Predicting Cryptocurrency Prices

Group 4
Sumit | Moises | Sachin

Objective

- Digital forms of money are emerging as an alternative for customary cash all over the world.
- Cryptocurrencies popularity and wide acceptance is an indication that cash is going to experience a noteworthy change.
- We propose a prototype to predict the direction and changes in prices of cryptocurrency (Bitcoin) based on the historical time-series bitcoin data and historical trends in Google search.

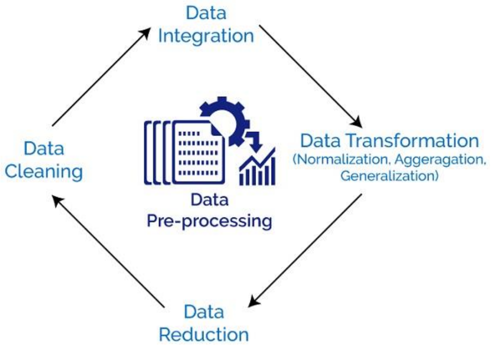
Dataset

1. **Historical data of cryptocurrency:**
    a. Dataset containing all the historical per minute prices for all cryptocurrencies as listed on CoinMarketCap.
    b. Every record in the dataset contains information for an opening price, high price, low price and closing price for a minute.
    c. Around 2.5 million record entries only for Bitcoin.
2. **Google trends data of Bitcoin:**
    a. Monthly search trend for Bitcoin for every month in the timeframe from 2013 to 2018.
    b. Daily search trend for Bitcoin for each month in the timeframe from 2013 to 2018.
    c. Average search per month for last 12 months [1,100,000 search/month].
    d. Total size is around 1GB (datasets combined)

Evaluation

- ➔ We plan to test our model by splitting the data into 80:20 ration where we will use 80% of the data for training the data and the remaining 20% will be used to compare the predicted prices with the actual prices.
- ➔ The accuracy can be determined by calculating the R-squared value (the percentage of the response variable variation that is explained by a linear model).
- ➔ Also, the accuracy of the model can be tested by calculating the Root Mean Square Error (RMSE) where lesser the value of RMSE means better the prediction model.

Data Preprocessing



- The data from Google trends from Jan-01-2013 to Apr-27-2013 will be ignored while constructing the model.
- Some missing values were fetched from Coin Market Cap and integrated with the data.
- Google trends data is integrated with the Bitcoin price data using record date as entity identifier.
- Converted Linux timestamp to date-format.
- Bitcoin prices for opening, high, low and closing prices have been normalized to values lying in range 0 to 1.
- The time series data collected per minute is reduced to data per day by recording the open, high, low and close price for the day for all cryptocurrencies.
- After this reduction, the data is more reduced to only Bitcoin cryptocurrency among all the cryptocurrencies recorded.

Technologies/Tools

Tools/Language

- Python 3.7
- Jupyter Notebook

Libraries

- Pandas
- Numpy
- Matplotlib
- Keras
- Tensorflow
- Statsmodels

Models

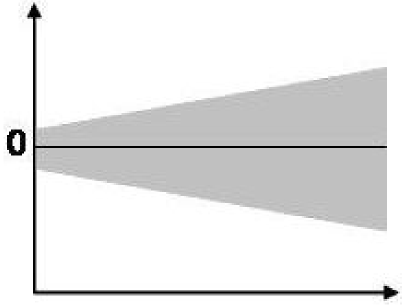
Linear Regression - Introduction

Simple linear regression is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables:

