Zealand

BITCOIN PRICE PREDICTION

Analysis of Random Forest, XGB Regressor, Facebook Prophet, and LSTM models utilizing historical data from 2014 and Wikipedia sentiment analysis.

Golden Bitcoin

Name: Kacper Jakub Bytnar

Supervisor: Jens Peter Andersen

School: Zealand - Sjællands Erhvervsakademi

Elective subject: Machine Learning

Count: 21 pages,

3 584 words,

20 787 characters without spacebars,

24 539 characters with spacebars

TABLE OF CONTENTS

INTRODUCTION

Bitcoin, a decentralized digital currency created in 2009, has been gaining popularity worldwide due to its unique features that allow transactions to be made without intermediaries. The blockchain technology that it operates on has been gaining attention from various industries for its potential to disrupt traditional financial systems. With its increasing adoption and use, it has become an important subject of research and analysis, particularly in the area of price prediction. This project aims to provide insights into the factors that influence bitcoin prices and to develop a predictive model that can forecast its future value.

MOTIVATION

As a bitcoin investor and enthusiast, I have been following the digital currency's price movements and researching the factors that affect its value for years. As a result of my research, I have gained significant knowledge and understanding of how bitcoin's price behaves in response to various factors such as market demand and supply, investor sentiment impacted by news or media coverage, adoption and regulations. Using my freshly gained machine learning skills, I hope to gain better insights into bitcoin's price movements. This project presents an exciting opportunity for me to kill two birds with one stone by honing my machine learning skills and potentially becoming a more successful investor. Additionally, this project could potentially provide a valuable tool for other investors and financial institutions to better understand and forecast bitcoin's future value, making it an exciting and worthwhile research area.

PROBLEM DEFINITION

Bitcoin's high level of volatility and complex set of underlying factors that influence its value make it very challenging to accurately predict its future price.
Therefore the main question this project aims to answer is:

Can machine learning techniques be used to accurately predict the future price of bitcoin?

In order to answer the big question, there are few sub-questions that can be explored:

- What are the key factors that impact bitcoin prices?
- How does historical price data, technical indicators, and general public sentiment contribute to predicting Bitcoin prices?
- What machine learning techniques can be used for Bitcoin price prediction, and which of them are the most effective?
- How the accuracy of our predictive model can be measured?
- Can the models be used as indicator for successful trading

METHODOLOGY

In my approach, I will utilize historical price data on a daily interval and gather Bitcoin’s page edits from Wikipedia and perform the sentiment analysis in order to create additional features for my dataset. I will extract various technical indicators, such as moving averages and trading volume, to enhance the feature set. To incorporate the influence of social media sentiment, I will perform sentiment analysis on Wikipedia page edit data and integrate the

sentiment as a feature in my models. Among the various strategies available to identify the target variable for predictions, my decision for this synopsis is to focus on forecasting the next-day closing price.

I will research and gain prediction insights from 4 different models using Random Forest Regression, the LSTM algorithm of RNN, XGB regressor and Prophet to compare the results. The prediction accuracy of these algorithms is evaluated using four metrics: RMSE, R2 Score, MAPE and DA. These metrics provide valuable insights into the models' performance. To ensure a fair comparison, all models will be evaluated using a single dataset that contains consistent values and features.

PLANNING

Week	Plan
18	• Collecting the necessary data for Bitcoin historical prices and bitcoin Wikipedia page • Preparing and cleaning the datasets • Perform exploratory data analysis to identify patterns and trends in the data
19	• Extract technical analysis features from the Bitcoin price data • Perform sentiment analysis on Wikipedia page edits to derive sentiment-related features • Conduct correlation analysis, utilize a Random Forest for feature selection based on importance scores then select the most significant features for further analysis
20	• Split the data into training and testing sets • Start with a baseline model using the Random Forest Regressor • Explore advanced models: XGB Regressor, Prophet and LSTM • Train and evaluate the models using train and test sets • Writing the Process/Findings section of the synopsis
21	• Compare the performance of the models • Evaluate and fine-tune the models by adjusting hyperparameters • Interpretate and visualize the results • Preparing conclusions and discussing the implications of the findings
22	• Finalizing the synopsis, including editing and proofreading • Preparing for the oral exam and practicing the presentation

PROCESS AND FINDINGS

DATA COLLECTION && EDA

I have started my research by looking for a suitable Bitcoin historical prices dataset. After going through and analyzing many datasets available on the internet I decided to work with the Bitcoin USD (BTC-USD) dataset from finance.yahoo.com. I employed the pandas' library to ingest the 'Daily_Bitcoin.csv' file extracting the data into a DataFrame (df). Subsequently, I excluded the final row, established the 'Date' column as the DataFrame's index, and transformed the index into a DateTime format, optimizing it for temporal analysis.

This dataset consists of historical daily price information. It includes following columns: 'Open', 'High', 'Low', 'Close', 'Adj Close', and 'Volume'. Each row represents a specific date, starting from September 17, 2014, and extending up to May 25, 2023.

Date	Open	High	Low	Close	Adj Close	Volume
2014-09-17	465.864014	468.174011	452.421997	457.334015	457.334015	21056800
2014-09-18	456.859985	456.859985	413.104004	424.440002	424.440002	34483200
2014-09-19	424.102997	427.834991	384.532013	394.795990	394.795990	37919700
2014-09-20	394.673004	423.295990	389.882996	408.903992	408.903992	36863600
2014-09-21	408.084991	412.425995	393.181000	398.821014	398.821014	26580100
...	...	...	...	...	...	...
2023-05-21	27118.423828	27265.917969	26706.921875	26753.826172	26753.826172	8647416921
2023-05-22	26749.892578	27045.734375	26549.734375	26851.277344	26851.277344	11056770492
2023-05-23	26855.960938	27434.683594	26816.179688	27225.726563	27225.726563	13697203143
2023-05-24	27224.603516	27224.603516	26106.576172	26334.818359	26334.818359	16299104428
2023-05-25	26329.460938	26591.519531	25890.593750	26476.207031	26476.207031	13851122697

3173 rows × 6 columns

```
corr = df.corr()
print(corr)
```

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999504	0.999099	0.998785	0.998785	0.710567
High	0.999504	1.000000	0.998994	0.999459	0.999459	0.714753
Low	0.999099	0.998994	1.000000	0.999366	0.999366	0.702189
Close	0.998785	0.999459	0.999366	1.000000	1.000000	0.709250
Adj Close	0.998785	0.999459	0.999366	1.000000	1.000000	0.709250
Volume	0.710567	0.714753	0.702189	0.709250	0.709250	1.000000

In fact, most of these columns are redundant for my predictions, therefore I drop Open, High, Low, and Adj Close.

```
df.drop(['Open', 'High', 'Low', 'Adj Close'], axis = 1, inplace = True)
```

Another step was gathering the wikipedia bitcoin's page edits for the same time period as my historical data. In order to achieve that I used mwclient library from python then stored revisions in revs variable.

```
site = mwclient.Site('en.wikipedia.org')
page = site.pages['Bitcoin']
revs = list(page.revisions())
revs[0]
```

Next, I used a pre-trained deep learning "sentiment-analysis" model from Python's transformers library which allowed me to analyze the average sentiment score for each day based on Wikipedia's Bitcoin page revisions. The sentiment score is labeled as avg_sentiment_score and described using values ranging from -1 to 1.

