BITCOIN TIME SERIES FORECASTING

Basic Analysis

ABSTRACT

We examine the bitcoin market's predictability over time horizons spanning of 6 years. We investigate a variety of machine learning models and discover that, while all models outperform a random classifier, recurrent neural networks and gradient boosting classifiers are particularly well-suited for the prediction tasks under consideration.

Sanket Patel

Member, SIME

Introduction

The following article is about Crypto/Stock prediction algorithms. In this study, I analyze the forecasting of the bitcoin market. Therefore, we utilize a variety of machine learning methods and consider a comprehensive set of potential market-predictive features.

Empirical asset pricing is a major branch of financial research. Machine learning methods have been applied increasingly within this domain, due to the ability to flexibly select amongst a potentially large number of features and to learn complex, high-dimensional relationships between features and targets. Although a considerable body of research has examined the pricing of equities and bonds, yielding a substantial number of potentially market-predictive factors, less attention has been paid to the novel stream of cryptocurrency pricing. In particular, the short-term predictability of the bitcoin market has not yet been analyzed comprehensively. Furthermore, most studies have solely considered technical features and have not analyzed the feature importance of the employed machine learning models. Against this backdrop, we tackle this research gap by comparatively analyzing different machine learning models for predicting market movements of the most relevant cryptocurrency bitcoin.

Related work

Financial market prediction is a prominent branch of financial research and has been studied extensively. There is mixed evidence regarding the predictability and efficiency of financial markets. An established approach to analyze return-predictive signals is to conduct regression analysis on possible signals to explain asset returns. However, linear regressions are not able to flexibly incorporate a large number of features and impose strong assumptions on the functional form of how signals indicate market movements. In contrast, machine learning methods, which often do not impose those restrictions, have been increasingly applied for financial market prediction. Among those, neural network-based methods may be expected to be particularly well-suited, as they are already described to be the predominant method for predicting the dynamics of financial markets.

Methodology

Bitcoin data is an example of time series. I implement data gathering, preprocessing, and model building using the Python programming language and the libraries of Data Processing (NumPy, Panda, DateTime), Data visualization (Matplotlib, Plotly, seaborn), and Machine Learning Libraries (Keras, scikit-learn, and XGBoost).

This Kernel is divided into two parts: -

- Data Exploration

- Time Series Analysis

And further for the Time Series Forecasting: -

- Time Series forecasting with LSTM
- Time Series forecasting with XGBoost

Data

I have collected Bitcoin Historic data (01/01/2016 – 23/03/2022) from Yahoo Finance in Excel format. This data serves as a very good dataset for time series forecasting. It consists of opening, closing, the lowest and highest price of the bitcoin on a particular day, including the volume traded that day.

Data is saved as BTC-USD.xlsx in local Computer

You can take download all types of stock and Crypto data from Yahoo Finance of a considerable range to work in this model

Software

Python 3.9.7

Jupyter Notebook and Dataspell (alternate – Google Collab and Datalore)

Microsoft Excel

Code Explained –

- Import basic Libraries for Data processing and visualization

```
import pandas as pd
import numpy as np
import datetime
from datetime import date, timedelta
import matplotlib.pyplot as plt
from plotly.offline import init_notebook_mode, iplot
from plotly import tools
import plotly as py
import plotly.graph_objs as go
import matplotlib.pyplot as plt
import seaborn as sns
```

- Reading data from excel sheet (basically CSV file is downloaded from the website, but I convert it into excel sheet to prevent the loss of data.

```
data = pd.read_excel("BTC-USD.xlsx" )
data.info()
data = data[["Date","Open","High","Low","Close","Volume"]]
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2274 entries, 0 to 2273
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       2274 non-null   datetime64[ns]
 1   Open       2274 non-null   float64
 2   High       2274 non-null   float64
 3   Low        2274 non-null   float64
 4   Close      2274 non-null   float64
 5   Adj Close  2274 non-null   float64
 6   Volume     2274 non-null   int64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 124.5 KB
```

- Now let's move towards the data cleaning process, our excel sheets might contain some missing value that needs to be solved so that we can take the average value which will later be analyzed and processed as Time Series

```
data.tail()
```

	Date	Open	High	Low	Close	Volume	Average_Price
2269	2022-03-19	41794.648438	42316.554688	41602.667969	42190.652344	19664853187	41976.130860
2270	2022-03-20	42191.406250	42241.164063	41004.757813	41247.824219	20127946682	41671.288086
2271	2022-03-21	41246.132813	41454.410156	40668.042969	41077.996094	24615543271	41111.645508
2272	2022-03-22	41074.105469	43124.707031	40948.281250	42358.808594	32004652376	41876.475586
2273	2022-03-23	42364.378906	42893.507813	41877.507813	42892.957031	25242943069	42507.087891