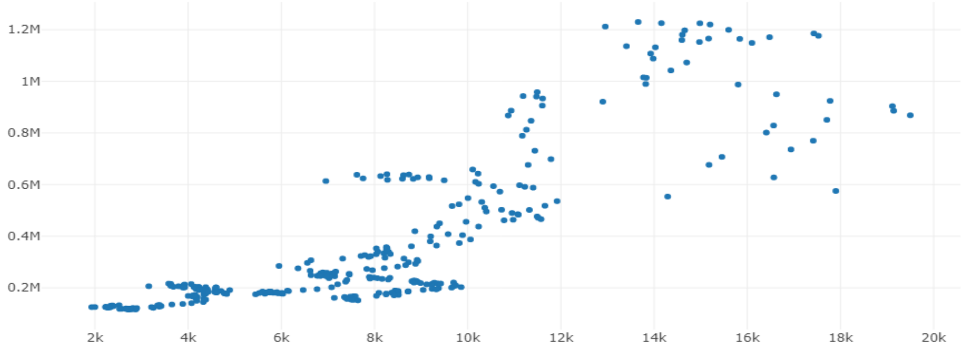
- One variable, denoted x, is regarded as the **predictor**, **explanatory**, or **independent** variable.
- The other variable, denoted y, is regarded as the **response**, **outcome**, or **dependent** variable.

Assumptions for Linear Regression:

- Linear relationship
- Multivariate normality
- No or little multicollinearity
- No auto-correlation
- Homoscedastic- Residuals are equal across the regression line



Linear Regression - Linear relationship

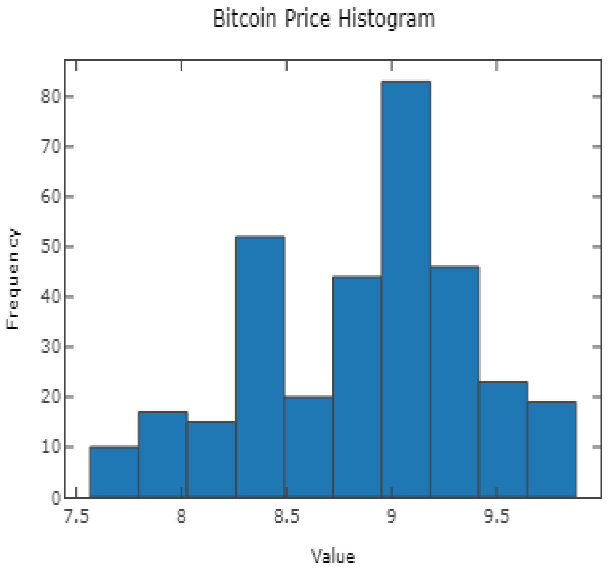



```
data.corr()
```

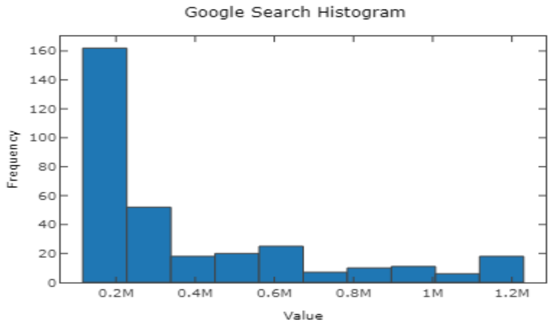
	Avg_5WeekSearch	Close
Avg_5WeekSearch	1.000000	0.874036
Close	0.874036	1.000000

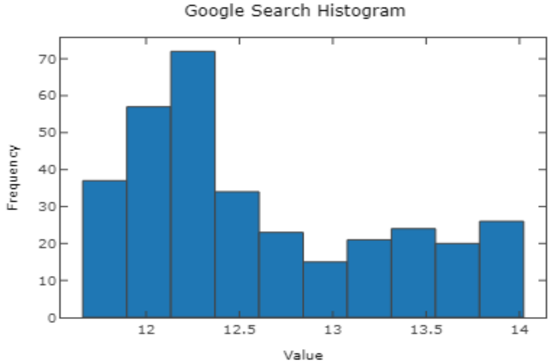
Linear Regression - Multivariate normality

Bitcoin price distribution:



Google Search





Linear Regression - Auto-correlation

OLS Regression Results

Dep. Variable:	Close	**R-squared:**	0.918
Model:	OLS	**Adj. R-squared:**	0.918
Method:	Least Squares	**F-statistic:**	2932.
Date:	Wed, 05 Dec 2018	**Prob (F-statistic):**	2.77e-144
Time:	21:34:12	**Log-Likelihood:**	184.16
No. Observations:	263	**AIC:**	-366.3
Df Residuals:	262	**BIC:**	-362.8
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Avg_5WeekSearch	0.8113	0.015	54.147	0.000	0.782	0.841

