REGRESSION ANALYSIS & FUTURE PRICE FORECASTING

Prediction Beyond and End of The Available Data

Machine learning algorithm to analyze financial time series

Sarit Maitra Apr 5, 2020 · 9 min read ★

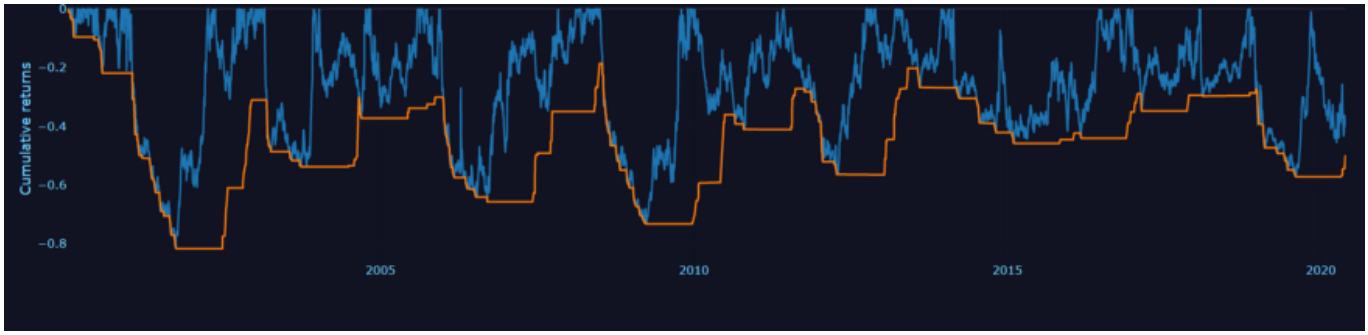


Image by author

Predictions into the future beyond the sample data in a complex task; especially when we are dealing with stochastic time-series with financial data which emerge from high frequency trading. A machine learning model's prediction performance is normally conducted :-

1. by splitting a given data set into an in-sample period; which is used for initial parameter estimation and model selection;

2. an out-of-sample period is used to evaluate forecasting performance

Out-of-sample forecast performance is generally considered more trustworthy and empirical evidence based compared to in-sample performance. In-sample performance tend to be more sensitive to outliers and data mining. Out-of-sample forecasts also better reflect the information available in real time. However, it can be noted that, in-sample return predictability does not necessarily imply out-of-sample return predictability. The key is that, model which learns from training set, whether able to generalize new data.

Let us explore and see how our real-life case study works in the scenario of never before seen data. We will use Dow Jones industrial index data for our investigation.

Data loading:

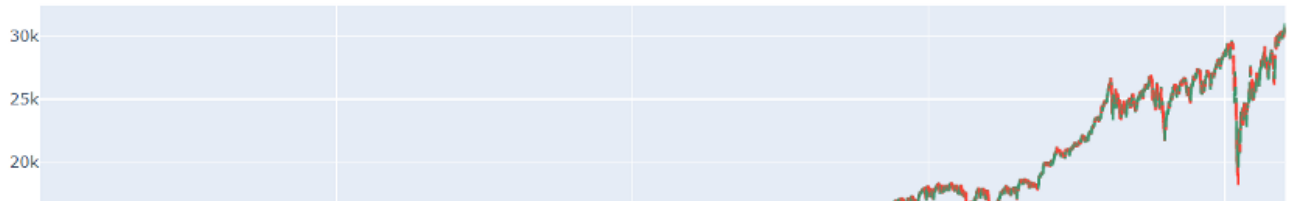
```
df = web.DataReader('^DJI', data_source = 'yahoo', start = '2000-01-
01')
print(df.head()); print('\n'); print(df.shape)
```

Date	High	Low	Open	Close	Volume	Adj Close
2000-01-03	11522.009766	11305.690430	11501.849609	11357.509766	169750000	11357.509766
2000-01-04	11350.059570	10986.450195	11349.750000	10997.929688	178420000	10997.929688
2000-01-05	11215.099609	10938.669922	10989.370117	11122.650391	203190000	11122.650391
2000-01-06	11313.450195	11098.450195	11113.370117	11253.259766	176550000	11253.259766
2000-01-07	11528.139648	11239.919922	11247.059570	11522.559570	184900000	11522.559570

```
(5287, 6)
```

Actual price visualization:

```
dataset = df.copy()
fig = go.Figure(data=[go.Candlestick(x=dataset.index,
open=dataset['Open'], high=dataset['High'], low=dataset['Low'],
close=dataset['Close'])])
fig.show()
```



Empirical Quantile of Daily Return:

Quantile forecasts are central to risk management decisions because of the widespread use of Value-at-Risk (VaR). It is the product of two factors:-

1. volatility forecasts, and
2. quantiles from the volatility forecasts.