- Now we have imported data from an excel sheet

```
data['Volume'].fillna(value=0, inplace=True)
data['Open'].fillna(method='ffill', inplace=True)
data['High'].fillna(method='ffill', inplace=True)
data['Low'].fillna(method='ffill', inplace=True)
data['Close'].fillna(method='ffill', inplace=True)
data['Average_Price'] = data["High"]/4 +  data["Low"]/4 + data["Open"]/4 + data["Close"]/4
daily_data = data
```

- Now we have completed the Data Processing part and now our Data Frame is without null value and also have an additional column

```
data.tail()
```

	Date	Open	High	Low	Close	Volume	Average_Price
2269	2022-03-19	41794.648438	42316.554688	41602.667969	42190.652344	19664853187	41976.130860
2270	2022-03-20	42191.406250	42241.164063	41004.757813	41247.824219	20127946682	41671.288086
2271	2022-03-21	41246.132813	41454.410156	40668.042969	41077.996094	24615543271	41111.645508
2272	2022-03-22	41074.105469	43124.707031	40948.281250	42358.808594	32004652376	41876.475586
2273	2022-03-23	42364.378906	42893.507813	41877.507813	42892.957031	25242943069	42507.087891

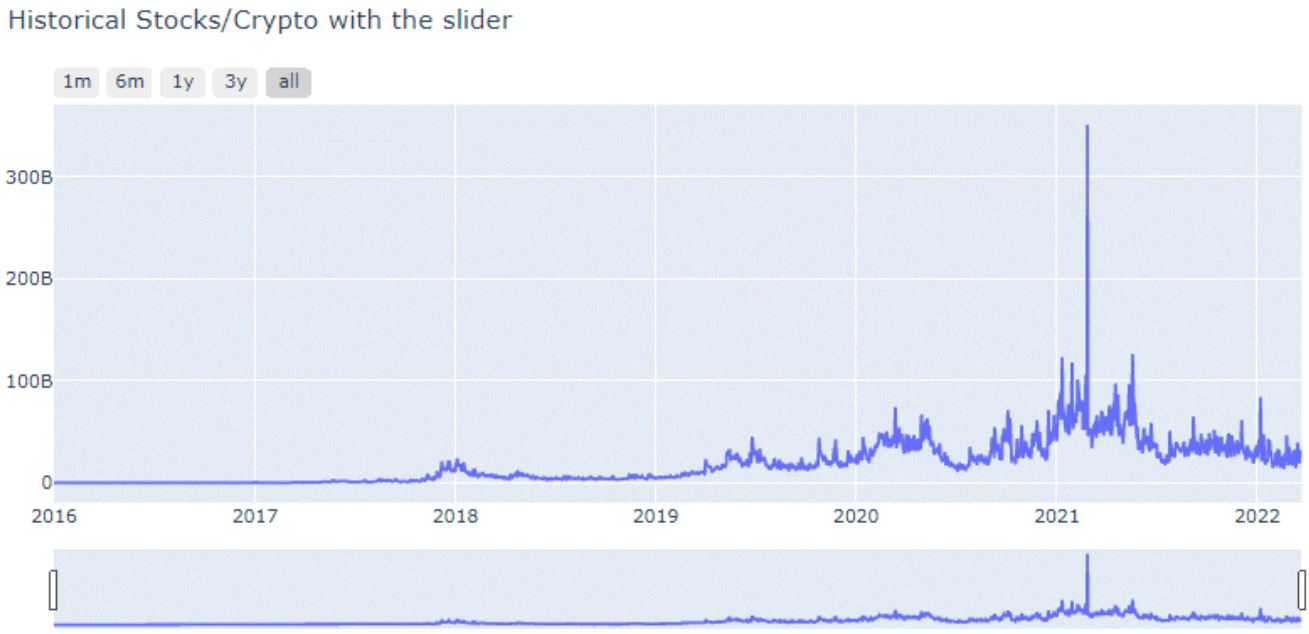
- Let's visualize 'open', 'close', and 'average price' of our data set

```
trace1 = go.Scatter(
    x=daily_data["Date"], y=daily_data["Open"].astype(float), mode="lines", name="Open"
)

trace2 = go.Scatter(
    x=daily_data["Date"],
    y=daily_data["Close"].astype(float),
    mode="lines",
    name="Close",
)
trace3 = go.Scatter(
    x=daily_data["Date"],
    y=daily_data["Average_Price"].astype(float),
    mode="lines",
    name="Average Price",
)
```

```
layout = dict(
    title="Historical Crypto/Stocks Prices with the Slider ",
    xaxis=dict(
        rangeselector=dict(
            buttons=list(
                [
                    dict(count=1, label="1m", step="month", stepmode="backward"),
                    dict(count=6, label="6m", step="month", stepmode="backward"),
                    dict(count=12, label="1y", step="month", stepmode="backward"),
                    dict(count=36, label="3y", step="month", stepmode="backward"),
                    dict(step="all"),
                ]
            )
        ),
        rangeslider=dict(visible=True),
        type="date",
    ),
)

data = [trace1, trace2, trace3]
fig = dict(data=data, layout=layout)
iplot(fig, filename="Time Series with Rangeslider")
```

