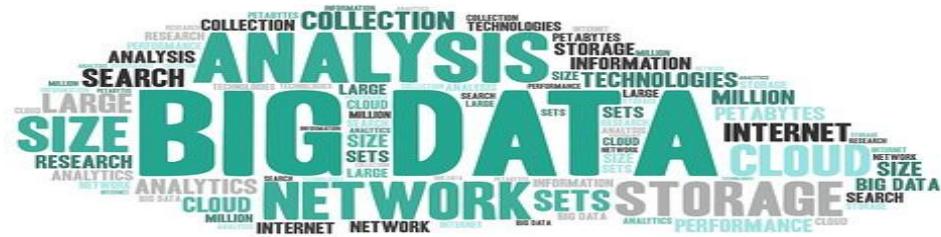


# Big Data Application Mini Project Report



Author: *Dilip Nikhil Francies*

## 1. Data Set – Bitcoin Closing Price Prediction

The dataset used for this project was downloaded from Kaggle. It contains 6 columns with a total shape of 6,723,281 x 6. The key features of the dataset are as follows:

- **Timestamp:** Provided in Unix format.
  - **Open, High, Low, Close:** Values of Bitcoin being traded (in USD).
  - **Volume:** Number of Bitcoins traded per second.
  - The data is provided at **one-minute intervals**.

Timestamp	Open	High	Low	Close	Volume
1.32541206E9	4.58	4.58	4.58	4.58	0.0
1.32541212E9	4.58	4.58	4.58	4.58	0.0
1.32541218E9	4.58	4.58	4.58	4.58	0.0
1.32541224E9	4.58	4.58	4.58	4.58	0.0
1.3254123E9	4.58	4.58	4.58	4.58	0.0
1.32541236E9	4.58	4.58	4.58	4.58	0.0
1.32541242E9	4.58	4.58	4.58	4.58	0.0
1.32541248E9	4.58	4.58	4.58	4.58	0.0
1.32541254E9	4.58	4.58	4.58	4.58	0.0
1.3254126E9	4.58	4.58	4.58	4.58	0.0
1.32541266E9	4.58	4.58	4.58	4.58	0.0
1.32541272E9	4.58	4.58	4.58	4.58	0.0
1.32541278E9	4.58	4.58	4.58	4.58	0.0
1.32541284E9	4.58	4.58	4.58	4.58	0.0
1.3254129E9	4.58	4.58	4.58	4.58	0.0
1.32541296E9	4.58	4.58	4.58	4.58	0.0
1.32541302E9	4.58	4.58	4.58	4.58	0.0
1.32541308E9	4.58	4.58	4.58	4.58	0.0
1.32541314E9	4.58	4.58	4.58	4.58	0.0
1.3254132E9	4.58	4.58	4.58	4.58	0.0

Using these features, along with additional technical indicators such as **moving averages**, **exponential moving averages**, **price momentum**, and **rate of change**, the goal was to predict the **closing price** after aggregating the dataset at a **daily level** using a regression model (not a time series model).

## 2. Environment Setup –

### Upload Dataset to AWS S3 Using AWS CLI

The first step was to upload the raw and processed datasets to S3 for storage and further processing. Here are the steps followed:

- 1) **Create an AWS Account:** Set up IAM roles, generate AWS access keys, and configure permissions.
- 2) **Install and Configure AWS CLI:** Download and set up AWS CLI on a Linux environment. Since free tier ec2 was taking forever and lack of resources on jetStream2, I used colab VM as my Linux environment.

```
PS C:\Users\dilip\OneDrive - Indiana University\BigData\PROJECT> aws s3api create-bucket --bucket big-data-project-dilip --region us-east-1
{
    "Location": "/big-data-project-dilip"
}

PS C:\Users\dilip\OneDrive - Indiana University\BigData\PROJECT> aws s3 ls
2024-12-04 21:46:23 big-data-project-dilip
PS C:\Users\dilip\OneDrive - Indiana University\BigData\PROJECT> aws s3 sync ./raw_data s3://big-data-project-dilip
upload: raw_data\btcusd_1-min_data.csv to s3://big-data-project-dilip/btcusd_1-min_data.csv
PS C:\Users\dilip\OneDrive - Indiana University\BigData\PROJECT> aws s3 ls s3://big-data-project-dilip --recursive
2024-12-04 21:48:02 349463898 btcusd_1-min_data.csv
PS C:\Users\dilip\OneDrive - Indiana University\BigData\PROJECT> 
```

