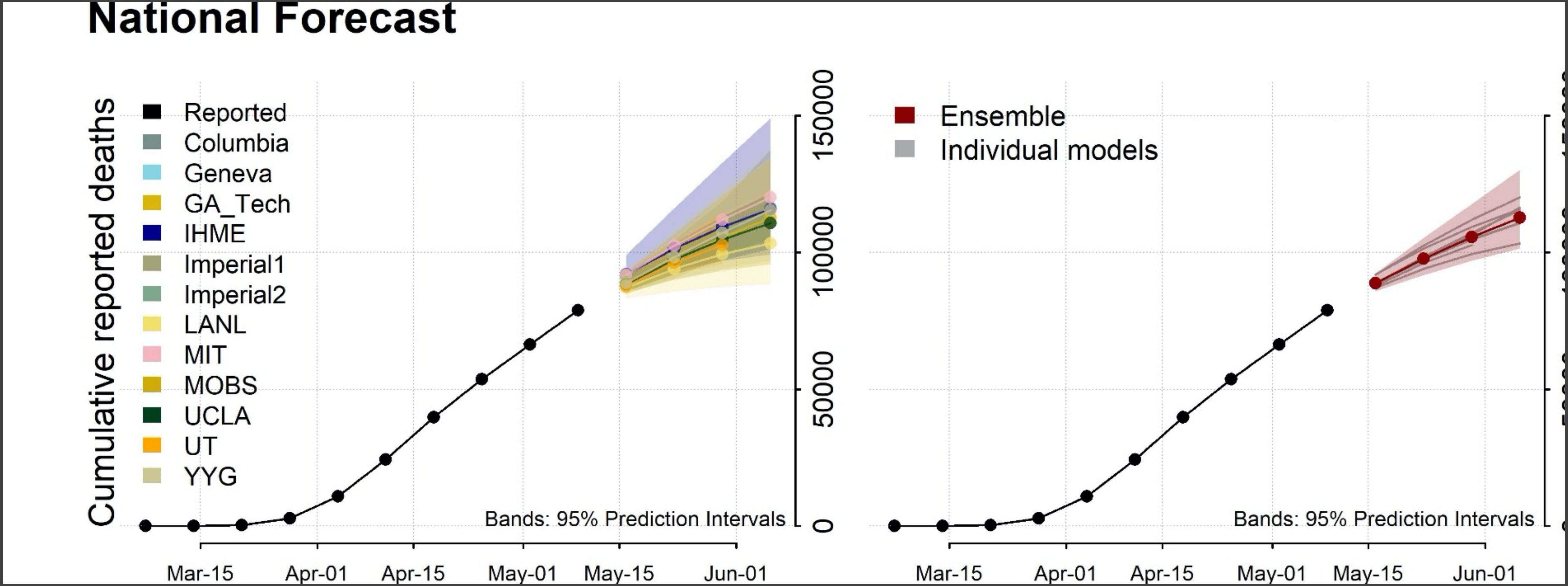
HiPAM

Hawai'i Pandemic Applied Modeling

Coach Blamey’s Auto Regressive Chat

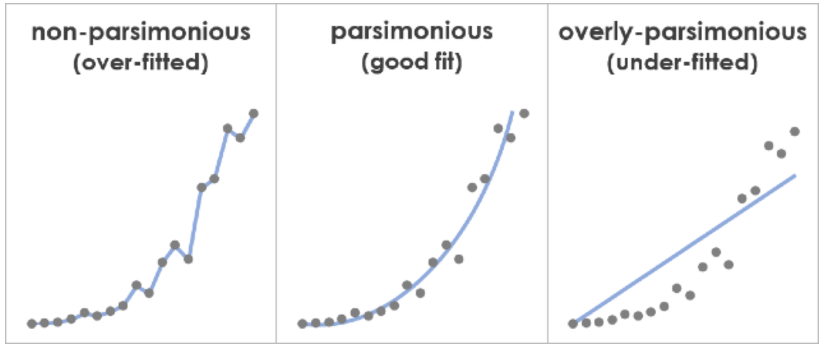
Modeling Covid-19

- Numerous classes of models are being used for Covid-19 forecasting
- Ensemble Models compensate for relative weaknesses in Individual Models while reinforcing their relative strengths (UMVUE).
- [COVID-19-Forecasts/COVID-19_Forecast_Model_Descriptions.md at master · cdcepi/COVID-19-Forecasts · GitHub]



The autoregressive model specifies that the response variable depends linearly on its own previous values, and a stochastic term.

