Predicting Stock prices using Gaussian Process Regression

END TERM REPORT

By

Aashay Goel

K18SJ

Roll No: 18

Ankit Kumar

K18SJ

Roll No: 23

Akshay P V

K18SJ

Roll No: 24

Department of Intelligent Systems
School of Computer Science Engineering
Lovely Professional University, Jalandhar

APRIL – 2020

STUDENT DECLARATION

This is to declare that this report has been written by me/us. No part of the report is copied from other sources. All information included from other sources have been duly acknowledged. I/We aver that if any part of the report is found to be copied, I/we are shall take full responsibility for it.

Aashay Goel
Roll No: 18
Ankit Kumar
Roll No: 23
Akshay P V
Roll No: 24

Place: LOVELY PROFESSIONAL UNIVERSITY
Date: 13/4/2020

BONAFIDE CERTIFICATE

Certified that this project report “ Predicting stock prices using Gaussian Process Regression ” is the bonafide work of “AASHAY GOEL, ANKIT KUMAR, AKSHAY P V ” who carried out the project work under my supervision.

Signature

DIPEN SAINI
Assistant Professor

Dept. Computer Science and

Engineering

Lovely professional University

Phagwara

Punjab

TABLE OF CONTENTS

Background and objectives of project assigned

Gaussian Processes:

A Gaussian process is a generalization of the Gaussian distribution - it represents a probability distribution over *functions* which is entirely specified by a mean and covariance *functions*. Mathematical definition would be then as follows:

Definition: *A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

Let x be some process $f(x)$. We write:

$$f(x) \sim GP(m(\cdot), k(\cdot, \cdot)),$$

where $m(\cdot)$ and $k(\cdot\ \cdot)$ are the mean and covariance functions, respectively:

$$m(x) = E[f(x)]$$
$$k(x_1, x_2) = E[(f(x_1) - m(x_1))(f(x_2) - m(x_2))].$$

We will assume that we have a training set $D = \{(x_i\ y_i)|i = 1\ \dots\ N\}$ where $x_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$. For sake of simplicity let X be the matrix of all inputs, and y the vector of targets. Also we should assume that the observations y_i from the proces $f(\ x)$ are noisy:

$$y_i = f(x_i) + \varepsilon_i, \qquad \text{where} \qquad \varepsilon_i \sim N(0, \sigma_n^2).$$

Regression with a GP is achieved by means of Bayesian inference in order to obtain a posterior distribution over functions given a suitable prior and training data. Then, given new test inputs, we can use the posterior to arrive at a predictive distribution conditional on the test inputs and the training data.

It is often convienient to assume that the GP prior distribution has mean of zero

$$f(x) \sim GP(0, k(\cdot, \cdot)).$$

Let $f = [f(x_1)\ \dots\ f(x_n)]$ be a vector of function values in the training set D. Their prior distribution is then:

$$f \sim GP(0, K(X, X)),$$

where $K(X X)_{i=}k(x_i x)$ is a covariance matrix evaluated using covariance function between given points (also known as *kernel* or *Gram* matrix). Considering the joint prior distribution between training and the test points, with locations given by matrix X_* and whose function values are f_* we can obtain that

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim N\left(0, \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right).$$

$$P(f, f_*|y) = \frac{P(y|f, f_*)P(f, f_*)}{P(y)} = \frac{P(y|f)P(f, f_*)}{P(y)},$$

where $P(y|f\ \ f_*) = P(y|f)$ since the likelihood is conditionally independent of f_* given f, and

$$y|f \sim N(f, \sigma_n^2 I_n),$$

where I_n is $N \times N$ identity matrix. So the desired predictive distribution is

$$P(f_*|y) = \int P(f, f_*|y)df = \frac{1}{P(y)} \int P(y|f)P(f, f_*)df.$$

Since these distributions are normal, the result of the marginal is also normal we have

$$E[f_*|y] = K(X_*, X)\Lambda^{-1}y,$$
$$Cov[f_*|y] = K(X_*, X_*) - K(X_*, X)\Lambda^{-1}K(X, X_*),$$

where y are test points and

$$\Lambda = K(X, X) + \sigma_n^2 I_N.$$

The computation of Λ^{-1} is the most computationally expensive in GP regression, requiring as mentioned earlier $O(N^3)$ time and also $O(\ \ \ N^2)$ space.

Covariance Function

A proper choice for the covariance function is important for encoding knowledge about our problem - several examples are given in Rasmussen and Williams (2006). In order to get valid covariance matrices, the covariance function should be symmetric and positive semi-definite, which implies that the all its eigenvalues are positive,

$$\int k(u, v)f(u)f(v)d\mu(u)d\mu(v) \geq 0,$$

for all functions f defined on appropriate space and measure μ.

The two most common choices for covariance functions are the *squared exponential* (also known as the *Gaussian* or *radial basis function* kernel):

$$k_{SE}(u, v, \sigma_l, \mu) = \mu \exp\left(-\frac{||u - v||^2}{2\sigma_l^2}\right),$$

which we will use in our problem and the *rational quadratic*:

$$k_{RQ}(u, v, \sigma_l, \alpha, \mu) = \mu \left(1 + \frac{||u - v||^2}{2\alpha\sigma_l^2}\right)^{-\alpha}.$$

In both cases, the hyperparameter σ_l governs the *characteristic length scale* of covariance function, indicating the degree of smoothness of underlying random functions and μ we can interpret as scaling hyperparameter. The rational quadratic can be interpreted as an infinite mixture of squared exponentials with different length-scales - it converges to a squared exponential with characteristic length-scale σ_l as $\alpha \to \infty$. In this project has been used classical Gasussian kernel.

Hyperparameters Optimization

For many machine learning algorithms, this problem has often been approached by minimizing a validation error through cross-validation, but in this case we will apply alternative approach, quite efficient for GP - maximizing the *marginal likelihood* of the observerd data with respect to the hyperparameters. This function can be computed by introducing latent function values that will be integrated over. Let θ be the set of hyperparameters that have to be optimized, and $K_X(\theta)$ be the covariance matrix computed by given covariance function whose hyperparameters are θ,

$$K_X(\theta)_{i,j} = k(x_i, x_j; \theta).$$

The marginal likelihood then can be wrriten as

$$p(y|X,\theta) = \int P(y|f,X)p(f|X,\theta)df,$$

where the distribution of observations $P(y|f\ X)$ is conditionally independent of the hyperparameters given the latent function f. Under the Gaussian process prior, we have that f $|X\theta \sim N(0K_X(\theta))$, or in terms of log-likelihood

$$\log P(f|X,\theta) = -\frac{1}{2}f'K_X^{-1}(\theta) - \frac{1}{2}\log|K_X(\theta)| - \frac{N}{2}\log 2\pi.$$

Since the distributions are normal, the marginalization can be done analytically to yield

$$\log P(y|X,\theta) = -\frac{1}{2}y'(K_X(\theta)+\sigma_n^2 I_N)^{-1}y - \frac{1}{2}\log|K_X(\theta)+\sigma_n^2 I_N| - \frac{N}{2}\log 2\pi.$$

This expression can be maximized numerically, for instance by a conjugate gradient or

like in our case - the default python's sklearn optimizer to yield the `fmin_l_bfgs_b`

selected hyperparameters:

$$\theta^* = \arg\max_\theta \log p(y|X,\theta).$$

The gradient of the marginal log-likelihood with respect to the hyperparameters-necessary for numerical optimization algorithms—can be expressed as

$$\frac{\partial \log P(f|X,\theta)}{\partial\theta_i} = -\frac{1}{2}y'K_X^{-1}(\theta)\frac{\partial K_X(\theta)}{\partial\theta_i}K_X^{-1}(\theta)y - \frac{1}{2}Tr\left(K_X^{-1}(\theta)\frac{\partial K_X(\theta)}{\partial\theta_i}\right).$$

See Rasmussen and Williams (2006) for details on the derivation of this equation.