Omnibus:	1.774	**Durbin-Watson:**	1.805
Prob(Omnibus):	0.412	**Jarque-Bera (JB):**	1.749
Skew:	0.198	**Prob(JB):**	0.417
Kurtosis:	2.942	**Cond. No.**	1.00

Linear Regression - Model

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=True)
regressor.fit(X_train, Y_train)
from sklearn.metrics import r2_score
Y_pred = regressor.predict(X_test)
r2_score(Y_test, Y_pred)
```

```
0.8182012069882014
```

Linear Regression - Result

```
print(regressor.coef_)
```

```
[0.68834675]
```

```
print(regressor.intercept_)
```

```
0.07684920329636741
```

```
from sklearn.metrics import r2_score
r2_score(Y_test, Y_pred)
```

```
0.8182012069882014
```

Linear Regression - Result

The picture can't be displayed.

Neural Network - Classification problem

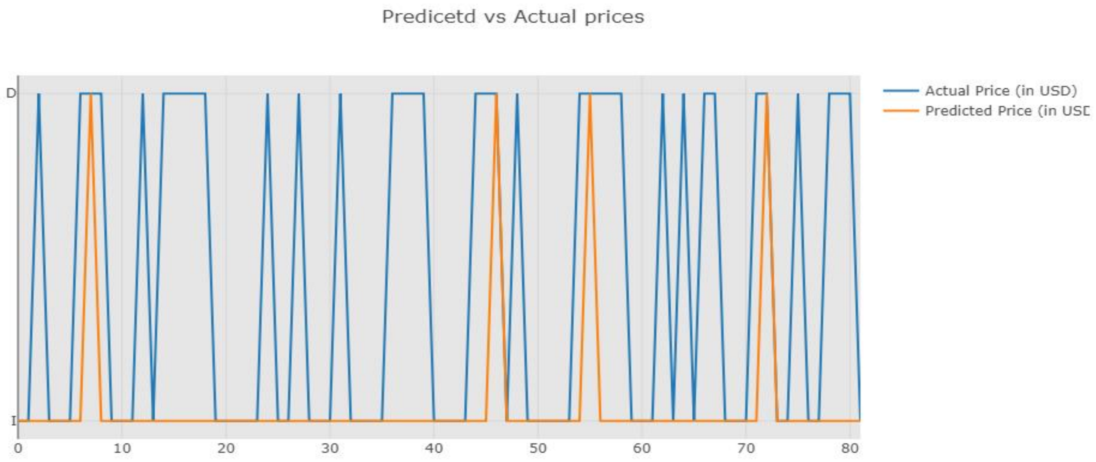
Labelled data into 3 classes - (D,I,S)

- D - If the value of bitcoin decreases on next day.
- I - If the value of bitcoin increases on next day.
- S - If the value of bitcoin remains same on next day.

Neural Network - Model

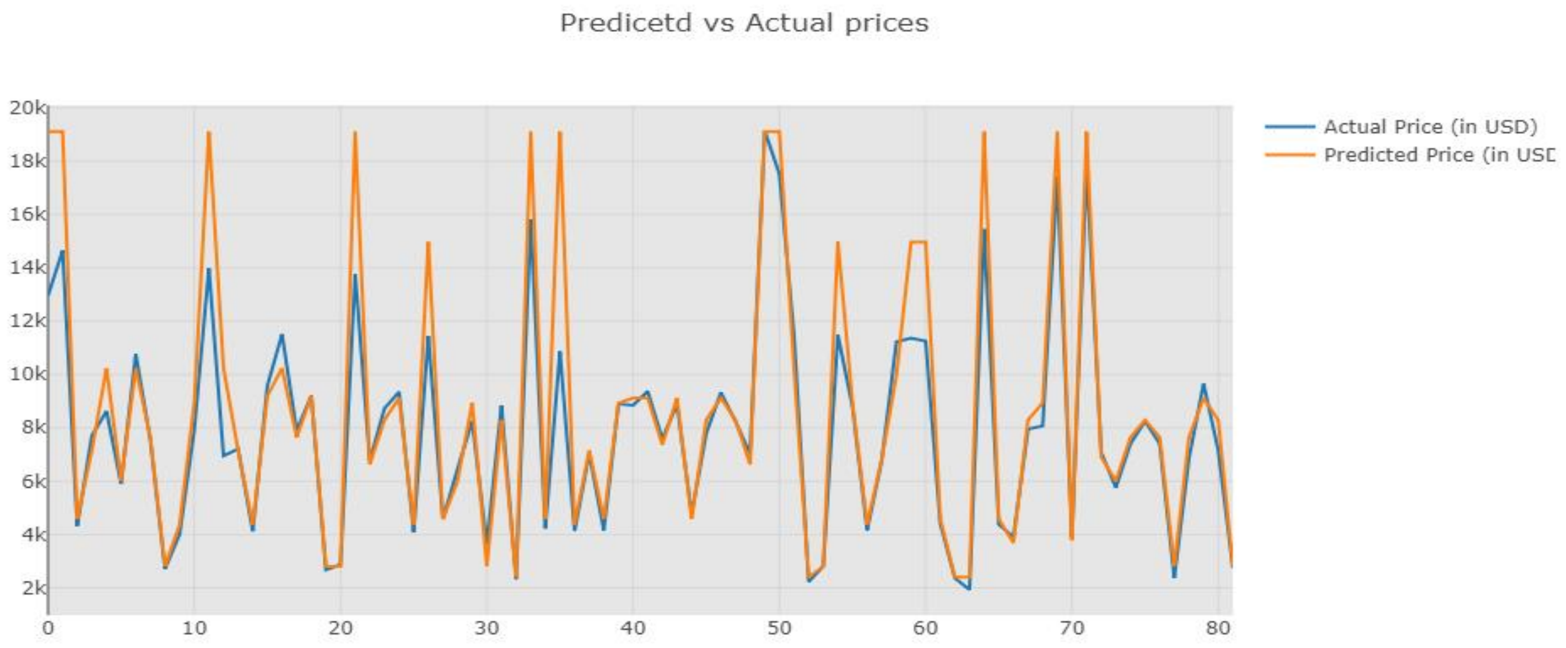
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(6, 6, 6), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
mlp.fit(X_train,y_train)
predictions = mlp.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

Neural Network - Result



	precision	recall	f1-score	support
D	1.00	0.11	0.20	36
I	0.59	1.00	0.74	46
avg / total	0.77	0.61	0.50	82

Neural Network - Regression Problem



ARIMA - Introduction

- ARIMA model is used on time-series data for forecasting.
- ARIMA (Auto Regressive Integrated Moving Average) Parameters:
  - p - the number of lag observations to include in the model, or lag order. (AR)
  - d - the number of times that the raw observations are differenced, or the degree of differencing. (I)
  - q - the size of the moving average window, also called the order of moving average.(MA)

ARIMA - Non Stationary vs Stationary

Non Stationary

In [8]:
```python
#seasonal_decompose(btc_month.close, freq=12).plot()
seasonal_decompose(btc_month.close).plot()
print("Dickey-Fuller test: p=%f" % adfuller(btc_month.close)[1])
plt.show()
```