All models are wrong, but some are useful – George Box



Data Wrangling/ Cleaning

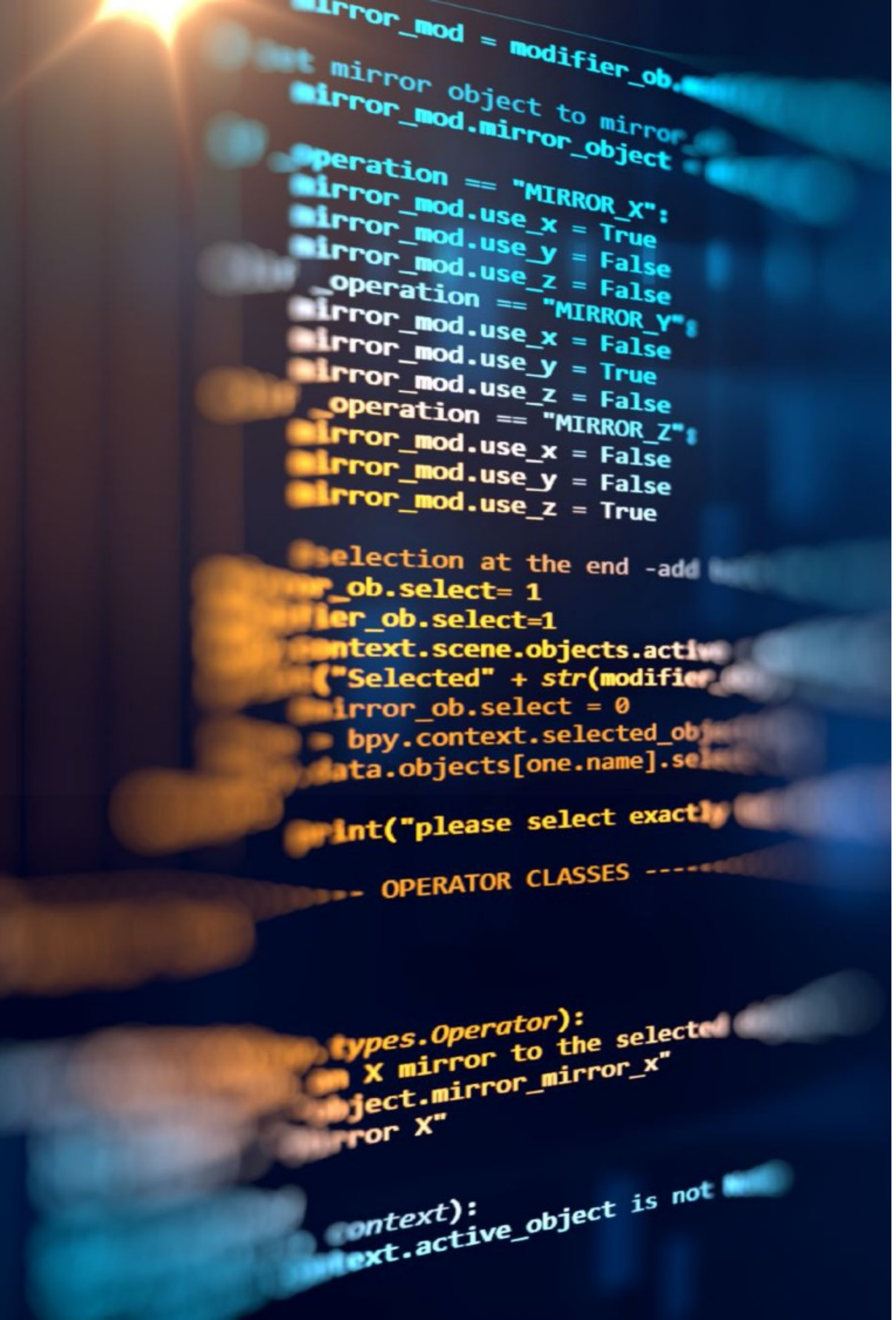
Data wrangling and data cleaning are two processes that we can perform on data to obtain meaningful data. However, the main difference between data wrangling and data cleaning is that data wrangling is the process of converting and mapping data from one format to another format to use that data to perform analyzing while data cleaning is the process of eliminating the incorrect data or to modify them.

[Disease Outbreak Control Division | COVID-19 (hawaii.gov)]

SPYDER

Spyder is a scientific integrated development environment written in Python. This software is designed for and by scientists who can integrate with Matplotlib, SciPy, NumPy, Pandas, Cython, IPython, SymPy, and other open-source software. Spyder is available through Anaconda (open-source distribution system) distribution on Windows, macOS, and Linux.