Forecasting Methodology

The main idea of this approach is to avoid representing the whole history as one time series. Each time series is treated as an independent input variable in the regression model. Consider a set of N real time series each of length M_i, $\{y^i\}$, $i = 1$ N and t $=1 ... M_i$. In this application each i represents a different year, and the series is the sequence of a particular prices during the period where it is traded. Considering the length of the stock market year, usually M will be equal to 252 and sometimes less if incomplete series is considered (for example this year) assuming that the series follow an annual cycle. Thus knowledge from past series can be transferred to a new one to be forecast. Each trade year of data is treated as a separate time series and the corresponding year is used as a independent variable in regression model.

The forecasting problem is that given observations from the complete series i $=1 ... N-1$ and (optionally) from a partial last series $\{y^N\}$, t $=1 ... M_N$ we want to extrapolate the last series until predetermined endpoint (usually a multiple of a quarter length during a year) - characterize the joint distribution of $\{y^N\}$, τ $=M_N+1 ... M_N+H$ for some H We are also given a set of non-stochastic explanatory variables specific to each series, $\{x^i\}$, where $x^i \in \mathbb{R}^d$. Our objective is to find an effective representation of $P(\{y^N\}_{\tau=M_N+1 \ldots M_N+H}|\{x^i, y^i\}^{i=1,\ldots,N})$, with τ i and t ranging, respectively over the forecatsing horizon, the available series and the observations within a series.

Data and Evaluation

For this project, three stocks/indices were selected:

- S&P 500 (GSPC),
- The Boeing Company (BA),
- Starbucks (SBUX).

The daily changes of adjusted closing prices of these stocks were examined and the historical data was downloaded in the form of csv file from the yahoo finance section. There are two sample periods taken for these three indices: first based on years 2008-2016 for prediction of whole year 2017 and second based on years 2008-2018 (up to end of second quarter) for prediction of the rest of the 2018 year. We have about 252 days of trading year per years since no data is observed on weekends. However, some years have more than 252 days of trading and some less so we choose to ignore 252+ days and for those with less trading days the remaing few we fill up with mean of the year to have

equal   for all years.

We choose to use adjusted close prices because we aim to predict the trend of the stocks not the prices. The adjusted close price is used to avoid the effect of dividends and splits because when stock has a split, its price drops by half. The adjusted close prices are standardized to zero mean and unit standard deviation. We also normalize the prices in each year to avoid the variation from previous years by subtracting the first day to start from zero.

WORK DIVISION

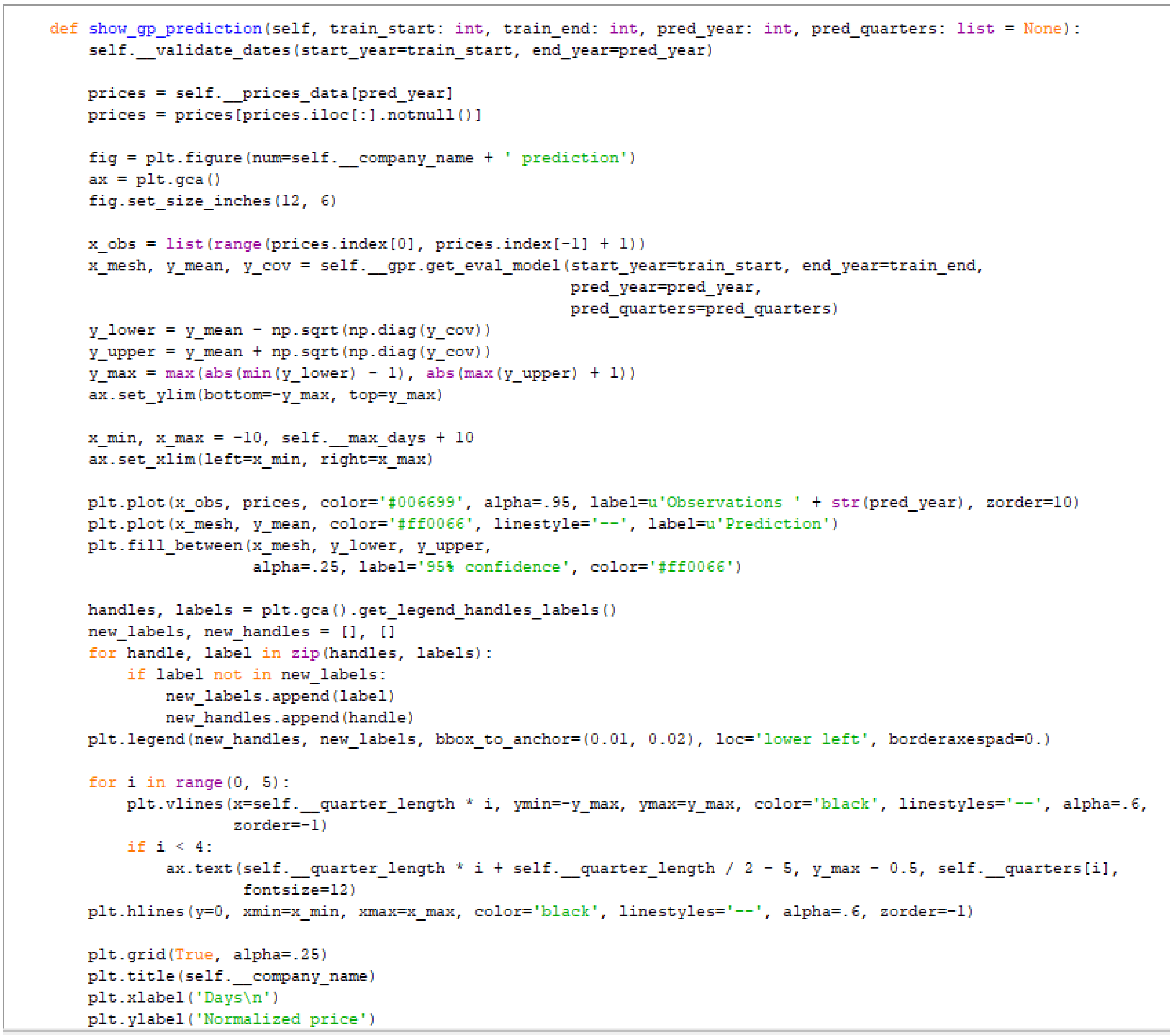
AASHAY GOEL : Training the model for predicting the next year result.