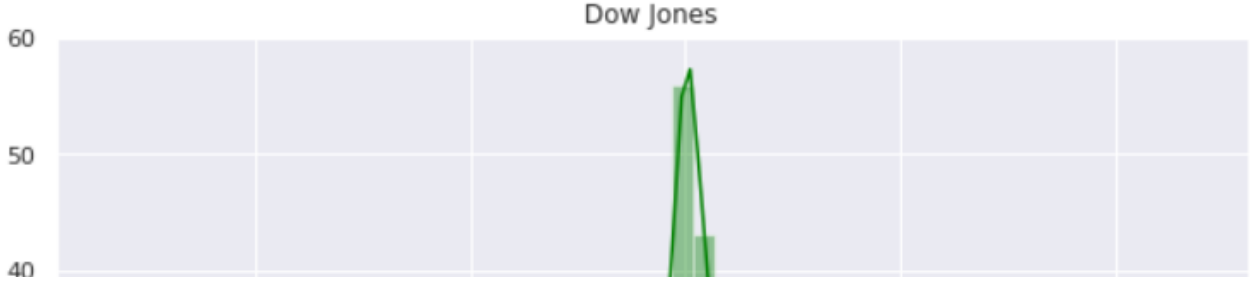
Daily VaR Using Empirical Quantiles:

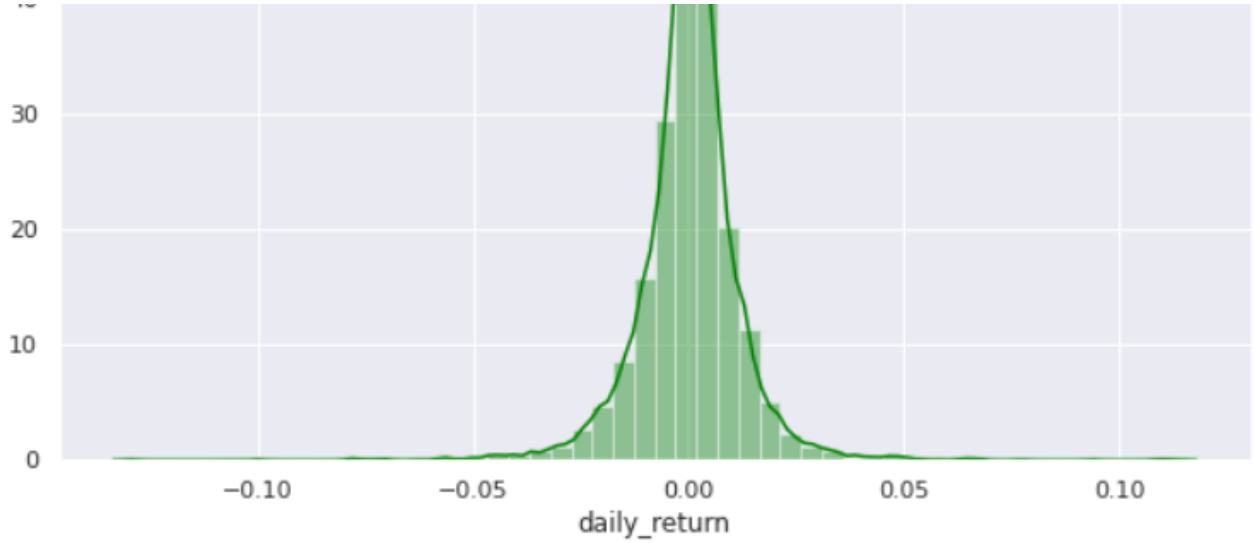
Hypothesis: daily returns are normally distributed. We can visualize the distribution plot on daily return as shown below.