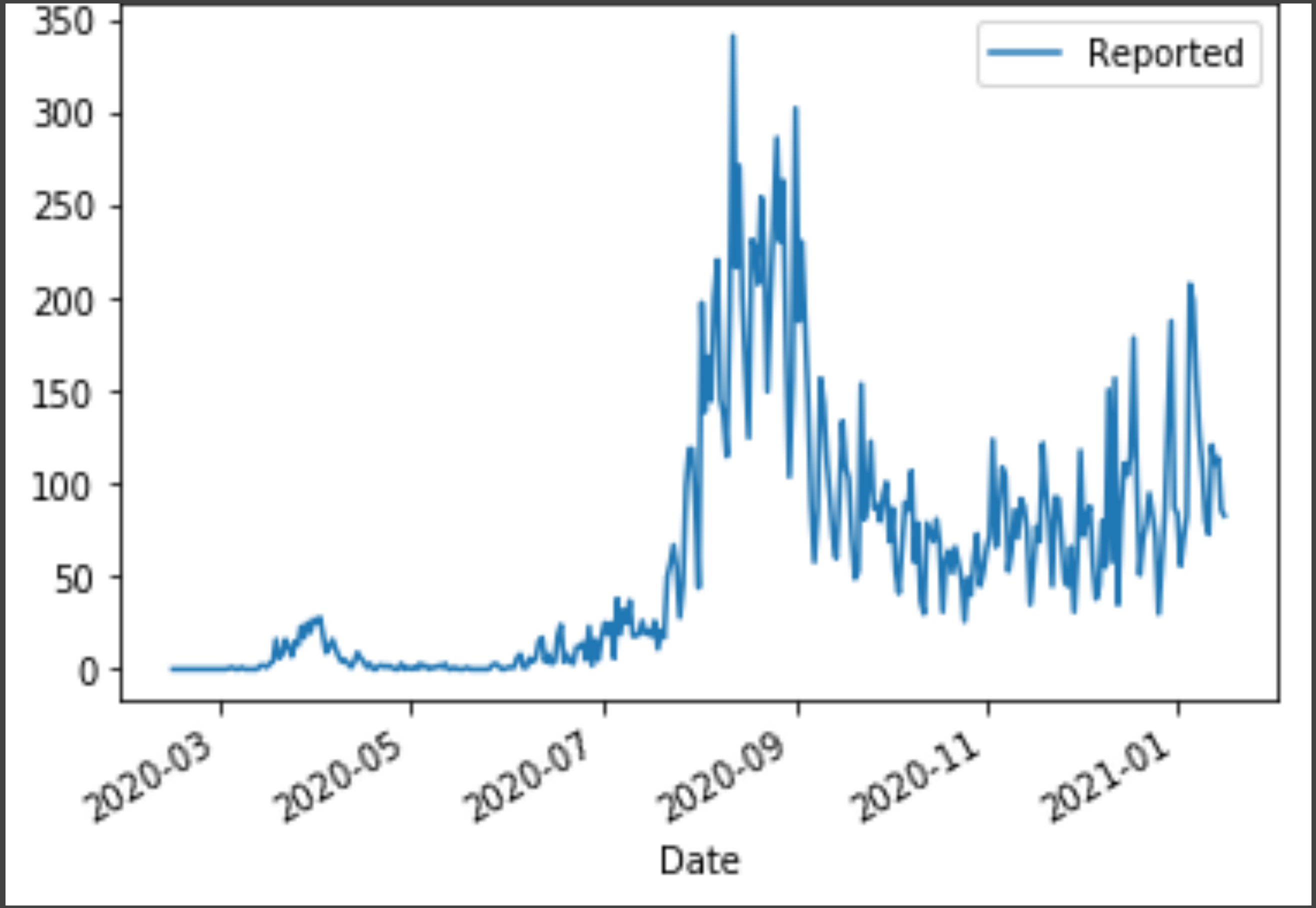
Software development environment



Python Libraries we will be using

- Pandas
- Matplotlib
- NumPy
- Statsmodels
- Pmdarima

Reading Data into Python




```
import pandas as pd

# lambda function that converts a string into datetime

dateparse = lambda dates: pd.datetime.strptime(dates, '%m/
%d/%Y')

# reads the csv into

df = pd.read_csv(r"C:
\TBLAMEY\UH\SP2021\Covid-19\Reported_hono.csv",
parse_dates=['Date'], date_parser=dateparse)

ts = df['Reported']

# sets the date column as the index

df = df.set_index('Date')

df.plot()
```

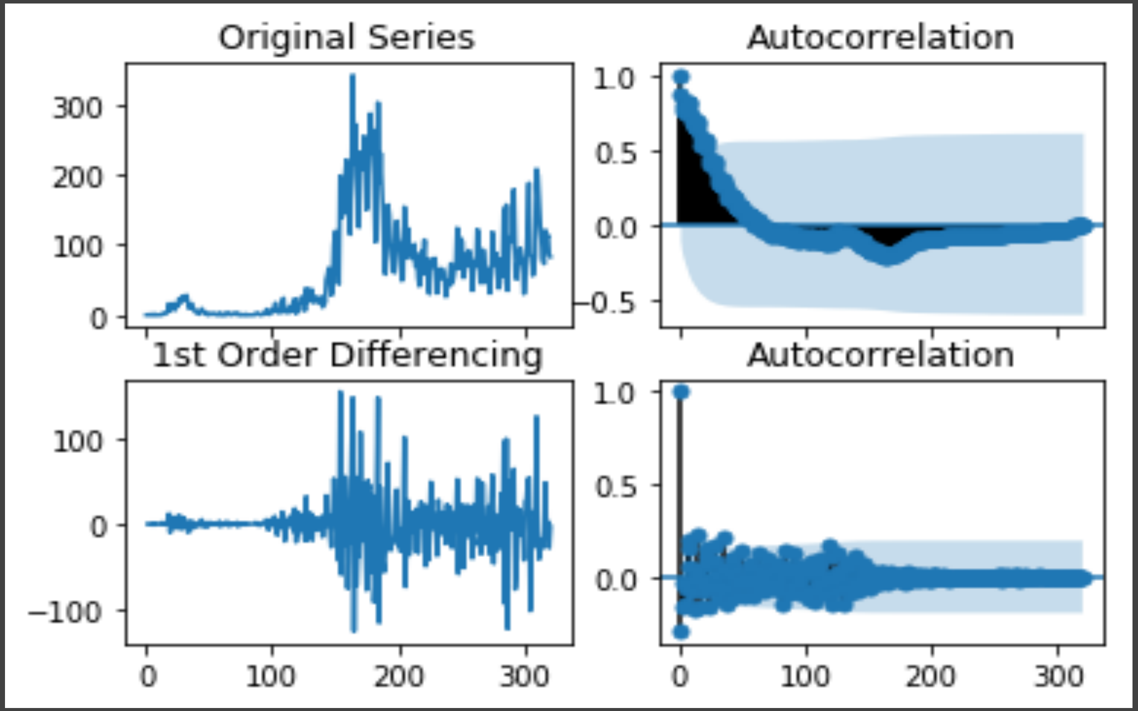
ARIMA Model

$$X_t = c + \epsilon_t + \sum_{i=1}^{p} \phi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$$

- Time series are a quite unique topic within forecasting.
- In time series the explanatory variables are often not known.
- Point is that explanatory variables are not very clear, nor is the explanatory data easily obtainable (as we have seen during the pandemic).
- One of the most used models when handling time series are ARIMA models (where the response variable is dependent only on prior time steps – Auto Regressive family).
- ARIMA models are actually a combination of two, (or three if you count differencing as a model) processes that are able to generate series data. Those two models are based on an Auto Regressive (AR) process and a Moving Average process.
- ARIMA models can work with data that isn't stationary, but instead has a trend.
- **When the data show a trend, we can remove the trend by differencing the time steps.**
- We are assuming no Seasonality (if we assumed Seasonality we would use a SARIMA model).

Testing for stationarity.

