           -1.1533j            1.2180           -0.30
21
AR.4           -0.3917           +1.1533j            1.2180            0.30
21
AR.5            1.2043           -0.5392j            1.3195           -0.06
70
AR.6            1.2043           +0.5392j            1.3195            0.06
70
AR.7            0.9823           -1.8008j            2.0513           -0.17
05
AR.8            0.9823           +1.8008j            2.0513            0.17
05
AR.9            5.0070           -0.0000j            5.0070           -0.00
00
----------------------------------------------------------------------------
--
```

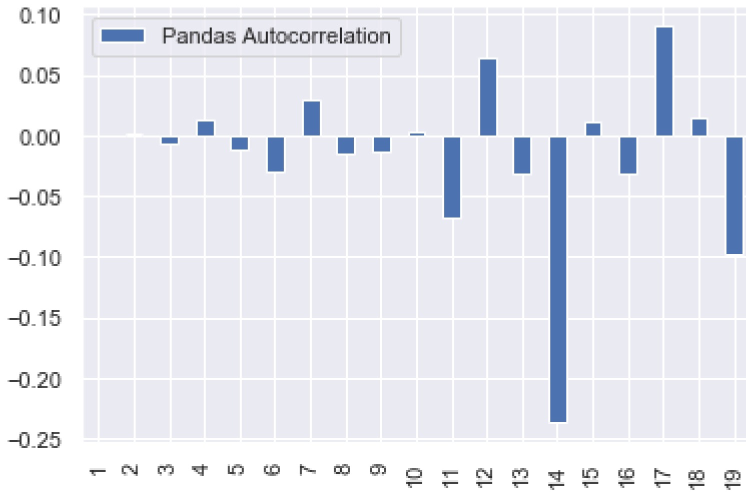





```
                  0
count    3036.000000
mean      -16.175859
std     75115.829895
min   -121005.889564
25%    -53349.738202
50%     -9845.241997
75%     35559.554853
max    275850.987992
```

Out[78]: <matplotlib.axes._subplots.AxesSubplot at 0x28415a449b0>





Hybrid ARIMA-ANN model: The Hybrid Model is implemented and MSE is a metric to evaluate the fitness of the model. We have implemented the Sequential model of Keras. It allows us to create models layer-by-layer

In [79]:

```
"""
Hybrid Model
"""
window_size = 50
def make_model(window_size):
    model1 = Sequential()
    model1.add(Dense(50, input_dim=window_size, init="uniform",
    activation="tanh"))
    model1.add(Dense(25, init="uniform", activation="tanh"))
    model1.add(Dense(1))
    model1.add(Activation("linear"))
    model1.compile(loss='mean_squared_error', optimizer='adam')
    return model1



model1 = make_model(50)

min_max_scaler1 = preprocessing.MinMaxScaler()
train = np.array(train).reshape(-1,1)

train_scaled = min_max_scaler1.fit_transform(train)
train_X,train_Y = [],[]
for i in range(0 , len(train_scaled) - window_size):
    train_X.append(train_scaled[i:i+window_size])
    train_Y.append(train_scaled[i+window_size])

new_train_X,new_train_Y = [],[]
for i in train_X:
    new_train_X.append(i.reshape(-1))
for i in train_Y:
    new_train_Y.append(i.reshape(-1))
new_train_X = np.array(new_train_X)
new_train_Y = np.array(new_train_Y)

model1.fit(new_train_X,new_train_Y, nb_epoch=500, batch_size=512, validation_
```