**Confirm Upload:** Verified the successful upload of both raw and processed data.

Name	Type	Last modified	Size	Storage class
btcusd_1-min_data.csv	CSV	December 4, 2024, 21:48:02 (UTC-05:00)	333.3 MB	Standard
processed_bitcoin_data.csv	CSV	December 14, 2024, 18:49:34 (UTC-05:00)	1.3 MB	Standard

Note: The picture contains raw and processed data uploaded. Have included so that there is no redundancy later in the report.

## Date Pipeline Tasks:

### Data Ingestion

The pipeline was designed to read data from S3, process it using **PySpark**, and write the results back to S3. Configuring the environment is shown below.

```
s3 = boto3.client('s3')

bucket_name = 'big-data-project-dilip'
file_key = 'btcusd_1-min_data.csv'

response = s3.get_object(Bucket=bucket_name, Key=file_key)
csv_content = response['Body'].read().decode('utf-8')

s3_url = f"s3://{bucket_name}/{file_key}"
s3_url
's3://big-data-project-dilip/btcusd_1-min_data.csv'

s3_url = f"s3://{bucket_name}/{file_key}"
obj = s3.get_object(Bucket=bucket_name, Key=file_key)
data = obj['Body'].read().decode('utf-8')

df_pandas = pd.read_csv(io.StringIO(data))
df = spark.createDataFrame(df_pandas)
df.show()
df.printSchema()

+-----+-----+-----+-----+
| Timestamp|Open|High| Low|Close|Volume|
+-----+-----+-----+-----+
| 1.32541206E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541212E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541218E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541224E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.3254123E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541236E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541242E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541248E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541254E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.3254126E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541266E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541272E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541278E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541284E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.3254129E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541296E9|4.58|4.58| 4.58| 4.58| 0.0|
| 1.32541302E9|4.58|4.58| 4.58| 4.58| 0.0|
```

*More information, code and steps, present in the notebook attached with this file.*

### Data Exploration and Preprocessing:

- Handled missing values (nulls).
- Converted Unix timestamps into proper datetime format.
- Created three new columns: **Year**, **Month**, and **Day**.

## Data Aggregation –

Here are some of the key metrics that I calculated.

*Monthly Average Closing Price:*

### 1. Monthly Average Closing Price

```
[ ] from pyspark.sql.functions import avg

monthly_avg_close = df_transformed.groupBy("Year", "Month").agg(avg("Close").alias("Avg_Close"))
monthly_avg_close.show()
```

Year	Month	Avg_Close
2012	10	11.630673163082575
2015	2	233.8848454861181
2017	3	1132.6591767473308
2014	4	462.5960842592671
2015	12	422.15270810932293
2016	7	660.7412473118395
2016	11	722.9622171997954
2012	8	10.92016935483949
2013	2	25.707080109127205
2012	4	4.980315277777693
2012	12	13.146735215053406
2014	10	364.2635499552096
2016	5	459.49873342293984
2014	12	343.1390557795864
2013	9	124.89339050926257
2013	10	152.31140501792743
2014	5	484.6043272849413
2016	2	401.459839680472
2017	7	2492.8033066756175
2013	12	797.2092573924576

only showing top 20 rows

*Total Volume Traded per month:*

### 2. Total Volume Traded Per Month

```
[ ] monthly_volume = df_transformed.groupBy("Year", "Month").agg({"Volume": "sum"})
monthly_volume.show()
```