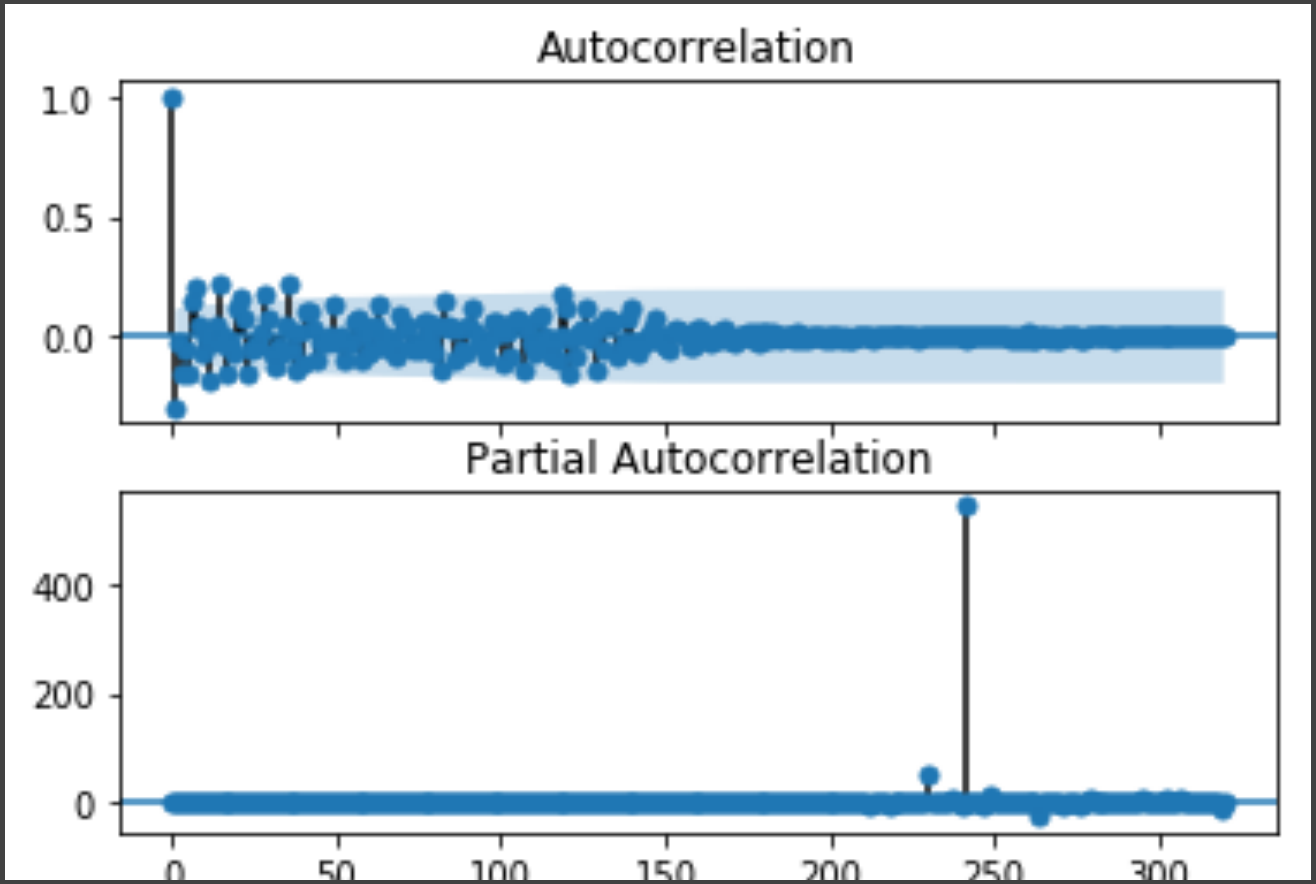
p-value: 0.178207 – Original Data (fail to reject Ho – Non Stationary)



p-value: 0.005823 – 1st Differenced Data (reject Ho – Stationary)

```
import pandas as pd
import numpy as np
import statsmodels as sm
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

#Ho: It is non-stationary vs H1: It is stationary
result = adfuller(ts.dropna())
print('p-value: %f' % result[1])

# Original Series plot
fig, axes = plt.subplots(2, 2, sharex=True)
axes[0, 0].plot(ts); axes[0, 0].set_title('Original Series')
plot_acf(ts, ax=axes[0, 1])

result = adfuller(ts.diff().dropna())
print('p-value: %f' % result[1])

# Differencing stabilizes the mean (Reducing Trend)
# 1st Order Differencing plot
axes[1, 0].plot(ts.diff()); axes[1, 0].set_title('1st Order Differencing')
plot_acf(ts.diff().dropna(), ax=axes[1, 1])
```

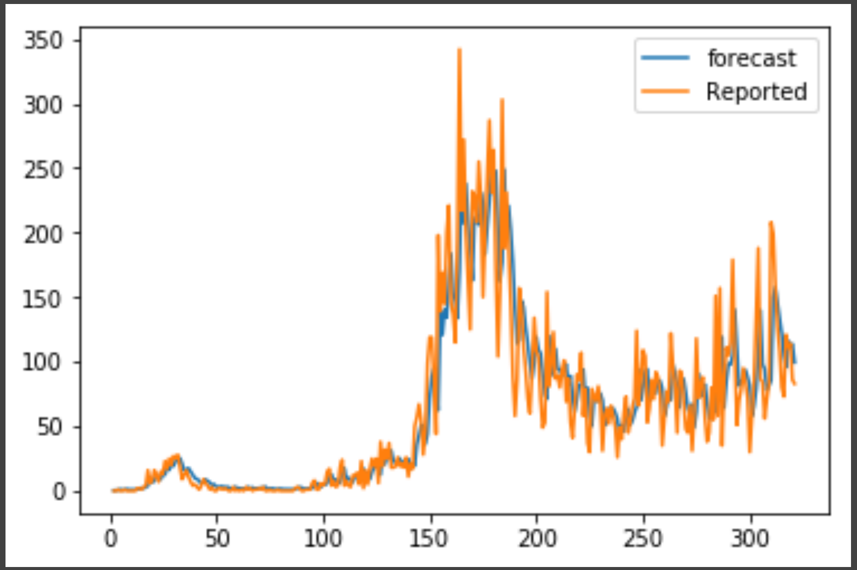
Order of the ARIMA (p,d,q)




```
import numpy as np
import statsmodels as sm
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Original Series plot
fig, axes = plt.subplots(2, 1, sharex=True)
plot_acf(ts.diff().dropna(), ax=axes[0])
plot_pacf(ts.diff().dropna(), ax=axes[1])
```

