Analysis of 77095 zip code

Imports and loading csv

In [297]:
```
#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 [298]:
```
df = pd.read_csv('Data Files/df_zillow_77095_prepped_fbprophet.csv')
```

In [299]:
```
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
```

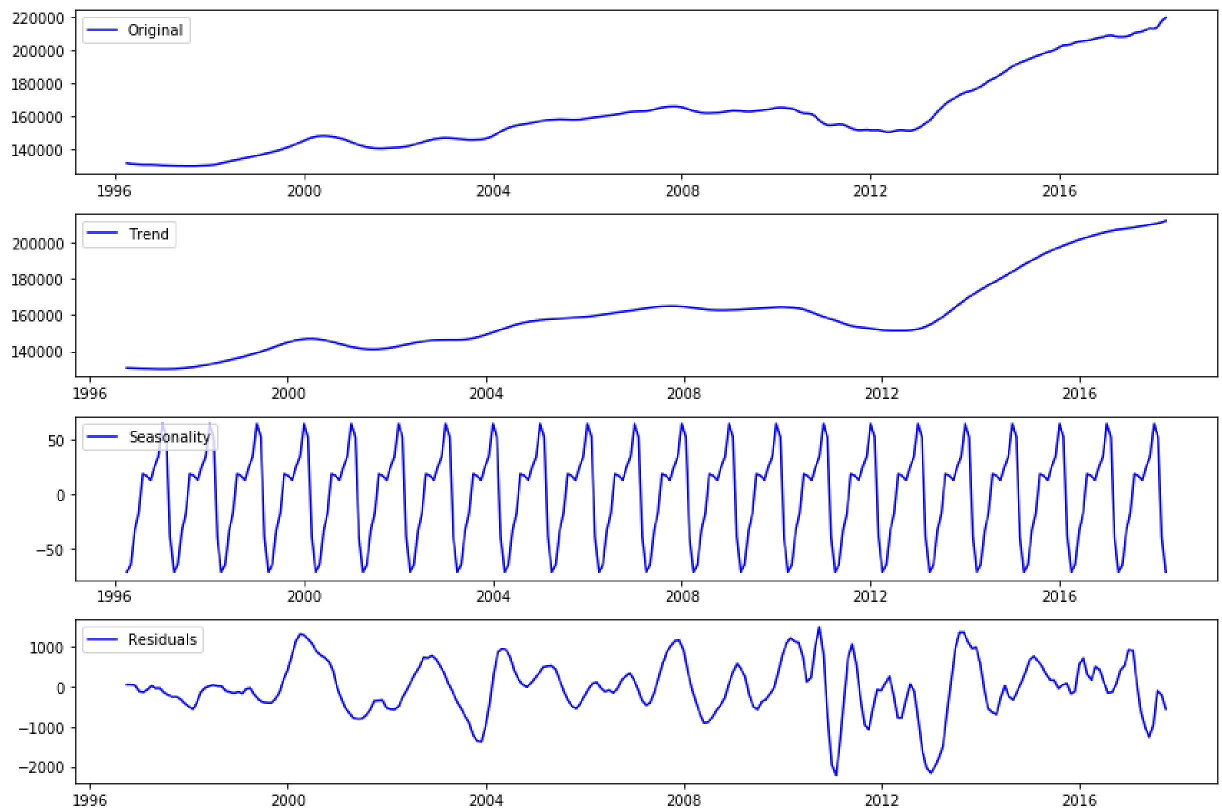
Decomposition and plots

In [300]:
```
df.index = pd.to_datetime(df['ds'])
df= df.drop(columns='ds')
```

In [301]:
```
decomposition = seasonal_decompose(df.y)
observed = decomposition.observed
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
```

In [302]:
```
register_matplotlib_converters()
```

In [303]:
```
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