Method: analytic quantiles by curve fitting to historical data here, Student's t distribution

```
dataset['daily_return'] = dataset['Adj Close'].pct_change()
round(dataset["daily_return"],2).quantile(0.05)
```

```
sns.despine(left=True)

# Plot a simple histogram with binsize determined automatically
sns.distplot(dataset['daily_return'], color="green")
plt.title('Dow Jones')
plt.grid(True)
plt.show()
```

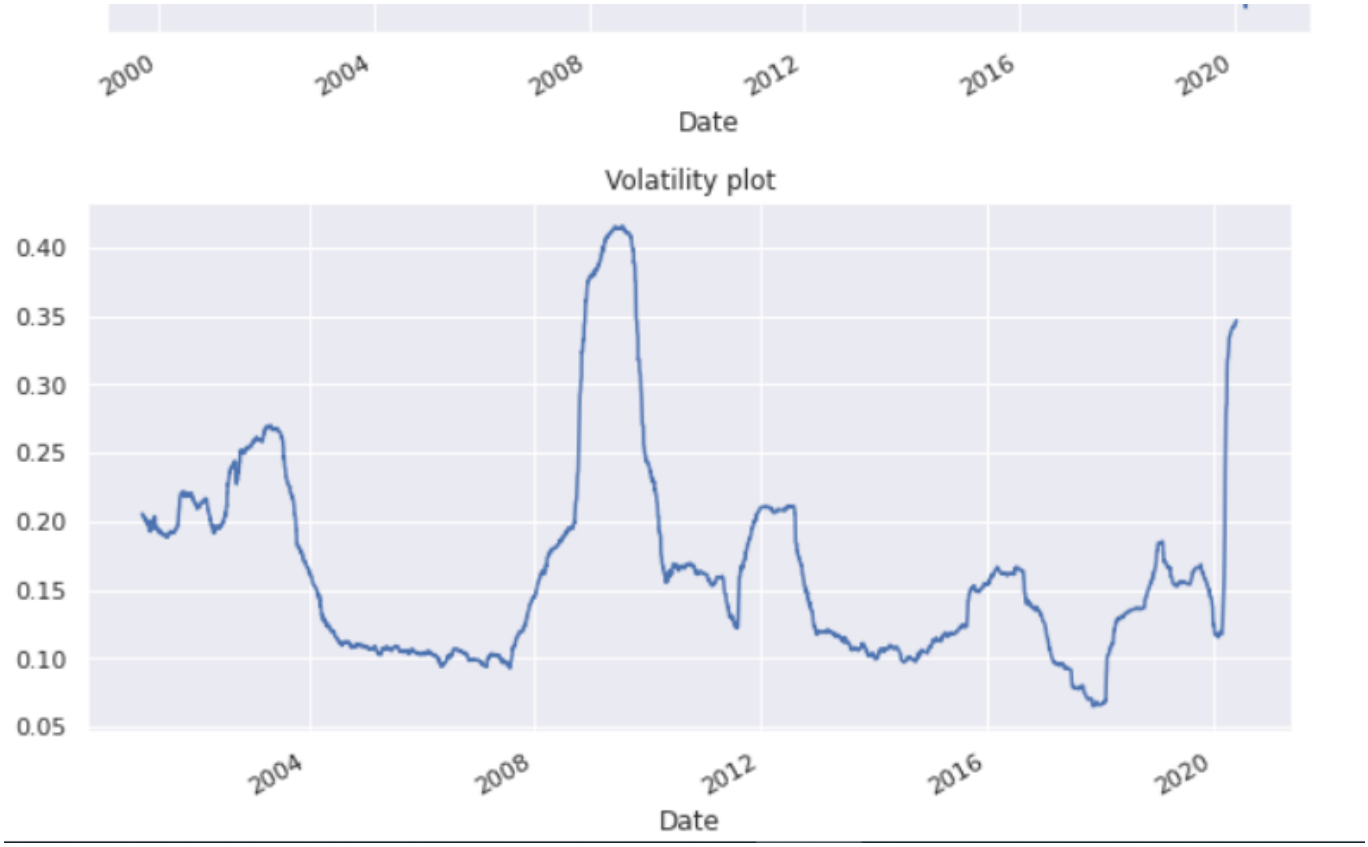




The 0.05 (p=0.05) empirical quantile of daily returns is at -0.02 (the 0.05 quantile is the 5th percentile).