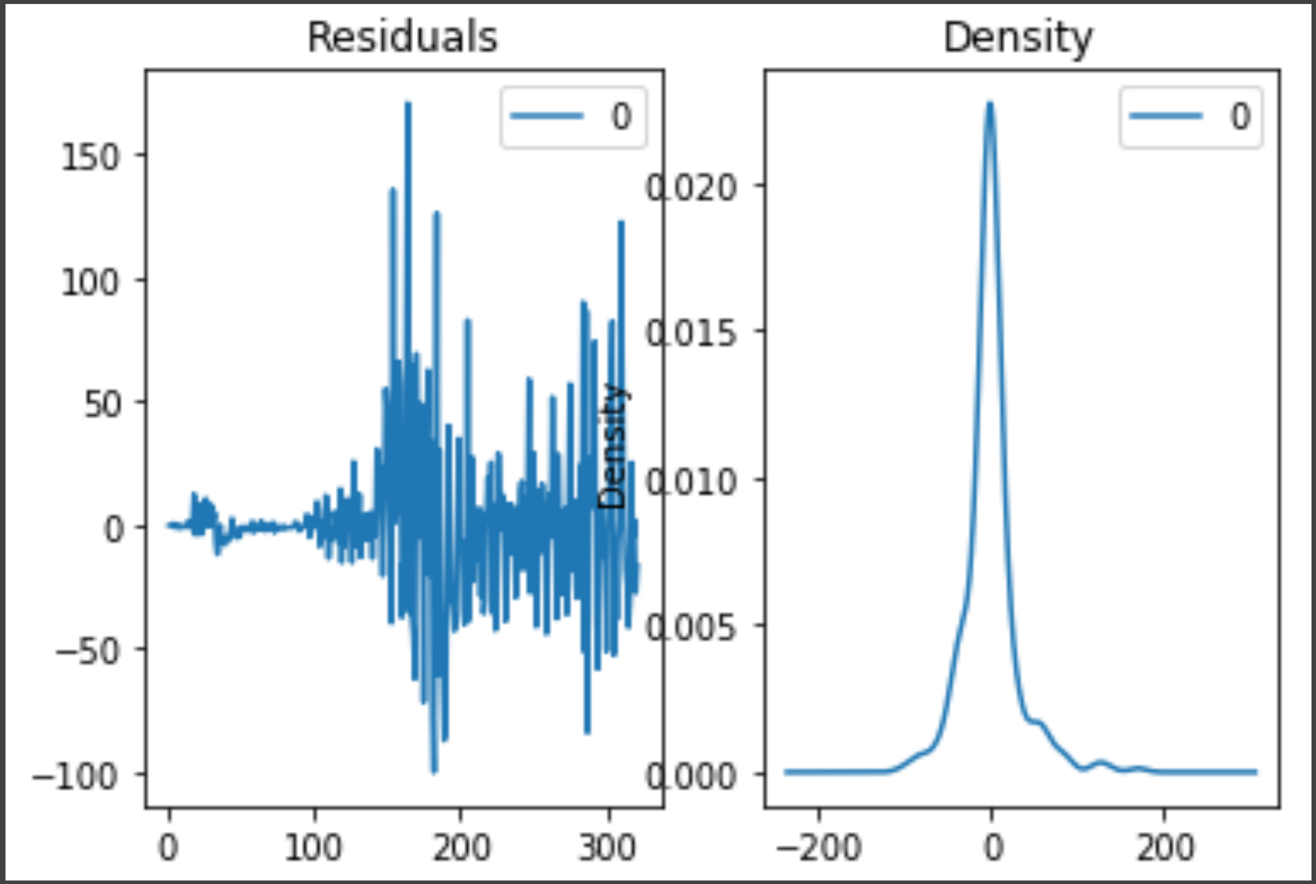
Fitting the ARIMA Model

```
ARIMA Model Results
==============================================================================
Dep. Variable:          D.Reported   No. Observations:              321
Model:              ARIMA(1, 1, 1)   Log Likelihood           -1555.944
Method:                    css-mle   S.D. of innovations         30.797
Date:             Mon, 18 Jan 2021   AIC                       3119.888
Time:                     13:41:13   BIC                       3134.974
Sample:                          1   HQIC                      3125.911

=====================================================================================
                coef   std err        z     P>|z|     [0.025     0.975]
-------------------------------------------------------------------------------------
const             0.3145     0.529      0.594     0.553     -0.723      1.352
ar.L1.D.Reported  0.3054     0.070      4.367     0.000      0.168      0.443
ma.L1.D.Reported -0.7884     0.038    -20.603     0.000     -0.863     -0.713
                        Roots
=============================================================================
          Real       Imaginary       Modulus      Frequency
-----------------------------------------------------------------------------
AR.1       3.2739      +0.0000j       3.2739       0.0000
MA.1       1.2684      +0.0000j       1.2684       0.0000
```