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

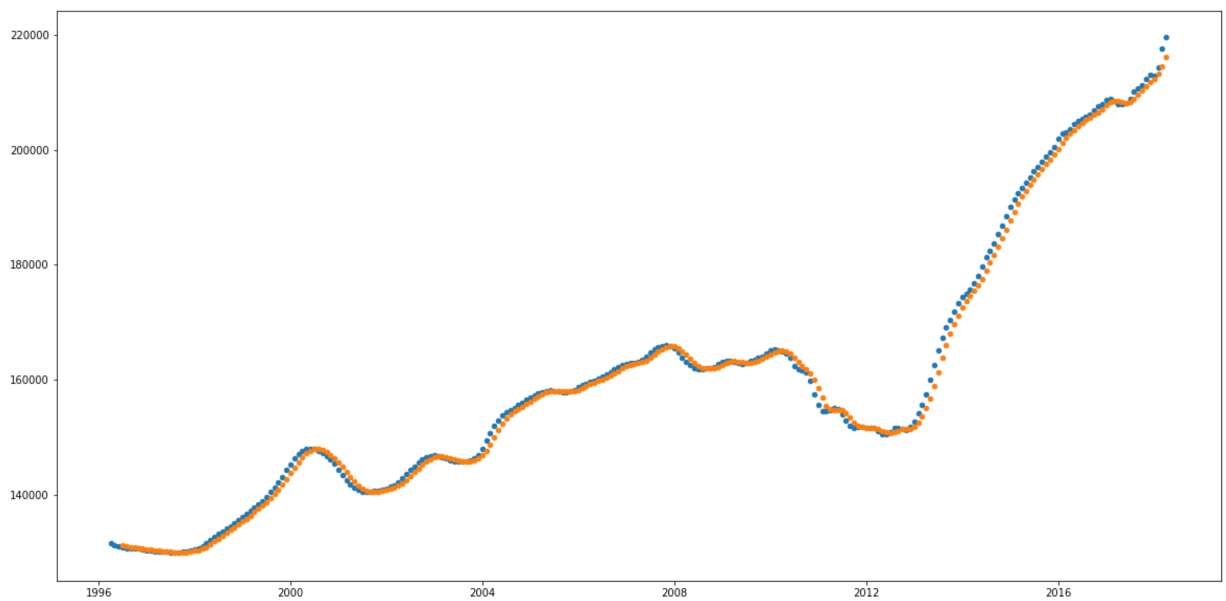
2) Plot data against itself with rolling avg to see visual of the graph.

In [304]:
```
df['roll_avg'] = df.rolling(window=4).mean()
df.corr()
```

Out[304]:

	y	roll_avg
y	1.000000	0.999088
roll_avg	0.999088	1.000000

In [305]:
```
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 [306]:
```
lr = LinearRegression()
lr.fit(df[['roll_avg']][4:], df['y'][4:])
```

Out[306]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [307]:
```
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 [308]:
```
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)
```

```
(0.5389081984501616, 0.9860089138645396, 12, 252, {'1%': -3.4565688966099373,
'5%': -2.8730786194395455, '10%': -2.5729189953388762}, 3500.599686448273)

Test Statistic                   0.538908
p-value                          0.986009
#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 HUGE 0.538908
- 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

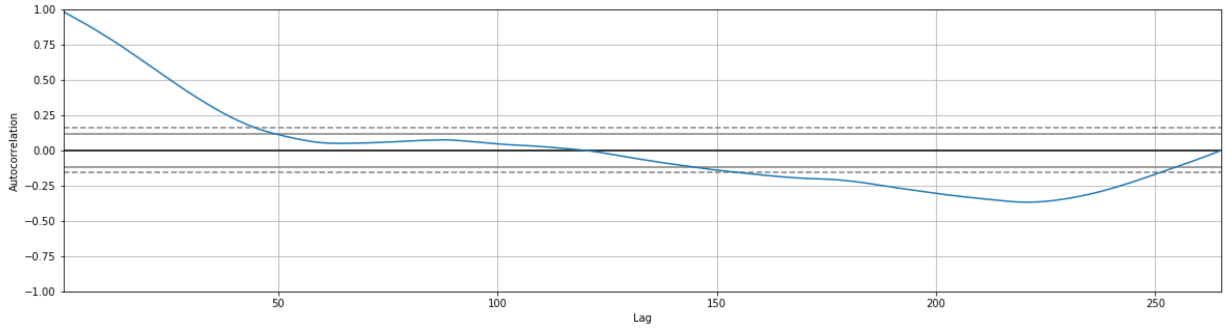
1. If p-value > 0.05: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.
    - Our current p-value is 0.986009
        - This means: p-value > 0.05: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.

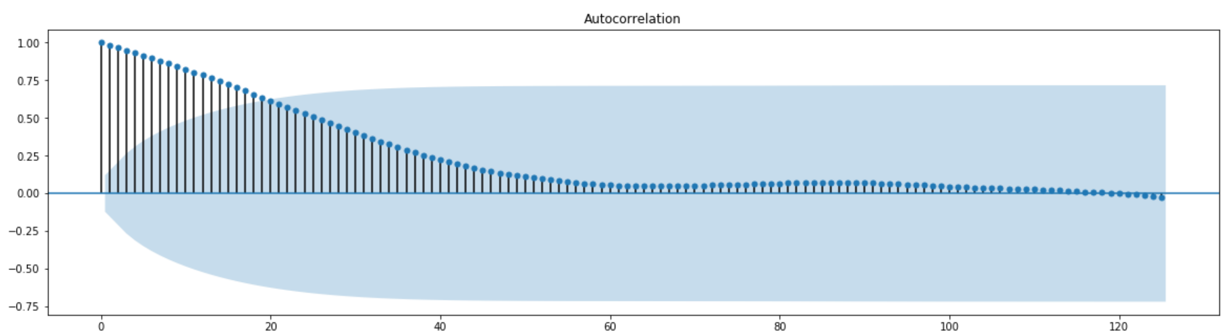
2. If p-value <= 0.05: Reject the null hypothesis (H0), the data does not have a unit root and is stationary.
    - Our goal is to make the data stationary

Auto-Correlation and Partial Auto-Correlation Check

In [309]:

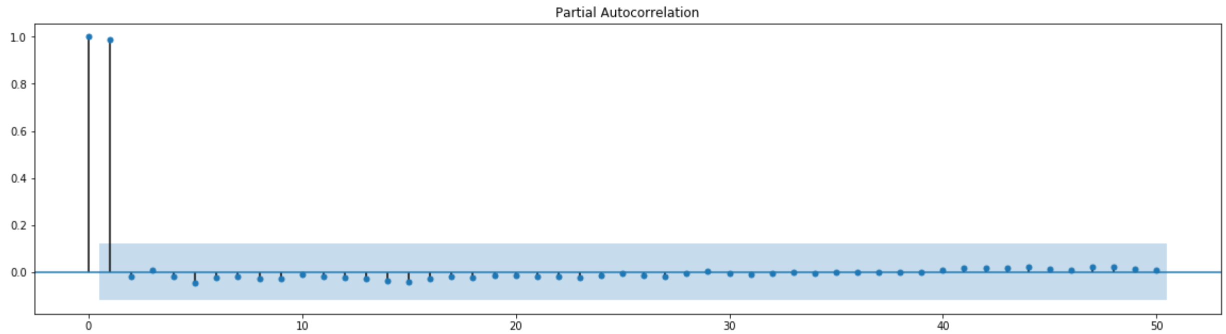
```
#ACF using plotting
plt.figure(figsize=(20, 5))
pd.plotting.autocorrelation_plot(df['y']);

