Analysis of 77072 zip code

Imports and loading csv

In [141]:
```
#Imports
import pandas as pd
import numpy as np
from pandas.plotting import register_matplotlib_converters
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
register_matplotlib_converters()

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

from scipy import stats
from random import gauss as gs
import datetime

from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose

#Supress default INFO logging
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import logging
logger = logging.getLogger()
logger.setLevel(logging.CRITICAL)
import logging, sys
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [142]:
```
df = pd.read_csv('Data Files/df_zillow_77072_prepped_fbprophet.csv')
```

In [143]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 265 entries, 0 to 264
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   ds      265 non-null    object 
 1   y       265 non-null    float64
dtypes: float64(1), object(1)
memory usage: 4.3+ KB
```

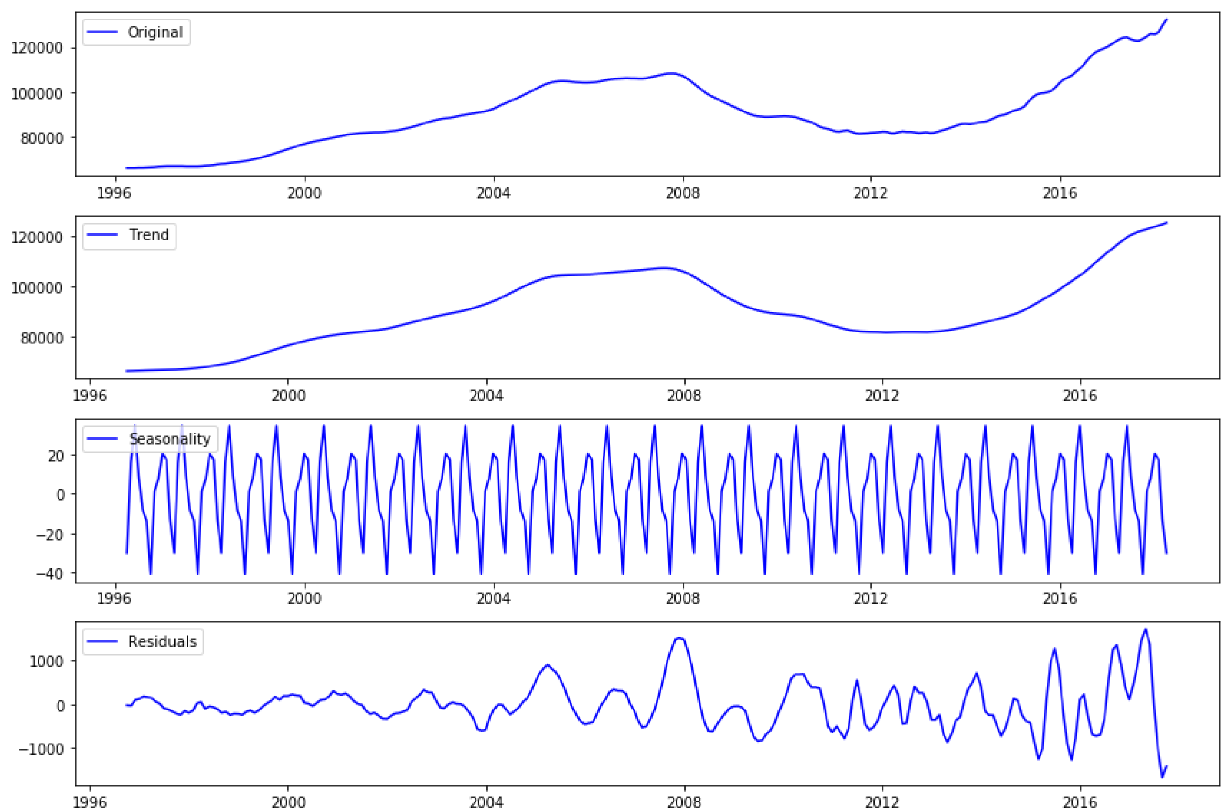
Decomposition and plots

In [144]:
```
df.index = pd.to_datetime(df['ds'])
df= df.drop(columns='ds')
```

In [145]:
```
decomposition = seasonal_decompose(df.y)
observed = decomposition.observed
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
```