```
Train on 2836 samples, validate on 150 samples
Epoch 1/500
2836/2836 [==============================] - 1s 185us/step - loss: 0.0595
- val_loss: 0.0378
Epoch 2/500
2836/2836 [==============================] - 0s 6us/step - loss: 0.0398 -
val_loss: 0.0384
Epoch 3/500
2836/2836 [==============================] - 0s 6us/step - loss: 0.0330 -
val_loss: 0.0358
Epoch 4/500
2836/2836 [==============================] - 0s 9us/step - loss: 0.0313 -
val_loss: 0.0295
Epoch 5/500
2836/2836 [==============================] - 0s 5us/step - loss: 0.0267 -
val_loss: 0.0265
Epoch 6/500
2836/2836 [==============================] - 0s 6us/step - loss: 0.0232 -
val_loss: 0.0227
```

In [80]:
```
test_extended = train.tolist()[-1*window_size:] + test_resid1
test_data1 = []
for i in test_extended:
    try:
        test_data1.append(i[0])
    except:
        test_data1.append(i)
```

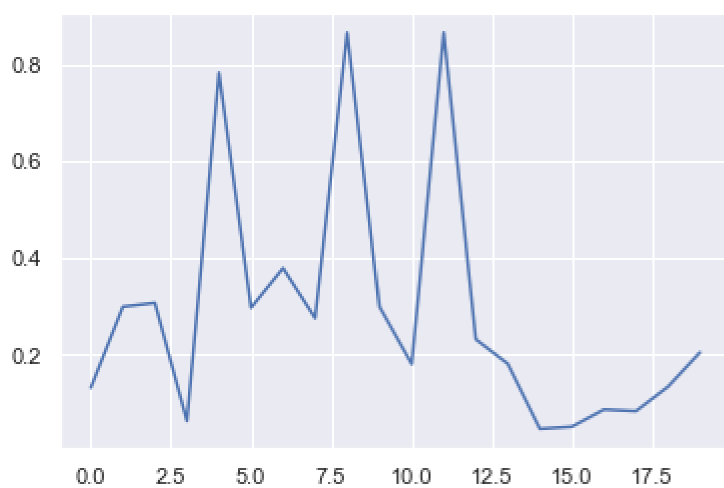
In [81]:
```
test_data1 = np.array(test_data1).reshape(-1,1)
min_max_scaler1 = preprocessing.MinMaxScaler()
test_scaled1 = min_max_scaler1.fit_transform(test_data1)
test_X,test_Y = [],[]
for i in range(0 , len(test_scaled1) - window_size):
    test_X.append(test_scaled1[i:i+window_size])
    test_Y.append(test_scaled1[i+window_size])
    new_test_X,new_test_Y = [],[]
for i in test_X:
    new_test_X.append(i.reshape(-1))
for i in test_Y:
    new_test_Y.append(i.reshape(-1))
```

In [82]:
```
new_test_X = np.array(new_test_X)
new_test_Y = np.array(new_test_Y)
```

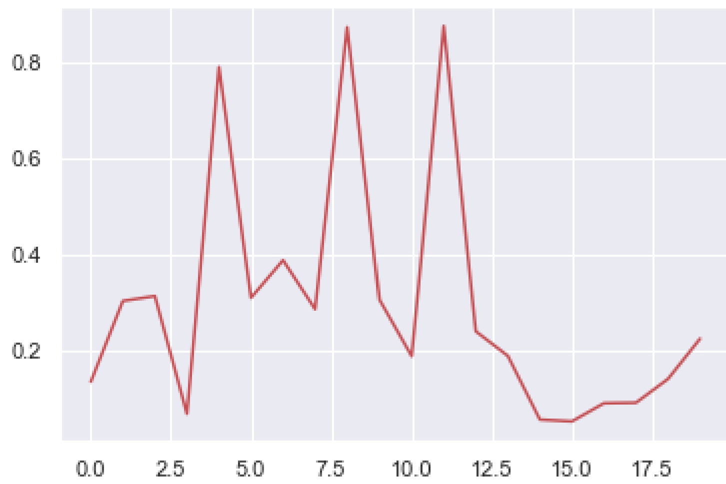
In [83]:
```
# making the model predictions
predictions1 = model1.predict(new_train_X)
predictions_rescaled=min_max_scaler1.inverse_transform(predictions1)
Y1 = pd.DataFrame(new_train_Y)
pred1 = pd.DataFrame(predictions1)
```

In [84]:
```
# plotting the graph for actual and predicted values
plt.plot(Y1.head(20))
```

Out[84]: [<matplotlib.lines.Line2D at 0x28414fcdf98>]



In [85]:
```
plt.plot(pred1.head(20) , color = 'r')
plt.show()
```



Conclusion

1. House prices decreased from Year 2010 and further decreased in Year 2011. House prices picked up in Year 2012 past June

2. The best time to sell houses is from the month of June through October (inclusive). House prices are at a peak during these months across all years

3. The best time to buy a house is in the months of December and January

4. Out of the 3 above models for SARIMA,we can observe that the Model 1 with order = (2, 0, 4) and seasonal_order = (3, 1, 2,12)12 gives the lowest Mean squared error value of 0.174 and also the actual values are very much aligned with predicted values. Hence Model1 for SARIMA gives us the best results

5. As a part of our research, we implemented a hybrid of ARIMA and ANN and cocluded that the hybrid results gives a better result/forecasting value in comparison to the individual model's performance

Citations

https://www.kaggle.com/rgrajan/time-series-exploratory-data-analysis-forecast (https://www.kaggle.com/rgrajan/time-series-exploratory-data-analysis-forecast)

https://towardsdatascience.com/forecasting-with-prophet-d50bbfe95f91 (https://towardsdatascience.com/forecasting-with-prophet-d50bbfe95f91)

https://www.kaggle.com/niyamatalmass/visualization-of-zillow-economics-data\ (https://www.kaggle.com/niyamatalmass/visualization-of-zillow-economics-data%5C)

https://github.com/Kanav123/ArimaAnnHybrid (https://github.com/Kanav123/ArimaAnnHybrid)

https://www.kaggle.com/sudhirnl7/real-estate-price-eda-arima-model (https://www.kaggle.com/sudhirnl7/real-estate-price-eda-arima-model)

Definitions From -

https://www.sciencedirect.com/science/article/pii/S0925231201007020#SEC3 (https://www.sciencedirect.com/science/article/pii/S0925231201007020#SEC3)

https://en.wikipedia.org/wiki/Artificial_neural_network (https://en.wikipedia.org/wiki/Artificial_neural_network)

https://www.zillow.com/corp/About.htm (https://www.zillow.com/corp/About.htm) \
https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/ (https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/)

https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average (https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average)

https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/ (https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/)

https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/ (https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/)

https://facebook.github.io/prophet/ (https://facebook.github.io/prophet/)

Contribution

Code written by self :50 %

Code referred from outside sources :50%

License