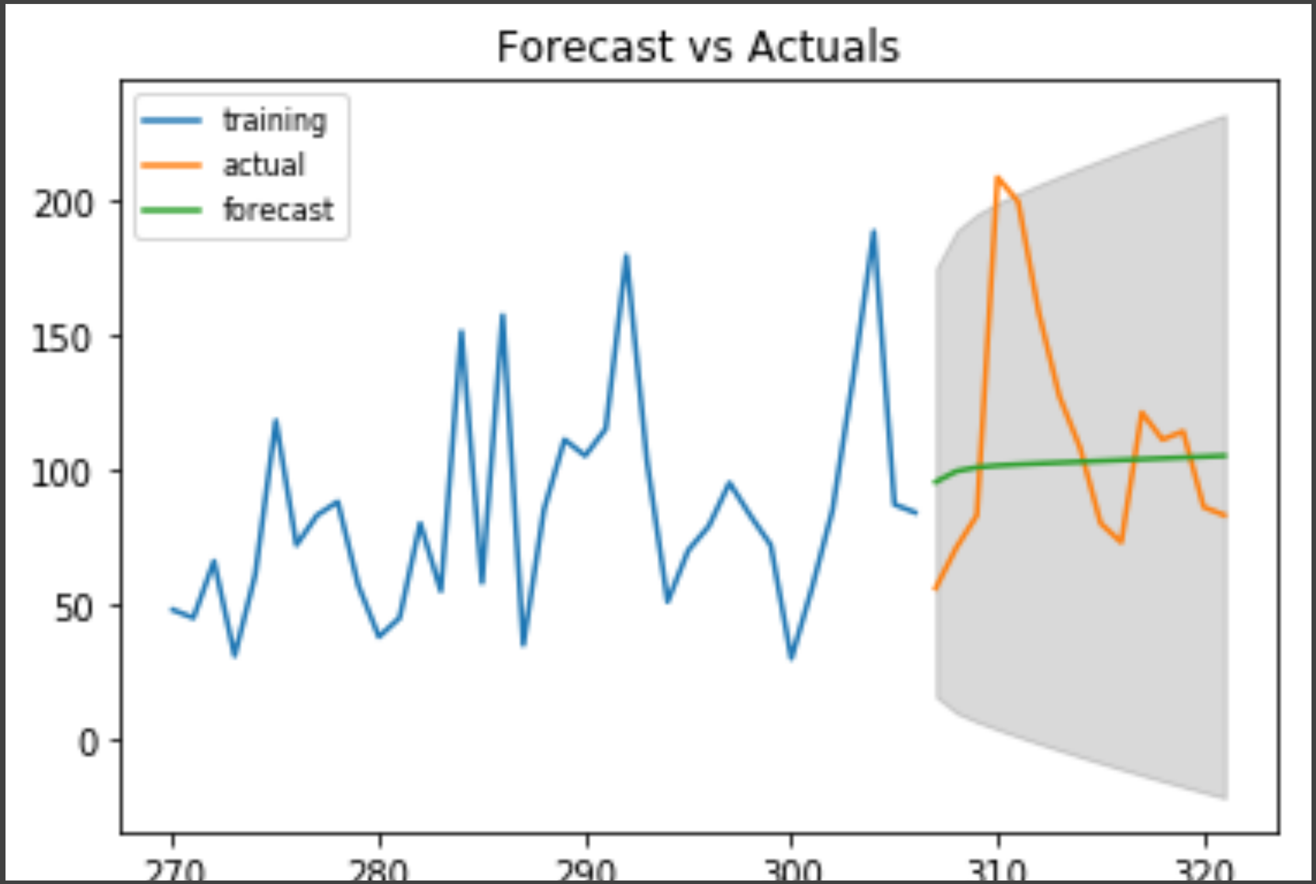

```python
import pandas as pd
import numpy as np
import statsmodels as sm
import matplotlib.pyplot as plt
from statsmodels.tsa.arima_model import ARIMA

# 1,1,1 ARIMA Model
model = ARIMA(ts, order=(1,1,1))
model_fit = model.fit(disp=0)
print(model_fit.summary())

# Actual vs Fitted
model_fit.plot_predict(dynamic=False)
plt.show()
```

Residual plots to ensure there are no patterns/ information left (constant mean and variance).




```
import pandas as pd
import numpy as np
import statsmodels as sm
import matplotlib.pyplot as plt
from statsmodels.tsa.arima_model import ARIMA

ts = df['Reported']

# 1,1,1 ARIMA Model
model = ARIMA(ts, order=(1,1,1))
model_fit = model.fit(disp=0)

# Plot residual errors
residuals = pd.DataFrame(model_fit.resid)
fig, ax = plt.subplots(1,2)
residuals.plot(title="Residuals", ax=ax[0])
residuals.plot(kind='kde', title='Density', ax=ax[1])
plt.show()
```