- *with 95% confidence, our worst daily loss will not exceed 2 % of the investment*
- *if we have a €1M investment, our one-day 5% VaR is 0.02 * €1M = € 2k*

```
dataset['daily_return'].plot(figsize = (10,5), grid=True)
plt.title('Daily return plot')
plt.show()

dataset['volatility'] = dataset['daily_return'].rolling(252).std()*(252**0.5)
dataset['volatility'].plot(figsize = (10,5), grid=True)
plt.title('Volatility plot')
plt.show()
```



Notice here from the above plots that, Gaussian White noise has constant volatility. The periods of high & low volatility and the extreme choppiness of the returns. Moreover, we also can see how volatility appears to be time dependent; periods of high volatility persist and periods of low volatility persist.

VaR : Monte Carlo Simulation

We will be generating the Dow Jones price movement based on the historical volatility. With this historical volatility, there will be an element of randomness.

Formula used to generate the next day's price:

> *The last days price and multiplying it by 1 + a random shock which is created by drawing a sample of the distribution given the historical volatility.*

Let us view the prices a year (252 days) back.

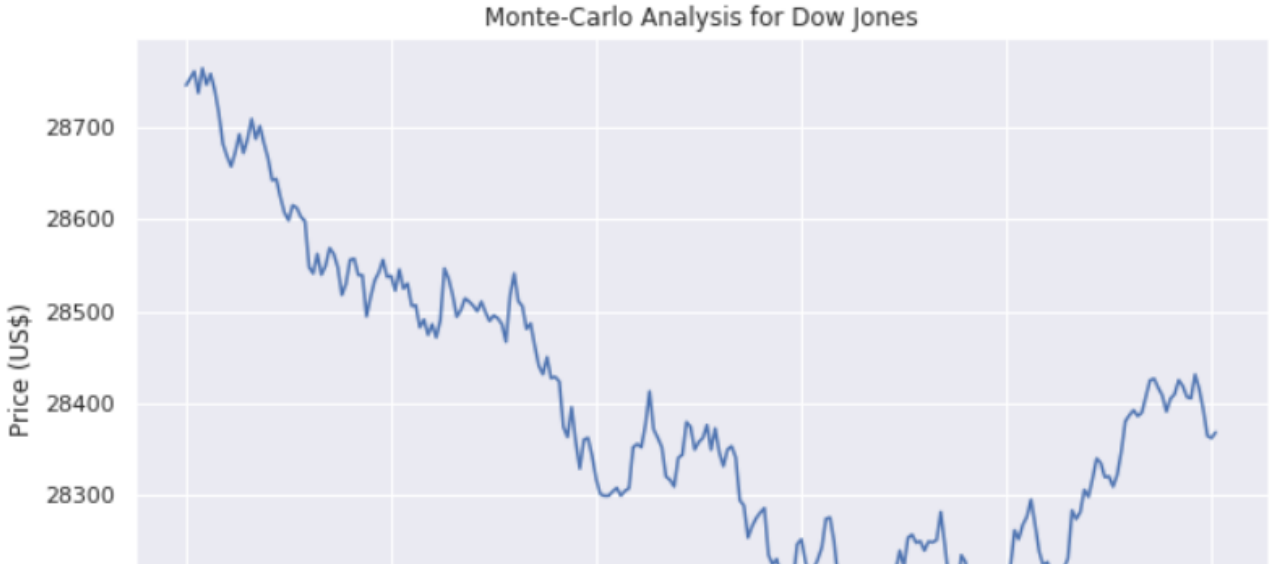
```
# resetting index
dataset.reset_index(inplace = True)
dataset = dataset.sort_values(by = 'Date', ascending=True)
print(dataset.loc[[(len(dataset) -252)]]) # closing price 252 days
back
```