- Let's analyze the volume of BTC

```
trace1 = go.Scatter(
    x=daily_data["Date"],
    y=daily_data["Volume"].astype(float),
    mode="lines",
    name="Bitcoin Price (Open)",
)

layout = dict(
    title="Historical Stocks/Crypto with the slider",
    xaxis=dict(
        rangeselector=dict(
            buttons=list(
                [
                    dict(count=1, label="1m", step="month", stepmode="backward"),
                    dict(count=6, label="6m", step="month", stepmode="backward"),
                    dict(count=12, label="1y", step="month", stepmode="backward"),
                    dict(count=36, label="3y", step="month", stepmode="backward"),
                    dict(step="all"),
                ]
            )
        ),
        rangeslider=dict(visible=True),
        type="date",
    ),
)

data = [trace1]
fig = dict(data=data, layout=layout)
iplot(fig, filename="Time Series with Rangeslider")
```



- Make time series with Average Price(Trend) and Date(Seasonal Variation)

```
data = daily_data
data = data.groupby([pd.Grouper(key="Date")]).first().reset_index()
data = data.set_index("Date")
data = data[["Average_Price"]]
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2274 entries, 2016-01-01 to 2022-03-23
Data columns (total 1 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Average_Price  2274 non-null   float64
dtypes: float64(1)
memory usage: 35.5 KB
```

- Split Data into two parts Training Data and Test Data by choosing a specific date then splitting into two different data frames

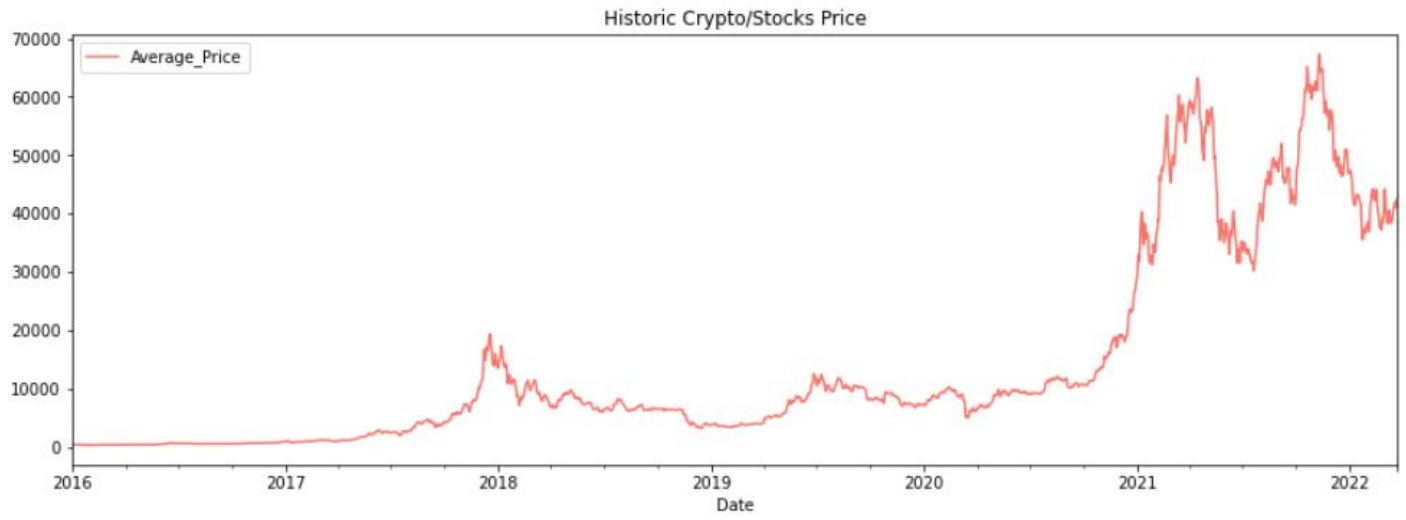
```
split_date = "2021-04-10"
data_train = data.loc[data.index <= split_date].copy()
data_test = data.loc[data.index > split_date].copy()
```