In [146]:
```
register_matplotlib_converters()
```

In [147]:
```
plt.figure(figsize=(12,8))
plt.subplot(411)
plt.plot(observed, label='Original', color="blue")
plt.legend(loc='upper left')
plt.subplot(412)
plt.plot(trend, label='Trend', color="blue")
plt.legend(loc='upper left')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality', color="blue")
plt.legend(loc='upper left')
plt.subplot(414)
plt.plot(residual, label='Residuals', color="blue")
plt.legend(loc='upper left')
plt.tight_layout()
```



I want to see if the data correlates with earlier data of

itself

1) Get rolling average with window of 4

```
- Couldn't see much with window of 1-3
```

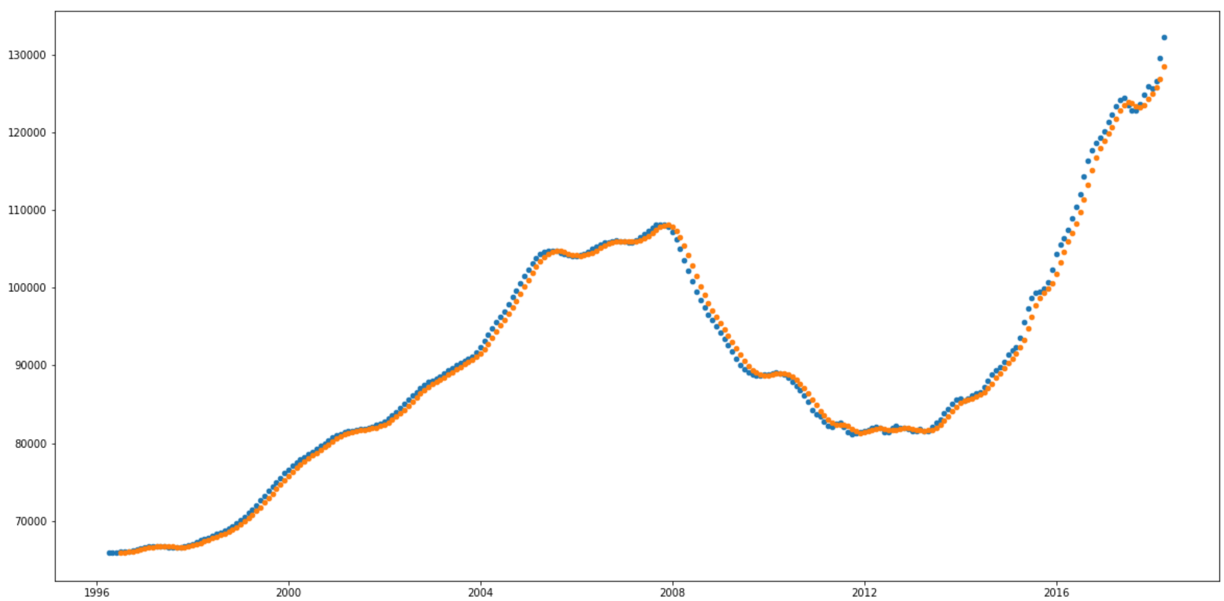
2) Plot data against itself with rolling avg to see visual of the graph.

In [148]:
```
df['roll_avg'] = df.rolling(window=4).mean()
df.corr()
```

Out[148]:

	y	roll_avg
y	1.000000	0.998385
roll_avg	0.998385	1.000000

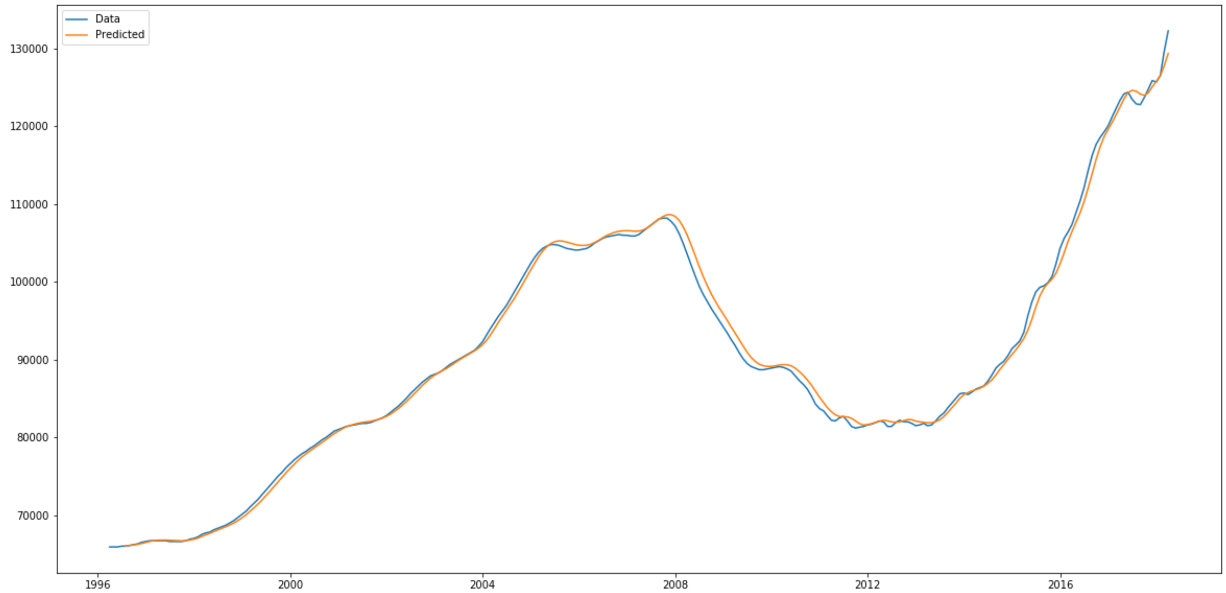
In [149]:
```
plt.figure(figsize=(20, 10))
plt.scatter(df.index[:265], df['y'][:265], s=20)
plt.scatter(df.index[1:265], df['roll_avg'][1:265], s=20);
```



In [150]:
```
lr = LinearRegression()
lr.fit(df[['roll_avg']][4:], df['y'][4:])
```

Out[150]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [151]:
```
plt.figure(figsize=(20, 10))
plt.plot(df.index[:265], df['y'][:265], label='Data')
plt.plot(df.index[4:265], lr.predict(df[['roll_avg']][4:265]),
         label='Predicted')
plt.legend();
```



Upon brief visual look, there might be some correlation. We will set up for our model by using the Dickey-Fuller test and ACF (Auto-correlation) and PACF (Partial-autocorrelation)

Checking for Stationarity

In [152]:
```
dftest = adfuller(df.y)
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used',
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dftest)
print()
print(dfoutput)
```

```
(-0.7060649545040951, 0.8451723171119757, 12, 252, {'1%': -3.4565688966099373,
'5%': -2.8730786194395455, '10%': -2.5729189953388762}, 3452.629967349372)

Test Statistic                  -0.706065
p-value                          0.845172
#Lags Used                      12.000000
Number of Observations Used    252.000000
Critical Value (1%)             -3.456569
Critical Value (5%)             -2.873079
Critical Value (10%)            -2.572919
dtype: float64
```

Dickey Fuller Test

```
- We see that test statistic value is -0.706065
- We see that the critical values are LESS than the test statistic. (-3.
45, -2.87, -2.57)
- From just the baseline data, the test statistic I have is MORE than th
e critical value.
- We accept the null that the time series is not stationary!
```

