Bitcoin price prediction

PHẠM VĂN VIỆT

Content

1.Understanding data

- Open: is the opening price of Bitcoin on that day.
- High: is the highest price that Bitcoin reached that day.
- Low: is the lowest price that Bitcoin reached that day.
- Close: is the closing price of Bitcoin on that day.
- Volume: is the trading volume of Bitcoin that day.

```
dataa.head()
```

	Date	Open	High	Low	Close	Volume
0	2016-01-01T00:00:00.000Z	430.721008	436.246002	427.515015	434.334015	36278900.0
1	2016-01-02T00:00:00.000Z	434.622009	436.062012	431.869995	433.437988	30096600.0
2	2016-01-03T00:00:00.000Z	433.578003	433.743011	424.705994	430.010986	39633800.0
3	2016-01-04T00:00:00.000Z	430.061005	434.516998	429.084015	433.091003	38477500.0
4	2016-01-05T00:00:00.000Z	433.069000	434.182007	429.675995	431.959991	34522600.0

```
dataa.tail()
```

	Date	Open	High	Low	Close	Volume
2984	2024-03-03T00:00:00.000Z	62031.578523	63230.209563	61435.023421	63167.370358	2.625381e+10
2985	2024-03-04T00:00:00.000Z	63137.004682	68537.029333	62386.518353	68330.415608	7.067047e+10
2986	2024-03-05T00:00:00.000Z	68341.057782	69170.628206	59323.908942	63801.197561	1.028029e+11
2987	2024-03-06T00:00:00.000Z	63776.051426	67637.929859	62848.671519	66106.802787	6.875023e+10
2988	2024-03-07T00:00:00.000Z	66099.741652	68029.918692	65655.534200	66925.483202	4.698954e+10

2.Loading and Data Processing

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
import warnings
warnings.filterwarnings("ignore")
```

```
dataa = pd.read_excel('Dataa_bitcoin.xlsx')
```

```
dataa['Date'] = pd.to_datetime(dataa['Date']).dt.date
```

```
dataa.head(1)
```

	Date	Open	High	Low	Close	Volume
0	2016-01-01	430.721008	436.246002	427.515015	434.334015	36278900.0

```
dataa.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2989 entries, 0 to 2988
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    2989 non-null   object 
 1   Open    2989 non-null   float64
 2   High    2989 non-null   float64
 3   Low     2989 non-null   float64
 4   Close   2989 non-null   float64
 5   Volume  2989 non-null   float64
dtypes: float64(5), object(1)
memory usage: 140.2+ KB
```

```
dataa.isnull().sum()
```

```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

3.Exploratory Data Analysis(EDA)

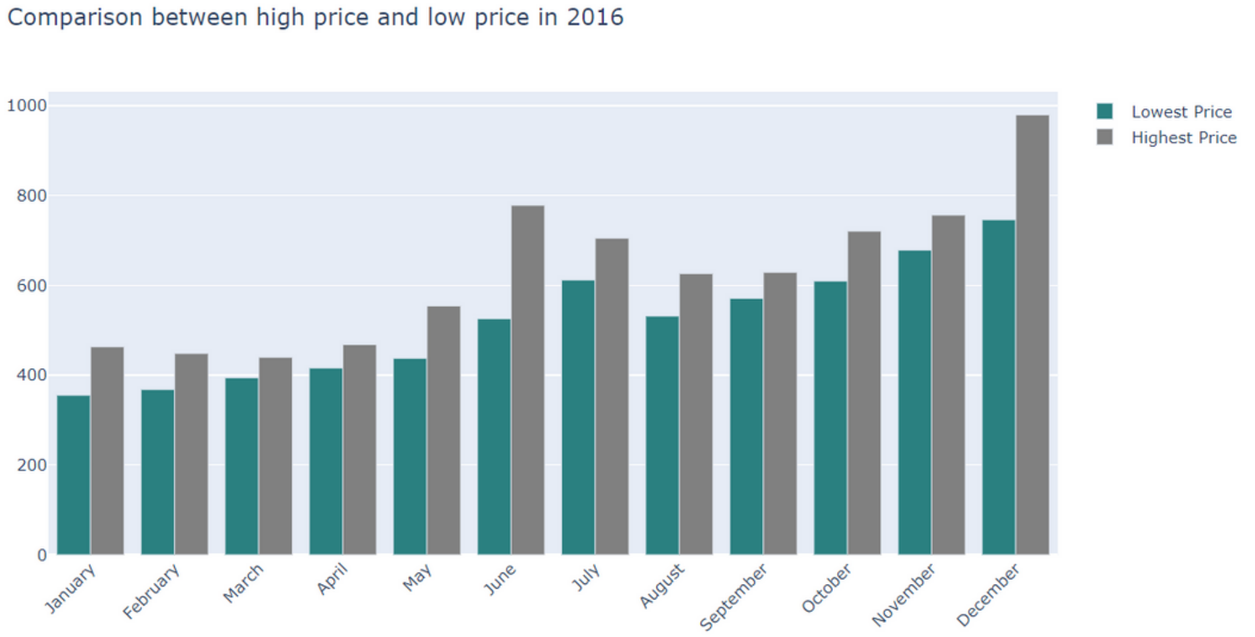
Analysis of Year 2016

```
dataa['Date'] = pd.to_datetime(dataa['Date'])
data_2016 = dataa[dataa['Date'].dt.year == 2016]
data_2016.drop('Volume', axis=1, inplace=True)
print(data_2016)
```

```
          Date        Open        High         Low       Close
0   2016-01-01  430.721008  436.246002  427.515015  434.334015
1   2016-01-02  434.622009  436.062012  431.869995  433.437988
2   2016-01-03  433.578003  433.743011  424.705994  430.010986
3   2016-01-04  430.061005  434.516998  429.084015  433.091003
4   2016-01-05  433.069000  434.182007  429.675995  431.959991
..         ...         ...         ...         ...         ...
361 2016-12-27  908.354004  940.047974  904.255005  933.197998
362 2016-12-28  934.830994  975.921021  934.830994  975.921021
363 2016-12-29  975.125000  979.396973  954.502991  973.497009
364 2016-12-30  972.534973  972.534973  934.833008  961.237976
365 2016-12-31  960.627014  963.742981  947.236023  963.742981

[366 rows x 5 columns]
```

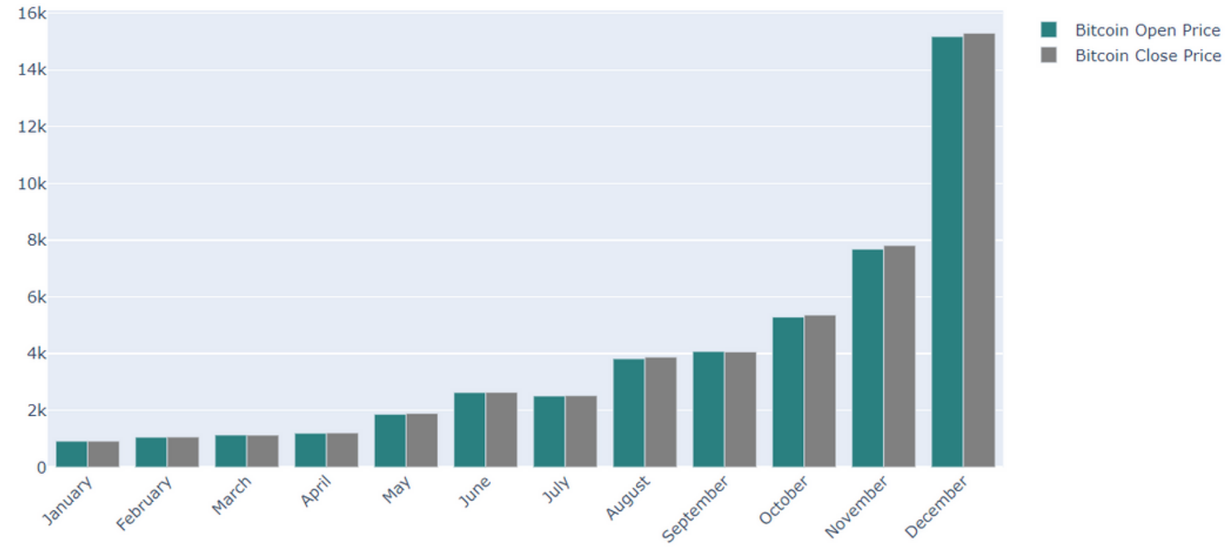
```
day_month1 = data_2016.groupby(data_2016['Date'].dt.strftime('%B'))[['Open','Close']].mean()


# Create a new list for the order of the months
new_name = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August',
            'September', 'October', 'November', 'December']

# Rearrange DataFrame in new order
day_month1 = day_month1.reindex(new_name, axis=0)


print(day_month1)


# Calculate the lowest value of the "Low" column and the highest value of the "High" column in each month
monthly_min_max = data_2016.groupby(data_2016['Date'].dt.strftime('%B')).agg({'Low': 'min', 'High': 'max'})

# Rearrange DataFrame in new order
monthly_min_max = monthly_min_max.reindex(new_name, axis=0)

print(monthly_min_max)
```

```
fig = go.Figure()

# Add columns to the chart
fig.add_trace(go.Bar(
    x=day_month1.index,
    y=day_month1['Open'],
    name='Bitcoin Open Price',
    marker_color='teal'
))
fig.add_trace(go.Bar(
    x=day_month1.index,
    y=day_month1['Close'],
    name='Bitcoin Close Price',
    marker_color='gray'
))

# Update the chart layout
fig.update_layout(barmode='group',
                  xaxis_tickangle=-45,
                  title='Comparison between opening price and closing price in 2016'
                 )


fig.show()
```