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





PACF

In [310]:
```
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 18 - 19 lags, the line goes into our confidence inte
rval (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

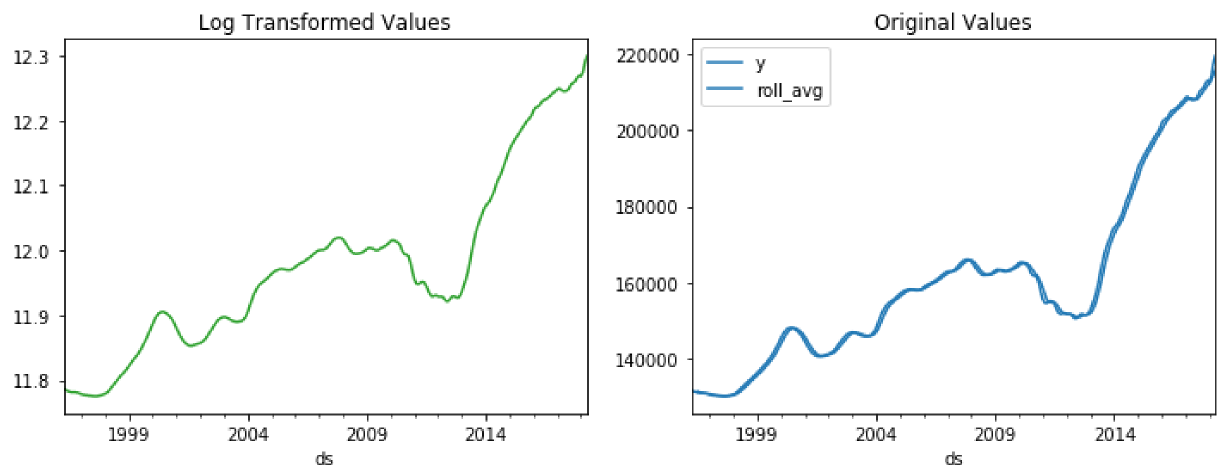
1. I will try the following
   - Log transform
   - Subtract rolling mean

- 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 [311]:
```
logged_df = df['y'].apply(lambda x : np.log(x))
```

In [312]:
```
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 [313]:
```
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)
```

```
(0.19842744574733442, 0.9721732561111457, 7, 257, {'1%': -3.4560535712549925,
'5%': -2.8728527662442334, '10%': -2.5727985212493754}, -2482.323050639333)

Test Statistic                   0.198427
p-value                          0.972173
#Lags Used                       7.000000
Number of Observations Used    257.000000
Critical Value (1%)             -3.456054
Critical Value (5%)             -2.872853
Critical Value (10%)            -2.572799
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 0.198427
    - Critical Value (1%) -3.456360
    - Critical Value (5%) -2.872987
    - Critical Value (10%) -2.572870
2. P value is 0.972173
    - 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 [314]:
```
#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 [315]:
```
logged_df_minus_roll_mean1.head()
```

Out[315]:
```
ds
1997-02-01   -0.004243
1997-03-01   -0.003969
1997-04-01   -0.003204
1997-05-01   -0.003277
1997-06-01   -0.002719
Name: y, dtype: float64
```

In [316]:
```
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)
```

```
(-2.8522643372022785, 0.051187269180728495, 7, 247, {'1%': -3.457105309726321,
'5%': -2.873313676101283, '10%': -2.5730443824681606}, -2422.4572474345664)

Test Statistic                  -2.852264
p-value                          0.051187
#Lags Used                       7.000000
Number of Observations Used    247.000000
Critical Value (1%)             -3.457105
Critical Value (5%)             -2.873314
Critical Value (10%)            -2.573044
dtype: float64
```

--- Observations from Dickey Fuller Test ---

We are getting close.

```
- Test statistic  is -2.852264 which is within the range of the critical
values, lower than 5% and 10%, I can attempt ARMA
- p value is 0.051187
    - p > 0.05, null hypothesis cannot be rejected
```

Differencing the data and re-running Dickey Fuller

In [317]:
```
logged_df_diff = logged_df.diff(periods=1)
```

In [318]:
```
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 [319]:
```
logged_df_diff_roll_mean1.head()
```

Out[319]:
```
ds
1997-03-01    0.000274
1997-04-01    0.000765
1997-05-01   -0.000072
1997-06-01    0.000557
1997-07-01   -0.000211
Name: y, dtype: float64
```

In [320]:
```
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)
```

```
(-6.006073330484669, 1.6143922029352868e-07, 6, 247, {'1%': -3.457105309726321,
'5%': -2.873313676101283, '10%': -2.5730443824681606}, -2405.8863287499526)