It was also useful to count the page edits for that day labeled as edit_count and calculate the percentage of edits with a negative score that day labeled as negative_percentage

```
!pip install transformers
!pip install xformers
from transformers import pipeline
sentiment_pipeline = pipeline("sentiment-analysis")

def find_sentiment(text):
    sent = sentiment_pipeline([text[:250]])[0]
    score = sent["score"]
    if sent["label"] == "NEGATIVE":
        score *= -1
    return score
```

```
df = df.merge(wiki_df, left_index=True, right_index=True)
df
```

	Close	Volume	edit_count	sentiment	neg_sentiment
2014-09-17	457.334015	21056800	5.033333	-0.232191	0.532718
2014-09-18	424.440002	34483200	5.066667	-0.232760	0.532718
2014-09-19	394.795990	37919700	5.200000	-0.235415	0.549385
2014-09-20	408.903992	36863600	5.200000	-0.233185	0.549385
2014-09-21	398.821014	26580100	5.233333	-0.204017	0.532718
...	...	...	...	...	...
2023-05-21	26753.826172	8647416921	0.100000	-0.033484	0.050000
2023-05-22	26851.277344	11056770492	0.066667	-0.000186	0.016667
2023-05-23	27225.726563	13697203143	0.066667	-0.000186	0.016667
2023-05-24	26334.818359	16299104428	0.133333	-0.033500	0.050000
2023-05-25	26476.207031	13851122697	0.133333	-0.033500	0.050000

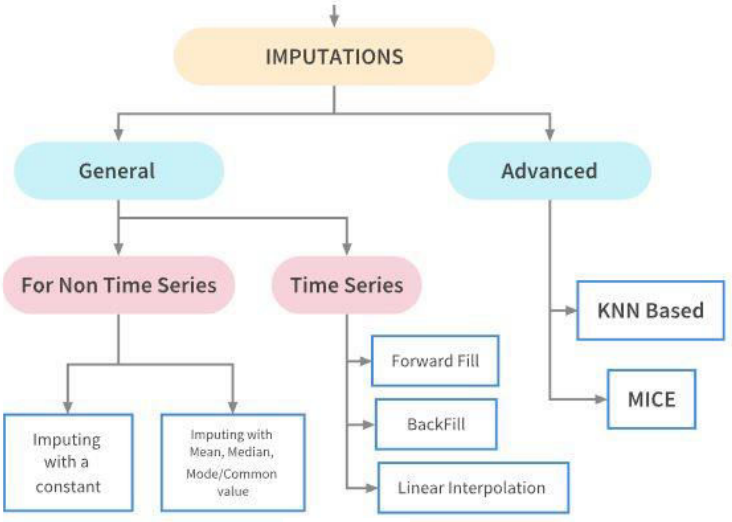
Lastly, all the new features were converted into the data frame and merged with the historical price dataset.

During the exploratory data analysis (EDA) of the historical Bitcoin price dataset using the Plotly library, intriguing patterns and trends emerged. The analysis revealed distinct cycles characterized by significant growth followed by declines, indicating strong market cycles. The graph demonstrated the volatility of Bitcoin through frequent price fluctuations, highlighting the need for risk management.

DATA PREPROCESSING & FEATURE ENGINEERING

HANDLING MISSING VALUES IN TIME-SERIES DATA

The best approach to handle missing value depends on the specific characteristics of the dataset. After researching this topic, I determined that using 'ffill' and 'bfill' proved to be the most suitable for my case. The 'ffill' technique replaces nulls with the most recently observed value, while the 'bfill' fills with the subsequently observed value.



Imputations picture

TARGET FEATURES FOR PREDICTIONS

```
df.fillna(df.mean(), inplace=True)
data = df.fillna(method='bfill')
final_data = data.fillna(method='ffill')
final_data.tail()
```

I have started the feature engineering process by researching different ways to determine the target variable. The choice was between next-day price, future price horizon, or price change. This was a very important decision since the target variable has a significant impact on the prediction process. For example, predicting the exact Bitcoin price requires regression algorithms, while predicting price change direction (up or down) is a classification problem necessitating the use of classification algorithms.

In my research, I have decided to predict the next day's closing price as my target variable, utilizing regression algorithms for the majority of my models. However, to showcase the distinction between regression and classification approaches, I will also employ the XGB Classifier, a classification algorithm.

Consequently, two prediction features have been generated:

```
df["tomorrow"] = df["Close"].shift(-1)
df["target"] = (df["tomorrow"] > df["Close"]).astype(int)
df
```

"tomorrow" - representing the next day's price, and

"target" - binary value of 0 or 1 indicating price movement direction (down and up, respectively).

TECHNICAL ANALYSIS FEATURES

In this research, I have utilized Bitcoin's historical price to extract a set of technical analysis indicators, which hold significant importance as predictor features. Technical analysis encompasses a trading discipline that entails the evaluation of investments and identification of trading opportunities through the analysis of statistical trends derived from trading activity, including price movements and trading volume. The extracted indicators encompass the (a) moving average (b) exponential moving average, (c) moving average convergence divergence, (d) relative strength index, (e) momentum, and (f) Aroon oscillator. These indicators contribute to the comprehensive analysis of the Bitcoin market and provide valuable insights for making informed trading decisions.

(a) **MA:** represents the average price over a specific time period (20 days). It helps smooth out short-term fluctuations and identifies long-term trends.

(b) **EMA:** The 12-day and 26-day exponential moving averages of the closing prices are computed. The EMA assigns more weight to recent prices, making it sensitive to short-term price movements.

(c) **MACD:** The indicator is obtained by taking the difference between the 12-day EMA and the 26-day EMA. It provides insights into the momentum and trend changes in the price series.

```
df['moving_avg'] = df['Close'].rolling(window=20).mean()
df['ema_12'] = df['Close'].ewm(span=12, adjust=False).mean()
df['ema_26'] = df['Close'].ewm(span=26, adjust=False).mean()
df['macd'] = df['ema_12'] - df['ema_26']
df['macd_signal'] = df['macd'].ewm(span=9, adjust=False).mean()
delta = df['Close'].diff()

gain = delta.where(delta > 0, 0)
loss = -delta.where(delta < 0, 0)

avg_gain = gain.rolling(window=14).mean()
avg_loss = loss.rolling(window=14).mean()

rs = avg_gain / avg_loss
df['rsi'] = 100 - (100 / (1 + rs))

aroon_up = 100 * ((25 - (df['Close'].rolling(25).apply(lambda x: x.argmax() + 1))) / 25)

aroon_down = 100 * ((25 - (df['Close'].rolling(25).apply(lambda x: x.argmin() + 1))) / 25)

df['aroon_oscillator'] = aroon_up - aroon_down
```

(d) **Momentum:** measures speed of price movement on a specific time period. It helps investors assess the strength of a trend.

$MOM_\zeta = C_p^{(i-(\zeta-1))} - C_p^{(i)}$, where ζ is the number of days.