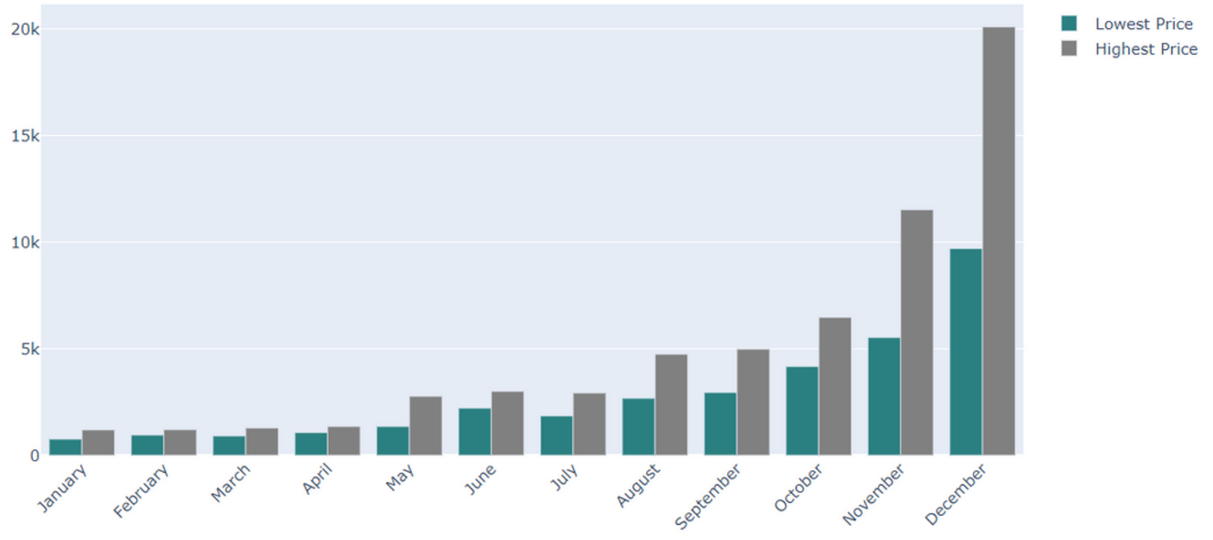



2016 analysis chart
1000
900
800
700
600
500
400
bitcoin value
Jan 2016
Mar 2016
May 2016
Jul 2016
Sep 2016
Nov 2016
Date
Parameters
Open
Close
High
Low

Comparison between opening price and closing price in 2017



Comparison between high price and low price in 2017

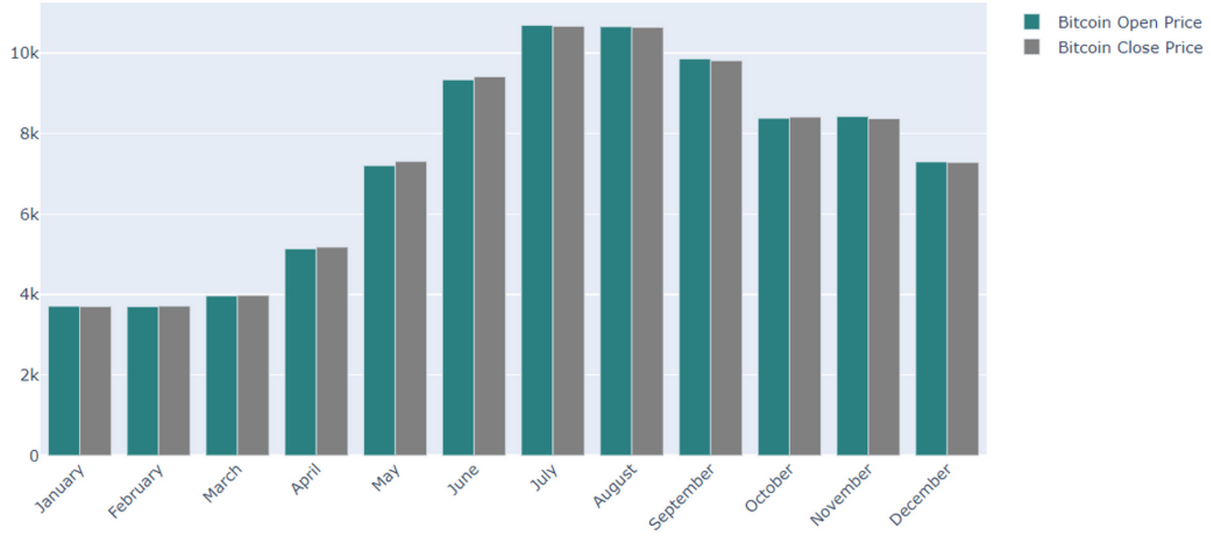



Comparison between opening price and closing price in 2018
Bitcoin Open Price
Bitcoin Close Price
20k
15k
10k
5k
0
January
February
March
April
May
June
July
August
September
October
November
December

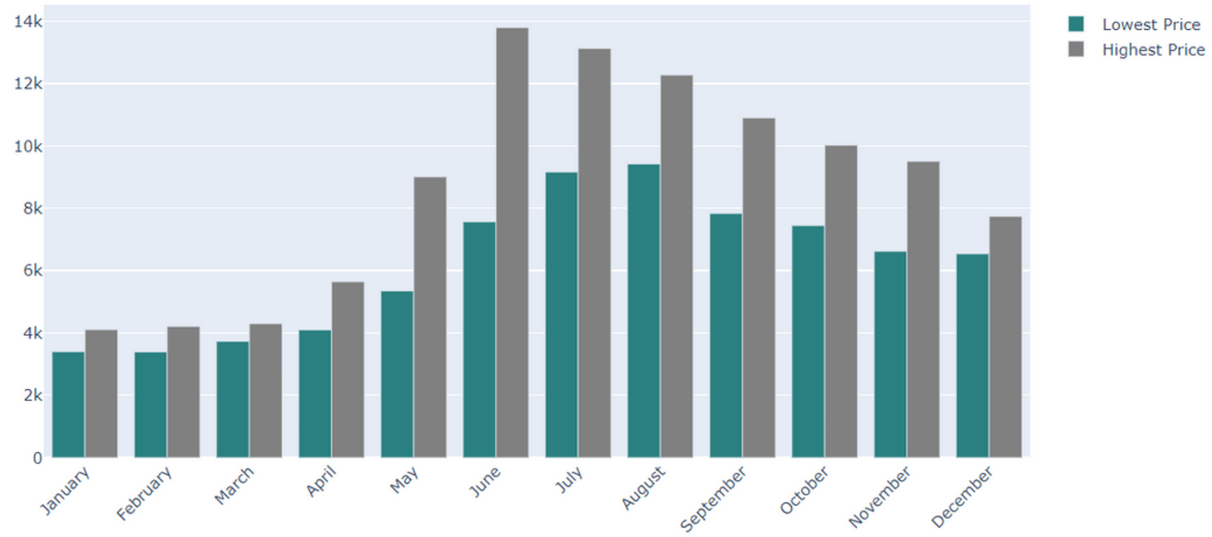

Comparison between high price and low price in 2018
Lowest Price
Highest Price
18k
16k
14k
12k
10k
8k
6k
4k
2k
0
January
February
March
April
May
June
July
August
September
October
November
December


2018 analysis chart
bitcoin value
Date
Parameters
Open
Close
High
Low
18k
16k
14k
12k
10k
8k
6k
4k
Jan 2018
Mar 2018
May 2018
Jul 2018
Sep 2018
Nov 2018