```
Dickey-Fuller test: p=0.998818
```

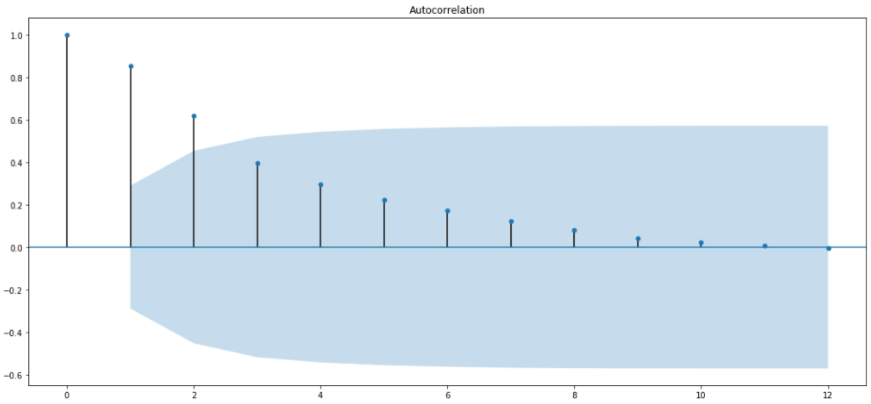
Stationary

In [12]:
```python
# Regular differentiation
btc_month['box_diff2'] = btc_month.box_diff_seasonal_12 - btc_month.box_diff_seasonal_12.shift(
1)

# STL-decomposition
seasonal_decompose(btc_month.box_diff2[13:]).plot()
print("Dickey-Fuller test: p=%f" % adfuller(btc_month.box_diff2[13:])[1])

plt.show()
```

```
Dickey-Fuller test: p=0.002295
```

ARIMA - Autocorrelation

```
In [13]:
#autocorrelation_plot(btc_month.close)
plot_acf(btc_month.close[13:].values.squeeze(), lags=12)

plt.tight_layout()
plt.show()
```



- There is a positive correlation with the first 10 lags that is perhaps significant for the first 2-3 lags.
- A good starting point for the AR parameter of the model may be 3.

ARIMA - Model Analysis

In [15]:

```
# Initial approximation of parameters
qs = range(0, 3)
ps = range(0, 3)
d=1
parameters = product(ps, qs)
parameters_list = list(parameters)
len(parameters_list)

# Model Selection
results = []
best_aic = float("inf")
warnings.filterwarnings('ignore')
for param in parameters_list:
    try:
        model = SARIMAX(btc_month.close_box, order=(param[0], d, param[1])).fit(disp=-1)
    except ValueError:
        print('bad parameter combination:', param)
        continue
    aic = model.aic
    if aic < best_aic:
        best_model = model
        best_aic = aic
        best_param = param
    results.append([param, model.aic])
```

```
bad parameter combination: (0, 0)
bad parameter combination: (2, 1)
```

In [16]:

```
# Best Models
result_table = pd.DataFrame(results)
result_table.columns = ['parameters', 'aic']
print(result_table.sort_values(by = 'aic', ascending=True).head())
```