(e) **RSI:** calculated using the average gain and average loss over a 14-day period, measures the magnitude of recent price changes to determine overbought or oversold conditions.

$$RSI = 100 - \frac{100}{1 + RS}$$, where $$RS = \frac{Avg.Gain}{Avg.Loss}$$

(f) **Aroon Oscillator:** represents the strength and direction of the current price trend. Calculated by following formula:

Aroon Up = 100*(x – days since x days high)/x
Aroon Down = 100*(x – days since x days low)/x
Aroon Oscillator = Aroon Up – Aroon Down

By incorporating these technical analysis factors into the prediction model, we can capture various aspects of market behavior, such as trends, momentum, and overbought/oversold conditions. These factors provide valuable insights for forecasting Bitcoin price movements and enhancing the accuracy of our predictions.

SENTIMENT ANALYSIS FEATURES

Furthermore, leveraging the sentiment analysis data, I have derived three additional columns as meaningful predictor features:

close_ratio_{horizon} represents the relative position of the current closing price compared to the historical average over a specific horizon, providing insight into potential trends or deviations.

edit_{horizon} captures the average number of edits or modifications made over a specific horizon.

trend{horizon}) represents the average value or trend of the target variable over a specific horizon. It helps identify the overall direction or movement of the target variable over time.

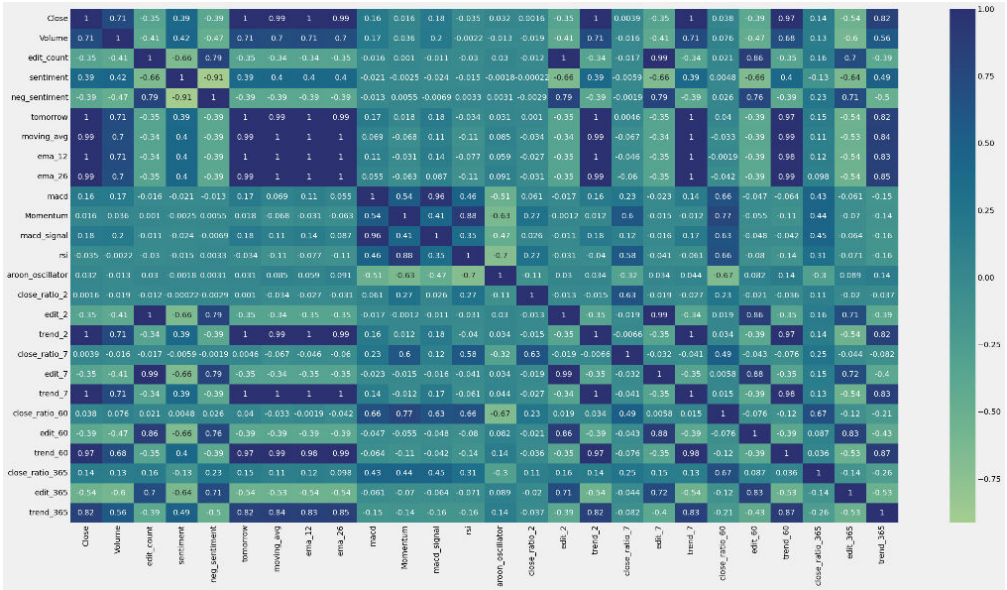
```
num_days = 14

if len(df) >= num_days:
    df['Momentum'] = (df['Close'] - df['Close'].shift(num_days)) / df['Close'].shift(num_days)

horizons = [2,7,60,365]

for horizon in horizons:
        rolling_averages = df.rolling(horizon, min_periods=1).mean()

        ratio_column = f"close_ratio_{horizon}"
        df[ratio_column] = df["Close"] / rolling_averages["Close"]

        edit_column = f"edit_{horizon}"
        df[edit_column] = rolling_averages["edit_count"]

        rolling = df.rolling(horizon, closed='left', min_periods=1).mean()
        trend_column = f"trend_{horizon}"
        df[trend_column] = rolling["tomorrow"]
```

FEATURE SELECTION

At this point I had 26 features and decided to select only the most relevant ones, to reduce noise, overfitting, dimensionality, computational complexity and most importantly enhance the model's ability to make accurate predictions. I have achieved this in 2 steps:

CORRELATION ANALYSIS by computing the correlation coefficients between each pair of features and the target variable and displaying the correlation matrix.

The correlation analysis of the closing price reveals that features such as "trend_2", "tomorrow", "trend_7", and "ema_12" exhibit strong positive correlations, indicating a significant influence on the closing price. Conversely "edit_count", "edit_2" and "edit_7" demonstrate negative correlations, suggesting a potential impact on price decreases.

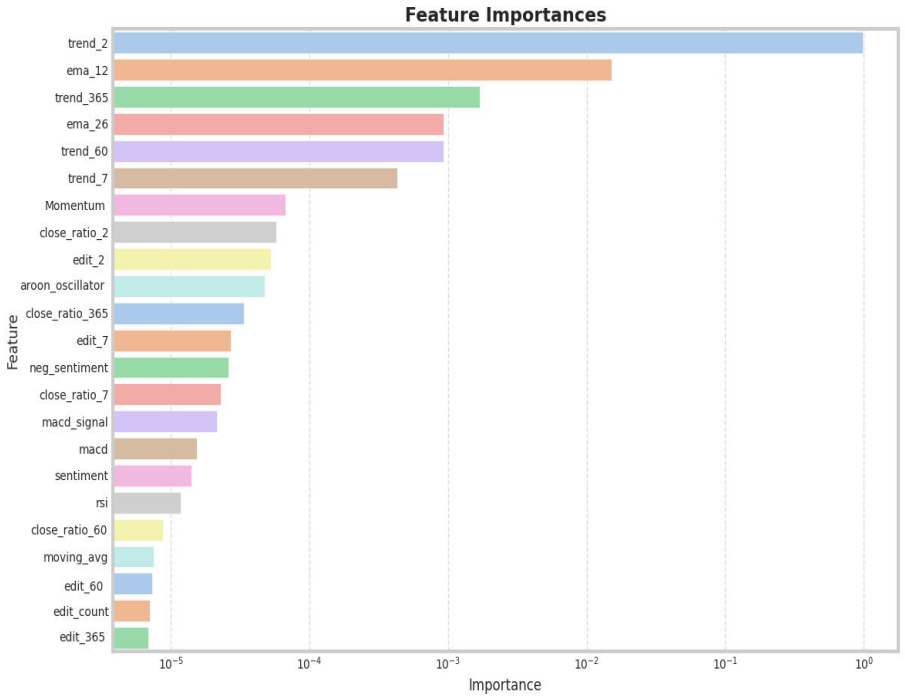


FEATURE SELECTION WITH RANDOM FOREST - a Random Forest regressor was employed for feature selection. The algorithm assigns importance scores to each feature, enabling the ranking and selection of the most significant ones. This approach allows for the identification of key predictors that contribute significantly to the model's predictive performance.