ANKIT KUMAR : Plotting the real time data from the csv sheet.

AKSHAY P V   : Plotting normalized prices of various year.

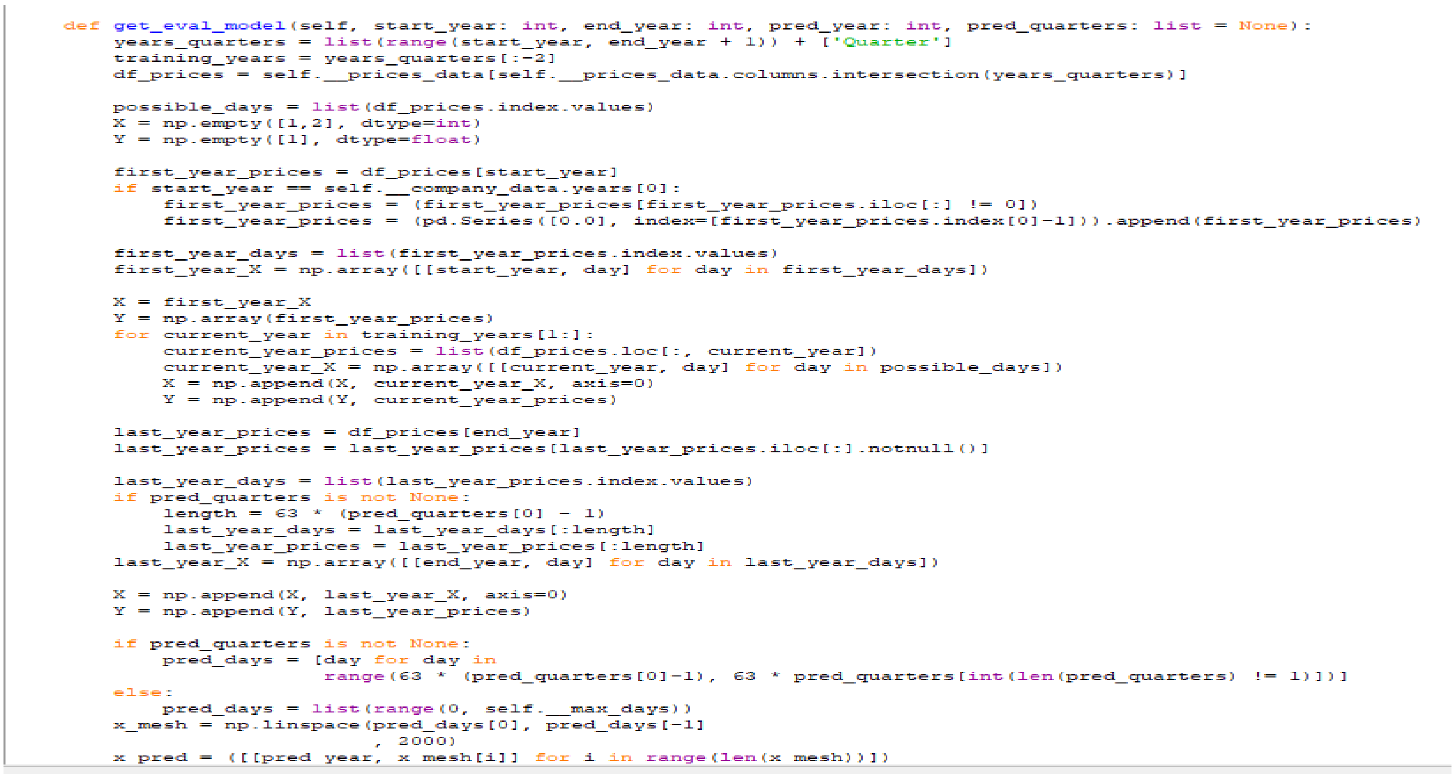
Implementation of the work

AKSHAY P V:

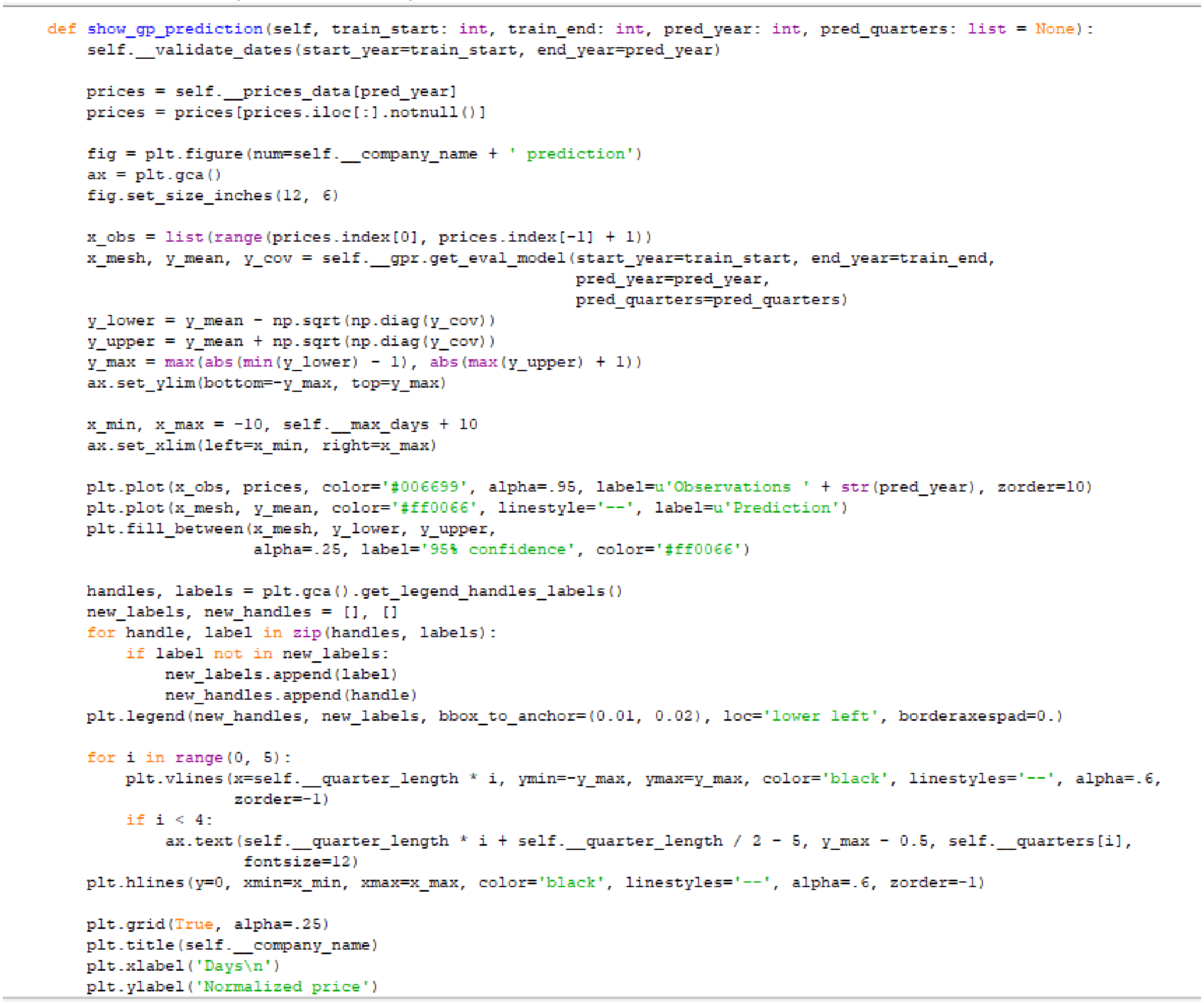


```
def show_gp_prediction(self, train_start: int, train_end: int, pred_year: int, pred_quarters: list = None):
    self.__validate_dates(start_year=train_start, end_year=pred_year)

    prices = self.__prices_data[pred_year]
    prices = prices[prices.iloc[:].notnull()]

    fig = plt.figure(num=self.__company_name + ' prediction')
    ax = plt.gca()
    fig.set_size_inches(12, 6)

    x_obs = list(range(prices.index[0], prices.index[-1] + 1))
    x_mesh, y_mean, y_cov = self.__gpr.get_eval_model(start_year=train_start, end_year=train_end,
                                                       pred_year=pred_year,
                                                       pred_quarters=pred_quarters)
    y_lower = y_mean - np.sqrt(np.diag(y_cov))
    y_upper = y_mean + np.sqrt(np.diag(y_cov))
    y_max = max(abs(min(y_lower) - 1), abs(max(y_upper) + 1))
    ax.set_ylim(bottom=-y_max, top=y_max)

    x_min, x_max = -10, self.__max_days + 10
    ax.set_xlim(left=x_min, right=x_max)

    plt.plot(x_obs, prices, color='#006699', alpha=.95, label=u'Observations ' + str(pred_year), zorder=10)
    plt.plot(x_mesh, y_mean, color='#ff0066', linestyle='--', label=u'Prediction')
    plt.fill_between(x_mesh, y_lower, y_upper,
                     alpha=.25, label='95% confidence', color='#ff0066')

    handles, labels = plt.gca().get_legend_handles_labels()
    new_labels, new_handles = [], []
    for handle, label in zip(handles, labels):
        if label not in new_labels:
            new_labels.append(label)
            new_handles.append(handle)
    plt.legend(new_handles, new_labels, bbox_to_anchor=(0.01, 0.02), loc='lower left', borderaxespad=0.)

    for i in range(0, 5):
        plt.vlines(x=self.__quarter_length * i, ymin=-y_max, ymax=y_max, color='black', linestyles='--', alpha=.6,
                   zorder=-1)
        if i < 4:
            ax.text(self.__quarter_length * i + self.__quarter_length / 2 - 5, y_max - 0.5, self.__quarters[i],
                    fontsize=12)
    plt.hlines(y=0, xmin=x_min, xmax=x_max, color='black', linestyles='--', alpha=.6, zorder=-1)

    plt.grid(True, alpha=.25)
    plt.title(self.__company_name)
    plt.xlabel('Days\n')
    plt.ylabel('Normalized price')
```

AASHAY GOEL:

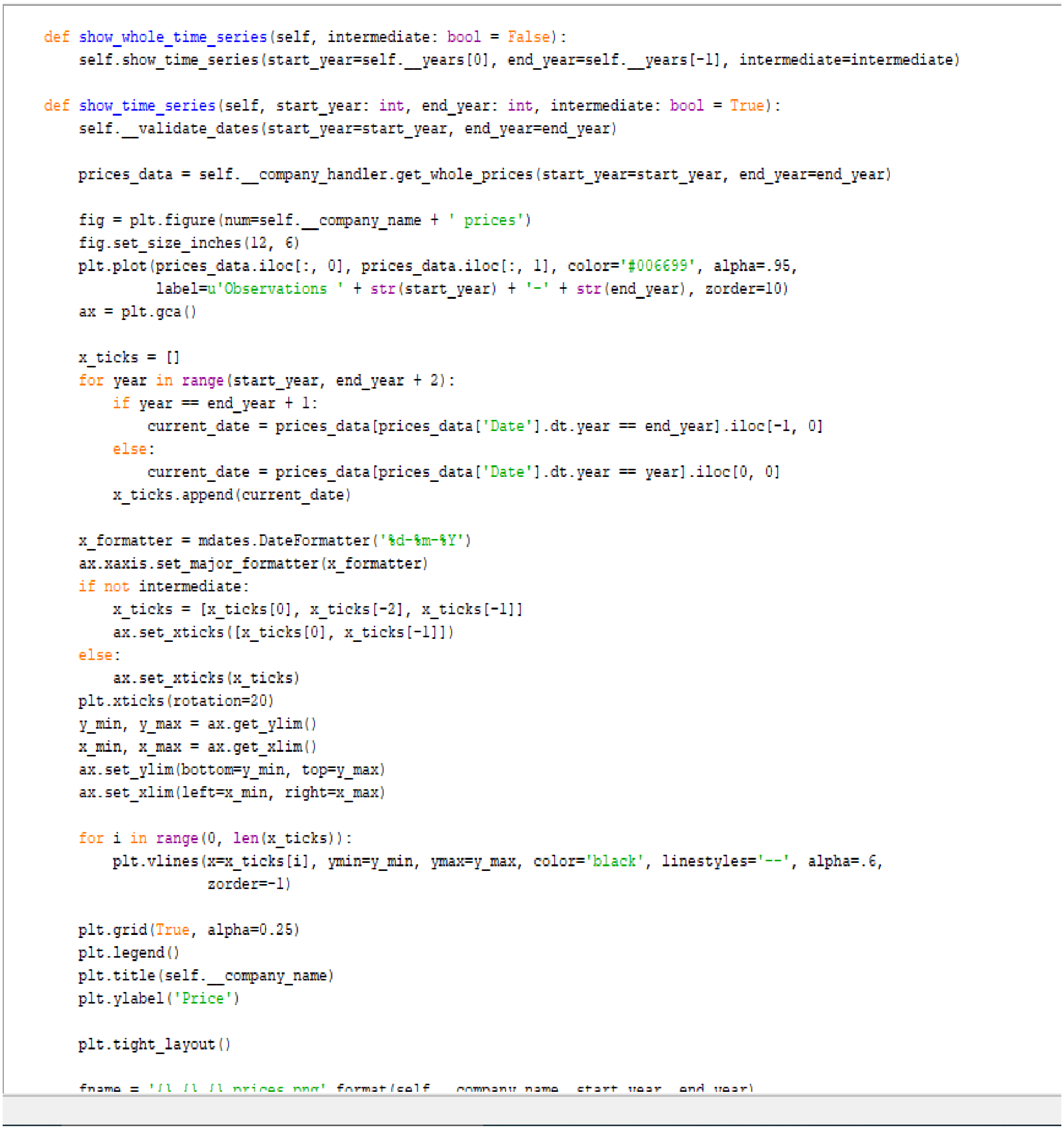


```python
def get_eval_model(self, start_year: int, end_year: int, pred_year: int, pred_quarters: list = None):
    years_quarters = list(range(start_year, end_year + 1)) + ['Quarter']
    training_years = years_quarters[:-2]
    df_prices = self.__prices_data[self.__prices_data.columns.intersection(years_quarters)]

    possible_days = list(df_prices.index.values)
    X = np.empty([1,2], dtype=int)
    Y = np.empty([1], dtype=float)

    first_year_prices = df_prices[start_year]
    if start_year == self.__company_data.years[0]:
        first_year_prices = (first_year_prices[first_year_prices.iloc[:] != 0])
        first_year_prices = (pd.Series([0.0], index=[first_year_prices.index[0]-1])).append(first_year_prices)

    first_year_days = list(first_year_prices.index.values)
    first_year_X = np.array([[start_year, day] for day in first_year_days])

    X = first_year_X
    Y = np.array(first_year_prices)
    for current_year in training_years[1:]:
        current_year_prices = list(df_prices.loc[:, current_year])
        current_year_X = np.array([[current_year, day] for day in possible_days])
        X = np.append(X, current_year_X, axis=0)
        Y = np.append(Y, current_year_prices)

    last_year_prices = df_prices[end_year]
    last_year_prices = last_year_prices[last_year_prices.iloc[:].notnull()]

    last_year_days = list(last_year_prices.index.values)
    if pred_quarters is not None:
        length = 63 * (pred_quarters[0] - 1)
        last_year_days = last_year_days[:length]
        last_year_prices = last_year_prices[:length]
    last_year_X = np.array([[end_year, day] for day in last_year_days])

    X = np.append(X, last_year_X, axis=0)
    Y = np.append(Y, last_year_prices)

    if pred_quarters is not None:
        pred_days = [day for day in
                     range(63 * (pred_quarters[0]-1), 63 * pred_quarters[int(len(pred_quarters) != 1)])]
    else:
        pred_days = list(range(0, self.__max_days))
    x_mesh = np.linspace(pred_days[0], pred_days[-1]
                         , 2000)
    x_pred = ([[pred_year, x_mesh[i]] for i in range(len(x_mesh))])
```



```python
def show_gp_prediction(self, train_start: int, train_end: int, pred_year: int, pred_quarters: list = None):
    self.__validate_dates(start_year=train_start, end_year=pred_year)

    prices = self.__prices_data[pred_year]
    prices = prices[prices.iloc[:].notnull()]

    fig = plt.figure(num=self.__company_name + ' prediction')
    ax = plt.gca()
    fig.set_size_inches(12, 6)

    x_obs = list(range(prices.index[0], prices.index[-1] + 1))
    x_mesh, y_mean, y_cov = self.__gpr.get_eval_model(start_year=train_start, end_year=train_end,
                                                        pred_year=pred_year,
                                                        pred_quarters=pred_quarters)
    y_lower = y_mean - np.sqrt(np.diag(y_cov))
    y_upper = y_mean + np.sqrt(np.diag(y_cov))
    y_max = max(abs(min(y_lower) - 1), abs(max(y_upper) + 1))
    ax.set_ylim(bottom=-y_max, top=y_max)

    x_min, x_max = -10, self.__max_days + 10
    ax.set_xlim(left=x_min, right=x_max)

    plt.plot(x_obs, prices, color='#006699', alpha=.95, label=u'Observations ' + str(pred_year), zorder=10)
    plt.plot(x_mesh, y_mean, color='#ff0066', linestyle='--', label=u'Prediction')
    plt.fill_between(x_mesh, y_lower, y_upper,
                     alpha=.25, label='95% confidence', color='#ff0066')

    handles, labels = plt.gca().get_legend_handles_labels()
    new_labels, new_handles = [], []
    for handle, label in zip(handles, labels):
        if label not in new_labels:
            new_labels.append(label)
            new_handles.append(handle)
    plt.legend(new_handles, new_labels, bbox_to_anchor=(0.01, 0.02), loc='lower left', borderaxespad=0.)

    for i in range(0, 5):
        plt.vlines(x=self.__quarter_length * i, ymin=-y_max, ymax=y_max, color='black', linestyles='--', alpha=.6,
                   zorder=-1)
        if i < 4:
            ax.text(self.__quarter_length * i + self.__quarter_length / 2 - 5, y_max - 0.5, self.__quarters[i],
                    fontsize=12)
    plt.hlines(y=0, xmin=x_min, xmax=x_max, color='black', linestyles='--', alpha=.6, zorder=-1)

    plt.grid(True, alpha=.25)
    plt.title(self.__company_name)
    plt.xlabel('Days\n')
    plt.ylabel('Normalized price')
```

ANKIT KUMAR:



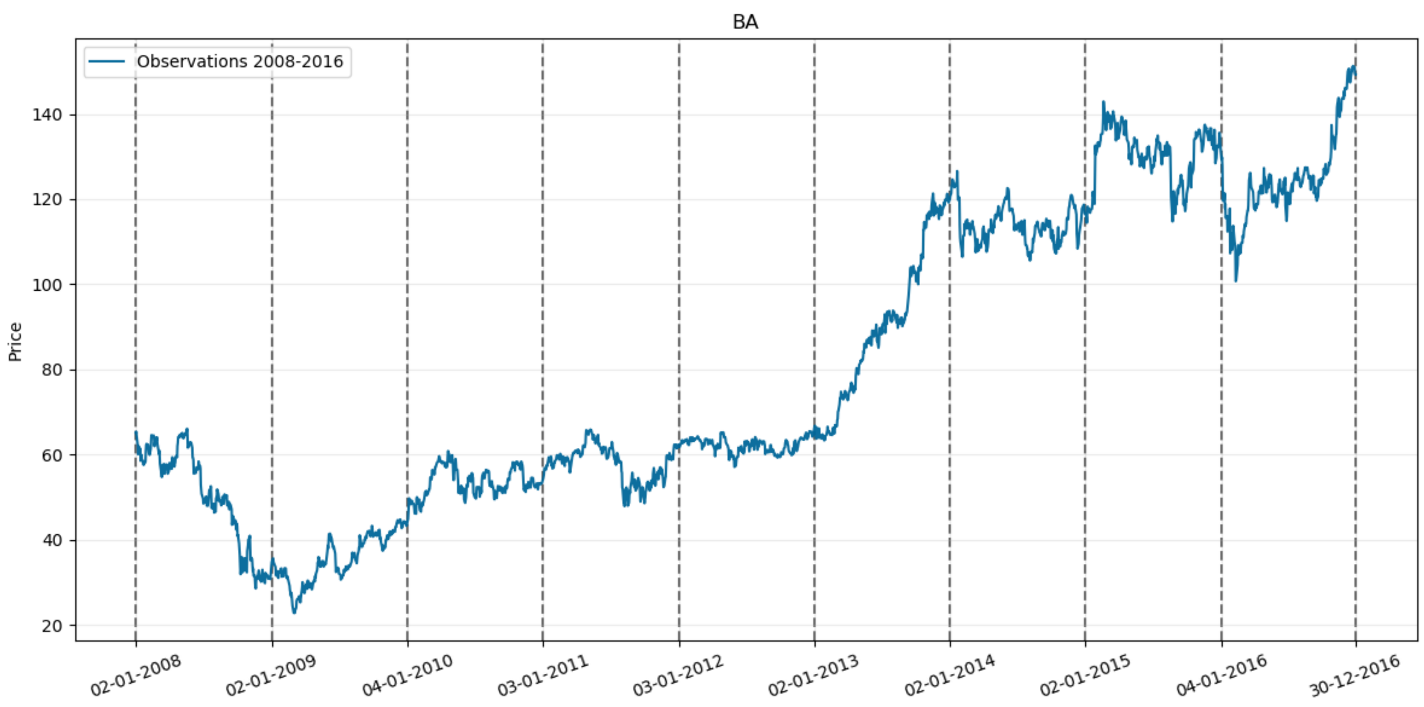
```
def show_whole_time_series(self, intermediate: bool = False):
    self.show_time_series(start_year=self.__years[0], end_year=self.__years[-1], intermediate=intermediate)

def show_time_series(self, start_year: int, end_year: int, intermediate: bool = True):
    self.__validate_dates(start_year=start_year, end_year=end_year)

    prices_data = self.__company_handler.get_whole_prices(start_year=start_year, end_year=end_year)

    fig = plt.figure(num=self.__company_name + ' prices')
    fig.set_size_inches(12, 6)
    plt.plot(prices_data.iloc[:, 0], prices_data.iloc[:, 1], color='#006699', alpha=.95,
             label=u'Observations ' + str(start_year) + '-' + str(end_year), zorder=10)
    ax = plt.gca()

    x_ticks = []
    for year in range(start_year, end_year + 2):
        if year == end_year + 1:
            current_date = prices_data[prices_data['Date'].dt.year == end_year].iloc[-1, 0]
        else:
            current_date = prices_data[prices_data['Date'].dt.year == year].iloc[0, 0]
        x_ticks.append(current_date)

    x_formatter = mdates.DateFormatter('%d-%m-%Y')
    ax.xaxis.set_major_formatter(x_formatter)
    if not intermediate:
        x_ticks = [x_ticks[0], x_ticks[-2], x_ticks[-1]]
        ax.set_xticks([x_ticks[0], x_ticks[-1]])
    else:
        ax.set_xticks(x_ticks)
    plt.xticks(rotation=20)
    y_min, y_max = ax.get_ylim()
    x_min, x_max = ax.get_xlim()
    ax.set_ylim(bottom=y_min, top=y_max)
    ax.set_xlim(left=x_min, right=x_max)

    for i in range(0, len(x_ticks)):
        plt.vlines(x=x_ticks[i], ymin=y_min, ymax=y_max, color='black', linestyles='--', alpha=.6,
                   zorder=-1)

    plt.grid(True, alpha=0.25)
    plt.legend()
    plt.title(self.__company_name)
    plt.ylabel('Price')

    plt.tight_layout()

    fname = '{} {} {} prices.png'.format(self.__company_name, start_year, end_year)
```

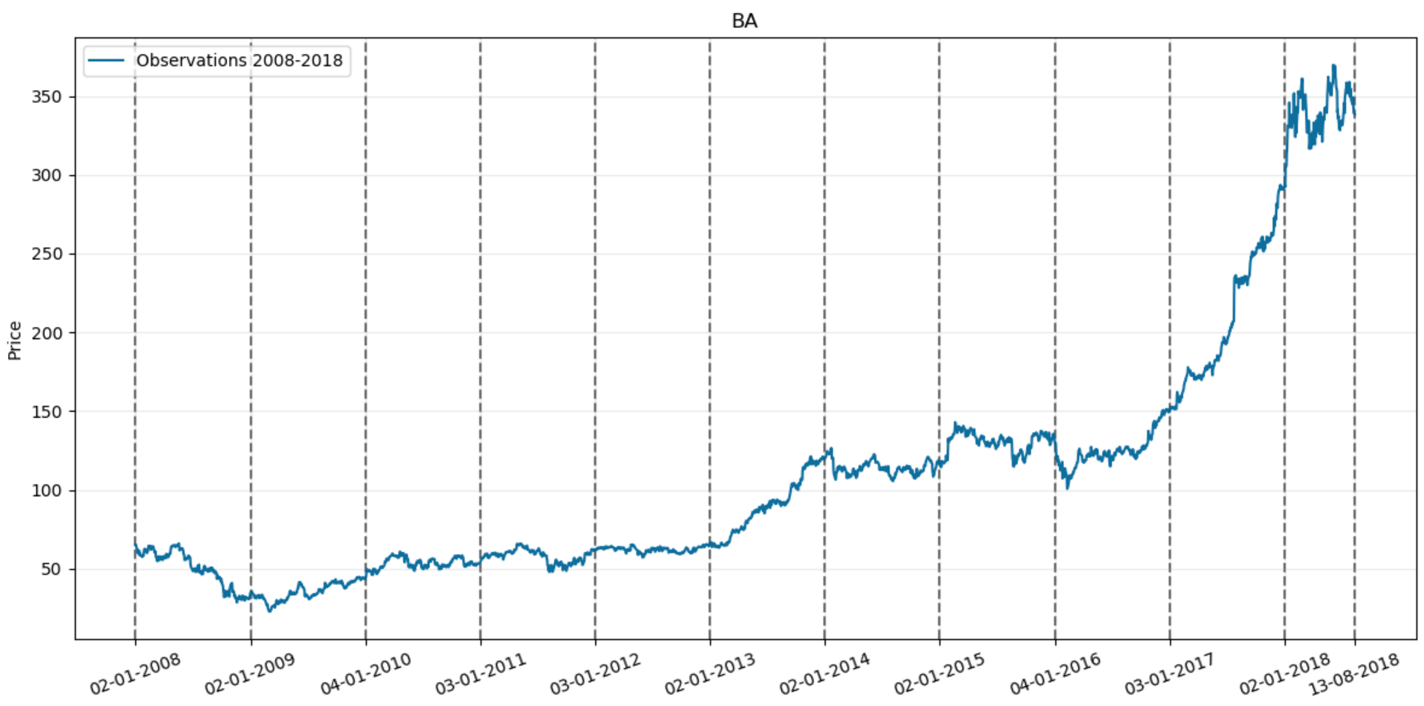
Technologies and Framework

1. sklearn.gaussian_process : to use Gaussian regression on the data
2. pandas : to work with csv file
3. numpy : to perform mathematical operations
4. os
5. matplotlib : to plot graphs