```
  parameters         aic
2     (1, 0) -221.187837
0     (0, 1) -220.803378
3     (1, 1) -219.227474
5     (2, 0) -219.218002
1     (0, 2) -219.002799
```

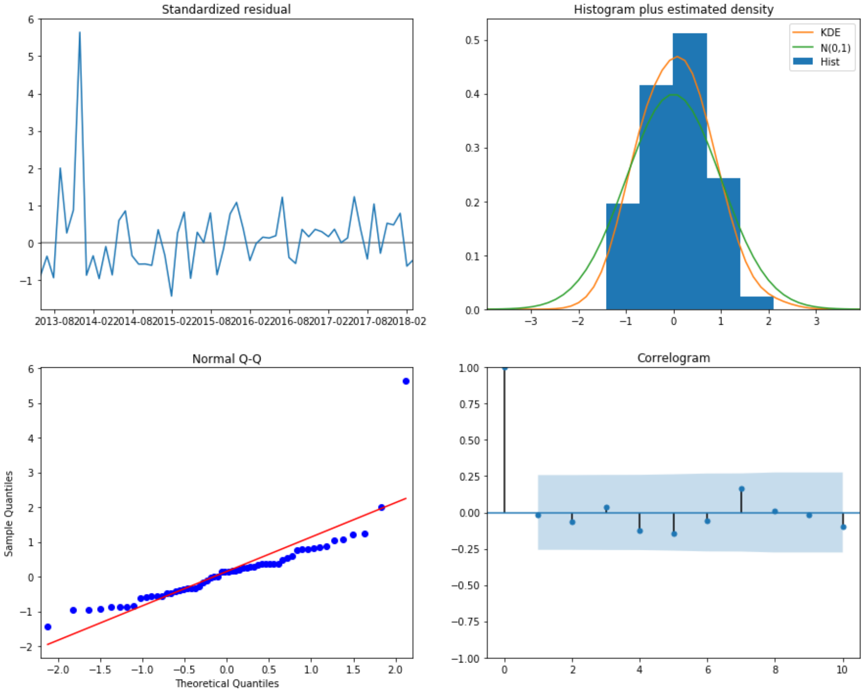
- Given a collection of models for the data, AIC (Akaike Information Criterion) estimates the quality of each model, relative to each of the other models. Thus, AIC provides a means for model selection.
- AIC should be lower.

ARIMA - Best Model

- The best model has following parameters:
  - p (AR) - 1
  - d (I) - 1
  - q (MA) - 0
- The AIC for this model is -221.188

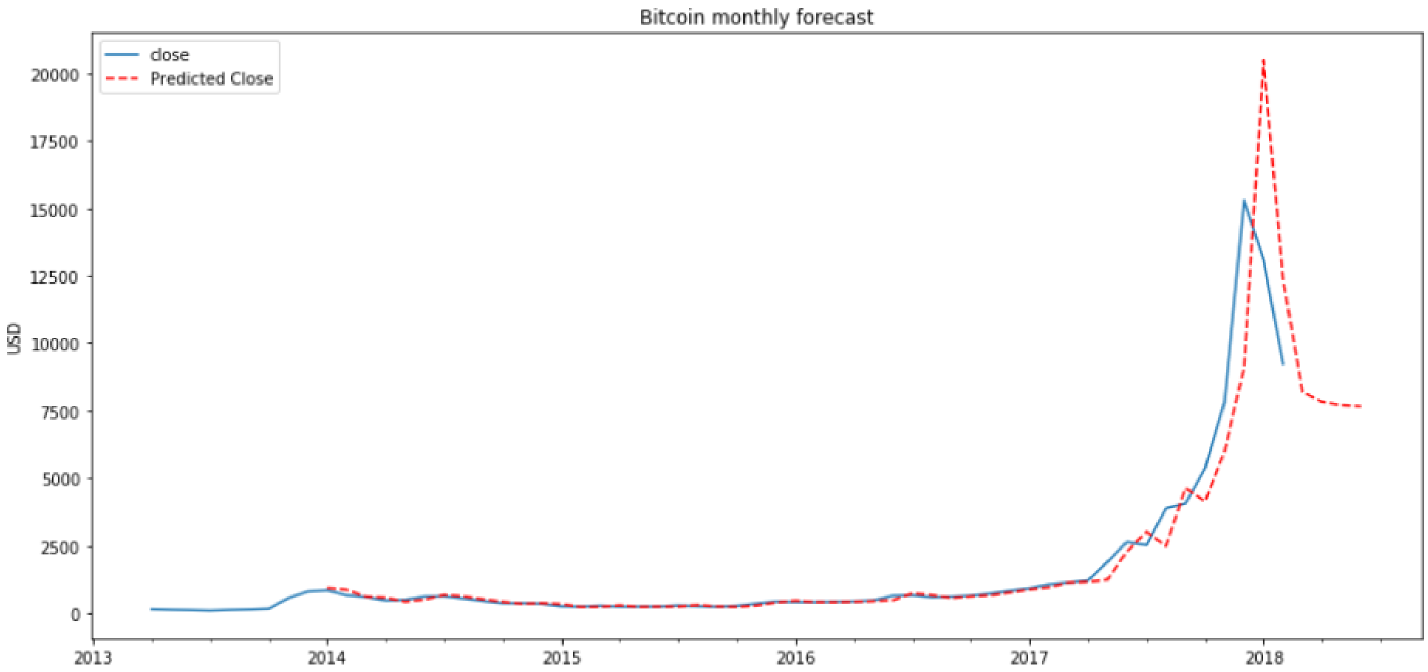
```
Statespace Model Results
==============================================================================
Dep. Variable:                      close box   No. Observations:                   59
Model:                         ARIMA(1, 1, 0)   Log Likelihood                 112.594
Date:                        Sat, 01 Dec 2018   AIC                           -221.188
Time:                                15:06:25   BIC                           -217.033
Sample:                            04-30-2013   HQIC                          -219.566
                                 - 02-28-2018
Covariance Type:                          opg
==============================================================================
```

