Imports

In [55]:

```
```

In [57]:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from matplotlib.pylab import rcParams
import numpy as np
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.seasonal import seasonal_decompose

from statsmodels.tsa.arima_model import ARMA
from sklearn.metrics import mean_squared_error
from pmdarima import auto_arima

from statsmodels.tsa.arima.model import ARIMA
import statsmodels.api as sm
from matplotlib.pylab import rcParams
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.stattools import pacf

from fbprophet import Prophet
from fbprophet.diagnostics import performance_metrics
from fbprophet.diagnostics import cross_validation
from fbprophet.plot import plot_cross_validation_metric

import warnings
warnings.filterwarnings("ignore")
```

Pre Process Function

In [58]:

```
# Prepares the data for modeling
def preprocess(init_data, exog=True, facebook=False, logged=False):

    # Log transform data if wanted
    if logged:
        data = init_data.copy()
        for i in range(1,len(init_data.columns)):
            col = init_data.columns[i]
            data[col] = np.log(init_data[col])
    else:
        data = init_data.copy()

    ## Drops unwanted columns and returns dataset

    # Preprocess specific to facebook prophet
    if facebook:
        fb = data.copy()
        fb['Date'] = pd.DatetimeIndex(fb['Date'])
        fb = fb.drop(['Open','High','Low','Close','Volume'],axis=1)
        fb = fb.rename(columns={'Date': 'ds','Adj Close':'y'})
        return fb

    # If using volume, splits data into 2 separate series for modeling
    elif exog:
        X = data.drop(['Open','High','Low','Close'],axis=1)
        X['Date'] = pd.to_datetime(X['Date'])
        X = X.set_index('Date')
        return X['Adj Close'],X['Volume']
    else:
        X = data.drop(['Open','High','Low','Close','Volume'],axis=1)
        X['Date'] = pd.to_datetime(X['Date'])
        X = X.set_index('Date')
        return X
```

In [59]:

```
def roi_calc(start,end):
    return round((end-start)/start*100,2)
```

In [60]:

```
# Returns amount of periods to difference data, using adfuller method
def return_d(data,alpha=0.05, plotting = False, output = False):
    # Uses differencing and adfuller method to find d value for ARIMA models
    diff = data.copy()
    d = 0

    # Iterates through a default range of 30 periods and finds the first period where d
    for j in range(30):
        dtest = adfuller(diff)
        if dtest[1] < alpha:

            # Plots differenced data
            if plotting:
                diff.plot()
                plt.xlabel('Date')
                plt.ylabel('Price')
                plt.title(('Differencing with Periods ='+str(j)))

            # Prints stat values
            if output:
                dfoutput = pd.Series(dtest[0:4], index=['Test Statistic', 'p-value', '#
                for key,value in dtest[4].items():
                    dfoutput['Critical Value (%s)'%key] = value
                print(dfoutput)
            d = j
            return d
        else:
            # Differences the data one additional period and checks adfuller method
            diff = diff.diff(periods=j+1).dropna()
    return d
```

In [61]:

```
# Returns suggestion for moving average's p
def return_p(data,alpha=0.05, plotting = False):
    # Determine if data needs differencing first
    d = return_d(data,alpha)
    new_data = data.copy()
    if d > 0:
        new_data = data.diff(periods=d).dropna()

    # Uses pacf to find p value for ARIMA models
    data_pacf = pacf(new_data)
    p = 0
    # Plots PACF
    if plotting:
        plot_pacf(new_data,alpha=alpha)
        plt.xlabel('Lags')
        plt.ylabel('PACF')

    # Returns first p value less than alpha
    for k in range(len(data_pacf)):
        if (abs(data_pacf[k])) < alpha:
            p = k-1
            return p

    return p
```

In [62]:

```
# Returns suggestion for auto regressive's q
def return_q(data, alpha=0.05, plotting = False):
    # Determine if data needs differencing first
    d = return_d(data,alpha)
    new_data = data.copy()
    if d > 0:
        new_data = data.diff(periods=d).dropna()

    # uses acf to find q value for ARIMA models
    data_acf = acf(new_data)
    q = 0

    # Plots ACF
    if plotting:
        plot_acf(new_data,alpha=alpha)
        plt.xlabel('Lags')
        plt.ylabel('ACF')

    # Returns first q value less than alpha
    for i in range(len(data_acf)):
        if (abs(data_acf[i])) < alpha:
            q = i-1
            return q
    return q
```

In [63]:

```
# Plots seasonal trends if any
def seasonal(data):
    decomposition = seasonal_decompose(data)

    # Gather the trend, seasonality, and residuals
    trend = decomposition.trend
    seasonal = decomposition.seasonal
    residual = decomposition.resid