Cross Validating your Model




```
from statsmodels.tsa.stattools import acf
# Create Training and Test
s = 270; d = 307
train = ts[s:d]; test = ts[d:]
# Build Model
model = ARIMA(ts, order=(1, 1, 1))
fitted = model.fit(disp=-1)
# Forecast
fc, se, conf = fitted.forecast(len(ts)-d, alpha=0.01)  # 99% conf
# Make as pandas series
fc_series = pd.Series(fc, index=test.index)
lower_series = pd.Series(conf[:, 0], index=test.index)
upper_series = pd.Series(conf[:, 1], index=test.index)
# Plot
#plt.figure(figsize=(10,5), dpi=100)
plt.plot(train, label='training')
plt.plot(test, label='actual')
plt.plot(fc_series, label='forecast')
plt.fill_between(lower_series.index, lower_series, upper_series, color='k', alpha=.15)
plt.title('Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```

Auto ARIMA Forecast

- import pmdarima as pm
- model = pm.auto_arima(ts, start_p=1, start_q=1,
- test='adf', # use adftest to find optimal 'd'
- max_p=3, max_q=3, # maximum p and q
- m=1, # frequency of series
- d=None, # let model determine 'd'
- seasonal=False, # No Seasonality
- start_P=0,
- D=0,
- trace=True,
- error_action='ignore',
- suppress_warnings=True,
- stepwise=True)
- print(model.summary())

- Fit ARIMA: order=(1, 1, 1); AIC=3119.888, BIC=3134.974, Fit time=0.047 seconds
- Fit ARIMA: order=(0, 1, 0); AIC=3189.163, BIC=3196.706, Fit time=0.000 seconds
- Fit ARIMA: order=(1, 1, 0); AIC=3162.134, BIC=3173.448, Fit time=0.031 seconds
- Fit ARIMA: order=(0, 1, 1); AIC=3136.600, BIC=3147.914, Fit time=0.016 seconds
- Fit ARIMA: order=(2, 1, 1); AIC=3121.111, BIC=3139.969, Fit time=0.085 seconds
- Fit ARIMA: order=(1, 1, 2); AIC=3121.571, BIC=3140.428, Fit time=0.069 seconds
- Fit ARIMA: order=(2, 1, 2); AIC=3092.257, BIC=3114.886, Fit time=0.156 seconds
- Fit ARIMA: order=(3, 1, 2); AIC=3081.968, BIC=3108.368, Fit time=0.161 seconds
- Fit ARIMA: order=(3, 1, 1); AIC=3108.247, BIC=3130.875, Fit time=0.090 seconds
- Fit ARIMA: order=(3, 1, 3); AIC=3064.358, BIC=3094.529, Fit time=0.527 seconds
- Fit ARIMA: order=(2, 1, 3); AIC=3062.935, BIC=3089.335, Fit time=0.342 seconds
- Fit ARIMA: order=(1, 1, 3); AIC=3119.239, BIC=3141.868, Fit time=0.120 seconds
- Total fit time: 1.643 seconds

ARIMA Model Results

==

Dep. Variable:	D.y	No. Observations:	321
Model:	ARIMA(2, 1, 3)	Log Likelihood	-1524.468
Method:	css-mle	S.D. of innovations	27.849
Date:	Mon, 18 Jan 2021	AIC	3062.935
Time:	13:50:56	BIC	3089.335
Sample:	1	HQIC	3073.476

==

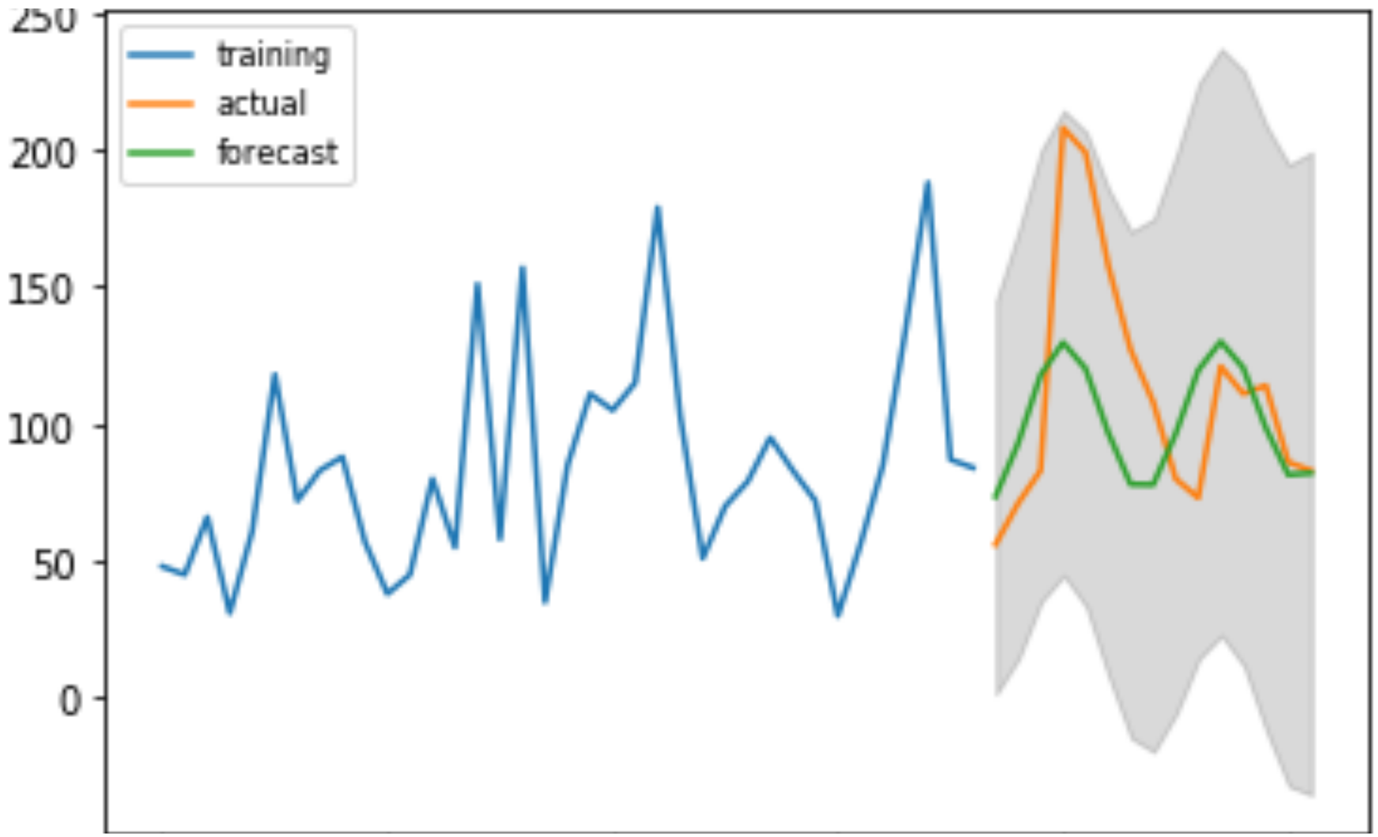
	coef	std err	z	P>|z|	[0.025	0.975]
const	0.3133	0.530	0.592	0.555	-0.725	1.351
ar.L1.D.y	1.2276	0.016	75.131	0.000	1.196	1.260
ar.L2.D.y	-0.9804	0.014	-69.048	0.000	-1.008	-0.953
ma.L1.D.y	-1.7980	0.049	-37.009	0.000	-1.893	-1.703
ma.L2.D.y	1.6163	0.068	23.936	0.000	1.484	1.749
ma.L3.D.y	-0.5633	0.044	-12.921	0.000	-0.649	-0.478

Roots

==

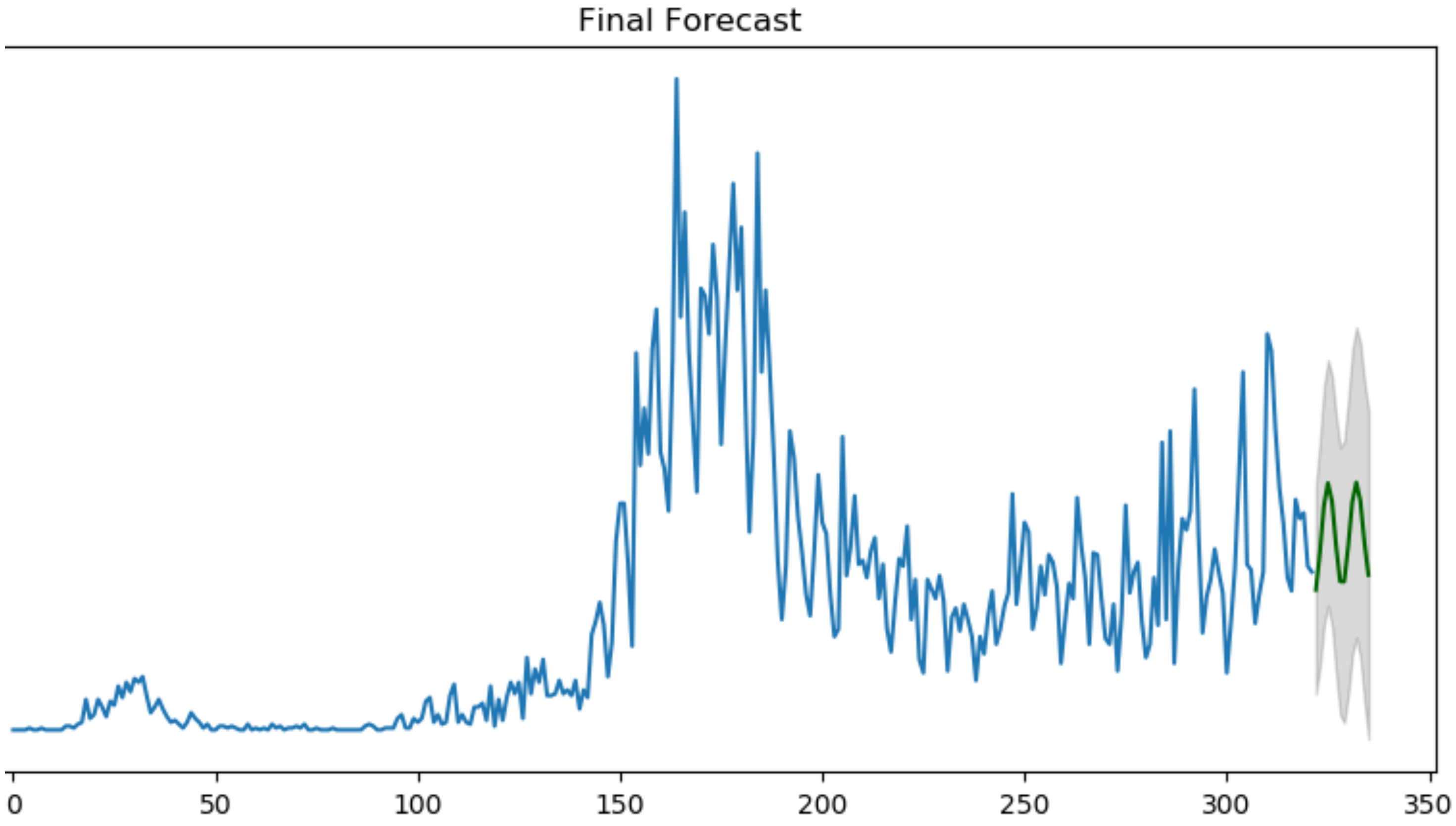
	Real	Imaginary	Modulus	Frequency
AR.1	0.6260	-0.7925j	1.0099	-0.1436
AR.2	0.6260	+0.7925j	1.0099	0.1436

Cross Validating your Model