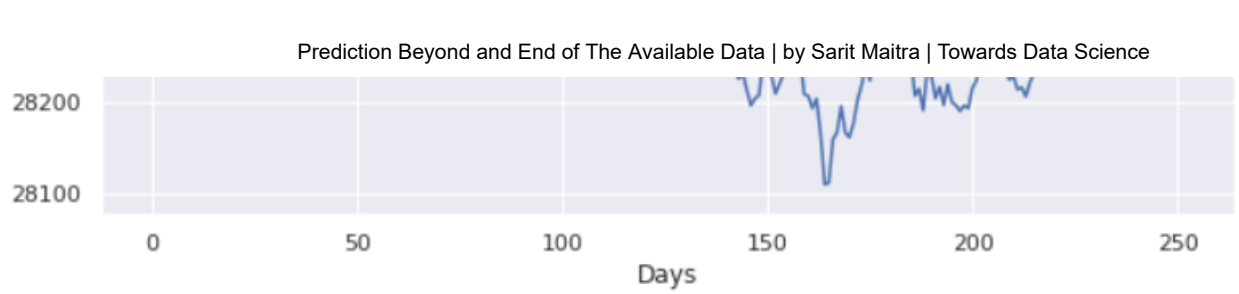
	Date	High	Low	Open	Close	Volume	Adj Close	daily_return	volatility
5035	2020-01-08	28866.179688	28522.509766	28556.140625	28745.089844	291750000	28745.089844	0.005647	0.117112

We see here that, the Close price was $28745.08.

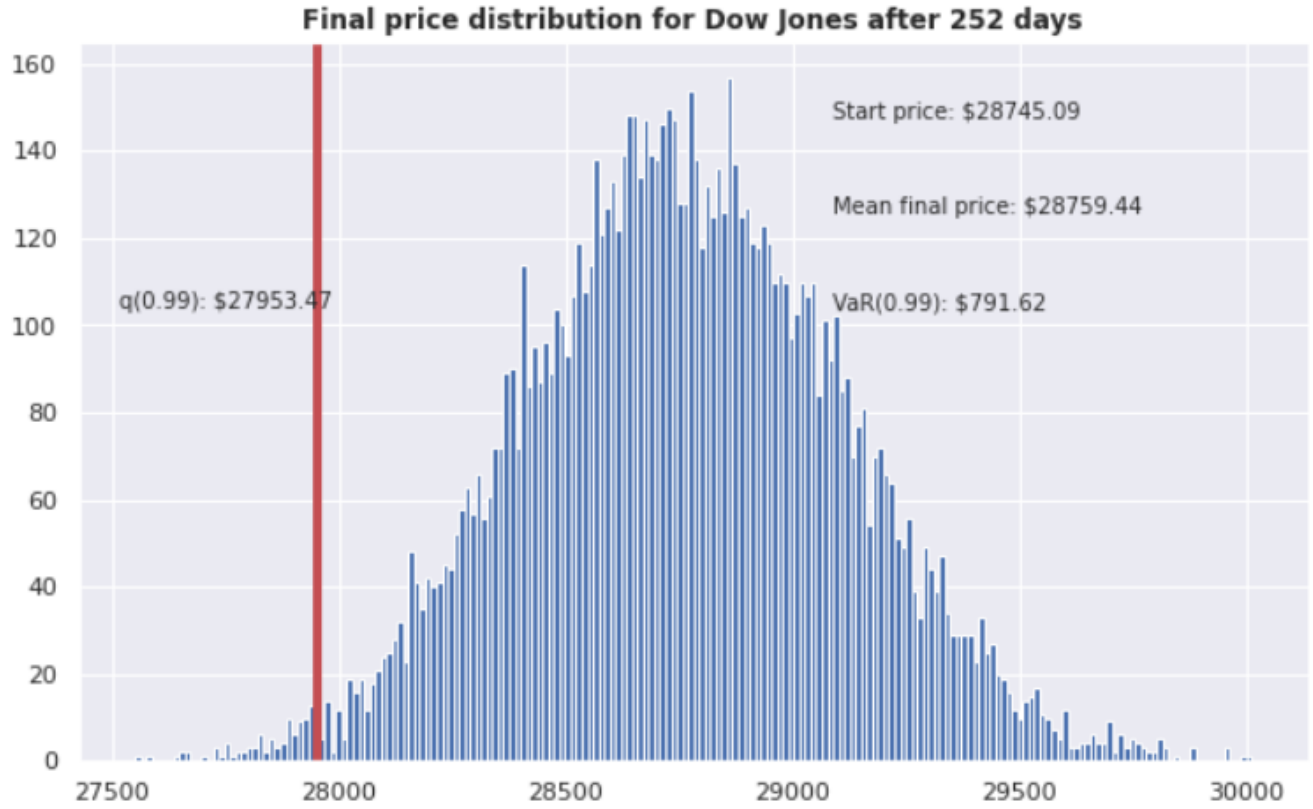
We run a simulation with 10,000 run to obtain the below output.

```
days = 252
start_price = 28745.089844 # Taken from above

#delta t
dt = 1/252
mu = dataset['daily_return'].mean() # mean return
sigma = dataset['daily_return'].std()  # volatility

def stock_monte_carlo(start_price, days, mu, sigma):
    price = np.zeros(days)
    price[0] = start_price
    shock = np.zeros(days)
    drift = np.zeros(days)
    for x in range(1,days):
        shock[x]=np.random.normal(loc=mu*dt,scale=sigma*np.sqrt(dt))
drift[x] = mu * dt
        price[x] = price[x-1] + (price[x-1] * (drift[x]+shock[x]))
    return price
plt.plot(stock_monte_carlo(start_price, days, mu, sigma))
plt.xlabel('Days'); plt.ylabel('Price (US$)')
plt.title('Monte-Carlo Analysis for Dow Jones')
plt.show()
```






```
runs = 10000
simulations = np.zeros(runs)
for run in range(runs):
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)
[days-1]
q = np.percentile(simulations,1)
plt.hist(simulations, bins = 200)
plt.figtext(0.6,0.8,s="Start price: $%.2f" %start_price)
plt.figtext(0.6,0.7,"Mean final price: $%.2f" % simulations.mean())
plt.figtext(0.6,0.6,"VaR(0.99): $%.2f" % (start_price -q,))
plt.figtext(0.15,0.6, "q(0.99): $%.2f" % q)
plt.axvline(x=q, linewidth=4, color='r')
plt.title(u"Final price distribution for Dow Jones after %s days"
%days, weight='bold')
plt.show()
```