Comparison between opening price and closing price in 2019

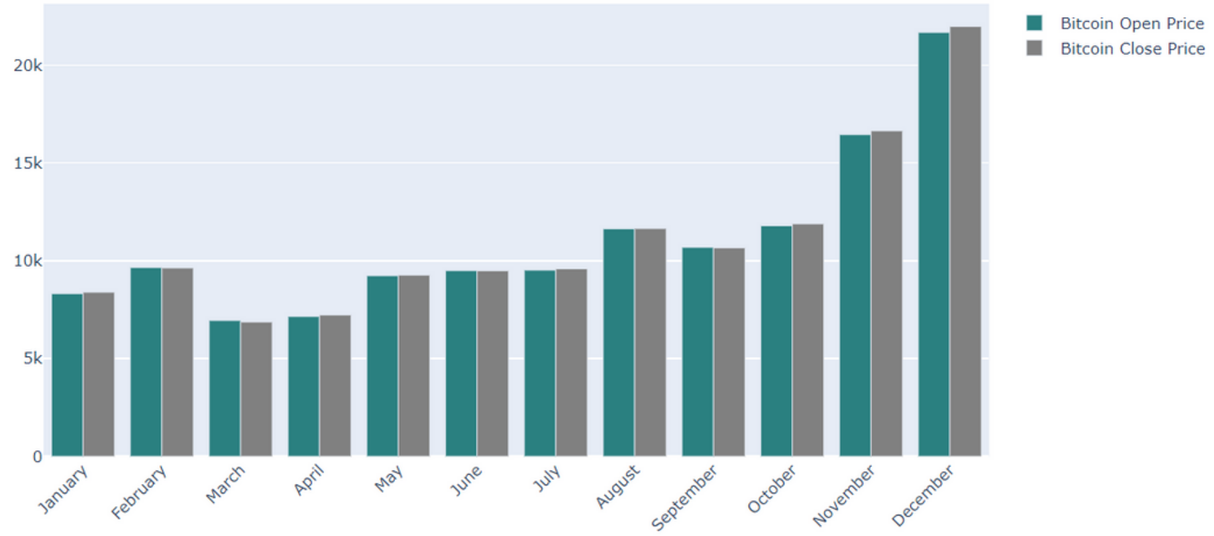


Comparison between high price and low price in 2019

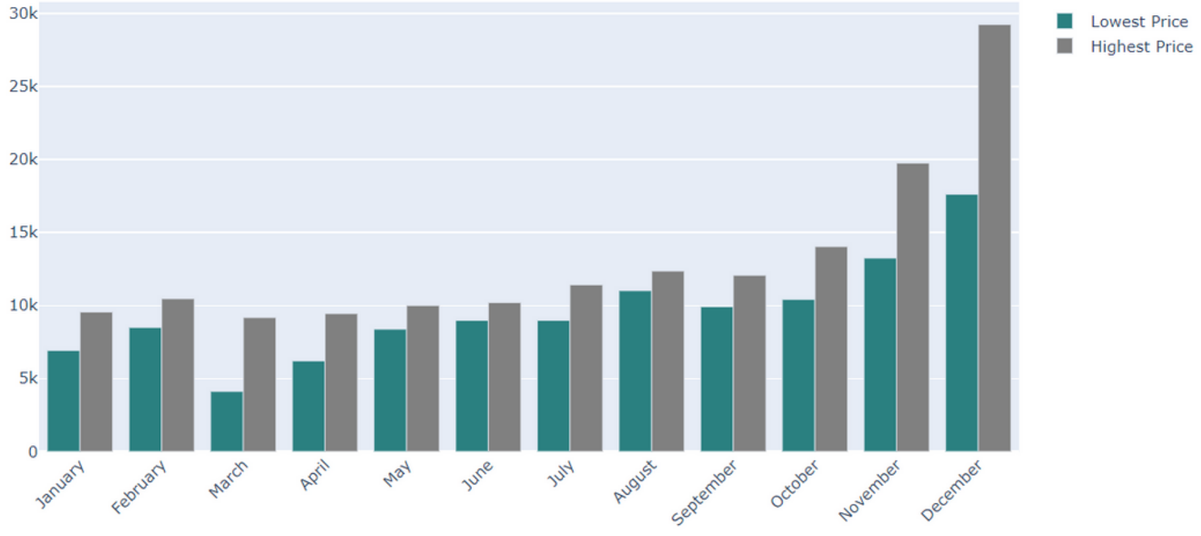


2019 analysis chart

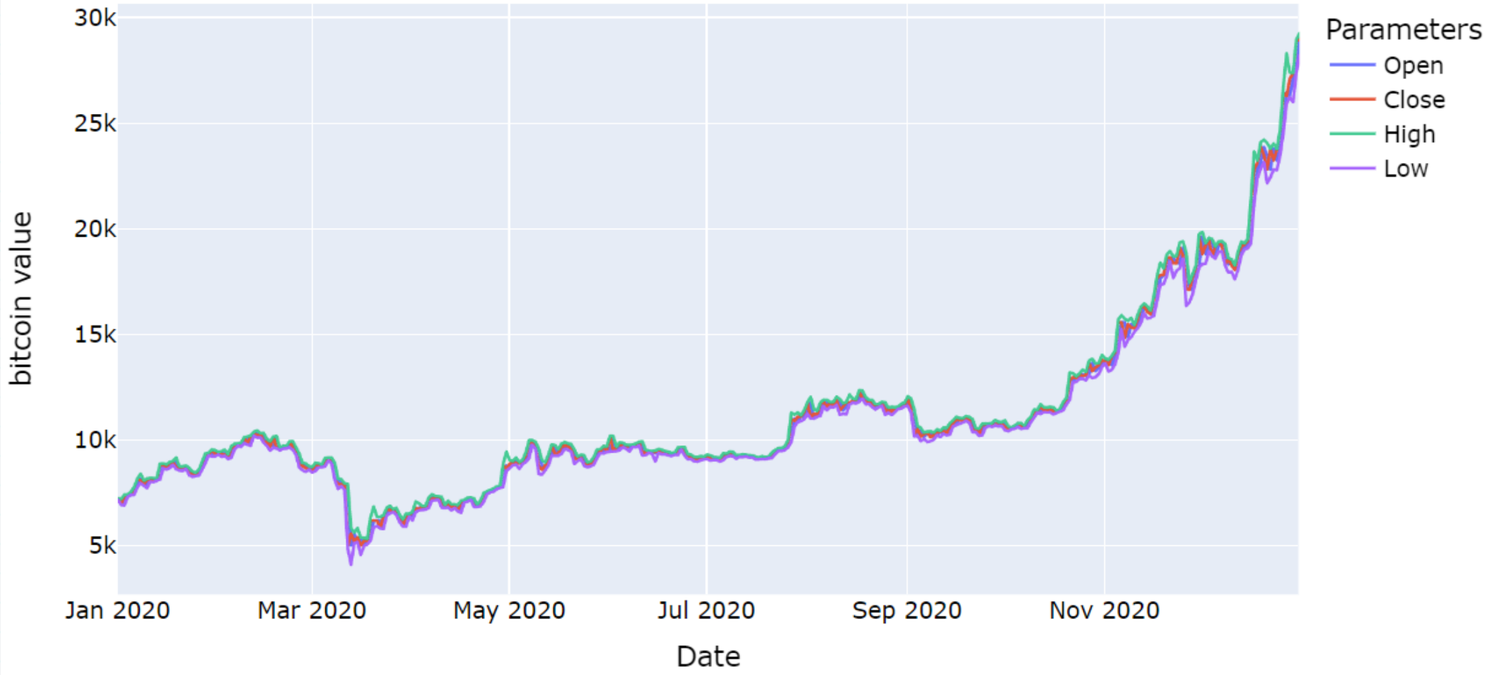
Comparison between opening price and closing price in 2020



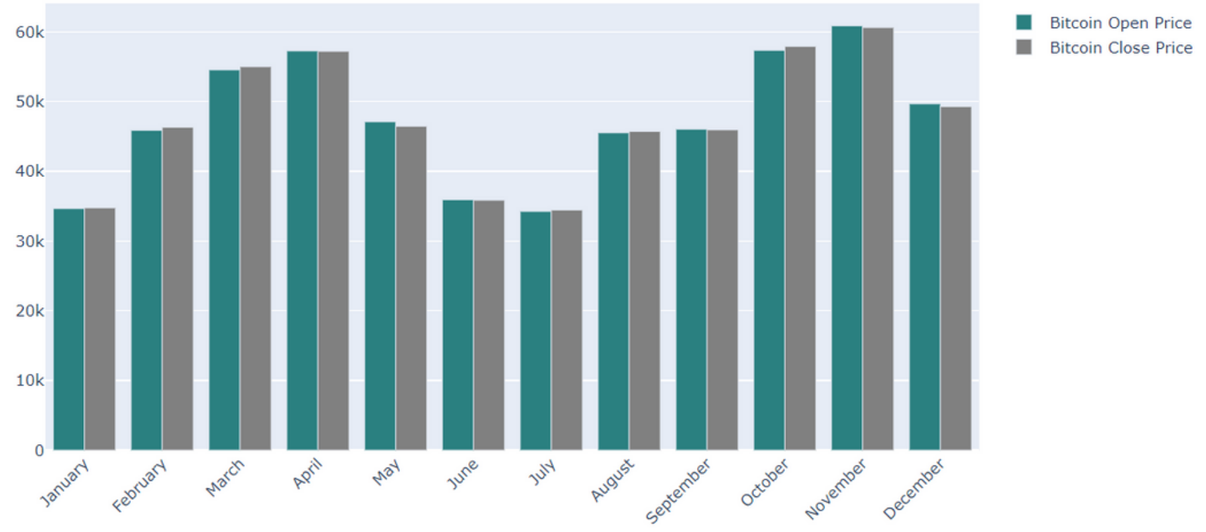
Comparison between high price and low price in 2020