    # Plot gathered statistics
    plt.figure(figsize=(12,8))
    plt.subplot(411)
    plt.plot(data, label='Original', color='blue')
    plt.legend(loc='best')
    plt.subplot(412)
    plt.plot(trend, label='Trend', color='blue')
    plt.legend(loc='best')
    plt.subplot(413)
    plt.plot(seasonal,label='Seasonality', color='blue')
    plt.legend(loc='best')
    plt.subplot(414)
    plt.plot(residual, label='Residuals', color='blue')
    plt.legend(loc='best')
    plt.tight_layout()
```

In [64]:

```
# Helper function to return order values for use in base model
def model_params(data, exog=True, logged=False):
    if exog:
        new_data, ex = preprocess(init_data=data,exog=exog,logged=logged)
    else:
        new_data = preprocess(init_data=data,exog=exog,logged=logged)
    p = return_p(new_data)
    q = return_q(new_data)
    d = return_d(new_data)
    return p,d,q
    print("Returns: p, d, q")
```

Train Test Split Function

In [65]:

```
# Splits train/test data specific to time series
def train_test(data,exog=True, percent =.75,facebook=False, logged=False, full=False):
    if full:
        exog=False
        length = len(data)+1
    else:
        length = int(len(data)*percent)
    if exog:
        X,Xv= preprocess(init_data=data,exog=exog,facebook=facebook,logged=logged)

        train, trainv = X.iloc[:length],Xv.iloc[:length]
        test, testv = X.iloc[length:],Xv.iloc[length:]
        return train,trainv,test,testv
    else:
        X = preprocess(init_data=data, exog=exog,facebook=facebook, logged=logged)

        if full:
            future_index = pd.date_range(start=X.index[-1], periods=60,freq='D')
            if facebook:
                future = pd.DataFrame(data=future_index,columns=['ds'])
                train = X.iloc[:length]
                test = future.copy()
            else:
                future = pd.DataFrame(data=future_index,columns=['Date'])
                train = X.iloc[:length]
                test = future.set_index('Date')
        else:
            train = X.iloc[:length]
            test = X.iloc[length:]
        return train, test
```

Base Model Function

In [66]:

```
def base_model(data,exog=True,percent = .75, plotting=False, summary=False, mse=False,
               return_rmse=False,logged=False, full=False, roi=False, return_roi = Fals
```

```
    # Failsafe just in case
    if full:
        exog=False
        mse=False
    else:
        roi=False


    # Get initial p,d,q from helper function
    p,d,q = model_params(data=data, exog=exog, logged=logged)

    # Containers for train and test splits
    trainpreds = pd.DataFrame()
    testpreds = pd.DataFrame()

    # Splits the data, depending on modeling with/without exogenous, and models using S
    if exog:
        train,trainv,test,testv = train_test(data=data,percent = percent,exog=exog,logg
        sarima = sm.tsa.SARIMAX(train,order=(p,d,q),trend='c',exog=trainv).fit()
        trainpreds = sarima.predict()
        forecast = sarima.get_forecast(len(test), index=test.index, exog=testv)
        testpreds = forecast.predicted_mean
        conf = forecast.conf_int(alpha=.10)
    else:
        train,test= train_test(data=data,percent=percent,exog=exog,logged=logged,full=f
        sarima = sm.tsa.SARIMAX(train,order=(p,d,q),trend='c').fit()
        trainpreds = sarima.predict()
        forecast = sarima.get_forecast(len(test), index=test.index)
        testpreds = forecast.predicted_mean
        conf = forecast.conf_int(alpha=.05)


    # Reverse transforms the data if log transformed initially
    if logged:
        itrain = np.exp(train)
        itest = np.exp(test)
        itrainpreds = np.exp(trainpreds)
        itestpreds = np.exp(testpreds)
        iconf = np.exp(conf)
    else:
        itrain = train.copy()
        itest = test.copy()
        itrainpreds = trainpreds.copy()
        itestpreds = testpreds.copy()
        iconf = conf.copy()

    # Plots the data and the forecasts
    if plotting:
        max_y1 = max(data['Adj Close'])
        max_y2 = max(itestpreds)
        max_y = max([max_y1,max_y2])
        figure = plt.figure(figsize=(15,15))
        plt.plot(itrain.append(itest), label='Original')
        plt.plot(itrainpreds.append(itestpreds),label='Model')

        # plots confidence interval
        conf_df = iconf.copy()
        conf_df.columns =  ['y1','y2']
        conf_df['X']=test.index
        plt.fill_between(conf_df['X'],conf_df['y1'],conf_df['y2'], alpha=.2)
        plt.xlabel('Date')
        plt.ylabel('Price')
        plt.title('Daily Price over Time')
        plt.ylim(top=(max_y*1.25))
        if full:

            # Cropping the output graphs
            lengthX = int(len(itrain)*.5)
            min_x = itrain.index[lengthX]
            plt.xlim(left=min_x)

            min_y1 = int(min(itrain['Adj Close'].iloc[lengthX:]))
            min_y2 = int(min(itestpreds))
            min_y = int(min([min_y1,min_y2])*.75)
            plt.ylim(bottom=min_y)

        plt.legend()
        plt.show();

        # Plots residual data
        sarima.plot_diagnostics()
    # Model Summary
    if summary:
        print(sarima.summary())

    # RMSE
    if mse:
        print('ARIMA Test RMSE: ', mean_squared_error(itest, itestpreds)**0.5)
        if return_rmse:
            return sarima,mean_squared_error(itest, itestpreds)**0.5

    # Return on investment
    if roi:
        start = itrainpreds[-1]
        end = itestpreds[-1]
        roival = roi_calc(start=start,end=end)
        print("ARIMA ROI: ", roival,"%")
        if return_roi:
            return sarima,roival

    return sarima
```

Auto Arima Function

In [102…

```
def create_auto_arima(data, exog=True,percent=.75, plotting=False, summary=False, mse=F
                      return_rmse = False,logged=False, full=False, roi = False, return

    trainpreds = pd.DataFrame()
    testpreds = pd.DataFrame()
    max_val=max(data['Adj Close'])
    # Failsafe just in case
    if full:
        exog=False
        mse=False
    else:
        roi = False

    # Train Test Split and Predictions
    if exog:
        train,trainv,test,testv = train_test(data=data,exog=exog, percent=percent,logge
        auto = auto_arima(y=train ,trace=trace, exog=trainv,stepwise=True, max_order=12
        testpreds, conf = auto.predict(len(test), index=test.index, exog=testv, return_

    else:
        train,test= train_test(data=data,exog=exog,percent=percent, logged=logged, full
        auto = auto_arima(y=train ,trace=trace,stepwise=True, max_order=12).fit(train)
        trainpreds = auto.predict()
        testpreds, conf = auto.predict(len(test), index=test.index, return_conf_int=Tru


    # Reverse transforms data if log transformed already
    if logged:
        itrain = np.exp(train)
        itest = np.exp(test)
        itestpreds = np.exp(testpreds)
        iconf = np.exp(conf)

    else:
        itrain = train.copy()
        itest = test.copy()
        itestpreds = testpreds.copy()
        iconf = conf.copy()


    # PLots the data and forecasts
    if plotting:
        plot_preds = pd.DataFrame(data=itestpreds, columns=['Adj Close'], index=itest.i
        figure = plt.figure(figsize=(10,10))

        max_y1 = max(data['Adj Close'])
        max_y2 = max(itestpreds)
        max_y = max([max_y1,max_y2])

        # Plots confidence interval
        conf_df = pd.DataFrame(data=iconf, columns = ['y1','y2'])
        conf_df['X']=test.index
        plt.fill_between(conf_df['X'],conf_df['y1'],conf_df['y2'], alpha=.2)

        plt.plot(itrain.append(itest), label='Original')
        plt.plot(plot_preds,label='Model')
        plt.xlabel('Date')
        plt.ylabel('Price')
        plt.title('Daily Price over Time')
        plt.ylim(top=(max_y*1.25))

        if full:

            # Cropping the output graphs
            lengthX = int(len(itrain)*.5)
            min_x = itrain.index[lengthX]
            plt.xlim(left=min_x)

            min_y1 = int(min(itrain['Adj Close'].iloc[lengthX:]))
            min_y2 = int(min(itestpreds))
            min_y = int(min([min_y1,min_y2])*.75)
            plt.ylim(bottom=min_y)



        plt.legend()
        plt.show();
        auto.plot_diagnostics()

    # Model Summary
    if summary:
        print(auto.summary())

    # RMSE
    if mse:
        print('Auto Arima Test RMSE: ', mean_squared_error(itest, itestpreds)**0.5)
        if return_rmse:
            return auto,mean_squared_error(itest,itestpreds)**.5

    # Return on Investment
    if roi:
        start = itestpreds[0]
        end = itestpreds[-1]
        roival = roi_calc(start=start,end=end)
        print("Auto ARIMA ROI: ", roival,"%")
        if return_roi:
            return auto, roival

    return auto
```

Prophet Function

In [68]:

```
def create_prophet(data,exog=False,percent=.75,plotting=False,summary=False, mse=False,
                   return_rmse=False,logged=False, full=False, roi=False, return_roi=Fa

    # Failsafe
    exog=False
    if full:
        roi = True
        mse = False

    # Train/Spit and
    fb, fbtest = train_test(data,exog=exog,percent=percent,facebook=True, logged=logged
    fb_model = Prophet(interval_width=.90, daily_seasonality=True)
    fb_model.fit(fb)

    if full:
        length = pd.date_range(start=fb.iloc[-1].ds, periods=60,freq='D')
        mse = False
    else:
        length = pd.date_range(start=fb.iloc[0].ds, end=fbtest.iloc[-1].ds,freq='D')
        roi = False
    ifuture = pd.DataFrame(data=length,columns=['ds'])
    iforecast = fb_model.predict(ifuture)
    forecast = iforecast.copy()