After conducting an in-depth feature analysis, it was determined that the features:

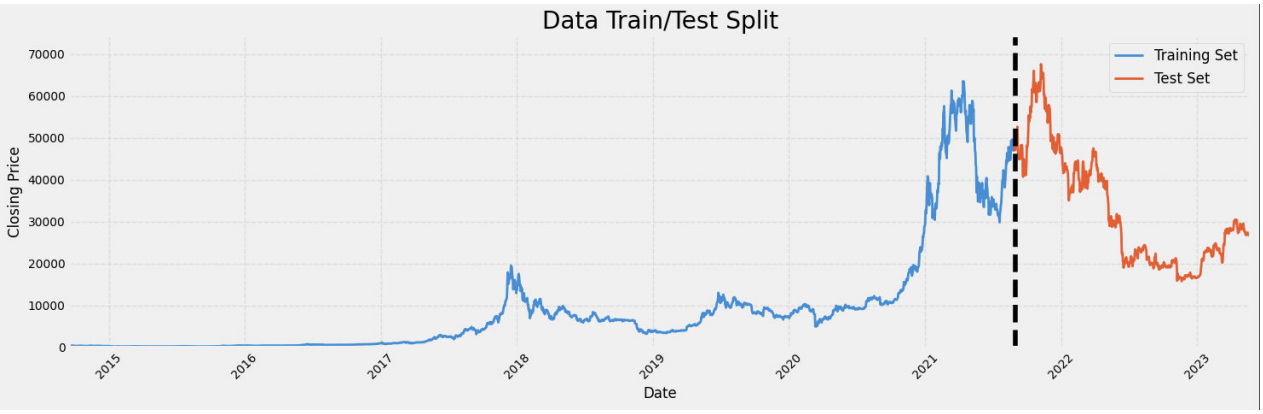
- "edit_365"
- "edit_60"
- "close_ratio_7"
- "close_ratio_2"
- "close_ratio_60"

exhibited limited relevance and were subsequently excluded from further analysis.



TRAINING METHODOLOGY

I will opt for a ***hold-out based validation***, also known as train-test split validation. This widely employed technique allows for the assessment of the model's performance in a reliable manner.



The training set serves as the foundation for model training, wherein the input features and their corresponding target values are provided. Through exposure to the training data, the model acquires an understanding of the underlying patterns and relationships, iteratively adjusting its internal parameters to minimize prediction errors. This iterative process involves careful consideration of the model's architecture and hyperparameters, ensuring optimal performance is achieved.

Once trained, the models are evaluated using the testing set, which contains unseen data. Predictions are compared against actual target values to assess the models' generalization abilities. Hold-out based validation ensures unbiased evaluation and reliable performance estimation on new, unseen data, which is crucial in evaluating its effectiveness in real-world scenarios.

```
df_train = final_data[final_data.index < "2021-09"]
df_test = final_data[final_data.index >= "2021-09"]

print('train shape :', df_train.shape)
print('validation shape :', df_test.shape)
```

```
train shape : (2541, 26)
validation shape : (632, 26)
```

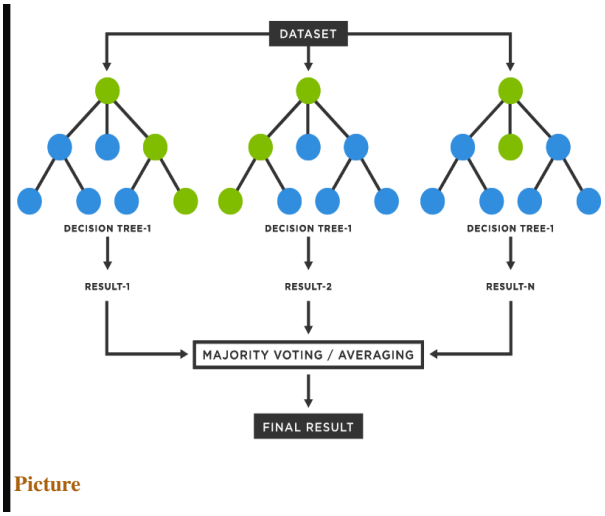
For this study, the data from 2014-09-17 to 2021-09 is used for training, while the data from 2021-09 onwards serves as the hold-out set.

MODELS SELECTION & TRAINING

BASELINE MODEL - RANDOM FOREST REGRESSOR

The Random Forest Regressor was chosen as the baseline model due to its ensemble learning approach, which combines multiple decision trees to enhance generalization and mitigate overfitting. It accommodates various data types without extensive preprocessing, offers feature importance estimation, and exhibits robustness to outliers and noisy data. Using Random Forest as a benchmark allows me effective performance evaluation of more advanced algorithms.

```
from sklearn.ensemble import RandomForestRegressor
df_Random_Forest = df.copy()


model2 = RandomForestRegressor(n_estimators=1000, min_samples_split=50, random_state=1)

train = df_Random_Forest.iloc[:size]
test = df_Random_Forest.iloc[size:]


predictors = ["moving_avg", "ema_12", "ema_26", "macd", "macd_signal",  "rsi",  "aroon_oscillator", "Volume","sentiment", "Close"]
model2.fit(train[predictors], train["tomorrow"])
```



Picture

Prediction results on training set

```
y_pred_training = model2.predict(train[predictors])
evaluate_model(train["tomorrow"], y_pred_training)
```

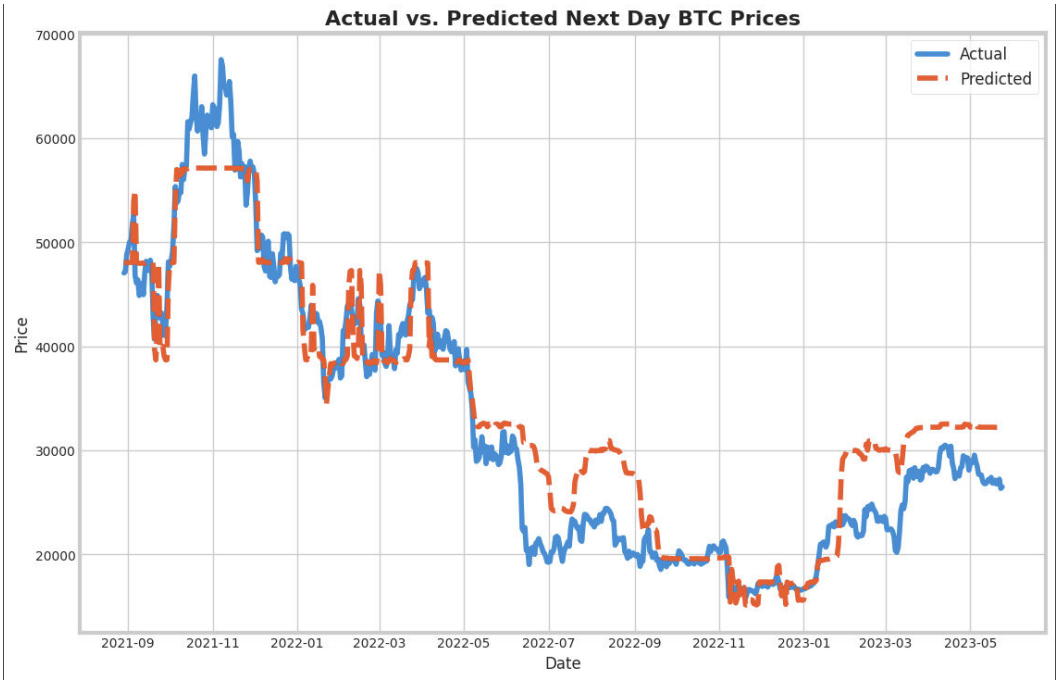
```
RMSE: 775.2919
R2 Score: 0.9963
MAPE: 2.72%
Directional Accuracy: 50.47%
```

Prediction results on test set

```
y_pred = model2.predict(test[predictors])
evaluate_model(test["tomorrow"], y_pred)
```

```
RMSE: 4121.3735
R2 Score: 0.9064
MAPE: 11.75%
Directional Accuracy: 66.14%
```

Test set predictions on the plot



XGB REGRESSOR

XGBoost Regressor is an algorithm that effectively captures complex relationships and non-linear patterns in the data, which is crucial for understanding Bitcoin's dynamic behavior. This model uses boosting ensemble technique that allows it to progressively improve prediction accuracy by learning from past errors. Additionally, it incorporates techniques like regularization and tree pruning to prevent overfitting and enhance model generalization. The availability of customizable hyperparameters enables me to fine-tune the model for optimal performance. All of these make it a well-suited choice for accurately predicting Bitcoin prices.