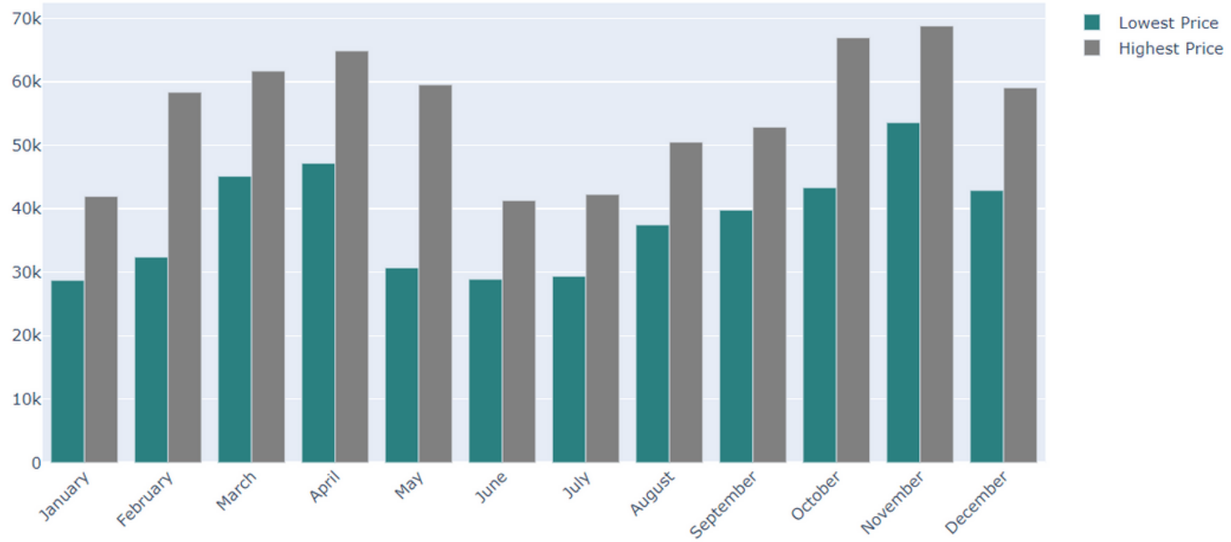
2020 analysis chart

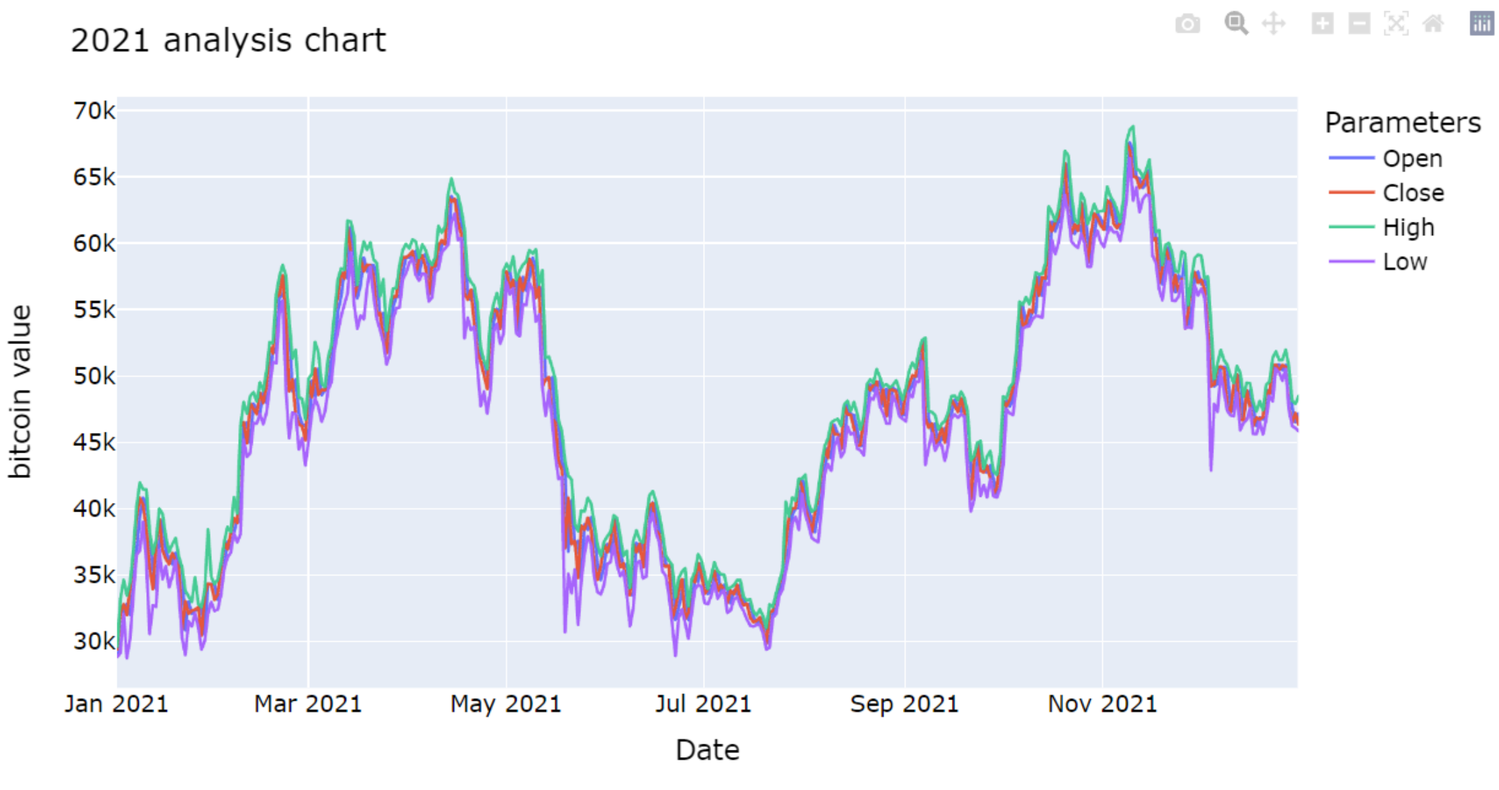


Comparison between opening price and closing price in 2021

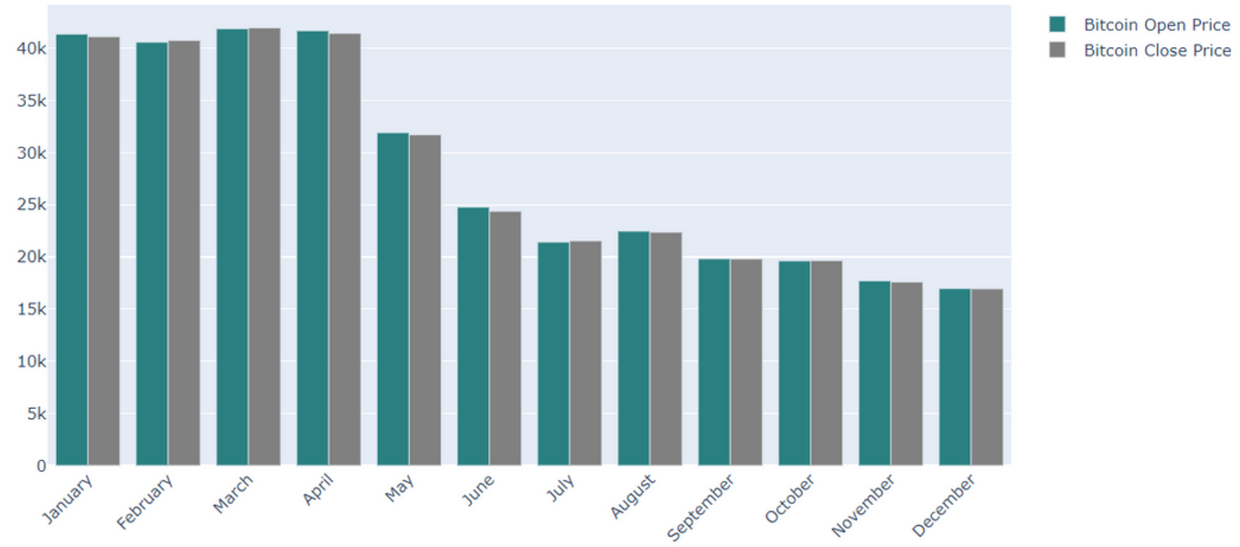


Comparison between high price and low price in 2021

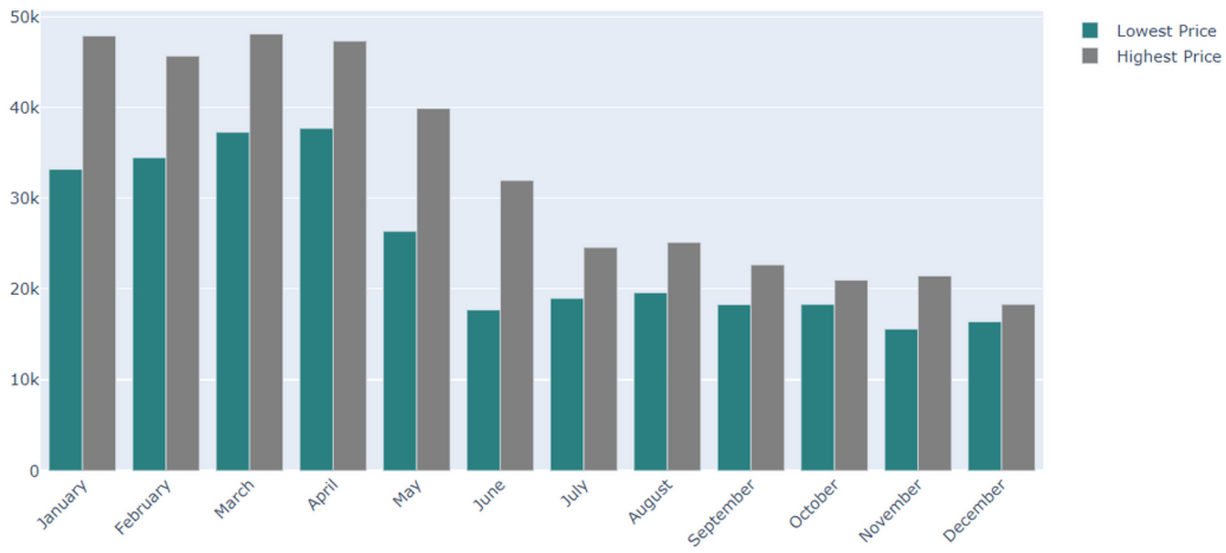




Comparison between opening price and closing price in 2022