ARIMA - Best Model Diagnosis



- Our primary concern is to ensure that the residuals of our model are uncorrelated and normally distributed with zero-mean.
- In the histogram (top right), the KDE line should follow the N(0,1) line (normal distribution with mean 0, standard deviation 1) closely.
- In the Q-Q-plot the ordered distribution of residuals (blue dots) should follow the linear trend of the samples taken from a standard normal distribution with N(0, 1).
- The standardized residual plot doesn't display any obvious seasonality.
- This is confirmed by the autocorrelation plot, which shows that the time series residuals have low correlation with lagged versions of itself.

ARIMA - Prediction Result

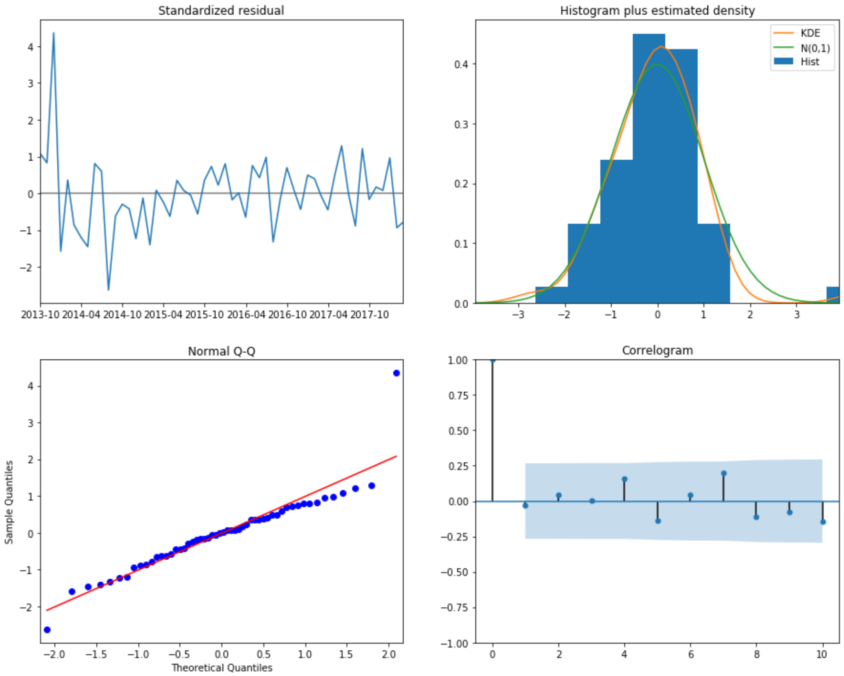


SARIMA - Introduction

- SARIMA is Seasonal ARIMA
- When dealing with seasonal effects, we make use of the seasonal ARIMA, which is denoted as *ARIMA(**p,d,q**)(**P,D,Q**)**s***.
- Here, (***p, d, q***) are the non-seasonal parameters described above, while (***P, D, Q***) follow the same definition but are applied to the seasonal component of the time series.
- The term ***s*** is the periodicity of the time series (4 for quarterly periods, 12 for yearly periods, etc.).

SARIMA - Best Model Diagnosis