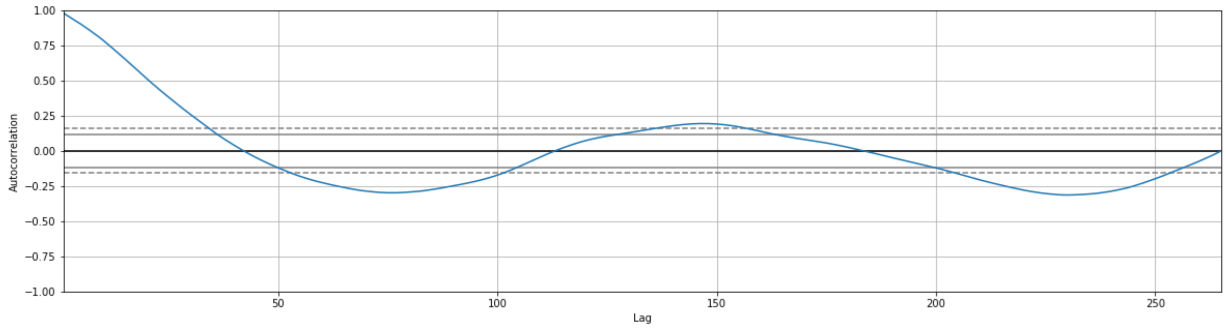
P-Value analysis

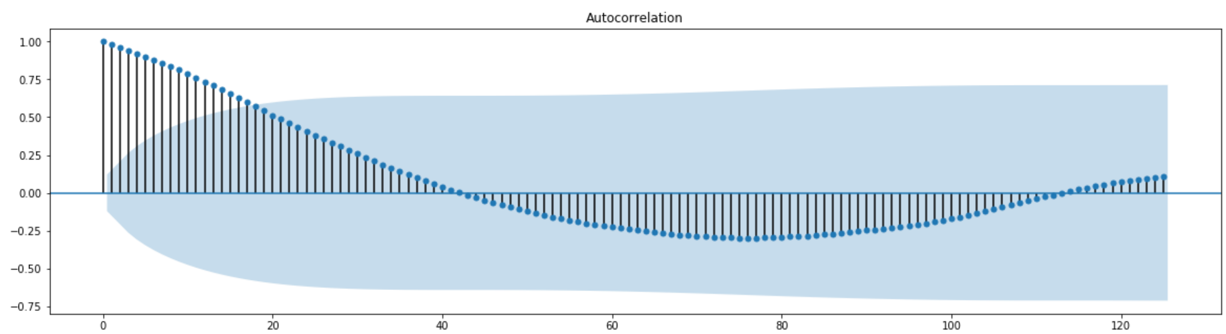
1. If p-value > 0.05: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.
    - Our current p-value is 0.845172
        - This means: p-value > 0.05: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.

2. If p-value <= 0.05: Reject the null hypothesis (H0), the data does not have a unit root and is stationary.
    - Our goal is to make the data stationary

Auto-Correlation and Partial Auto-Correlation Check

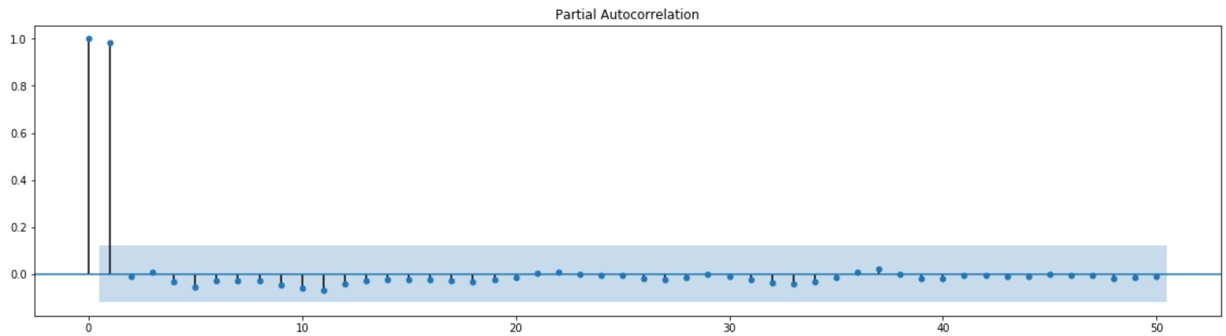
In [153]:
```
#ACF using plotting
plt.figure(figsize=(20, 5))
pd.plotting.autocorrelation_plot(df['y']);

#Statsmodels ACF
rcParams['figure.figsize'] = 20, 5
plot_acf(df['y'], lags=125, alpha=0.05);
```





PACF

In [154]:
```
pacf(df['y'], nlags=20)
rcParams['figure.figsize'] = 20, 5
plot_pacf(df['y'], lags=50, alpha=0.05);
```



Observations of ACF and PACF

We see the following:

- We know that the ACF describes the autocorrelation between an observation and another observation at a prior time step that includes direct and indirect dependence information.

```
  - After about 17-18 lags, the line goes into our confidence interv
al (light blue area).
  - This can be due to seasonality of every 18 months in our data.
```

- We know that the PACF only describes the direct relationship between an observation and its lag.

```
  - PACF cuts off after lags = 2
  - This means there are no correlations for lags beyond 2
```

** Granted the data is not stationary, we will have to transform the data to make it stationary and satisfy the Dicky-Fuller test**