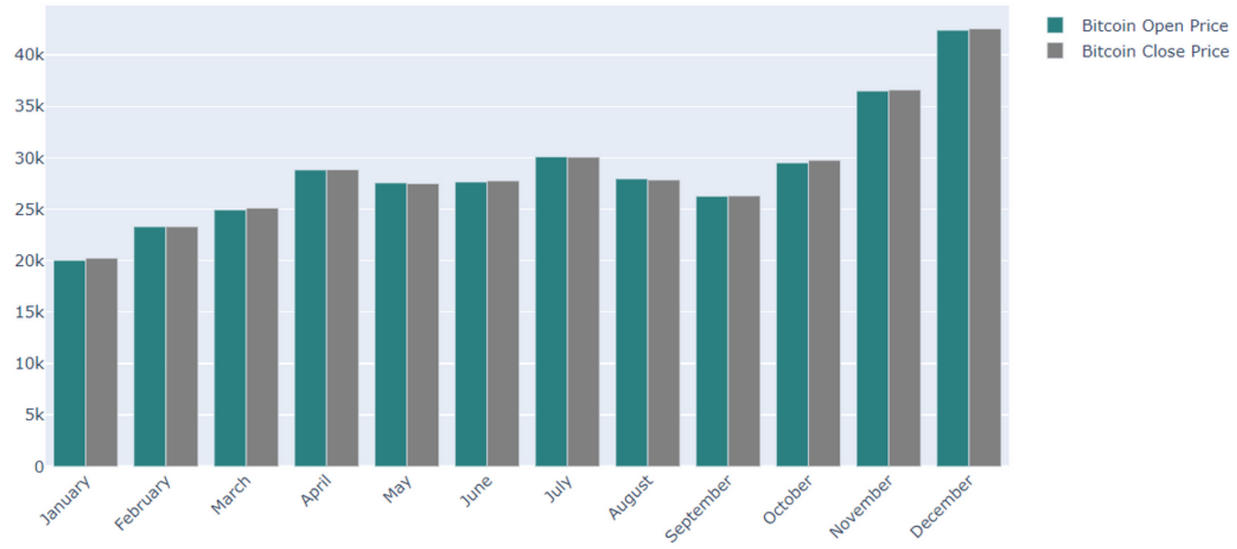


Comparison between high price and low price in 2022

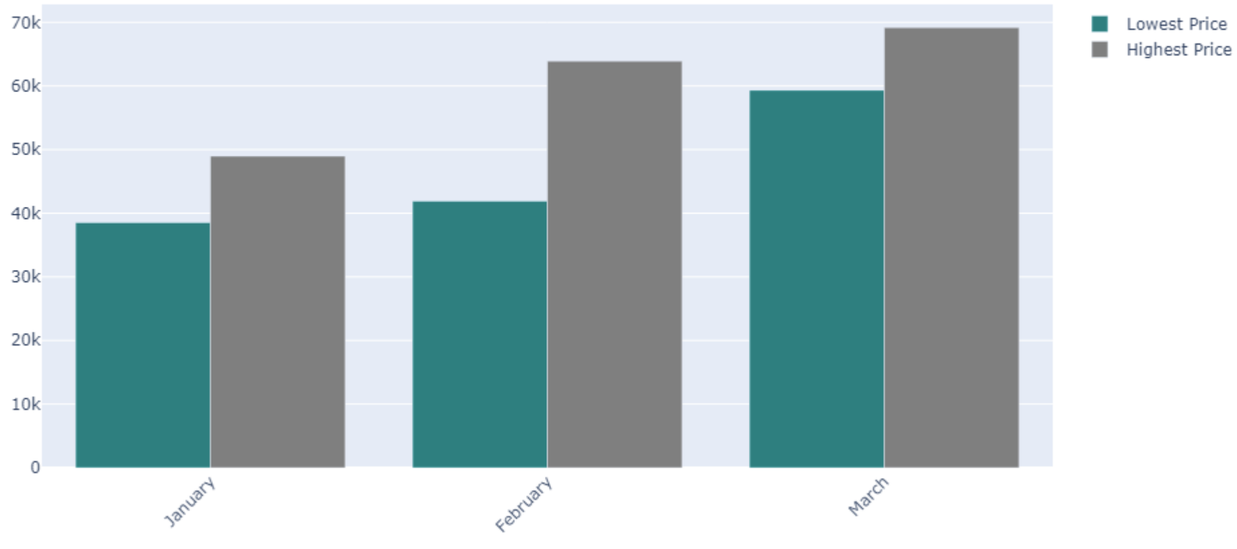


2022 analysis chart

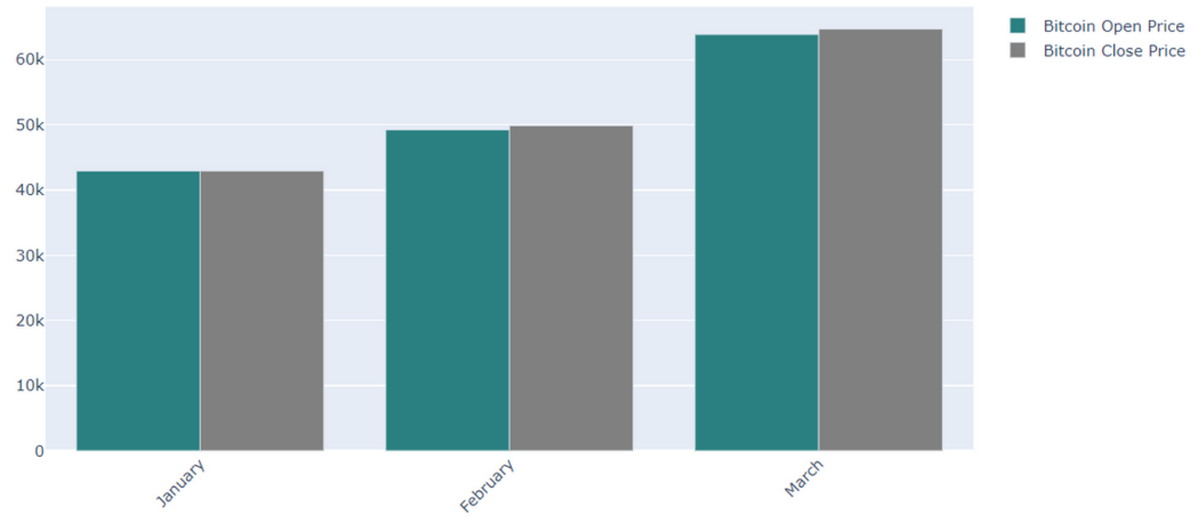
Comparison between opening price and closing price in 2023



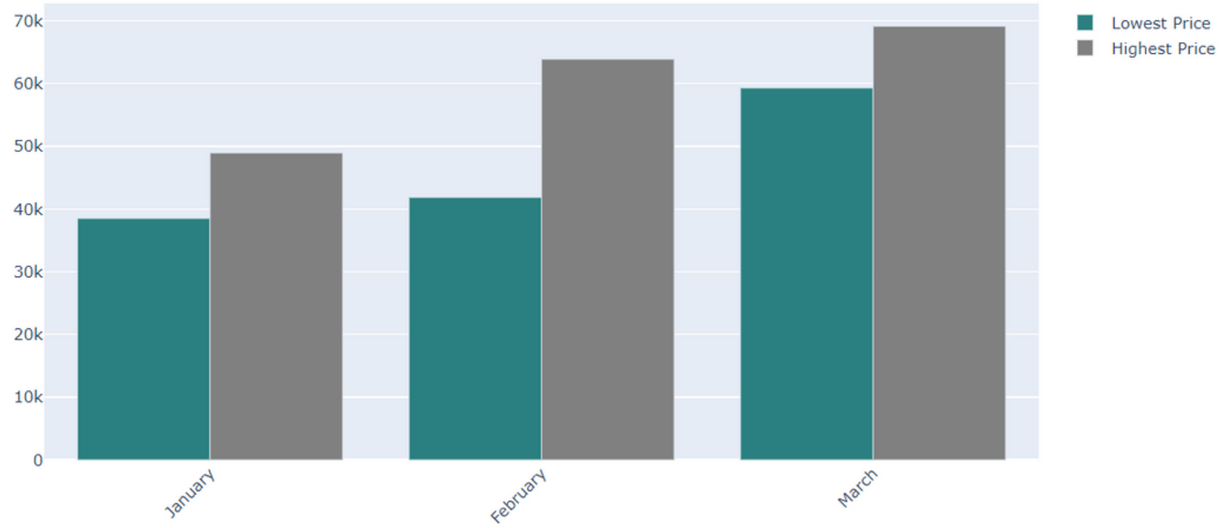
Comparison between high price and low price in 2023