- Shape the training and test dataset properly for proper fitting in model

```
training_set = data_train.values
training_set = np.reshape(training_set, (len(training_set), 1))
from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler()
training_set = sc.fit_transform(training_set)
X_train = training_set[0 : len(training_set) - 1]
y_train = training_set[1 : len(training_set)]
X_train = np.reshape(X_train, (len(X_train), 1, 1))
color_pal = [
    "#F8766D",
    "#D39200",
    "#93AA00",
    "#00BA38",
    "#00C19F",
    "#00B9E3",
    "#619CFF",
    "#DB72FB",
]
```

- Graph before splitting

```
_ = data.plot(
    style="", figsize=(15, 5), color=color_pal[0], title="Historic Crypto/Stocks Price"
)
```



- Graph after splitting

```
_ = (
    data_test.rename(columns={"Average_Price": "Test Set"})
    .join(data_train.rename(columns={"Average_Price": "Training Set"}), how="outer")
    .plot(figsize=(15, 5), title="BTC Weighted_Price Price (USD) by Hours", style="")
)
```

Predicting using LSTM

In the first section, we use LSTM (Long short-term memory). LSTM units are units of a recurrent neural network (RNN). An RNN composed of LSTM units is often called an LSTM network (or just LSTM). A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

- Let's import Keras libraries and packages and build our model

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import Activation


model = Sequential()
model.add(LSTM(128, activation="sigmoid", input_shape=(1, 1)))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss="mean_squared_error", optimizer="adam")
model.fit(X_train, y_train, epochs=100, batch_size=50, verbose=2)
```

```
Epoch 92/100
39/39 - 0s - loss: 0.0036 - 93ms/epoch - 2ms/step
Epoch 93/100
39/39 - 0s - loss: 0.0034 - 87ms/epoch - 2ms/step
Epoch 94/100
39/39 - 0s - loss: 0.0034 - 78ms/epoch - 2ms/step
Epoch 95/100
39/39 - 0s - loss: 0.0035 - 86ms/epoch - 2ms/step
Epoch 96/100
39/39 - 0s - loss: 0.0035 - 93ms/epoch - 2ms/step
Epoch 97/100
39/39 - 0s - loss: 0.0033 - 93ms/epoch - 2ms/step
Epoch 98/100
39/39 - 0s - loss: 0.0034 - 82ms/epoch - 2ms/step
Epoch 99/100
39/39 - 0s - loss: 0.0034 - 88ms/epoch - 2ms/step
Epoch 100/100
39/39 - 0s - loss: 0.0033 - 93ms/epoch - 2ms/step

<keras.callbacks.History at 0x1d969b1c460>
```

- Now we have built our model

```
model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_15 (LSTM)              (None, 128)               66560