De-trending and transforming the data

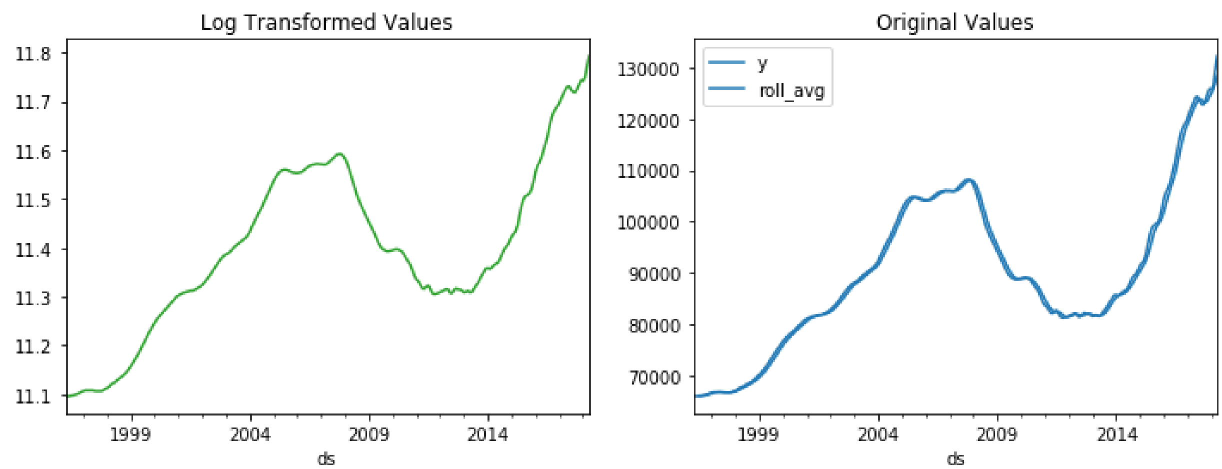
1. I will try the following
    - Log transform
    - Subtract rolling mean

- Run Dickey-Fuller test with each transform to see if I can rejefct/accept the null hypothesis
- Null-Hypothesis for Dickey-Fuller test is: The null-hypothesis for the test is that the time series is not stationary. So if the test statistic is less than the critical value, we reject the null hypothesis and say that the series is stationary.

Log-transform on data and testing for stationarity

In [155]:
```
logged_df = df['y'].apply(lambda x : np.log(x))
```

In [156]:
```
ax1 = plt.subplot(121)
logged_df.plot(figsize=(12,4) ,color="tab:green", title="Log Transformed Values"
ax2 = plt.subplot(122)
df.plot(color="tab:blue", title="Original Values", ax=ax2);
```



In [157]:
```
dftest = adfuller(logged_df)
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used'
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dftest)
print()
print(dfoutput)
```

```
(-1.1458067631277982, 0.6964635146228962, 12, 252, {'1%': -3.4565688966099373,
'5%': -2.8730786194395455, '10%': -2.5729189953388762}, -2273.0439965308883)

Test Statistic                  -1.145807
p-value                          0.696464
#Lags Used                      12.000000
Number of Observations Used    252.000000
Critical Value (1%)             -3.456569
Critical Value (5%)             -2.873079
Critical Value (10%)            -2.572919
dtype: float64
```

Observations after log-transform

1. Test Statistic is still larger than Critical Values. We accept the null-hypothesis that the time series is not stationary!

    - Test Statistic -1.145807
    - Critical Value (1%) -3.456360
    - Critical Value (5%) -2.872987
    - Critical Value (10%) -2.572870

2. P value is 0.696464
    - This means: p-value > 0.05: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.

Subtracting Rolling Mean from logged data and a better window size

In [158]:
```
#Try breakdown with data minus rollmean. It looks like there is seasonality but i
# Window of 11

logged_df_roll_mean = logged_df.rolling(window=11).mean()
logged_df_minus_roll_mean1 = logged_df - logged_df_roll_mean
logged_df_minus_roll_mean1.dropna(inplace=True)
```

In [159]:
```
logged_df_minus_roll_mean1.head()
```

Out[159]:
```
ds
1997-02-01    0.007671
1997-03-01    0.006574
1997-04-01    0.005477
1997-05-01    0.004380
1997-06-01    0.003421
Name: y, dtype: float64
```

In [160]:
```python
dftest = adfuller(logged_df_minus_roll_mean1)
# Extract and display test results in a user friendly manner
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used'
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dftest)
print()
print(dfoutput)
```

```
(-1.596689364254372, 0.48525631721789897, 12, 242, {'1%': -3.457664132155201,
'5%': -2.8735585105960224, '10%': -2.5731749894132916}, -2216.5553842190684)

Test Statistic                  -1.596689
p-value                          0.485256
#Lags Used                      12.000000
Number of Observations Used    242.000000
Critical Value (1%)             -3.457664
Critical Value (5%)             -2.873559
Critical Value (10%)            -2.573175
dtype: float64
```