```
from statsmodels.tsa.stattools import acf
# Create Training and Test
s = 270; d = 307
train = ts[s:d]; test = ts[d:]
# Build Model
model = ARIMA(ts, order=(2, 1, 3))
fitted = model.fit(disp=-1)
# Forecast
fc, se, conf = fitted.forecast(len(ts)-d, alpha=0.01)  # 99% conf
# Make as pandas series
fc_series = pd.Series(fc, index=test.index)
lower_series = pd.Series(conf[:, 0], index=test.index)
upper_series = pd.Series(conf[:, 1], index=test.index)
# Plot
#plt.figure(figsize=(10,5), dpi=100)
plt.plot(train, label='training')
plt.plot(test, label='actual')
plt.plot(fc_series, label='forecast')
plt.fill_between(lower_series.index, lower_series, upper_series, color='k', alpha=.15)
plt.title('Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```

Finally Forecasting (prospectively)




```
# Forecast
n_periods = 14
fc, confint = model.predict(n_periods=n_periods, return_conf_int=True)
index_of_fc = np.arange(len(ts), len(ts)+n_periods)

# make series for plotting purpose
fc_series = pd.Series(fc, index=index_of_fc)
lower_series = pd.Series(confint[:, 0], index=index_of_fc)
upper_series = pd.Series(confint[:, 1], index=index_of_fc)

# Plot
plt.figure(figsize=(10,5), dpi=100)

plt.plot(ts)
plt.plot(fc_series, color='darkgreen')
plt.fill_between(lower_series.index,
        lower_series,
        upper_series,
        color='k', alpha=.15)

plt.title("Final Forecast")
plt.show()
```