HYPER PARAMETER TUNING

I utilized RandomizedSearchCV, a method for hyperparameter tuning, to optimize the performance of the XGB Regressor model. By defining a grid of hyperparameters and performing a randomized search, I have tested various combinations to find the best one. The search involved key hyperparameters and utilized a 5-fold cross-validation strategy to evaluate multiple iterations.

Hyperparameter tuning described above allowed me to come up with the most accurate XGB Regressor setting I could do with my data.

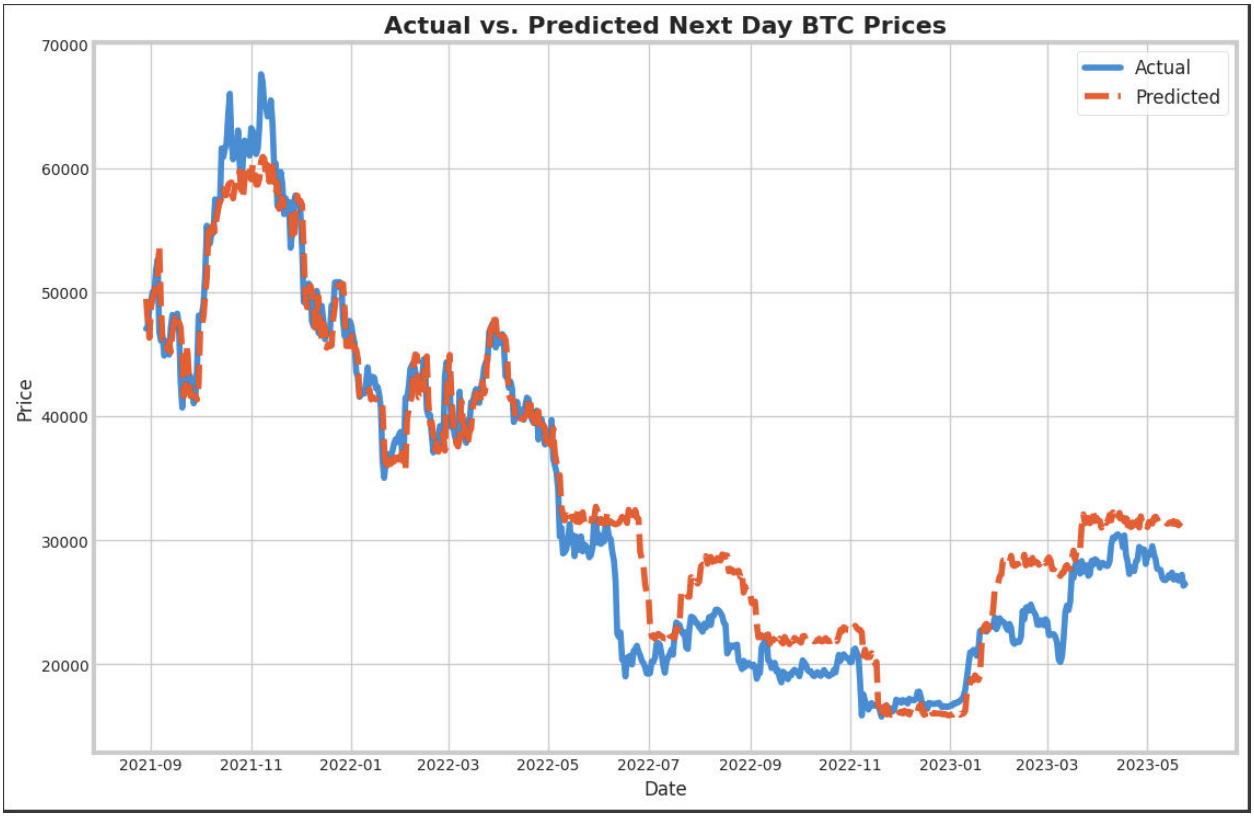
```
[949]   validation_0-rmse:423.66743     validation_1-rmse:3465.03362
```

```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=50,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.01, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=1000, n_jobs=None, num_parallel_tree=None,
             objective='reg:linear', predictor=None, ...)
```

```
## Hyper Parameter Optimization Grid

params={
 "learning_rate"    : [0.01, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
 "max_depth"        : [1, 3, 4, 5, 6, 7],
 "n_estimators"     : [int(x) for x in np.linspace(start=500, stop=2000, num=10)],
 "min_child_weight" : [int(x) for x in np.arange(3, 15, 1)],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],
 "subsample"        : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],
 "colsample_bytree" : [0.5, 0.6, 0.7, 0.8, 0.9, 1],
 "colsample_bylevel": [0.5, 0.6, 0.7, 0.8, 0.9, 1],

}

model  = RandomizedSearchCV(
                reg,
                param_distributions=params,
                n_iter=10,
                n_jobs=-1,
                cv=5,
                verbose=3,
                error_score='raise'
)

model.fit(x_train, y_train,eval_set=[(x_test, y_test)])

print(f"Model Best Score : {model.best_score_}")
print(f"Model Best Parameters : {model.best_estimator_.get_params()}")
```

```
reg = xgb.XGBRegressor(base_score=0.5, booster='gbtree',
                       n_estimators=1000,
                       early_stopping_rounds=50,
                       objective='reg:linear',
                       max_depth=3,learning_rate=0.01)
reg.fit(x_train, y_train,
        eval_set=[(x_train, y_train), (x_test, y_test)],
        verbose=100)
```

PREDICTIONS ON UNSEEN DATA



Training period predictions results:

```
y_pred_train = reg.predict(x_train)

evaluate_model(y_train, y_pred_train)
```

```
RMSE: 435.8183
R2 Score: 0.9989
MAPE: 5.85%
Directional Accuracy: 58.60%
```

Test period predictions results:

```
y_pred = reg.predict(x_test)  #

evaluate_model(y_test, y_pred)
```

```
RMSE: 3455.9054
R2 Score: 0.9342
MAPE: 10.29%
Directional Accuracy: 63.78%
```

FACEBOOK PROPHET

Developed by Facebook in 2017, Prophet was specifically designed to excel in forecasting business data, offering a promising alternative to traditional models like ARIMA. Its ability to capture the intricate dynamics of time series data, especially in the presence of strong seasonal patterns, makes it another great choice for analyzing Bitcoin prices.