--- Observations from Dickey Fuller Test ---

We are getting close.

```
- Test statistic is -1.596689 which is higher than crit values
-p value is 0.485256 , I cannot reject null
```

Differencing the data and re-running Dickey Fuller

In [161]:
```python
logged_df_diff = logged_df.diff(periods=1)
```

In [162]:
```python
logged_df_diff_roll_mean = logged_df_diff.rolling(window=11).mean()
logged_df_diff_roll_mean1 = logged_df_diff - logged_df_diff_roll_mean
logged_df_diff_roll_mean1.dropna(inplace=True)
```

In [163]:
```python
logged_df_diff_roll_mean1.head()
```

Out[163]:
```
ds
1997-03-01   -0.001097
1997-04-01   -0.001097
1997-05-01   -0.001097
1997-06-01   -0.000959
1997-07-01   -0.002323
Name: y, dtype: float64
```

In [164]:
```
dftest = adfuller(logged_df_diff_roll_mean1)
# Extract and display test results in a user friendly manner
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used'
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dftest)
print()
print(dfoutput)
```

```
(-4.556078938977154, 0.0001555133374368479, 11, 242, {'1%': -3.457664132155201,
'5%': -2.8735585105960224, '10%': -2.5731749894132916}, -2206.246504084256)

Test Statistic                   -4.556079
p-value                           0.000156
#Lags Used                       11.000000
Number of Observations Used     242.000000
Critical Value (1%)              -3.457664
Critical Value (5%)              -2.873559
Critical Value (10%)             -2.573175
dtype: float64
```

--- Observations of Dickey-Fuller Test ---

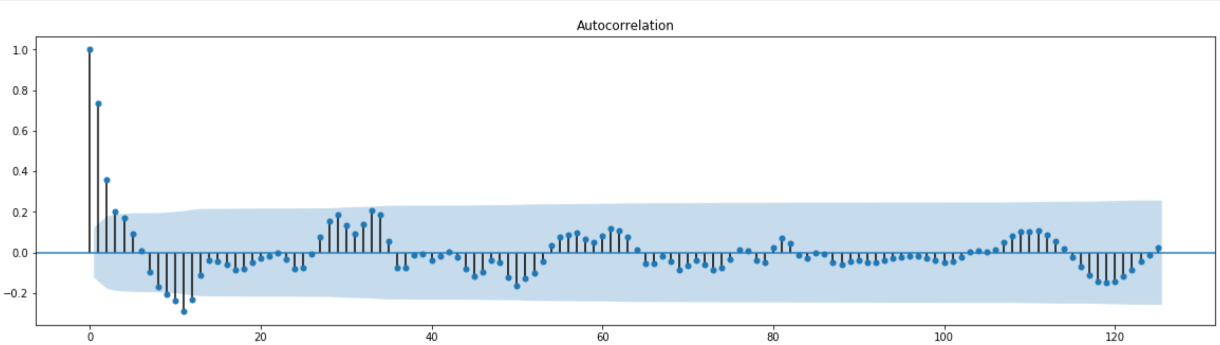
- We see Test Statistic is less than the Critical values, this satisfies the stationarity assumption. We can reject the null and say series is stationary.

```
- Test Statistic                 -4.556079
- Critical Value (1%)            -3.458247
- Critical Value (5%)            -2.873814
- Critical Value (10%)           -2.573311
```

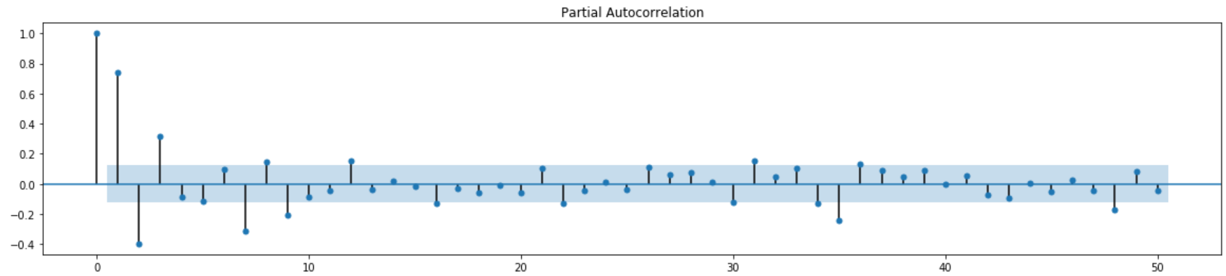
- We see that p-value = 0.000156. Since p <= 0.05, I can reject the null hypothesis (H0 = series is non-stationary). The data does not have a unit root and is stationary.