- The four plots analyze the residual after applying the chosen parameters. There is no long- or short-term trend remaining in the autocorrelation factor(ACF). However, the histogram plot (upper right) shows that the residual is not perfectly normally distributed: it has a long right tail.
- Q-Q Normal Plot, the graph is between the actual distribution of residual quantiles and a perfectly normal distribution residuals. If the graph is perfectly overlapping on the diagonal, the residual is normally distributed.
- The correlations are very low (the y axis goes from +.1 to -.1) and don't seem to have a pattern.The gray areas are confidence bands (e.g. tell you whether the correlation is significant).



SARIMA - Validation

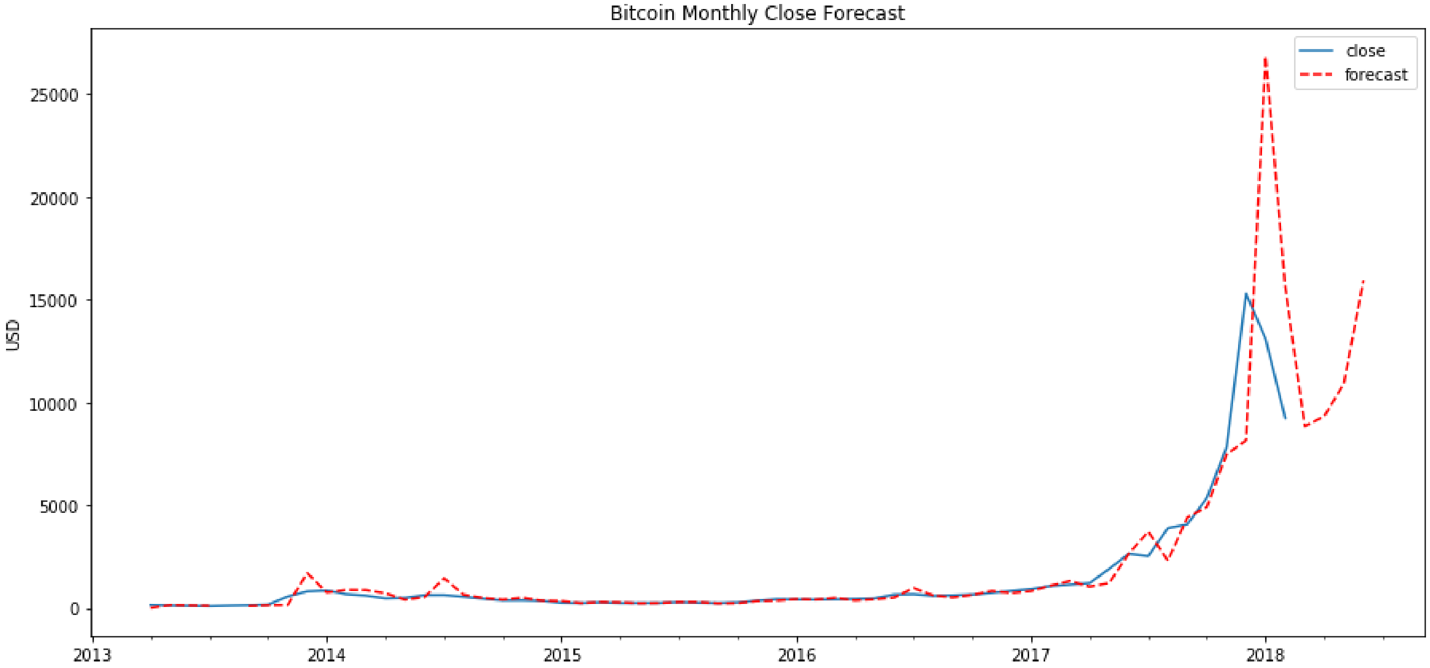
- A simple indicator of how accurate our forecast is is the root mean square error (RMSE).
- The RMSE represents the square root of the second sample moment of the differences between predicted values and observed values or the quadratic mean of these differences.
- Lower the value, the better the predictions for that model.

In [28]:

```
y_forecasted = btc_month2.forecast
y_truth = btc_month2['2015-01-01':'2017-01-01'].close

# Compute the root mean square error
rmse = np.sqrt(((y_forecasted - y_truth) ** 2).mean())
print('Mean Squared Error: {}'.format(round(rmse, 2)))
```

```
Mean Squared Error: 85.18
```

SARIMA - Prediction Result




ANY
QUESTIONS?