Year	Month	sum(Volume)
2012	10	83104.04747408001
2015	2	351708.969556257
2017	3	321058.7148016618
2014	4	1517371.75097255554
2015	12	422649.12320094294
2016	7	118452.46361068907
2016	11	154162.8690078005
2012	8	82848.67116894989
2013	2	123510.87774068979
2012	4	16484.167772440003
2012	12	91377.29088790013
2014	10	599895.4417664655
2016	5	135188.60074122954
2014	12	294441.8571654428
2013	9	322672.50941772043
2013	10	625096.5716852888
2014	5	302311.93173702917
2016	2	216664.02664584914
2017	7	442045.6083083977
2013	12	953353.1568645568

only showing top 20 rows

*Top 10 days with Highest Volume –*

### 3. Top 10 Days with Highest Volume

```
[ ] daily_volume = df_transformed.groupBy("Year", "Month", "Day").agg({"Volume": "sum"}) \
    .orderBy("sum(Volume)", ascending=False)
daily_volume.show(10)
```

```
→+---+---+---+-----+
|Year|Month|Day|      sum(Volume)|
+---+---+---+-----+
|2013|   12| 18| 127286.48653306|
|2015|     1| 14|115814.46241601992|
|2014|     2| 25|112848.33758712989|
|2015|     11|   4|100714.05054385003|
|2013|   12|   7| 92891.97224274017|
|2015|     1| 15| 89350.93392063005|
|2013|     11| 19| 84192.38187954016|
|2015|     11|   5| 80972.91950993001|
|2014|     10|   6| 79712.11416376993|
|2015|     11|   3| 76952.78107203002|
+---+---+---+-----+
only showing top 10 rows
```

*Monthly High and Low Prices –*

### 4. Monthly High and Low Prices

```
[ ] from pyspark.sql.functions import max, min

monthly_high_low = df_transformed.groupBy("Year", "Month") \
    .agg(max("High").alias("Monthly_High"), min("Low").alias("Monthly_Low"))
monthly_high_low.show()
```

```
→+---+---+---+-----+
|Year|Month|Monthly_High|Monthly_Low|
+---+---+---+-----+
|2012|   10|      12.99|       9.5|
|2015|     2|     267.92|    208.48|
|2017|     3|    1350.0|    891.33|
|2014|     4|      548.0|    339.79|
|2015|    12|     467.8|    348.64|
|2016|     7|    704.99|    605.5|
|2016|    11|    755.07|    670.32|
|2012|     8|      16.41|       7.1|
|2013|     2|     34.24|    19.5|
|2012|     4|      5.43|    4.69|
|2012|    12|     13.94|    12.24|
|2014|    10|    417.99|    275.0|
|2016|     5|      548.5|    435.0|
|2014|    12|     383.0|    304.99|
|2013|     9|     134.85|    115.0|
```

## Days with Maximum Price Volatility –

```
5. Days with Maximum Price Volatility

Price volatility is calculated as the difference between the High and Low prices for the day.

[ ] from pyspark.sql.functions import abs

daily_volatility = df_transformed.withColumn("Volatility", abs(df_transformed["High"] - df_transformed["Low"])) \
    .groupBy("Year", "Month", "Day") \
    .agg({"Volatility": "max"}) \
    .orderBy("max(Volatility)", ascending=False)

daily_volatility.show(10)

+---+---+---+-----+
|Year|Month|Day| max(Volatility)|
+---+---+---+-----+
|2021| 10 | 11 | 4747.760000000002 |
|2022| 1  | 24 | 4664.68 |
|2021| 10 | 28 | 4380.379999999997 |
|2021| 10 | 18 | 3078.410000000035 |
|2021| 4  | 14 | 3004.760000000002 |
|2021| 12 | 4  | 2985.910000000035 |
|2021| 7  | 26 | 2622.339999999965 |
|2024| 2  | 28 | 2532.0 |
|2021| 11 | 3  | 2508.080000000017 |
|2021| 4  | 18 | 2457.139999999994 |
+---+---+---+-----+
only showing top 10 rows
```

## Store Processed Data Back to S3 (0.5 Point)

Once these values were calculated the data frame was converted to csv, downloaded and uploaded to S3 bucket under a separate folder “Aggregated data” as shown below.