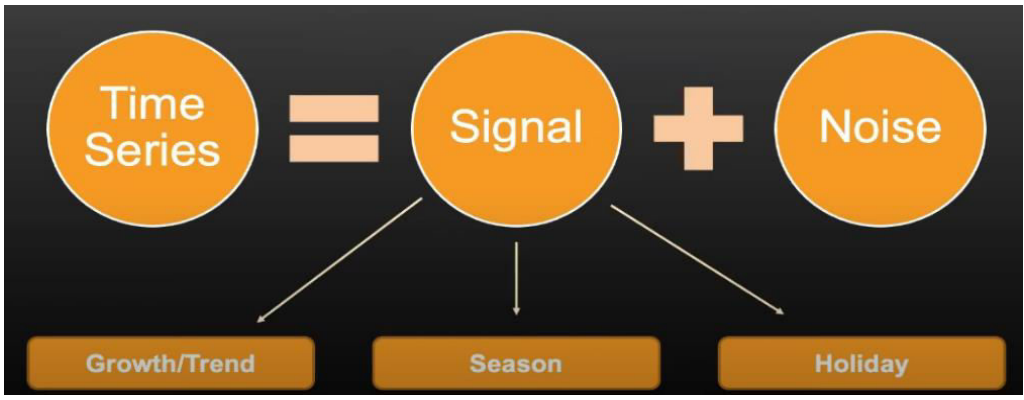
Prophet's inherent flexibility allows it to accurately model these complex seasonal patterns, providing valuable insights into the underlying dynamics of the market. The model's decomposable time series framework, consisting of trend, seasonality, and holiday components, further enhances its effectiveness. They are integrated in the following equation: $y(t) = g(t) + s(t) + h(t) + \varepsilon t$, where

g(t): piecewise linear or logistic growth curve that captures non-periodic changes

s(t): periodic changes

h(t): effects of holidays (user provided)

εt: denotes the error term, accounting for any unusual changes



MODEL TRAINING & PREDICTIONS

I have implemented a new DataFrame and divided it into train and test sets to comply with Prophet's specific column naming convention requirements ('ds' and 'y').

```
[77] # Create a new DataFrame with 'ds' and 'y' columns
     prophet_df = pd.DataFrame({'ds': final_data.index, 'y': final_data['tomorrow']})
     # Display the new DataFrame
     print(prophet_df)

                        ds           y
     2014-09-17 2014-09-17   424.440002
     2014-09-18 2014-09-18   394.795990
     2014-09-19 2014-09-19   408.903992
     2014-09-20 2014-09-20   398.821014
     2014-09-21 2014-09-21   402.152008
```

```
split_date = "2021-08-31"

train_filt = prophet_df['ds'] <= split_date
test_filt = prophet_df['ds'] > split_date

df_train_prophet = prophet_df[train_filt]
df_test_prophet = prophet_df[test_filt]
```

To enhance the performance of the Facebook Prophet model, have updated with additional features used in previous models. In a loop, each feature was iterated over and added as a regressor to the Prophet model.

```
# Create the baseline prophet model with confidence internal of 95%
model_fbp = Prophet(interval_width=0.85,n_changepoints=7)
exogenous_features = list(x_train.columns)


for feature in exogenous_features:
  model_fbp.add_regressor(feature)
  df_train_prophet = pd.concat([df_train_prophet, x_train[feature]], axis=1
  df_test_prophet = pd.concat([df_test_prophet, x_test[feature]], axis=1)

model_fbp.fit(df_train_prophet)
```

CROSS VALIDATION & EVALUATION

To assess the model's performance, I utilized a cross-validation function that compares the actual and predicted values. Considering the dataset's duration of 632 days, I allocated

```
# Cross validation
df_cv = cross_validation(model_fbp, initial='500 days', period='60 days', horizon = '30 days', parallel="processes")
df_cv.head()
```

500 days for initial model training. Subsequently, I evaluated the model's performance over a 60-day period.

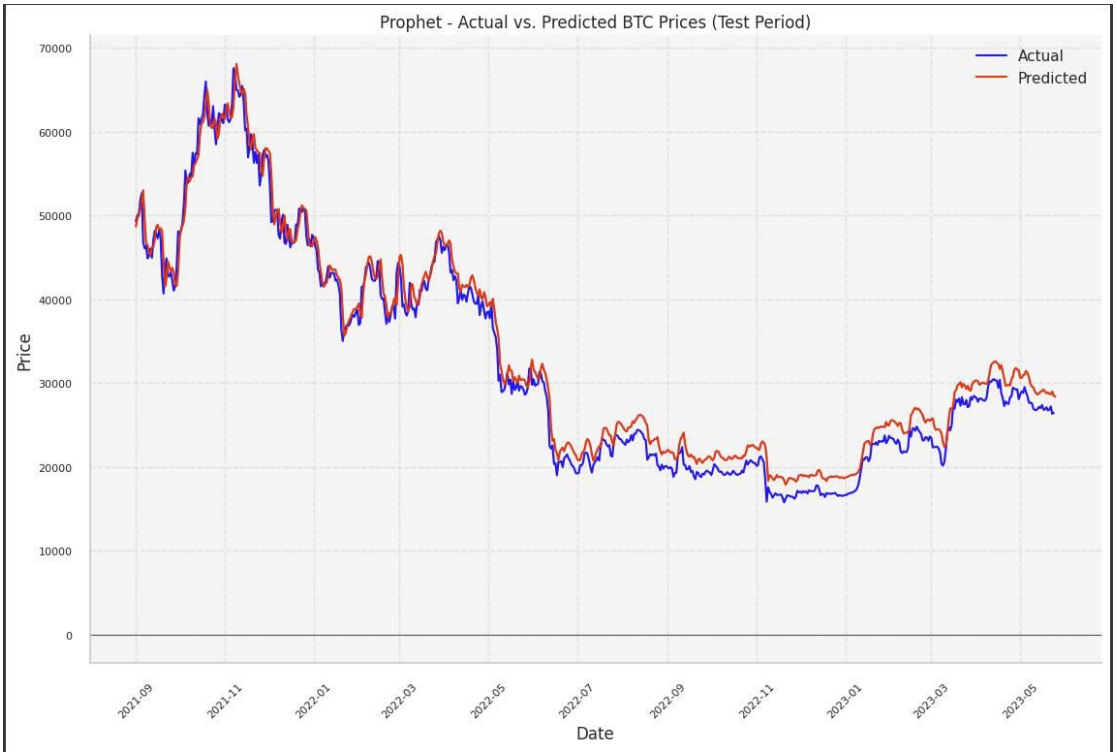
PROPHET'S PREDICTION RESULTS ON TEST DATA

Training set:

```
RMSE: 676.1943751924362
R2: 0.9972344070375135
MAPE: 13.197764158177968
Directional Accuracy: 0.4990153603781016
```

Testing set:

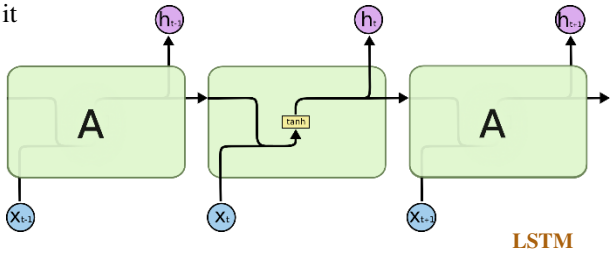
```
RMSE: 2034.7793814919721
R2: 0.9771427392264778
MAPE: 6.571844838004
Directional Accuracy: 0.488888
```



Model's evaluation metrics and the chart analysis makes it clear that the predictions made by Prophet demonstrate a high level of accuracy. Prophet has proven to be the most accurate and best-performing model on the provided dataset and an exceptionally powerful machine learning model that excels in forecasting Bitcoin price movements with remarkable accuracy.