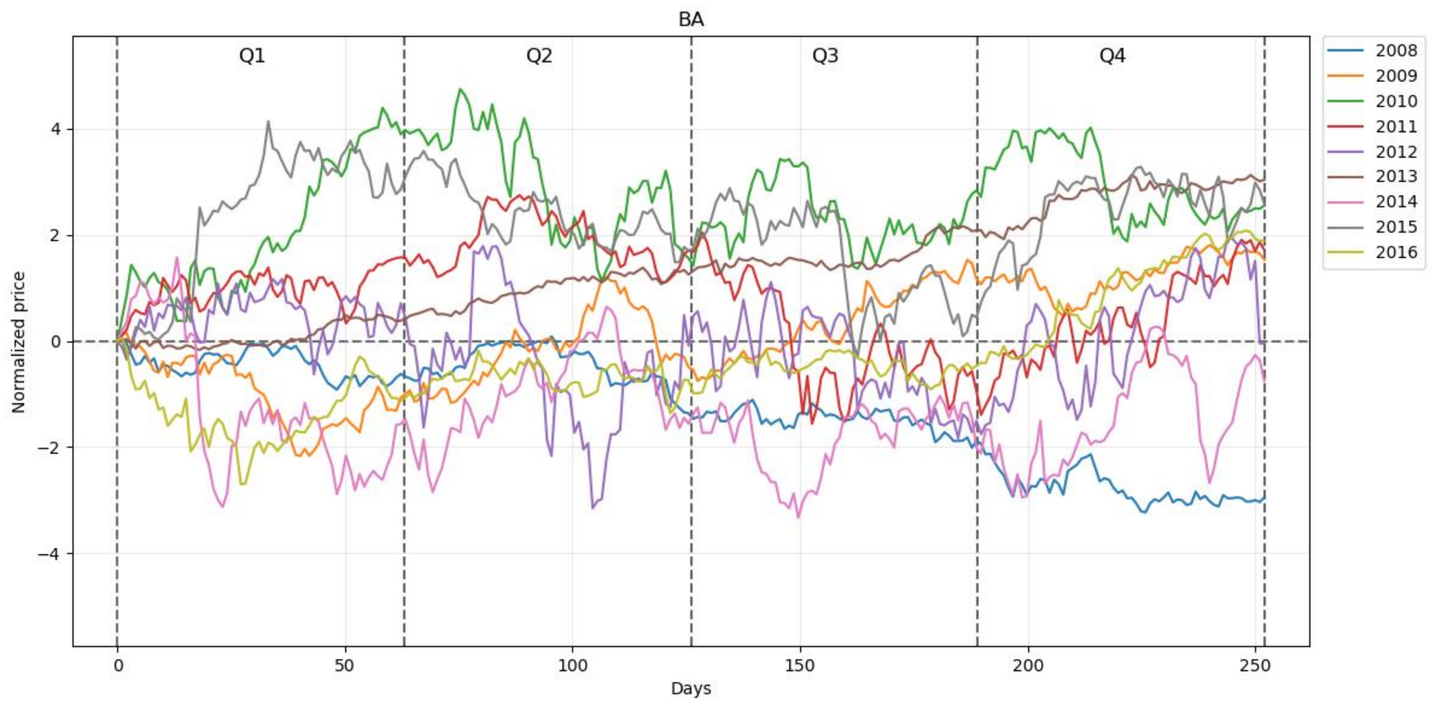
OUTPUT:

Plotting of the initial prices:

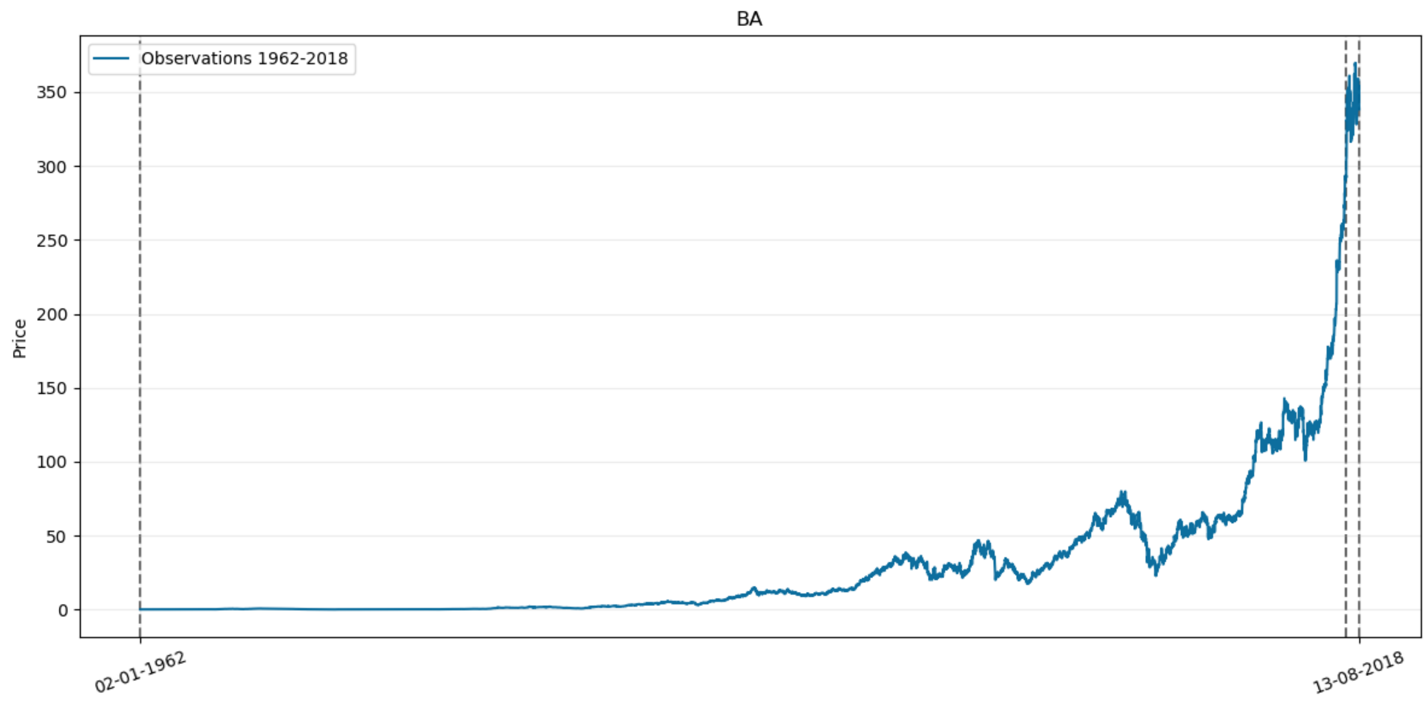




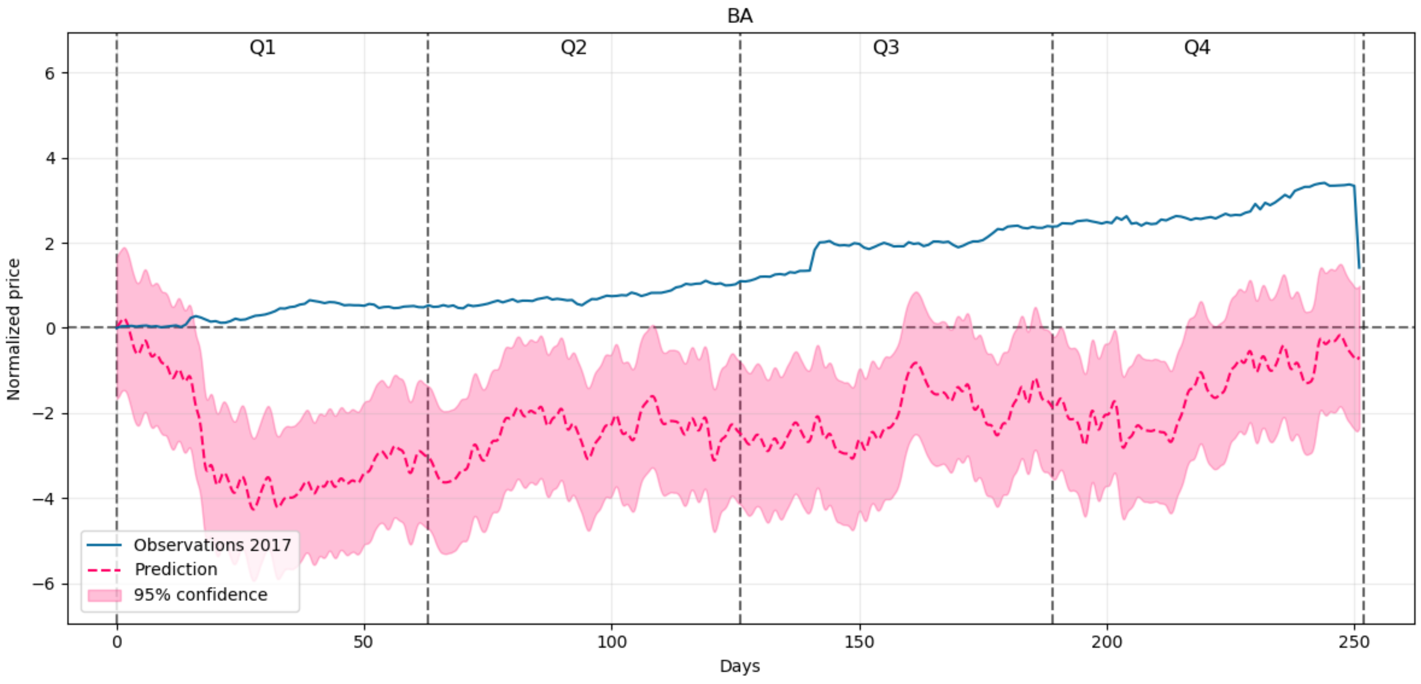
Normalized Graph:

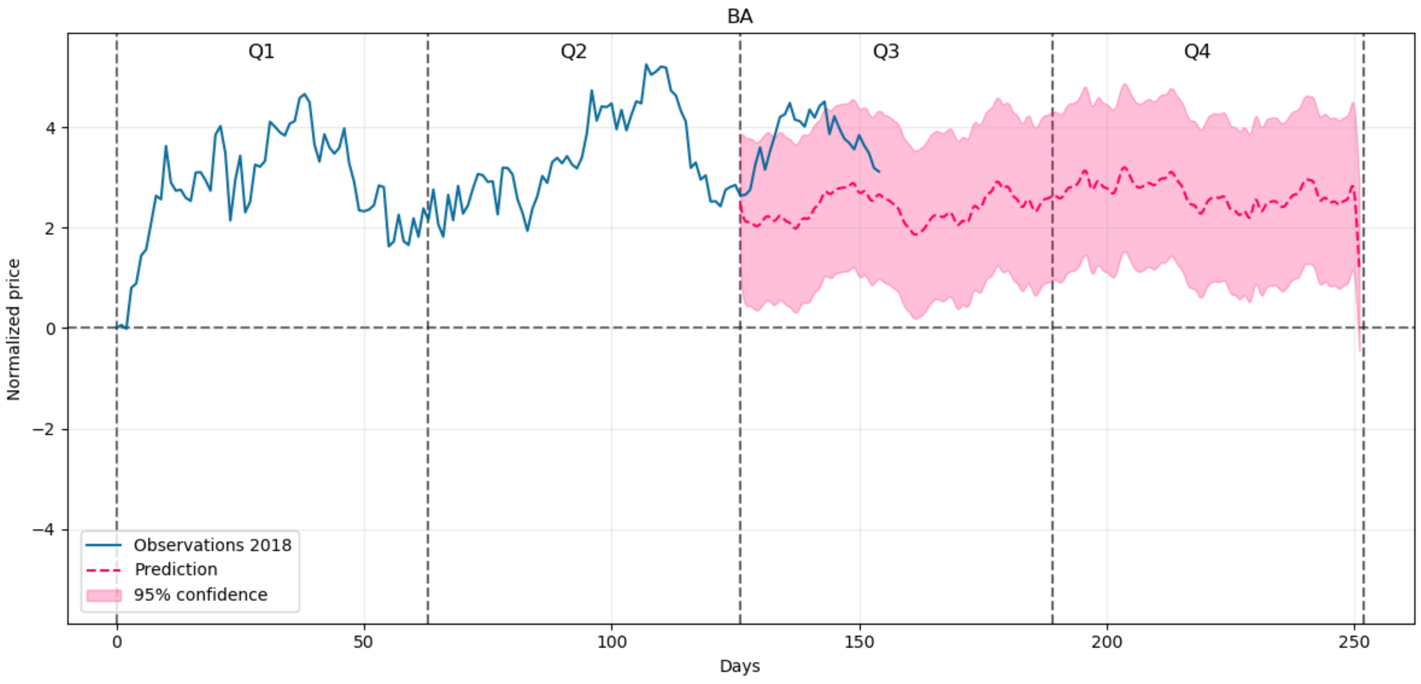


Whole graph:



Predicting the stock prices:


BA
Q1
Q2
Q3
Q4
Normalized price
Days
Observations 2017
Prediction
95% confidence


BA
Q1
Q2
Q3
Q4
Normalized price
Days
Observations 2018
Prediction
95% confidence