 dropout_15 (Dropout)        (None, 128)               0

 dense_15 (Dense)            (None, 1)                 129

=================================================================
Total params: 66,689
Trainable params: 66,689
Non-trainable params: 0
_________________________________________________________________
```

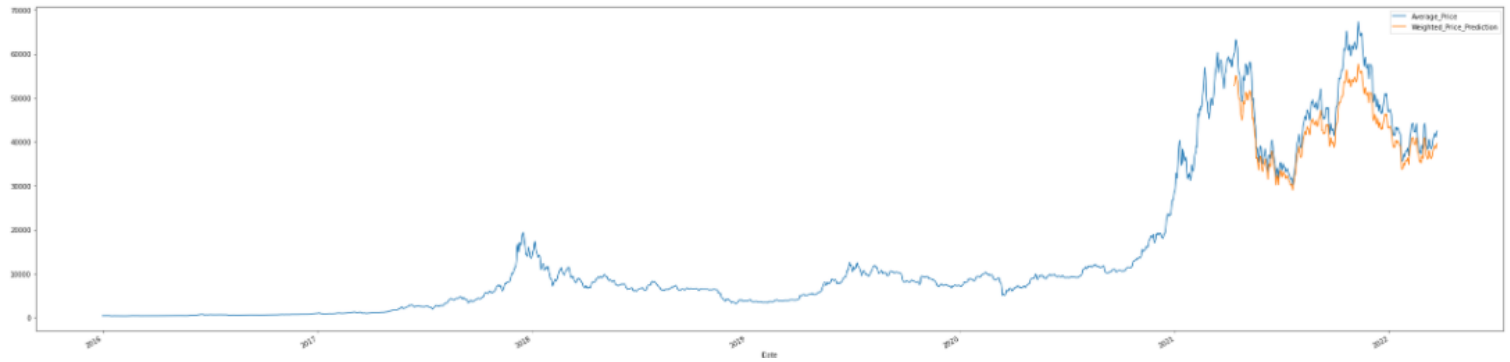
- Fit your dataset in the model and concatenate the predicated model with our original dataset

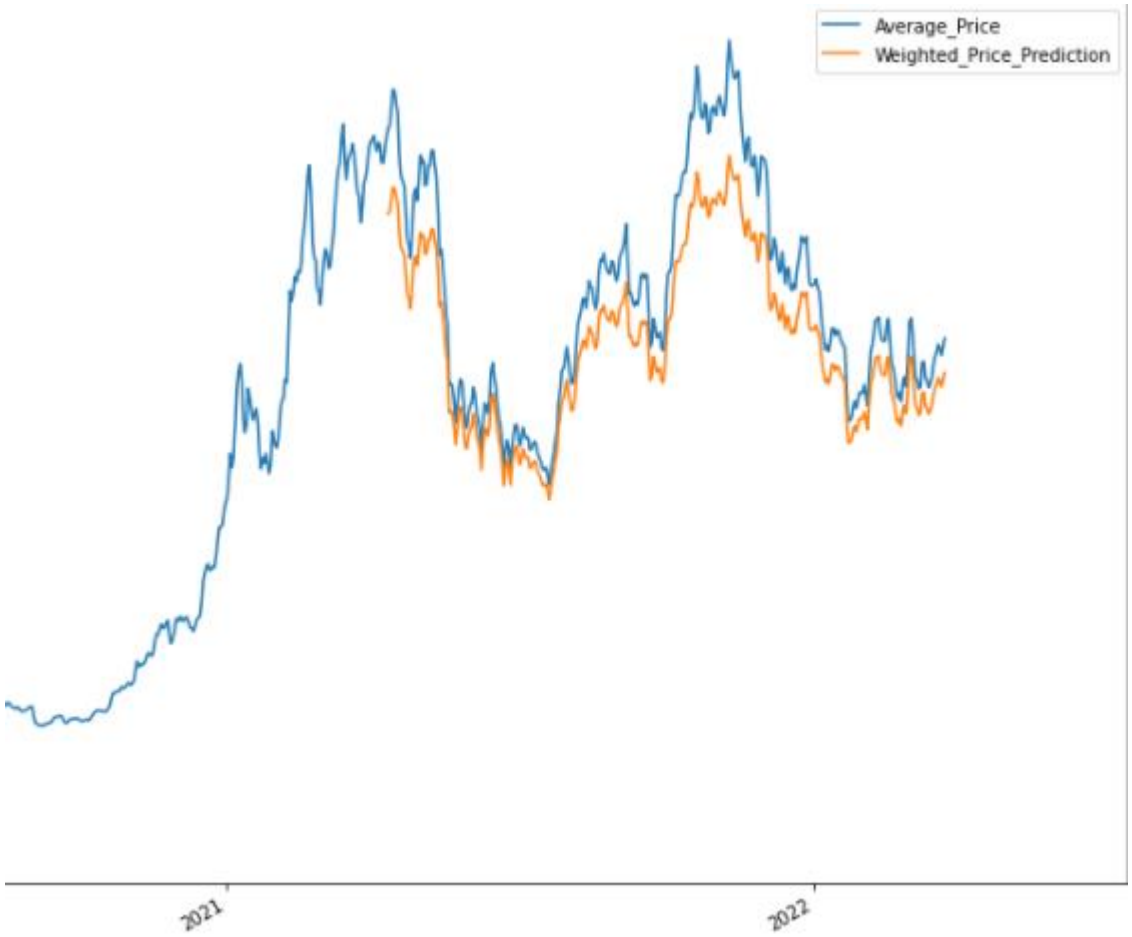
```
test_set = data_test.values
inputs = np.reshape(test_set, (len(test_set), 1))
inputs = sc.transform(inputs)
inputs = np.reshape(inputs, (len(inputs), 1, 1))
predicted_BTC_price = model.predict(inputs)
predicted_BTC_price = sc.inverse_transform(predicted_BTC_price)
```