The above plot indicates that, Dow Jones price was almost stable since last 1 year; starting Adj Close price was $ 28745, and the average final price over 10,000 runs is almost similar ($ 28759). The red line indicates the VaR at the 99% confidence interval.

We are going to perform ML to predict the future Adj Close price. Let us do some feature engineering which are critical for the performance of the model.

> Time series components are highly important to analyzing the variable of interest in order to understand its behavior, what patterns it has, and to be able to choose and fit an appropriate time-series model. Time series predictors, on he other hand, may help some models to recognize additional patterns and improve the quality of forecasts. Both time series components and features are key to interpreting the behavior of the time series, analyzing its properties, identifying possible causes, and more...Roman Josue de las Heras Torres

Feature Engineering

```
dq = df.copy()
dq['ho'] = dq['High'] - dq['Open']
dq['lo'] = dq['Low'] - dq['Open']
dq['oc'] = dq.Open - dq.Close
dq['hl'] = dq.High - dq.Low
dq['volume_gap'] = dq.Volume.pct_change()
dq['day_of_week'] = dq.index.dayofweek
dq['day_of_month'] = dq.index.day
ema_12 = dq['Adj Close'].ewm(span=10).mean()
ema_26 = dq['Adj Close'].ewm(span=26).mean()
dq['ROC'] = ((dq['Adj Close'] - dq['Adj Close'].shift(5)) / (dq['Adj
Close'].shift(5)))*100
delta = dq['Adj Close'].diff()
window = 14
```

```
up_days = delta.copy()
up_days[delta<=0]=0.0
down_days = abs(delta.copy())
down_days[delta>0]=0.0
RS_up = up_days.rolling(window).mean()
RS_down = down_days.rolling(window).mean()
dq['rsi'] = 100-100/(1+RS_up/RS_down)
dq['macd'] = ema_12 - ema_26

#print(dataset)

# lag featuers

lags = 3
# Create the shifted lag series of prior trading period close values
for i in range(0, lags):
    dq["Lag%s" % str(i+1)] = dq["Adj Close"].shift(i+1).pct_change()

# target variable
future_pred = int(15)
dq['prediction'] = dq['Adj Close'].shift(-future_pred)
dq.head()
```

Relative Strength Index (RSI) & Moving Average Convergence Divergence (MACD)

Interested may visit ***here*** to get details technical indicators.