ACF and PACF

In [165]:
```
#Statsmodels ACF
rcParams['figure.figsize'] = 20, 5
plot_acf(logged_df_diff_roll_mean1, lags=125, alpha=0.05);
```



In [166]:
```
#PACF plot
rcParams['figure.figsize'] = 20, 4
plot_pacf(logged_df_diff_roll_mean1, lags=50, alpha=0.05);
```



--- Observations of ACF and PACF ---

1. After about 4 lags, the line goes into our confidence interval (light blue area).
    - This can be due to seasonality of every 4 months in our data.
2. PACF trails off after 3-4 lags.
    - Also slight slight sinusoidal behavior but nothing crazy
    - This means there are no high correlations for lags beyond 2-3
3. Based on above information and that the data is stationary, we can use the p and q values for the ARMA model
    - p = 4 (per ACF)
    - q = 2,3,4 (per PACF)

ARMA Modeling

In [167]:
```python
# Instantiate & fit model with statsmodels
#p = num lags - ACF
p = 4

 # q = lagged forecast errors - PACF
q = 4

# Fitting ARMA model and summary
ar = ARMA(logged_df_minus_roll_mean1,(p, q)).fit()
ar.summary()
```

Out[167]:

ARMA Model Results

Dep. Variable:	y	**No. Observations:**	255
Model:	ARMA(4, 4)	**Log Likelihood**	1196.797
Method:	css-mle	**S.D. of innovations**	0.002
Date:	Thu, 29 Apr 2021	**AIC**	-2373.594
Time:	14:17:07	**BIC**	-2338.181
Sample:	02-01-1997	**HQIC**	-2359.349
	- 04-01-2018		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0159	0.011	1.464	0.145	-0.005	0.037
ar.L1.y	2.6469	0.101	26.121	0.000	2.448	2.845
ar.L2.y	-2.4693	0.281	-8.798	0.000	-3.019	-1.919
ar.L3.y	0.7846	0.281	2.797	0.006	0.235	1.334
ar.L4.y	0.0297	0.101	0.294	0.769	-0.168	0.228
ma.L1.y	-0.4997	0.081	-6.133	0.000	-0.659	-0.340
ma.L2.y	-0.5621	0.092	-6.123	0.000	-0.742	-0.382
ma.L3.y	0.1243	0.063	1.968	0.050	0.001	0.248
ma.L4.y	0.6447	0.067	9.687	0.000	0.514	0.775

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.0440	-0.0000j	1.0440	-0.0000
AR.2	0.9467	-0.4502j	1.0483	-0.0707
AR.3	0.9467	+0.4502j	1.0483	0.0707
AR.4	-29.3860	-0.0000j	29.3860	-0.5000
MA.1	0.8781	-0.4785j	1.0000	-0.0794
MA.2	0.8781	+0.4785j	1.0000	0.0794
MA.3	-0.9745	-0.7755j	1.2454	-0.3930
MA.4	-0.9745	+0.7755j	1.2454	0.3930

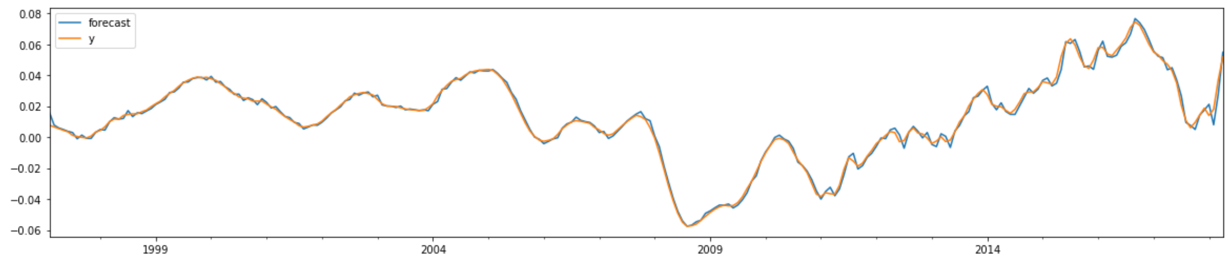
In [168]:
```
r2_score(logged_df_minus_roll_mean1, ar.predict())
```

Out[168]:
```
0.9930234354656668
```

- Ths means that 99.3 percent of the variation in the y data is due to variation in the x data
- This might indicate overfitting, but we chose our params from a stationary time series ACF and PACF.