```
data_test["Weighted_Price_Prediction"] = predicted_BTC_price
data_all = pd.concat([data_test, data_train], sort=True)
final_data = data_all
final_data = final_data.reset_index()
final_data = final_data.rename(columns={"Weighted_Price_Prediction": "lstm"})
final_data = final_data[["Date", "Average_Price", "lstm"]]
```

- Graph of Average Price and Predicted Price

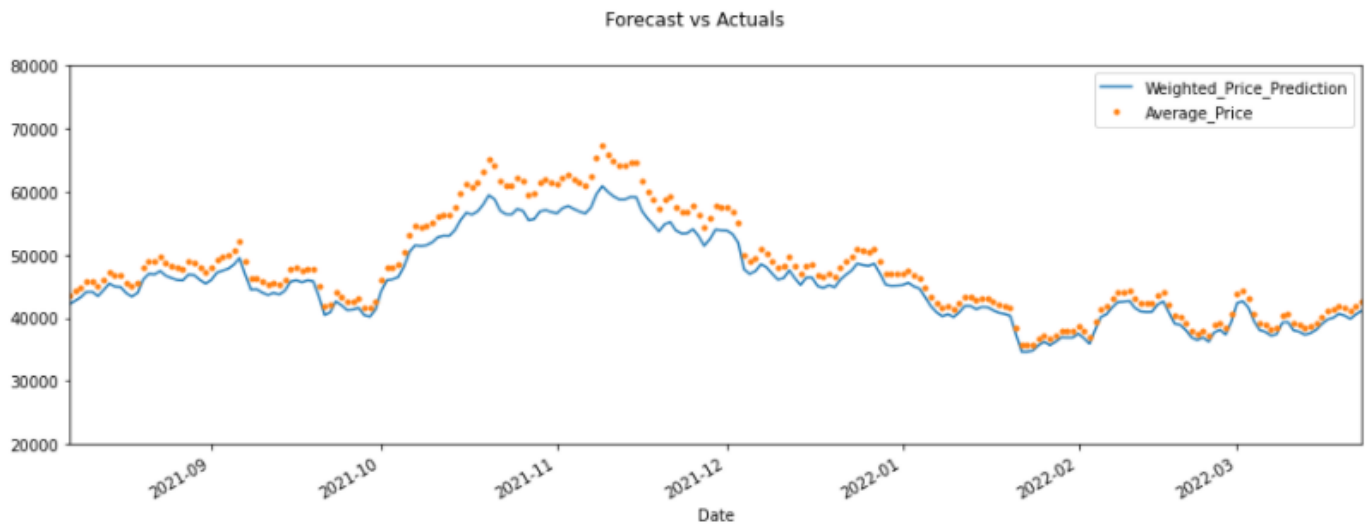
```
_ = data_all[["Average_Price", "Weighted_Price_Prediction"]].plot(figsize=(40, 10))
```





- Average price and forecasted price over a specific interval of time

```
f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)
_ = data_all[["Weighted_Price_Prediction", "Average_Price"]].plot(
    ax=ax, style=["-", "."]
)
ax.set_xbound(lower="08-07-2021", upper="23-03-2022")
ax.set_ylim(20000, 80000)
plot = plt.suptitle("Forecast vs Actuals")
```



❖ Time Series forecasting with XGBoost

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost is a powerful and versatile tool. Let's see, How well does XGBoost perform when used to predict future values of a time-series like Bitcoin prices

In XG Boost we have to make X_train, y_train, X_test, y_test so we have created a function name

```
def create_features(df, label=None):
    """
    Creates time series features from datetime index
    """
    df["date"] = df.index
    df["hour"] = df["date"].dt.hour
    df["dayofweek"] = df["date"].dt.dayofweek
    df["quarter"] = df["date"].dt.quarter
    df["month"] = df["date"].dt.month
    df["year"] = df["date"].dt.year
    df["dayofyear"] = df["date"].dt.dayofyear
    df["dayofmonth"] = df["date"].dt.day
    df["weekofyear"] = df["date"].dt.weekofyear

    X = df[
        [
            "hour",
            "dayofweek",
            "quarter",
            "month",
            "year",
            "dayofyear",
            "dayofmonth",
            "weekofyear",
        ]
    ]
    if label:
        y = df[label]
        return X, y
    return X
```

```
X_train, y_train = create_features(data_train, label="Average_Price")
X_test, y_test = create_features(data_test, label="Average_Price")
```

- Import XGBoost module and fit our dataset in it