Target Variable

We have created target variable to predict future 15 days values which are not available in the current data-set.

```
dq = dq.drop(['High','Low','Open','Close','Volume', 'Adj Close'],1)
dq.dropna(inplace=True)

# creating X,y set
X = dq.drop(['prediction'],1)
X_fcast = X[-future_pred:] # future prediction set
X = X[:-future_pred] # removing last 15 rows
y = y[:-future_pred]

# splitting train/test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=42)
```

No, let us fit a linear (OLS) model as base model and observe the model we obtain:

```
ols = linear_model.LinearRegression().fit(X_train, y_train)
# Explained variance score: 1 is perfect prediction
print('Training Variance score (R^2)', r2_score(y_train,
ols.predict(X_train)))
# Explained variance score: 1 is perfect prediction
print('Test Variance score (R^2): ', r2_score(y_test,
ols.predict(X_test)))
```

```
Training Variance score (R^2) 0.23788335169817953
Test Variance score (R^2):  0.22862820875255985
```

Extreme Gradient Boosting

We fit the model using XGBRegressor and subsequently check the confidence level on test set, also obtain the predicted values (15 days). XGBoost has the ability to deal with missing values in data. Moreover, data scaling is not necessary for tree based algorithms.

```
reg = XGBRegressor(objective ='reg:squarederror',
n_jobs=-1).fit(X_train, y_train)
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_train,
reg.predict(X_train)))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, reg.predict(X_test)))
```

```
Variance score: 0.56
Variance score: 0.47
```

Though the performance has improved from base model, but we are not quite there considering we have used regression algorithm. At this stage we can go back look into the features and work more on relevant features.

Once we are satisfied with the accuracy, we may work on future forecast using the forecast data kept separated. The procedure is shown below.

```
prediction = reg.predict(X_fcast)
print('\033[4mExpected Close price for next 15 days\033[0m')
print(prediction);
```

```
Expected Close price for next 15 days
[24754.184 27600.414 24645.855 25840.373 25377.803 29219.156 29659.37
 26990.732 27710.    27163.414 27743.066 26941.684 26479.3   27216.568
 30789.85 ]
```

We need to have dates with the above values for better understanding. Actually, we have predicted for 2 weeks as shown below. Below lines of codes generates future dates for the obtained predicted values.

```
d = df[['Adj Close']].tail(len(prediction));
d.reset_index(inplace = True)
d = d.append(pd.DataFrame({'Date': pd.date_range(start =
d.Date.iloc[-1],
periods = (len(d)+1), freq = 'D', closed = 'right')}))
d = d.tail(future_pred);
d.set_index('Date', inplace = True)
prediction = pd.DataFrame(prediction)
prediction.index = d.index
prediction.rename(columns = {0: 'Forecasted_price'}, inplace=True)
prediction
```

Anticipated price

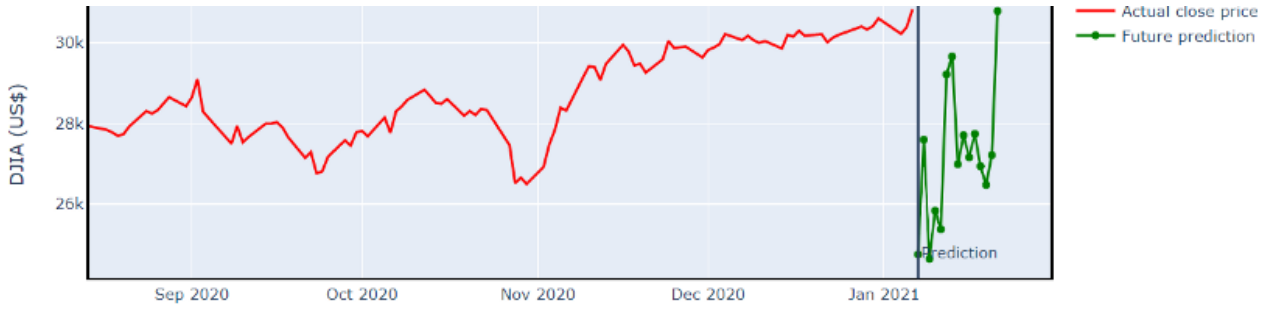
Date	Forecasted_price
2021-01-07	24754.183594
2021-01-08	27600.414062
2021-01-09	24645.855469
2021-01-10	25840.373047
2021-01-11	25377.802734
2021-01-12	29219.156250