    # Plots the data and forecasts
    if plotting:
        fb_model.plot(forecast,uncertainty=True, figsize=(10,10));
        plt.plot(fb.append(fbtest).set_index('ds'), c='red', label = 'Actual', alpha=.2
        plt.legend()
        plt.xlabel('Date')
        if logged:
            plt.ylabel('Log Price')
            plt.title('Daily Log Price over Time')
        else:
            plt.ylabel('Price')
            plt.title('Daily Price over Time')
        max_y = int(max(forecast['yhat']*1.25))
        plt.ylim(top=max_y)
        plt.show();

    # RMSE
    if mse:
        train_error = pd.concat([forecast.set_index('ds'),fb.set_index('ds')], join='in
        test_error = pd.concat([forecast.set_index('ds'),fbtest.set_index('ds')], join=
        if logged:
            itest_error_y = np.exp(test_error.y)
            itest_error_yhat = np.exp(test_error.yhat)
            print("Logged Prophet Test RMSE:", mean_squared_error(itest_error_y,itest_e
            if return_rmse:
                return fb_model,mean_squared_error(itest_error_y,itest_error_yhat)**.5
        else:
            print("Prophet Test RMSE:", mean_squared_error(test_error.y,test_error.yhat
            if return_rmse:
                return fb_model,mean_squared_error(test_error.y,test_error.yhat)**.5

    # Return on investment
    if roi:
        start = fb['y'].iloc[-1]
        end = forecast['yhat'].iloc[-1]
        roival = roi_calc(start=start,end=end)
        print("Prophet ROI: ", roival,"%")
        if return_roi:
            return fb_model, roival

    return fb_model
```

Best Model Function

In [142…

```
def best_model(data,percent=.75, plotting=False):
    models = ['ARIMA', 'Logged_ARIMA','Auto_ARIMA','Logged_Auto_ARIMA','Prophet','Logge

    # Runs all the models, with and without log transforming the data
    sarima1,s1 = base_model(data,exog=True, percent=percent,mse=True,return_rmse=True)
    sarima2,s2 = base_model(data,exog=True, percent=percent,mse=True,return_rmse=True,

    auto1,a1 = create_auto_arima(data,exog=True, percent=percent,mse=True,return_rmse=T
    auto2,a2 = create_auto_arima(data,exog=True, percent=percent,mse=True,return_rmse=T

    fb1, f1 = create_prophet(data,exog=False, percent=percent,mse=True,return_rmse=True
    fb2, f2 = create_prophet(data,exog=False, percent=percent,mse=True,return_rmse=True

    # Determines best model using lowest RMSE
    rmses = [s1,s2,a1,a2,f1,f2]
    best_index = rmses.index(min(rmses))
    rmses = np.array([s1,s2,a1,a2,f1,f2])
    rmses_rounded = np.around(rmses,decimals=2)

    # Fits the best version of the model using the full data
    if best_index == 0:
        model, growth = base_model(data,exog=True,percent=percent,full=True,roi=True,re
        rmse = rmses[best_index]
        model_name=models[best_index]
        print('Best Model:',model_name)
    if best_index == 1:
        model, growth = base_model(data,exog=True,percent=percent,full=True,roi=True,re
        rmse = rmses[best_index]
        model_name=models[best_index]
        print('Best Model:',model_name)
    if best_index == 2:
        model, growth = create_auto_arima(data,exog=True,percent=percent,full=True,roi=
        rmse = rmses[best_index]
        model_name=models[best_index]
        print('Best Model:',model_name)
    if best_index == 3:
        model, growth = create_auto_arima(data,exog=True,percent=percent,full=True,roi=
                                          trace=False,plotting=plotting)
        rmse = rmses[best_index]
        model_name=models[best_index]
        print('Best Model:',model_name)
    if best_index == 4:
        model, growth = create_prophet(data,exog=False,percent=percent,full=True,roi=Tr
        rmse = rmses[best_index]
        model_name=models[best_index]
        print('Best Model:',model_name)
    if best_index == 4:
        model, growth = create_prophet(data,exog=False,percent=percent,full=True,roi=Tr
        rmse = rmses[best_index]
        model_name=models[best_index]
        print('Best Model:',model_name)

    # Converts the output into a dataframe
    columns = models.copy()
    columns.extend(['Best_Model', 'Best_RMSE','Expected_60day_Growth(%)'])
    full_data = np.append(rmses_rounded,[model_name,rmses_rounded[best_index],growth])

    best_df = pd.DataFrame(columns=columns)
    best_df.loc[0] = full_data

    return best_df
```