**Note:** The csv files are also attached to this document for your reference

Name	Type	Last modified	Size	Storage class
average_vol.csv	CSV	December 15, 2024, 13:39:55 (UTC-05:00)	306.0 B	Standard
avg_close_by_signal.csv	CSV	December 15, 2024, 13:39:55 (UTC-05:00)	62.0 B	Standard
avg_high_low_by_month.csv	CSV	December 15, 2024, 13:39:56 (UTC-05:00)	6.6 KB	Standard
daily_volatility.csv	CSV	December 15, 2024, 13:39:56 (UTC-05:00)	117.0 KB	Standard
daily_volume.csv	CSV	December 15, 2024, 13:39:56 (UTC-05:00)	128.6 KB	Standard
highest_volatility.csv	CSV	December 15, 2024, 13:39:56 (UTC-05:00)	163.0 B	Standard
month_over_month.csv	CSV	December 15, 2024, 13:39:57 (UTC-05:00)	3.9 KB	Standard
monthly_avg_close.csv	CSV	December 15, 2024, 13:39:57 (UTC-05:00)	3.9 KB	Standard
monthly_high_low.csv	CSV	December 15, 2024, 13:39:57 (UTC-05:00)	3.4 KB	Standard
monthly_volume.csv	CSV	December 15, 2024, 13:39:57 (UTC-05:00)	3.9 KB	Standard

## Data Analysis Using Spark SQL

1. Register the processed data as a temporary SQL table and start querying.

```
[ ] # Register the DataFrame as a temporary SQL table  
bitcoin.createOrReplaceTempView("bitcoin")
```

Here are some of the insights that I derived from the processed data –

1. *Identify the Days with the Highest Average Volume –*

### 1. Identify the Days with the Highest Average Volume

```
[ ] query_1 = spark.sql("""  
    SELECT Date, Avg_Volume  
    FROM bitcoin  
    ORDER BY Avg_Volume DESC  
    LIMIT 10  
""")  
query_1.write.csv("average_vol.csv", header=True, mode="overwrite")  
query_1.show()
```

```
→ +-----+-----+  
|      Date|  Avg_Volume|  
+-----+-----+  
|2013-12-18| 88.39339342573611|  
|2015-01-14| 80.42671001112494|  
|2014-02-25| 78.36690110217354|  
|2015-11-04| 69.94031287767363|  
|2013-12-07| 64.50831405745845|  
|2015-01-15| 62.0492596671042|  
|2013-11-19| 58.466931860791775|  
|2015-11-05| 56.23119410411807|  
|2014-10-06| 55.35563483595134|  
|2015-11-03| 53.43943130002084|  
+-----+-----+
```

2. Calculate the month over month average closing price –

```
2. Calculate Month-over-Month Average Closing Price Growth

query_2 = spark.sql("""
    SELECT
        YEAR(Date) AS Year,
        MONTH(Date) AS Month,
        AVG (Avg_Close)
    FROM bitcoin
    GROUP BY YEAR(Date), MONTH(Date)
    ORDER BY Year, Month
""")
query_2.write.csv("month_over_month.csv", header=True, mode="overwrite")
query_2.show()
```

Year	Month	avg(Avg_Close)
2012	1	6.232117250278735
2012	2	5.234546695402287
2012	3	4.954453181003575
2012	4	4.980315277777753
2012	5	5.041769041218648
2012	6	5.050802214814821

3. Find the Top5 days with Highest Price Volatility –

```
3. Find the Top 5 Days with the Highest Price Volatility

query_3 = spark.sql("""
    SELECT
        Date,
        (Avg_High - Avg_Low) AS Volatility
    FROM bitcoin
    ORDER BY Volatility DESC
    LIMIT 5
""")
query_3.write.csv("highest_volatility.csv", header=True, mode="overwrite")
query_3.show()
```

Date	Volatility
2021-02-23	209.6799652778791
2021-05-19	207.7766736110425
2021-01-11	196.13825000001088
2021-05-20	177.53968055564474
2024-08-05	162.9416666666657