```
import xgboost as xgb
from xgboost import plot_importance, plot_tree

model = xgb.XGBRegressor(
    objective="reg:linear",
    min_child_weight=10,
    booster="gbtree",
    colsample_bytree=0.3,
    learning_rate=0.1,
    max_depth=5,
    alpha=10,
    n_estimators=100,
)
model.fit(
    X_train,
    y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],
    early_stopping_rounds=50,
    verbose=False,
)
```

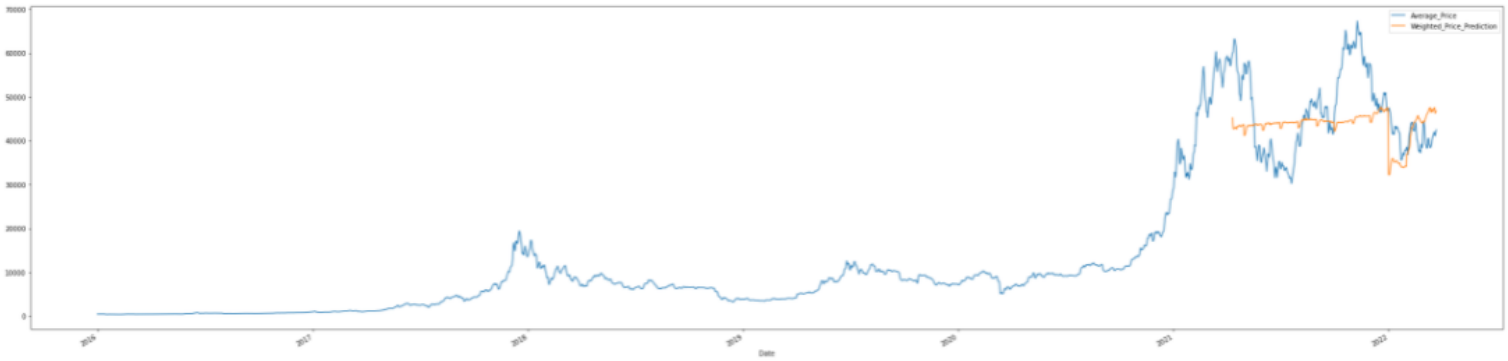
```
XGBRegressor(alpha=10, base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.3, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.1, max_delta_step=0,
             max_depth=5, min_child_weight=10, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=8,
             num_parallel_tree=1, objective='reg:linear', predictor='auto',
             random_state=0, reg_alpha=10, reg_lambda=1, scale_pos_weight=1,
             subsample=1, tree_method='exact', validate_parameters=1,
             verbosity=None)
```

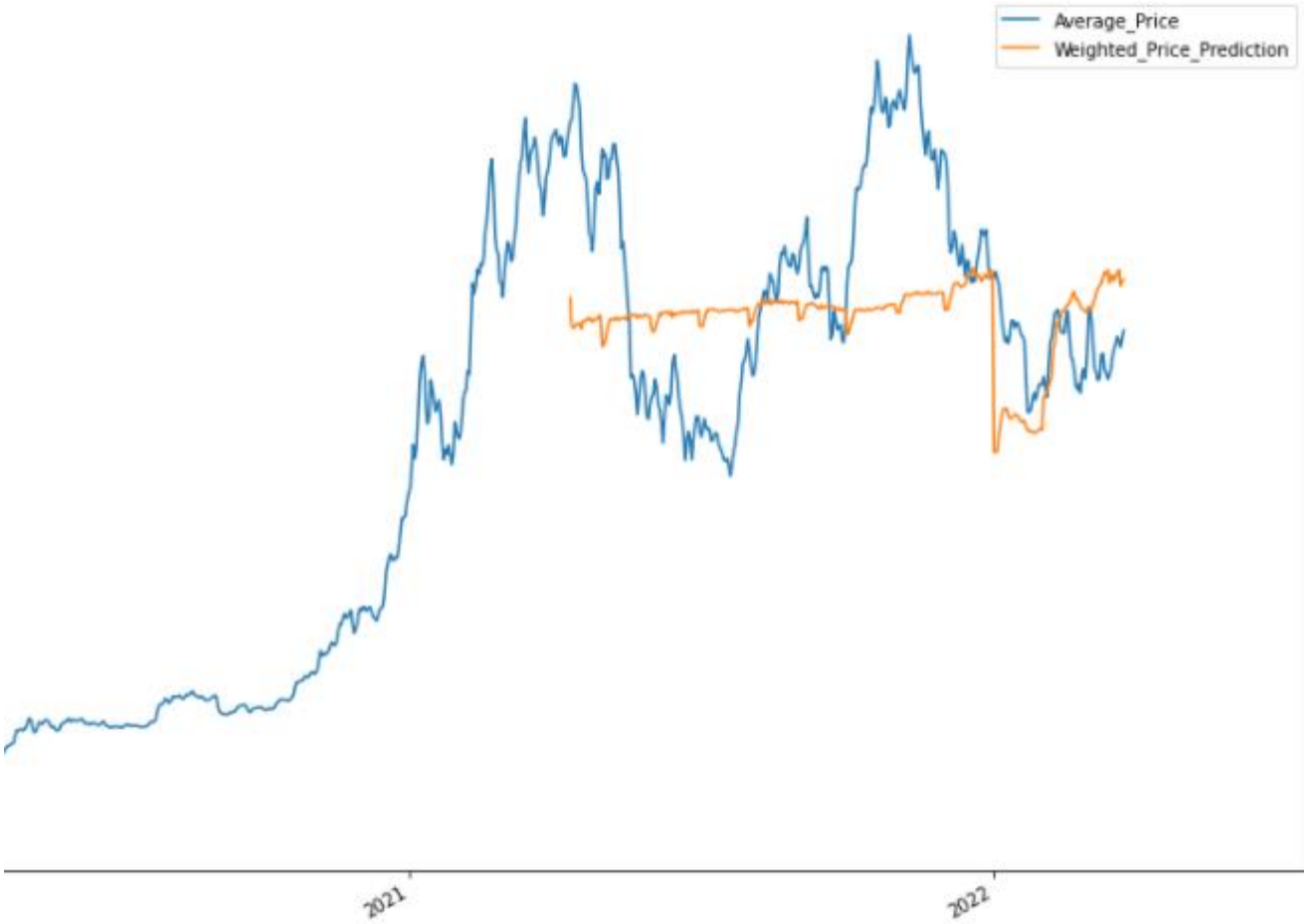
- Let's apply our model and concatenate predicted output with the data set obtained from the LSTM model