LSTM (LONG SHORT-TERM MEMORY) NEURAL NETWORK

LSTM model excels in ability to capture long-term dependencies, model intricate non-linear relationships, and handle variable-length sequences. These factors make the LSTM model exceptionally well-suited for accurately capturing the complex patterns and fluctuations in my Bitcoin data. Hence it seems to offers promising potential in delivering reliable and accurate predictions for Bitcoin price movements.



LSTM

I have implemented the LSTM model using Keras for capturing temporal dependencies in sequential data. The architecture consists of multiple layers:

The first LSTM layer is the main input layer with 150 units (memory cells) and returns the full sequence output.

The second LSTM layer has 100 units and returns the full sequence output.

The third LSTM layer has 50 units and produces the final sequence output.

Dropout layers with a 0.2 rate are added after each LSTM layer to prevent overfitting.

```
lstm_input = Input(shape=(backcandles, 8), name='lstm_input')
inputs = LSTM(150, name='first_layer')(lstm_input)
inputs = Dense(1, name='dense_layer')(inputs)
output = Activation('linear', name='output')(inputs)
model = Model(inputs=lstm_input, outputs=output)
adam = optimizers.Adam()
model.compile(optimizer=adam, loss='mse')
model.fit(x=data_X_train, y=data_y_train, batch_size=15, epochs=30, shuffle=True, validation_split = 0.1)
```

A single-unit dense layer is added for the final output and Linear activation is used.

Model was complied with Adam optimizer, MSE loss function is employed for regression tasks.

```
[2] model.summary()

Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_input (InputLayer)     [(None, 8, 22)]           0

 lstm_26 (LSTM)              (None, 8, 150)            103800

 dropout_22 (Dropout)        (None, 8, 150)            0

 lstm_27 (LSTM)              (None, 8, 100)            100400

 dropout_23 (Dropout)        (None, 8, 100)            0

 lstm_28 (LSTM)              (None, 50)                30200

 dropout_24 (Dropout)        (None, 50)                0

 dense_7 (Dense)             (None, 1)                 51

 activation_3 (Activation)   (None, 1)                 0

=================================================================
Total params: 234,451
Trainable params: 234,451
Non-trainable params: 0
_________________________________________________________________
```

Scaling the data was necessary to ensure stable and efficient training since LSTM models are sensitive to the scale of input features, and large differences in feature magnitudes can lead to difficulties in convergence during training. Hence I have used MinMaxScaler() to prepare my data for training and then inverse it afterward to evaluate it fairly.

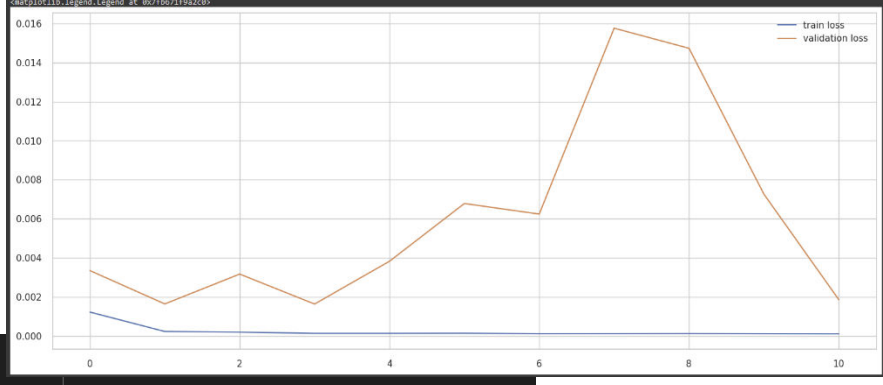
```
from sklearn.preprocessing import MinMaxScaler

x_train_scaler = MinMaxScaler(feature_range=(0,1))
y_train_scaler = MinMaxScaler(feature_range=(0,1))
x_test_scaler = MinMaxScaler(feature_range=(0,1))
y_test_scaler = MinMaxScaler(feature_range=(0,1))


x_train_scaled = x_train_scaler.fit_transform(x_train)
y_train_scaled = y_train_scaler.fit_transform(y_train.values.reshape(-1,1))
x_test_scaled = x_test_scaler.fit_transform(x_test)
y_test_scaled = y_test_scaler.fit_transform(y_test.values.reshape(-1,1))
```

```
test_predict_inv = y_test_scaler.inverse_transform(y_pred2)
train_predict_inv = y_train_scaler.inverse_transform(y_pred)
```

I encountered difficulties with overfitting in my model, where the training loss continued to decrease while the validation loss fluctuated. Early stopping in the model training process helped me to overcome the challenge of overfitting what can be observed in the loss plot. By monitoring the validation loss and stopping the training if it did not improve after a certain number of epochs.

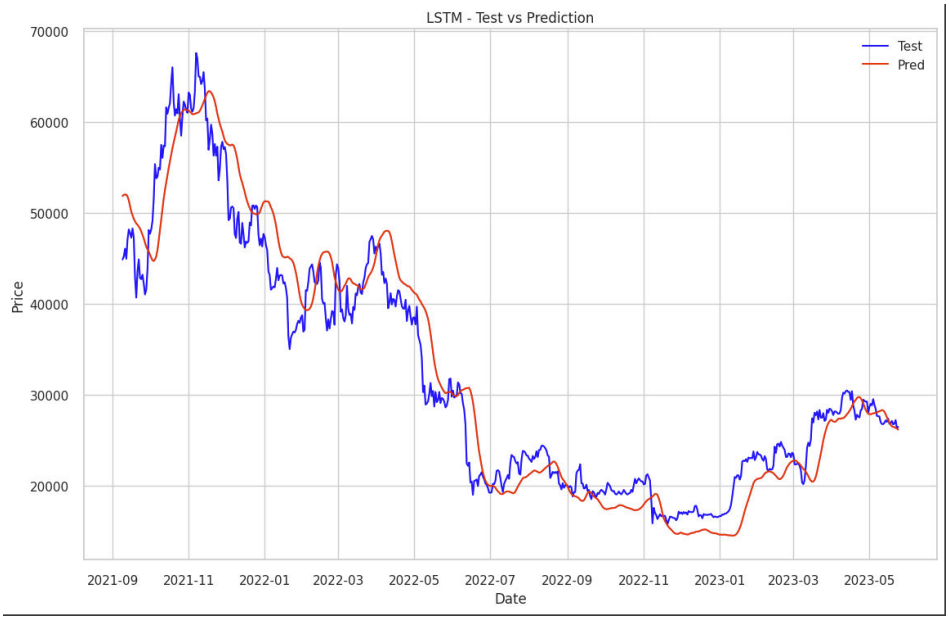
```
plt.figure(figsize=(16,7))
plt.plot(history.history["loss"], label= "train loss")
plt.plot(history.history["val_loss"], label= "validation loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb671f9a2c0>
```




```
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=7, restore_best_weights=True)

history = model.fit(x_train_shaped, y_train_scaled, batch_size=15, epochs=30, validation_split=0.1, callbacks=[early_stopping])
```