4. Analyze average close price for the Signal –

4. Analyze the Average Close Price by Signal (Buy or Sell)

```
[ ] query_4 = spark.sql("""
    SELECT signal, AVG(Avg_Close) as Avg_Close
    FROM bitcoin
    GROUP BY signal
    ORDER BY signal
""")
query_4.write.csv("avg_close_by_signal.csv", header=True, mode="overwrite")
query_4.show()
```

signal	Avg_Close
0.0	14522.95077594446
1.0	14641.298888375683

5. Determine the average high and low prices for each month:

5. Determine Average High and Low Prices for Each Month

```
[ ] query_5 = spark.sql("""
    SELECT YEAR(Date) AS Year, MONTH(Date) AS Month, AVG(Avg_High) AS Avg_High, AVG(Avg_Low) AS Avg_Low
    FROM bitcoin
    GROUP BY YEAR(Date), MONTH(Date)
    ORDER BY Year, Month
""")
query_5.write.csv("avg_high_low_by_month.csv", header=True, mode="overwrite")
query_5.show()
```

Year	Month	Avg_High	Avg_Low
2012	1	6.232149060314576	6.232061918737517
2012	2	5.234590038314165	5.234526340996157
2012	3	4.95449126344085	4.954436603942641
2012	4	4.980339351851827	4.980292129629603
2012	5	5.041785394265243	5.041759856630836
2012	6	5.9598763888888975	5.959721064814821
2012	7	7.778474238351243	7.778117831541208
2012	8	10.921000224014344	10.919631496415777
2012	9	11.398095138888866	11.397714814814794
2012	10	11.63096841397852	11.630384184587841
2012	11	11.321442673107857	11.32105107689208
2012	12	13.146929211469573	13.146575044802903
2013	1	15.261499327957022	15.260730958781396
2013	2	25.708068700396833	25.706108878968262
2013	3	56.72128494623658	56.6964872311828
2013	4	127.38757800925929	127.01545925925932
2013	5	117.93066151423689	117.7400828853046

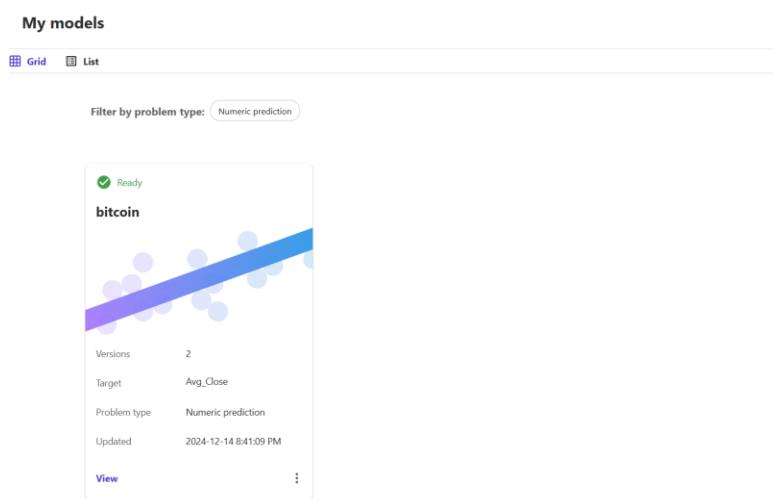
## Machine Learning with AWS Sage Maker Autopilot:

1. To automate the model training and evaluation, I used **AWS SageMaker Autopilot** with the following steps:
  - a. Upload Processed Data: Uploaded the day-level aggregated data to S3. *[code provided in the notebook]*
2. AWS Sage maker was pretty easy and straightforward to use.

Some of the difficulties that I particularly faced were –

- a. Making sure the column names adhered to naming convention *[no capital letter, no % marks, or any special characters, long wait times for the model to train and get predictions etc.]*
3. Here are some of the screenshots of the pipeline.

### Experiment:



## Information about the dataset for training:

My models / bitcoin / version 2

Select Build Analyze Predict Deploy + Create new version

**Select a column to predict**

Choose the target column. The model that you build predicts values for the column that you select.

Target column: Avg\_Close

Value distribution: A histogram showing the distribution of Avg\_Close values, ranging from 8.36 to 68729.12.

**Model type**

SageMaker Canvas automatically recommends the appropriate model type for your analysis.

Numeric prediction: For the Avg.Close, your model predicts numeric values.