-Future work: investigate more tweaks to the model

In [169]:
```
#plot of ARMA model
fig = ar.plot_predict()
```



Change the params, maybe it will affect r^2

In [170]:
```
# Try p = 4 and q = 2


# Instantiate & fit model with statsmodels
#p = num lags - ACF
p = 4

 # q = lagged forecast errors - PACF
q = 2

# Fitting ARMA model and summary
ar = ARMA(logged_df_minus_roll_mean1,(p, q)).fit()
ar.summary()
```

Out[170]:

ARMA Model Results

Dep. Variable:	y	**No. Observations:**	255
Model:	ARMA(4, 2)	**Log Likelihood**	1189.035
Method:	css-mle	**S.D. of innovations**	0.002
Date:	Thu, 29 Apr 2021	**AIC**	-2362.069
Time:	14:17:08	**BIC**	-2333.739
Sample:	02-01-1997	**HQIC**	-2350.674
	- 04-01-2018		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	0.0158	0.010	1.597	0.112	-0.004	0.035
ar.L1.y	1.1351	0.123	9.251	0.000	0.895	1.376
ar.L2.y	-0.3223	0.262	-1.231	0.219	-0.835	0.191
ar.L3.y	0.4592	0.284	1.619	0.107	-0.097	1.015
ar.L4.y	-0.3089	0.140	-2.209	0.028	-0.583	-0.035
ma.L1.y	1.0760	0.120	8.988	0.000	0.841	1.311
ma.L2.y	0.6758	0.076	8.867	0.000	0.526	0.825

Roots

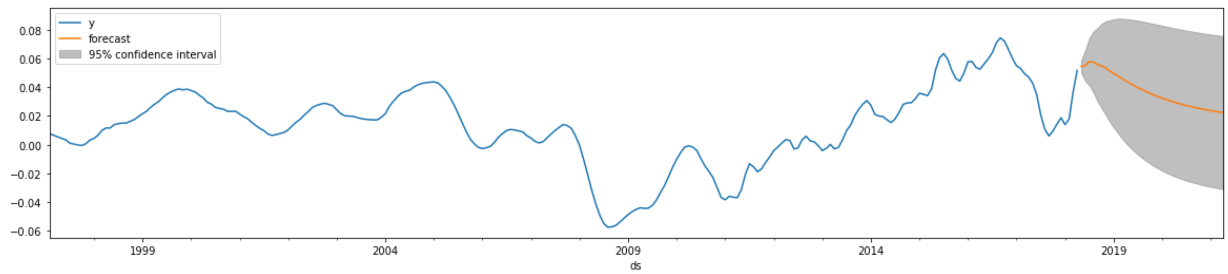
	Real	**Imaginary**	**Modulus**	**Frequency**
AR.1	-0.5156	-1.3517j	1.4467	-0.3080
AR.2	-0.5156	+1.3517j	1.4467	0.3080
AR.3	1.0638	-0.0000j	1.0638	-0.0000
AR.4	1.4542	-0.0000j	1.4542	-0.0000
MA.1	-0.7961	-0.9197j	1.2164	-0.3635
MA.2	-0.7961	+0.9197j	1.2164	0.3635

In [171]:
```
#Slightly lower r^2...hmm
r2_score(logged_df_minus_roll_mean1, ar.predict())
```

Out[171]: 0.9925556183138879

Forecasting

In [172]:
```
#plot of ARMA model
fig, ax = plt.subplots()
ax = logged_df_minus_roll_mean1.plot(ax=ax)
fig = ar.plot_predict('2018-05-01', '2021-04-01', dynamic=True, ax=ax, plot_insam
plt.show()
```



Lower prices might be good indication to buy.

In [173]:
```
#Future work, try SARIMAX prediction
# Need to install modules properly for SARIMAX to work.
```