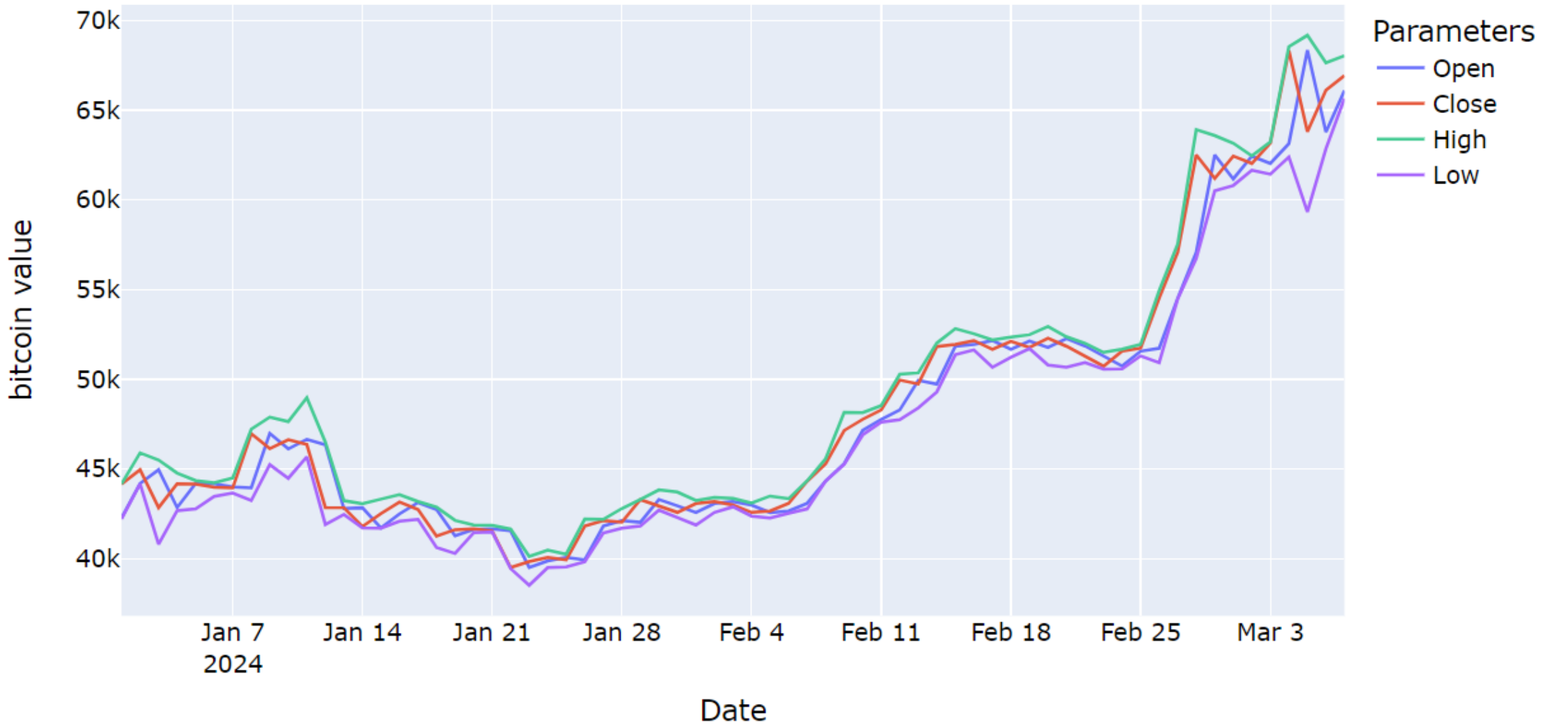
Comparison between opening price and closing price in 2024



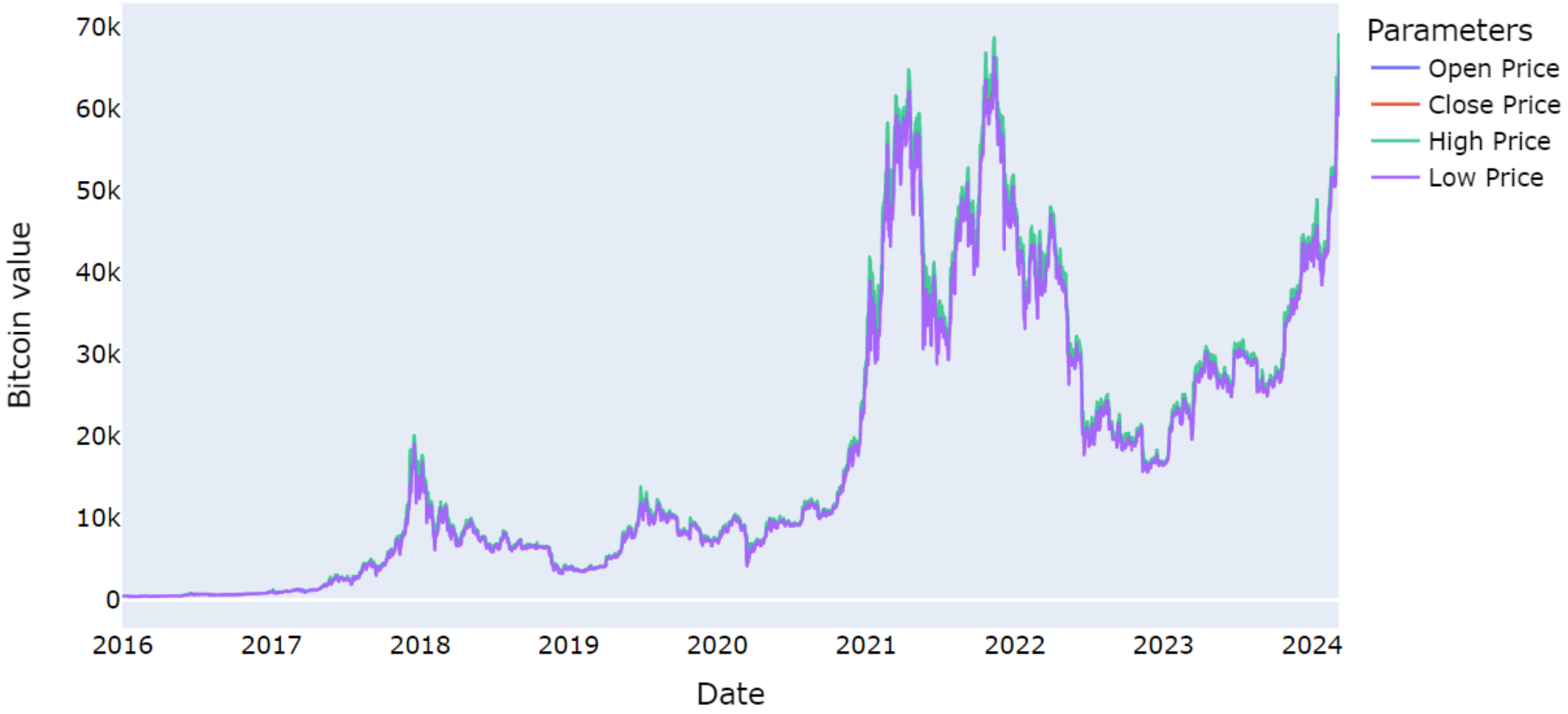
Comparison between high price and low price in 2024



2024 analysis chart



Analysis chart 2016 - 2024



4. Feature Engineering & Data Wrangling

```
# Lets First Take all the Close Price
closedataa = dataa[['Date','Close']]
print("Number of close: ", closedataa.shape)
```

```
Number of close:  (2989, 2)
```

```
start_date = '2021-01-01'
end_date = '2024-03-07'
data_filtered = closedataa[(closedataa['Date'] >= start_date) & (closedataa['Date'] <= end_date)]

print("Number of data points between 2021 and 7/3/2024: ", data_filtered.shape[0])

print(data_filtered)
```

```
Number of data points between 2021 and 7/3/2024:  1162
           Date         Close
1827 2021-01-01  29374.151889
1828 2021-01-02  32127.267939
1829 2021-01-03  32782.024466
1830 2021-01-04  31971.913522
1831 2021-01-05  33992.429344
...         ...           ...
2984 2024-03-03  63167.370358
2985 2024-03-04  68330.415608
2986 2024-03-05  63801.197561
2987 2024-03-06  66106.802787
2988 2024-03-07  66925.483202
```

```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

data_filtered = closedataa[(closedataa['Date'] >= start_date) & (closedataa['Date'] <= end_date)]

data_filtered['Close_scaled'] = scaler.fit_transform(data_filtered[['Close']])

print(data_filtered)
```

```
           Date         Close  Close_scaled
1827 2021-01-01  29374.151889      0.258585
1828 2021-01-02  32127.267939      0.310982
1829 2021-01-03  32782.024466      0.323444
1830 2021-01-04  31971.913522      0.308026
1831 2021-01-05  33992.429344      0.346480
...         ...           ...           ...
2984 2024-03-03  63167.370358      0.901737
2985 2024-03-04  68330.415608      1.000000
2986 2024-03-05  63801.197561      0.913800
2987 2024-03-06  66106.802787      0.957680
2988 2024-03-07  66925.483202      0.973261

[1162 rows x 3 columns]
```