Configure model

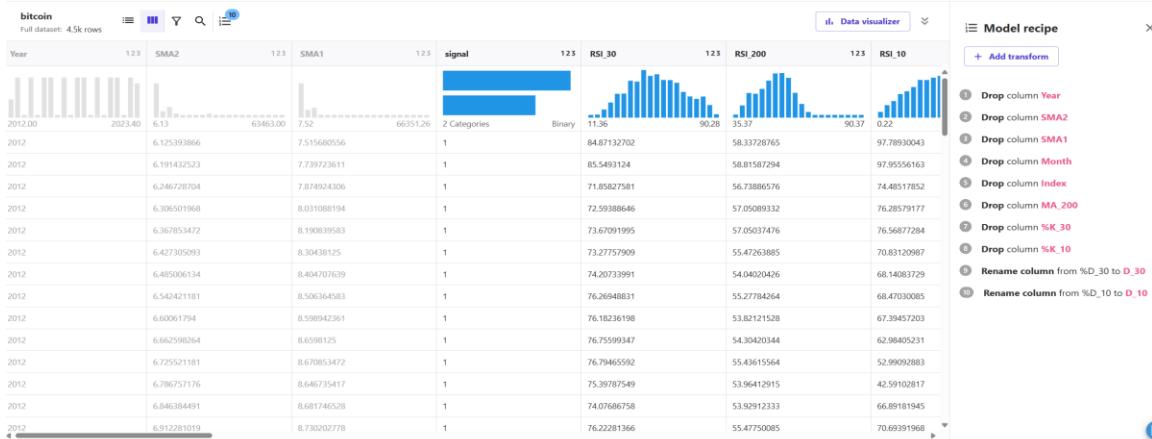
Standard build: Preview model

bitcoin: Full dataset: 4.5k rows

Data visualizer

Column name	Data type	Feature type	Missing	Mismatched	Unique	Mode	
D_30	Numeric	-	0.00% (0)	0.00% (0)	4,495	0.01	
D_10	Numeric	-	0.00% (0)	0.00% (0)	4,494	9.42	
Avg_Volume	Numeric	-	0.00% (0)	0.00% (0)	4,490	0	
Avg_Open	Numeric	-	0.00% (0)	0.00% (0)	4,492	276.8	
Avg_Low	Numeric	-	0.00% (0)	0.00% (0)	4,492	276.8	
Avg_High	Numeric	-	0.00% (0)	0.00% (0)	4,492	276.8	
Avg_Close	Target	Numeric	-	0.00% (0)	0.00% (0)	4,492	276.8
%K_30	Numeric	-	0.00% (0)	0.00% (0)	4,490	100	
%K_10	Numeric	-	0.00% (0)	0.00% (0)	4,484	100	

## Dataset Summary Visualization:



## Modelling, Feature Importance and Performance Metrics:

Model status: Standard build

RMSE: 21.499

The model often predicts a value that is within +/- 21.499 of the actual value for Avg\_Close.