```
data_test["Weighted_Price_Prediction"] = model.predict(X_test)
data_all = pd.concat([data_test, data_train], sort=False)
final_data = pd.merge(final_data, data_all, sort=False)
final_data = final_data.rename(columns={"Weighted_Price_Prediction": "xgboost"})
final_data = final_data[["Date", "Average_Price", "lstm", "xgboost"]]
```

- Graph of Average Price and Predicted Price

```
_ = data_all[["Average_Price", "Weighted_Price_Prediction"]].plot(figsize=(40, 10))
```





Conclusions

This study demonstrates that machine learning models can predict the movements of the bitcoin market. The forecasting accuracy of the bitcoin market predictability is somewhat limited. A limited bitcoin market predictability is comparable to findings related to the market predictability of other financial assets, such as stocks. This may be due to multiple reasons, for instance, an immediate market reaction to the utilized features or a potentially large amount of information beyond these features that influence the bitcoin market. Furthermore, our results are consistent with the findings that the bitcoin market has become more efficient over the last few years. Since trading results based on the market predictions are negative after transaction costs, our work does not represent a challenge to bitcoin market efficiency. However, in this study, we analyze the predictability of the bitcoin market movement and do not train our models to maximize trading performance. Nevertheless, our results indicate that empirical trading strategies should be implemented based on models with more extended prediction horizons.

Next, we find that RNN(LSTM) models are particularly well-suited for predicting the short-term bitcoin market. Yet, as the field of machine learning is evolving constantly, we may speculate about future specialized machine learning models performing even better on this task.

Sources and Documentation

Data Processing (NumPy, Pandas, Datetime)

Data Visualization (Plotly, seaborn, Matplotlib)

Machine learning (Keras, sci-kit-learn, XGBoost)

Data Collection (Yahoo Finance)

Source Code

Learning (Coursera)

Other Websites Used for this Article

https://towardsdatascience.com/cryptocurrency-price-prediction-using-deep-learning-70cfca50dd3a

https://medium.com/@nutanbhogendrasharma/simple-sequence-prediction-with-lstm-69ff0f4d57cd

https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

Time Series forecasting with XGBoost by Rob Mulla

https://www.sciencedirect.com/science/article/pii/S2405918821000027

Other Articles to read

Time Series Forecasting with Prophet by Rob Mulla

Bitcoin Price. Prediction by ARIMA by Артём

https://medium.com/@josemarcialportilla/using-python-and-auto-arima-to-forecast-seasonal-time-series-90877adff03c

https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/

https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b

https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/

https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

https://towardsdatascience.com/using-lstms-to-forecast-time-series-4ab688386b1f

https://mlwhiz.com/blog/2017/12/26/How_to_win_a_data_science_competition/

https://www.analyticsvidhya.com/blog/2018/05/generate-accurate-forecasts-facebook-prophet-python-r/