5.Model Training and Evaluate

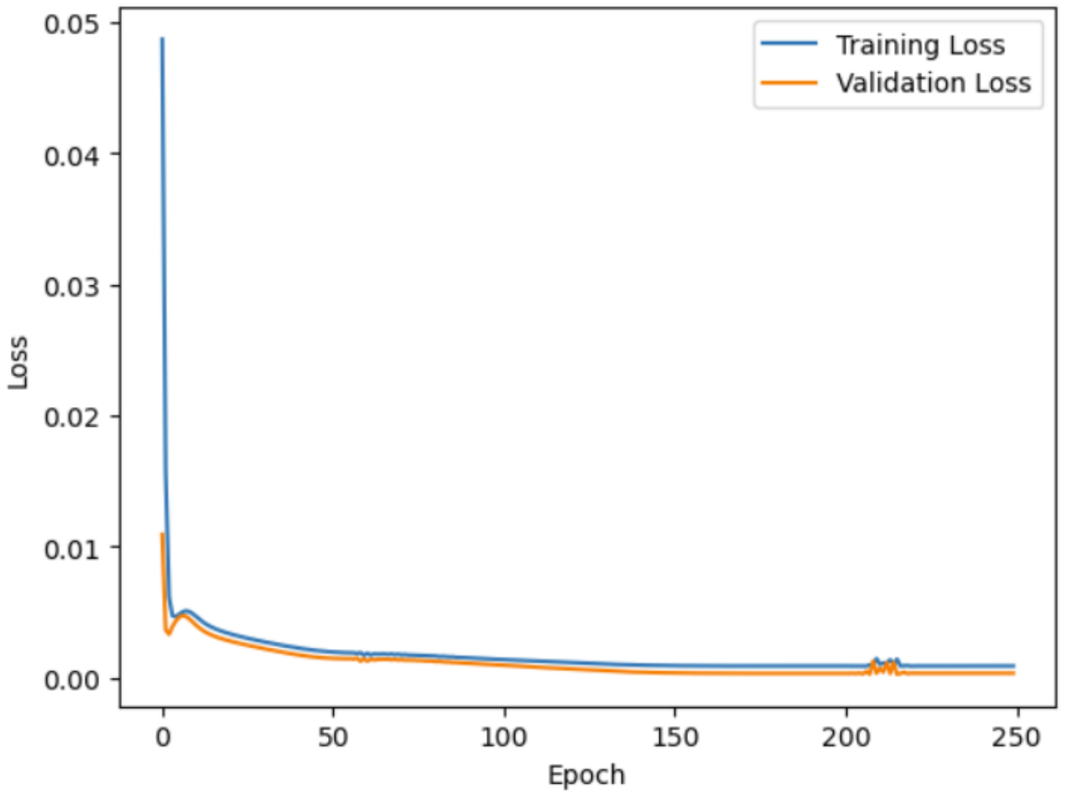
```
# number data tranning
total_samples = len(data_filtered)
train_size = int(total_samples * 0.6)
```

```
print(train_size)
```

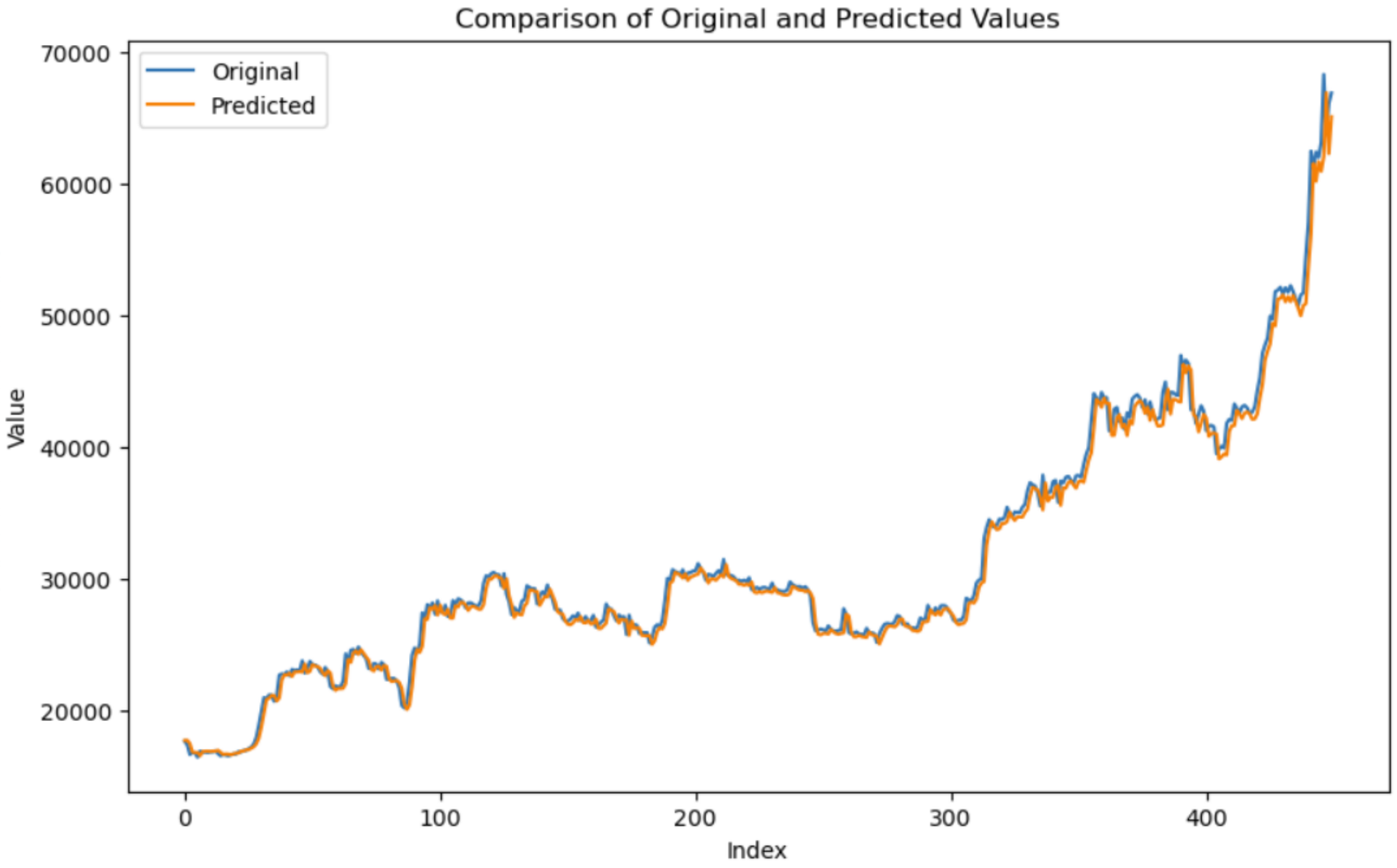
```
697
```

```
# Split data into training set and test set
train_data = data_filtered.iloc[:train_size]
test_data = data_filtered.iloc[train_size:]
```

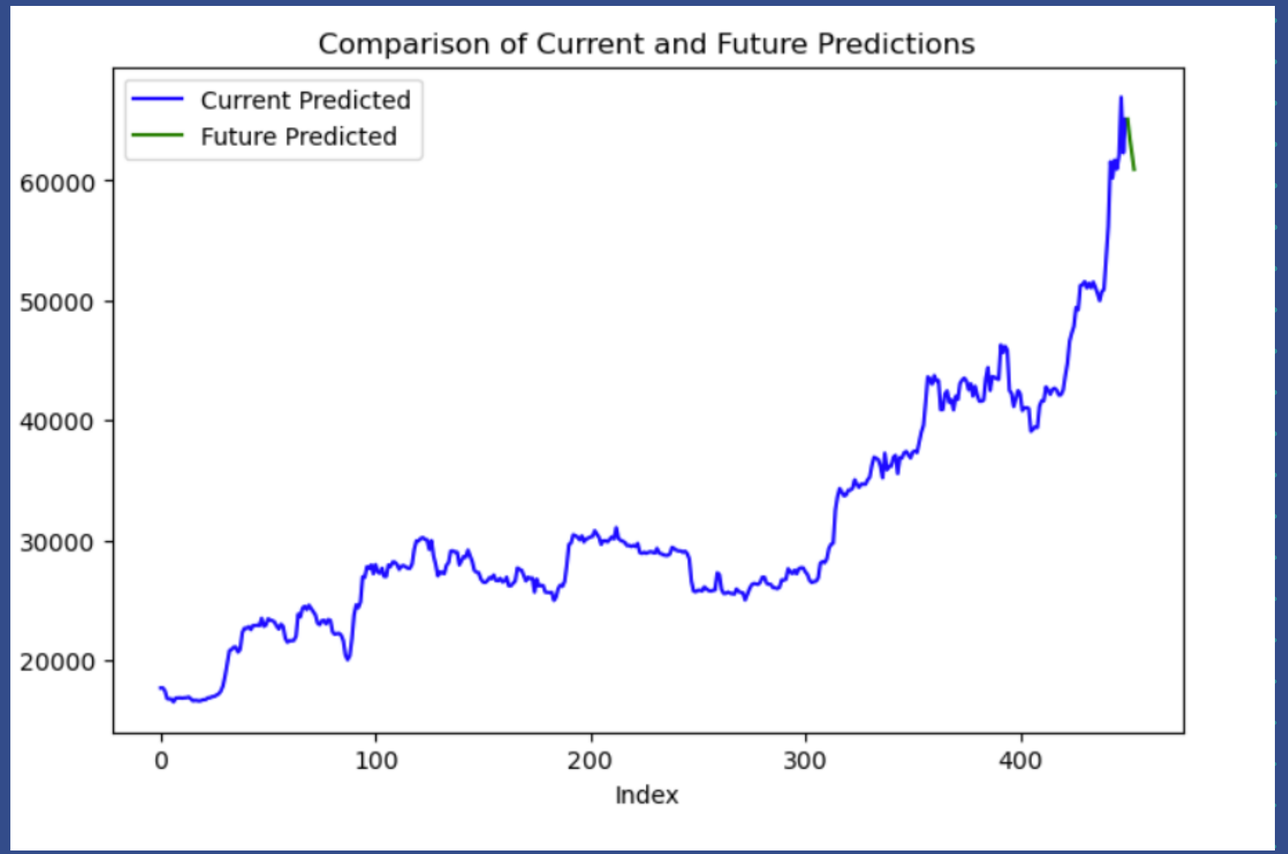
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# convert time series data into training data for the LSTM model
def create_dataset(X, y, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        v = X.iloc[i:(i + time_steps)].values
        Xs.append(v)
        ys.append(y.iloc[i + time_steps])
    return np.array(Xs), np.array(ys)