EVALUATION OF LSTM PREDICTION ON UNSEEN DATA



Train set:

```
evaluate_model(y_test_reshaped[8:-1], test_predict_inv[:-1])
```

```
RMSE: 3718.2441
R2 Score: 0.9229
MAPE: 9.43%
Directional Accuracy: 43.98%
```

Test set:

```
evaluate_model(y_train[8:-1], train_predict_inv[:-1])
```

```
RMSE: 1921.6981
R2 Score: 0.9776
MAPE: 28.51%
Directional Accuracy: 63.90%
```

ALL MODELS EVALUATION TABLE

For clear comparison and analysis of all the models, I created a table in Python that provides an overview of their performance metrics

Training Evaluation Metrics:

Model	RMSE	R2 Score	MAPE	Directional Accuracy
Random Forest	775.29	0.9963	2.72%	50.47%
XGB Regressor	435.81	0.9989	5.85%	58.60%
Prophet	676.19	0.9972	13.19%	49.90%
LSTM	1921.69	0.9776	28.51%	63.90%

Testing Evaluation Metrics:

Model	RMSE	R2 Score	MAPE	Directional Accuracy
Random Forest	4121.37	0.9064	11.75%	66.14%
XGB Regressor	3455.9	0.9342	10.29%	63.78%
Prophet	2034.78	0.9771	6.57%	48.88%
LSTM	3718.24	0.9229	9.43%	43.98%

CONCLUSION

WHAT ARE THE KEY FACTORS THAT IMPACT BITCOIN PRICES?

The key factors influencing Bitcoin prices include supply and demand dynamics, market adoption and acceptance, regulatory developments, macroeconomic factors, investor sentiment, technological advancements, and external events. These factors collectively contribute to the volatility and complexity of Bitcoin price movements.

WHICH OF THESE FACTORS WOULD BE THE MOST SUFFICIENT TO USE FOR MACHINE LEARNING MODELS?

Historical price data is closely tied to supply and demand dynamics and helps us analyze past price patterns and trends. Technical indicators, influenced by factors like market adoption and acceptance can provide additional insights into market trends and potential price reversals. Lastly, sentiment analysis captures general public sentiment that includes external events which often shape public sentiment impacting Bitcoin prices. Additionally, sentiment analysis can also reflect market adoption and acceptance as positive or negative sentiment towards Bitcoin can influence its price.

WHAT MACHINE LEARNING TECHNIQUES CAN BE USED FOR BITCOIN PRICE PREDICTION, AND WHICH OF THEM ARE THE MOST EFFECTIVE?

There are many techniques to predict bitcoin price, as well as many ways to establish target feature like next-day price or price change direction. In my research, I was predicting the next day's closing price utilizing regression algorithms such as Random Forest Regressor, XGB Regressor, Facebook Prophet and LSTM. All of these turned out to be powerful tools for price prediction, however prophet performed the best for me with significantly lower

RMSE value than other models. Each model's performance depends on the training data and features so it is possible to achieve even better results with other algorithms by different training approaches.

HOW THE ACCURACY OF OUR PREDICTIVE MODEL CAN BE MEASURED?

In the case of regression models for Bitcoin price prediction, important evaluation metrics include RMSE (average deviation between the predicted values and the actual values), R2 score (t represents the proportion of the variance in the target variable), MAPE (average percentage difference between the predicted values and the actual values), and directional accuracy (percentage of correct predictions in terms of the direction of price movements).

CAN MACHINE LEARNING TECHNIQUES BE USED TO ACCURATELY PREDICT THE FUTURE PRICE OF BITCOIN?

Based on the research findings - yes, machine learning techniques such as Facebook Prophet or LSTM models, can be used to predict the future price of Bitcoin with a certain level of accuracy. By utilizing historical price data, market indicators, and other relevant features, machine learning models can analyze patterns and trends to make predictions about Bitcoin's future price movements. The evaluation of these models, proves their ability to provide reasonably accurate predictions.

Through this research, I have gained a deep understanding of how machine learning works and have honed my skills to solve a wide range of machine learning problems. This means that I now have the knowledge and expertise to apply these techniques in real-world scenarios, such as predicting stock prices or understanding customer behavior.

REFLECTION

Now that I have completed the research and gained a deeper understanding of the subject, I can take a step back and ask myself some important questions and consider possible areas for improvement.

BETTER QUESTIONS: Considering the insights gained, I might be able to identify additional or more refined questions that would further enhance the depth of the research. However I am happy with the initial problem definition and my results.

METHOD SELECTION: I believe that the models chosen for the research were indeed among the best options for accurate price prediction using regression techniques. However, I have now come to realize that there are other approaches, such as classification-based predictions, that may provide even better results for real-world trading predictions.

PLANNING IMPROVEMENT: Upon reflection, I am pleased to note that the research plan was comprehensive and efficient, and I successfully achieved all the intended goals as initially outlined. This reinforces my confidence in my planning abilities and affirms that with proper organization and foresight, I can effectively execute future research projects.

BIBLIOGRAPHY

Bytnar, K. (2023). Kacper Bytnar Synopsis - Bitcoin Price Prediction.

https://github.com/KacperBytnar/Bitcoin_Price_Prediction/tree/main

GitHub repository that contains the entire code for this Bitcoin Price Research

Yahoo Finance. (n.d.). Bitcoin Historical Data. Retrieved from Yahoo Finance: https://finance.yahoo.com/quote/BTC-USD/history?period1=1410912000&period2=1684972800&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true

The dataset from Yahoo Finance was used to gather historical Bitcoin price data for the analysis.

Aurélien Géron Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd

edition, O'Reilly 2019

Used for libraries like Scikit-Learn and Keras.

Shutterstock. (n.d.). Face of Crypto Currency Golden Bitcoin Isolated. Retrieved from Shutterstock https://www.shutterstock.com/pl/image-photo/face-crypto-currency-golden-bitcoin-isolated-744111121

The Bitcoin image used on the title page of the project was sourced from Shutterstock.

XGBoost Documentation. (n.d.). XGBoost Parameters. Retrieved from XGBoost Documentation:

https://xgboost.readthedocs.io/en/stable/parameter.html

The XGBoost parameter documentation was referenced for understanding and implementing XGBoost in the project.

Facebook Prophet Documentation. (n.d.). Quick Start with Prophet. Retrieved from Facebook Prophet Documentation:

The Prophet quick start guide provided by Facebook was utilized for implementing the Prophet model in the project.

Colah's Blog. (n.d.). Understanding LSTM Networks. Retrieved from Colah's Blog: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-chain.png - LSTM Picture

LSTM Picture and understanding LSTM networks was referred to for gaining insights into LSTM and its application in the project.

TIBCO. (n.d.). What is a Random Forest? Retrieved from TIBCO:

https://www.tibco.com/reference-center/what-is-a-random-forest

https://www.tibco.com/sites/tibco/files/media_entity/2021-05/random-forest-diagram.svg - Picture

The TIBCO reference provided information about Random Forests and picture which was utilized in the project.

Kaggle. (n.d.). A Guide to Handling Missing Values in Python. Retrieved from Kaggle:

https://www.kaggle.com/code/parulpandey/a-guide-to-handling-missing-values-in-python

Used for Imputations picture and handling nulls

https://medium.com/analytics-vidhya/forecasting-using-facebooks-prophet-library-ce628e76586b

Used for Prophet logo and model implementation examples