Test Statistic                 -6.006073e+00
p-value                         1.614392e-07
#Lags Used                      6.000000e+00
Number of Observations Used     2.470000e+02
Critical Value (1%)            -3.457105e+00
Critical Value (5%)            -2.873314e+00
Critical Value (10%)           -2.573044e+00
dtype: float64
```

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

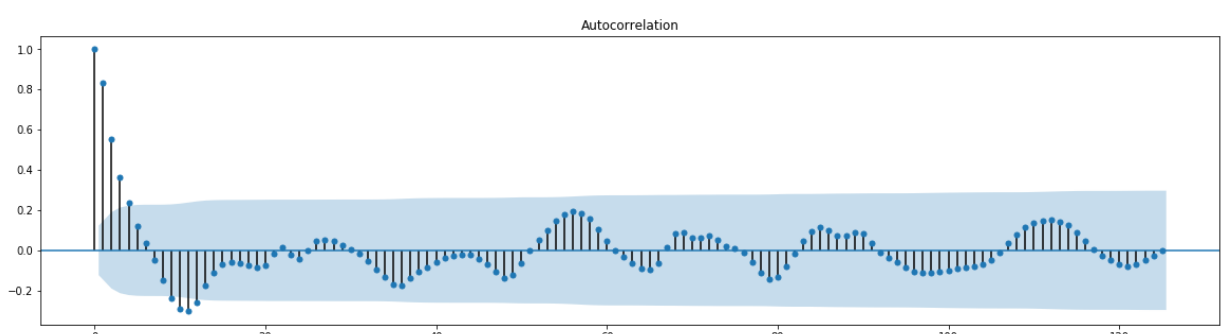
- We see Test Statistic is less than the Critical values, this satisfies the stationarity assumption. We can reject the null and say series is stationary.

```
- Test Statistic              -6.00607
- Critical Value (1%)         -3.458247
- Critical Value (5%)         -2.873814
- Critical Value (10%)        -2.573311
```

- We see that p-value = 1.614392e-07 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.

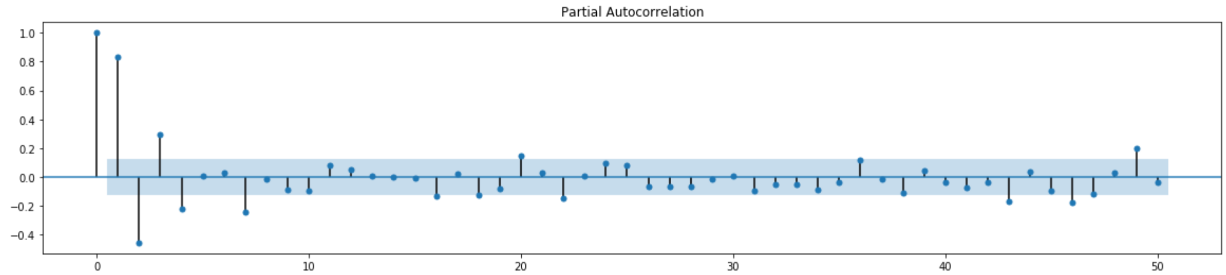
ACF and PACF

In [321]:
```
#Statsmodels ACF
rcParams['figure.figsize'] = 20, 5
plot_acf(logged_df_diff_roll_mean1, lags=125, alpha=0.05);
```



In [322]:

```
#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 - 5 lags, the line goes into our confidence interval (light blue area).
   - This can be due to seasonality of every 4 - 5 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 3-4
3. Based on above information and that the data is stationary, we can use the p and q values for the ARMA model
   - p = 5 (per ACF)
   - q = 4 (per PACF)

ARMA Modeling

In [323]:
```python
# Instantiate & fit model with statsmodels
#p = num lags - ACF
p = 5

 # q = lagged forecast errors - PACF
# 4 and 3 did not work, 2 did
q = 2

# Fitting ARMA model and summary
ar = ARMA(logged_df_diff_roll_mean1,(p, q)).fit()
ar.summary()
```

Out[323]:

ARMA Model Results

Dep. Variable:	y	**No. Observations:**	254
Model:	ARMA(5, 2)	**Log Likelihood**	1303.054
Method:	css-mle	**S.D. of innovations**	0.001
Date:	Thu, 29 Apr 2021	**AIC**	-2588.108
Time:	14:19:15	**BIC**	-2556.272
Sample:	03-01-1997	**HQIC**	-2575.301
	- 04-01-2018		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	0.0001	0.000	0.276	0.782	-0.001	0.001
ar.L1.y	0.4786	0.072	6.673	0.000	0.338	0.619
ar.L2.y	-0.5452	0.074	-7.396	0.000	-0.690	-0.401
ar.L3.y	0.8595	0.057	15.176	0.000	0.748	0.970
ar.L4.y	-0.4746	0.074	-6.407	0.000	-0.620	-0.329
ar.L5.y	0.1541	0.071	2.178	0.030	0.015	0.293
ma.L1.y	0.9449	0.035	26.857	0.000	0.876	1.014
ma.L2.y	0.9528	0.034	28.052	0.000	0.886	1.019