TIME_STEPS = 15

# Create a dataset for the LSTM model
X_train, y_train = create_dataset(train_data[['Close_scaled']], train_data['Close_scaled'], TIME_STEPS)
X_test, y_test = create_dataset(test_data[['Close_scaled']], test_data['Close_scaled'], TIME_STEPS)

# Build LSTM model
model = Sequential([
    LSTM(units=64, input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(units=1)
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=250,
    batch_size=20,
    validation_data=(X_test, y_test),
    shuffle=False
)
```




```
Train RMSE: 1630.5997603811068
Test RMSE: 973.2248940910694
Train MAE: 1153.0804946570438
Test MAE: 618.5295387361955
Train R2 Score: 0.985483275528965
Test R2 Score: 0.989735946516029 3
```


Comparison of Original and Predicted Values
Original
Predicted
70000
60000
50000
40000
30000
20000
Value
0
100
200
300
400
Index

```
Date 2024-03-08: 65114.90
Date 2024-03-09: 63528.98
Date 2024-03-10: 62253.72
Date 2024-03-11: 60928.52
```



USE BEST-PRACTICES

1. HYPERPARAMETER TUNING

```python
from sklearn.model_selection import RandomizedSearchCV
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor
from sklearn.metrics import r2_score

# Function to create model, required for KerasRegressor
def create_model(units=64):
    model = Sequential([
        LSTM(units=units, input_shape=(X_train.shape[1], X_train.shape[2])),
        Dense(units=1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

# Wrap Keras model for use with RandomizedSearchCV
model = KerasRegressor(build_fn=create_model, verbose=0)

# Define hyperparameters grid
param_grid = {
    'units': [32, 64, 128],
    'batch_size': [15, 20, 25],
    'epochs': [150, 200, 250, 270]
}

# Perform Randomized Search
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=10, cv=3)
random_result = random_search.fit(X_train, y_train)

# Print best parameters
print("Using:", random_result.best_params_)
```

```
Using: {'units': 128, 'epochs': 150, 'batch_size': 25}
```

R-squared: 0.9916014203338646
Mean Squared Error: 0.0002807252341578

2. REGULARIZERS

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras import regularizers

# Build LSTM model with regularization
model = Sequential([
    LSTM(units=64, input_shape=(X_train.shape[1], X_train.shape[2])),
    Dropout(0.1),  # Dropout layer to prevent overfitting
    Dense(units=1, kernel_regularizer=regularizers.l2(0.02))  # L2 regularization
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

history = model.fit(
    X_train, y_train,
    epochs=250,
    batch_size=20,
    validation_data=(X_test, y_test),
    shuffle=False
)
```

```
15/15 [==============================] - 0s 3ms/step
Test MSE: 0.0013838249587237899
Test R-square: 0.9585995031950161
```

3.OPTIMIZATION USING RMSPROP AND ADAGRAD

```python
from tensorflow.keras.optimizers import RMSprop

# Build LSTM model with RMSprop optimizer
model = Sequential([
    LSTM(units=64, input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(units=1)
])

# Compile the model with RMSprop optimizer
optimizer = RMSprop(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mean_squared_error')

history = model.fit(
    X_train, y_train,
    epochs=250,
    batch_size=20,
    validation_data=(X_test, y_test),
    shuffle=False
)
```

```
15/15 [==============================] - 0s 3ms/step
Test MSE: 0.00049302939325896344
Test R-squared: 0.9852498239289638
```

3.OPTIMIZATION USING RMSPROP AND ADAGRAD

```
from tensorflow.keras.optimizers import Adagrad

# Build LSTM model
model = Sequential([
    LSTM(units=64, input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(units=1)
])

# Compile the model with Adagrad optimizer
model.compile(optimizer=Adagrad(), loss='mean_squared_error')

history = model.fit(
    X_train, y_train,
    epochs=250,
    batch_size=20,
    validation_data=(X_test, y_test),
    shuffle=False
)
```

```
15/15 [==============================] - 2s 3ms/step
Test MSE: 0.00151142364195998 2
Test R-square: 0.9547820775557849
```

Conclusion

Conclusion

- Hyperparameter tuning using randomsearch:

After performing hyperparameter tuning using random search, we see a significant improvement in the R-squared coefficient of determination, increasing from 0.9897 to 0.9916. This shows that the model is more optimal in predicting test data after hyperparameter tuning.

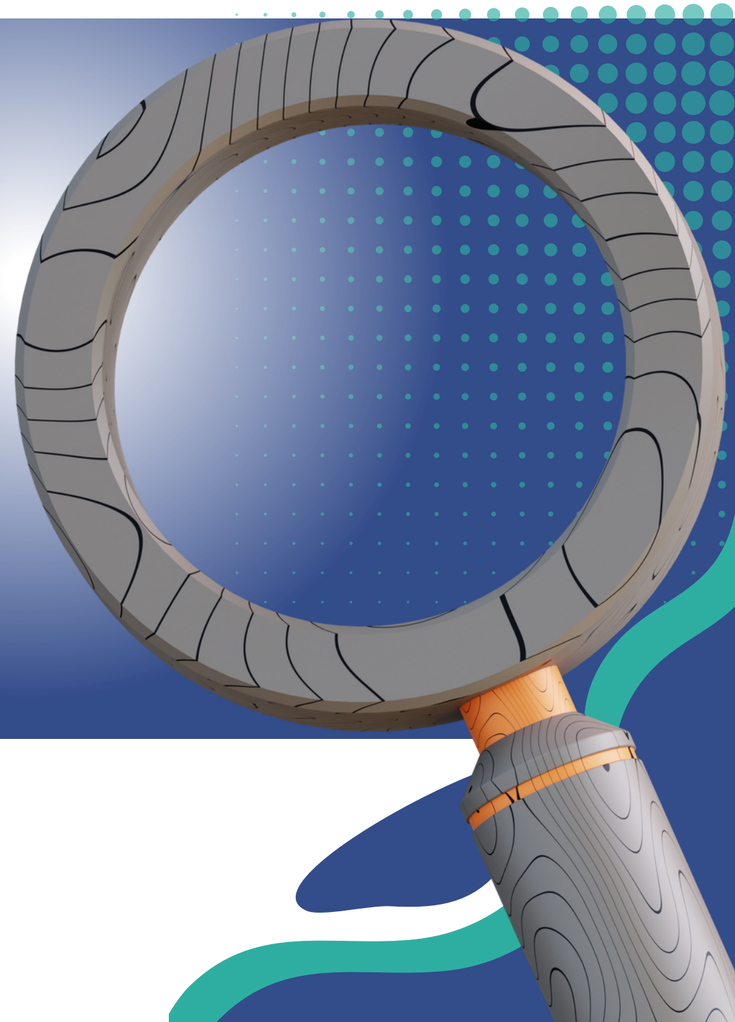
- Regularizers

After applying the regularizers technique, we see a significant decrease in the model accuracy. Test R-squared decreased from 0.9897 to 0.9586. This shows that using regularizers can reduce the model's prediction performance on the test data set.

- Optimization using RMSprop and Adagrad:

When optimizing with RMSprop, we see an improvement in model accuracy, with Test R-squared increasing to 0.9852 from 0.9897. This shows that RMSprop produces a better model for prediction. However, when using Adagrad, we see a decrease in model accuracy. Test R-squared decreased to 0.9547 from 0.9897. This shows a poor performance of this optimization method for the test data set.

In summary, in this case, fine-tuning hyperparameters using the random search method is the best choice to improve the accuracy of the machine learning model.




Thank You