2021-01-13	29659.369141
2021-01-14	26990.732422
2021-01-15	27710.000000
2021-01-16	27163.414062
2021-01-17	27743.066406
2021-01-18	26941.683594
2021-01-19	26479.300781
2021-01-20	27216.568359
2021-01-21	30789.849609

Visualization

The plot adding observed (last 2 months) and predicted values (future 1 month) helps us to understand if the model is been able to capture the pattern from historical data. The future forecast looks realistic here.

```
fig = go.Figure()
n = prediction.index[0]
fig.add_trace(go.Scatter(x = df.index[-100:], y = df['Adj Close']
[-100:], marker = dict(color ="red"), name = "Actual close
price"))fig.add_trace(go.Scatter(x = prediction.index, y =
prediction['Forecasted_price'], marker=dict(color = "green"), name =
"Future prediction"))
fig.update_xaxes(showline = True, linewidth = 2, linecolor='black',
mirror = True, showspikes = True,)
fig.update_yaxes(showline = True, linewidth = 2, linecolor='black',
mirror = True, showspikes = True,)
fig.update_layout(title= "15 days days DJIA Forecast",yaxis_title =
'DJIA (US$)',hovermode = "x",hoverdistance = 100, spikedistance =
1000,shapes = [dict(x0 = n, x1 = n, y0 = 0, y1 = 1, xref = 'x', yref
= 'paper',line_width = 2)],annotations = [dict(x = n, y = 0.05, xref
= 'x', yref = 'paper', showarrow = False,xanchor = 'left', text =
'Prediction')])
fig.update_layout(autosize = False, width = 1000, height = 400,)
fig.show()
```



We can clearly see a poor prediction from above plot.

Besides working on features, we need to revisit and look at our target variable which is not shifted in this experiment and we may have induced look ahead bias by doing so. We can try experimenting with df['Adj Close'].shift(-1) which gives future prediction.

> When we're ready to forecast the future in real time, we should of course use all the available data for estimation, so that the most recent data is used.

Summary

We may try other algorithms like support vector or random forest. Although, XGB can learn much more complex nonlinear representations but need more data set to perform better. We have not applied any fundamental variables, all the predictor variables used here are likely having a linear relationships with target variables.

> Tree based algorithms are best among the supervised machine learning algorithms for prediction problems. These methods are able to capture complex relationships in the data by learning many simple rules, one rule at a time.

Cross validation and algorithm optimization techniques need to be applied to finalize a robust model performance. By looking at the bias/variance we may adjust over-fitting or under-fitting issues; we may also introduce early stopping rounds which have not been

exercised here. It would be a good idea to test few other model and may introduce an ensemble of best models to check the performance and accuracy score.

Moreover, holding data out for validation purposes is an important diagnostic test because it gives the best indication of the accuracy that can be expected when forecasting the future. It may be a good practice to hold out at least 20% of data for validation purposes. However, with a lot of data availability, holding out 50% can also be tried.

I can be reached here.

References:

1. *Bergmeir, Christoph, and José M. Benítez. "On the use of cross-validation for time series predictor evaluation." Information Sciences 191 (2012)*
2. *Bergmeir, Christoph, Rob J. Hyndman, and Bonsoo Koo. "A note on the validity of cross-validation for evaluating autoregressive time series prediction." Computational Statistics & Data Analysis 120 (2018)*
3. *Burman, Prabir, Edmond Chow, and Deborah Nolan. "A cross-validatory method for dependent data." Biometrika 81.2 (1994)*

Time Series Forecasting | Supervised Learning | Machine Learning