Roots

	Real	**Imaginary**	**Modulus**	**Frequency**
AR.1	-0.4283	-0.9357j	1.0290	-0.3183
AR.2	-0.4283	+0.9357j	1.0290	0.3183
AR.3	1.4196	-0.0000j	1.4196	-0.0000
AR.4	1.2590	-1.6533j	2.0781	-0.1464
AR.5	1.2590	+1.6533j	2.0781	0.1464
MA.1	-0.4959	-0.8964j	1.0244	-0.3304
MA.2	-0.4959	+0.8964j	1.0244	0.3304

In [324]:
```
r2_score(logged_df_diff_roll_mean1, ar.predict())
```

Out[324]:
```
0.7993982126148949
```

- Ths means that 0.799 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

Change the params, maybe it will affect r^2

In [326]:

```python
# Try p = 4 and q = 3


# 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_diff_roll_mean1,(p, q)).fit()
ar.summary()
```

Out[326]:

ARMA Model Results

Dep. Variable:	y	**No. Observations:**	254
Model:	ARMA(4, 2)	**Log Likelihood**	1300.760
Method:	css-mle	**S.D. of innovations**	0.001
Date:	Thu, 29 Apr 2021	**AIC**	-2585.521
Time:	14:19:17	**BIC**	-2557.222
Sample:	03-01-1997	**HQIC**	-2574.136
	- 04-01-2018		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0001	0.000	0.313	0.755	-0.001	0.001
ar.L1.y	0.4312	0.072	6.015	0.000	0.291	0.572
ar.L2.y	-0.4330	0.051	-8.447	0.000	-0.533	-0.332
ar.L3.y	0.7940	0.051	15.582	0.000	0.694	0.894
ar.L4.y	-0.3925	0.067	-5.821	0.000	-0.525	-0.260
ma.L1.y	0.9419	0.033	28.532	0.000	0.877	1.007
ma.L2.y	0.9232	0.068	13.660	0.000	0.791	1.056

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-0.4118	-0.9505j	1.0359	-0.3151
AR.2	-0.4118	+0.9505j	1.0359	0.3151
AR.3	1.4233	-0.5906j	1.5410	-0.0626
AR.4	1.4233	+0.5906j	1.5410	0.0626
MA.1	-0.5101	-0.9072j	1.0407	-0.3315
MA.2	-0.5101	+0.9072j	1.0407	0.3315

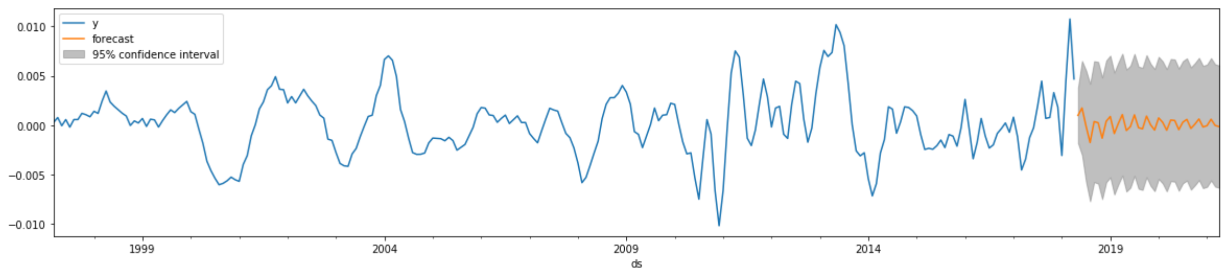
In [327]:
```
#slightly lower r^2, nothing too crazy
r2_score(logged_df_diff_roll_mean1, ar.predict())
```

Out[327]: 0.7956777549104934

This r^2 is not too low nor too high. Could use more data to figure out if it is overfitting or not.

Forecasting

In [328]:
```
#plot of ARMA model
fig, ax = plt.subplots()
ax = logged_df_diff_roll_mean1.plot(ax=ax)
fig = ar.plot_predict('2018-05-01', '2021-04-01', dynamic=True, ax=ax, plot_insa
plt.show()
```



Prices look lower per prediction, could be good to invest.

In [329]:
```
#Future work, try SARIMAX prediction - can account for seasonality
# Need to install modules properly for SARIMAX to work.
```