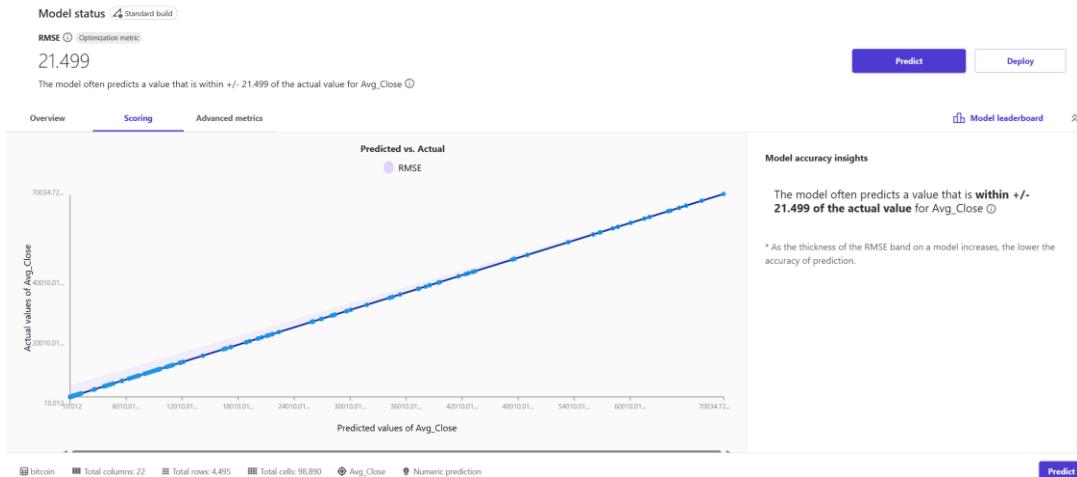
Predict Deploy

Overview Scoring Advanced metrics Model leaderboard

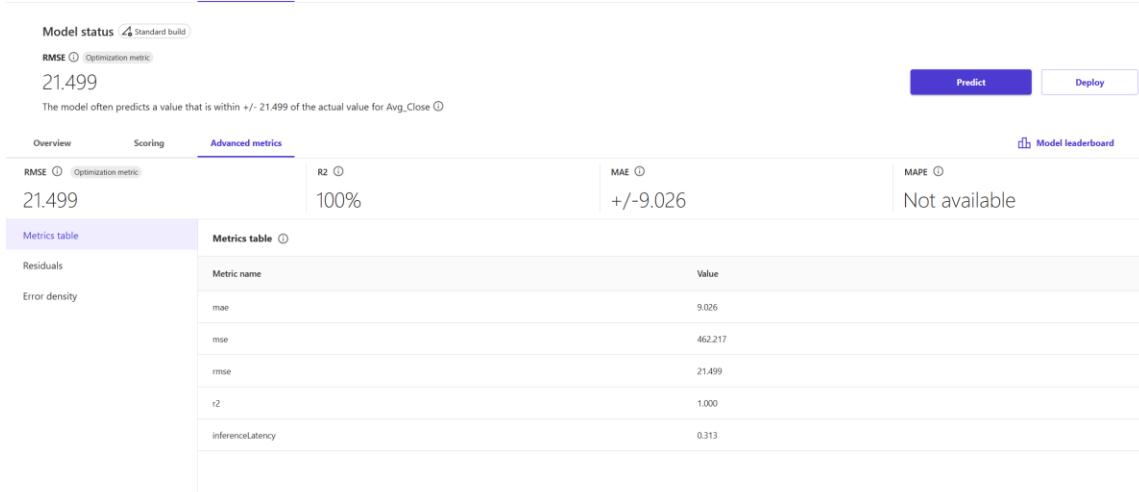
Column impact: Avg\_Low (32.967%), Avg\_Open (19.605%), Avg\_High (18.664%), EMA\_10 (7.357%), MA\_10 (6.425%), MA\_30 (4.233%), EMA\_30 (4.089%), EMA\_200 (4.089%).

Impact of Avg\_Low on prediction of Avg\_Close: A scatter plot showing the relationship between Avg\_Low (X-axis) and Impact on prediction of Avg\_Close (Y-axis). The X-axis ranges from -9414.96 to 70501.34, and the Y-axis ranges from -3585.03 to 14981.17. The data points show a strong positive correlation.

## Actual vs Predicted Values:



## Advanced Metrics for better comprehension of the model performance:



## Using the Model to predict on a sample Data:

The interface shows a "Predict" button at the top right. Below is a form for "Predict target values". It has a "Single prediction" tab selected. A table lists input columns (Index, Date, Avg\_Low, Avg\_Open, Avg\_High, Avg\_Volume, SMA1, SMA2) and their corresponding values (1, 07/18/12 12:00 am, 276.8, 276.8, 276.8, 0, 10.10209028, 10.01025961). To the right, a "Avg.Close Prediction" section shows a large bold value of 212.046. A legend indicates "New prediction" (blue bar) and "Last prediction" (grey bar). Below the prediction are two horizontal bars, both labeled 212.046.

Select Build Analyze Predict Deploy

Predict target values Single prediction

Modify values to predict Avg.Close in real time.

Filter columns

Column	Value
Index	1
Date	07/18/12 12:00 am
Avg_Low	276.8
Avg_Open	276.8
Avg_High	276.8
Avg_Volume	0
SMA1	10.10209028
SMA2	10.01025961

**Avg.Close Prediction**

**212.046**

New prediction Last prediction

212.046

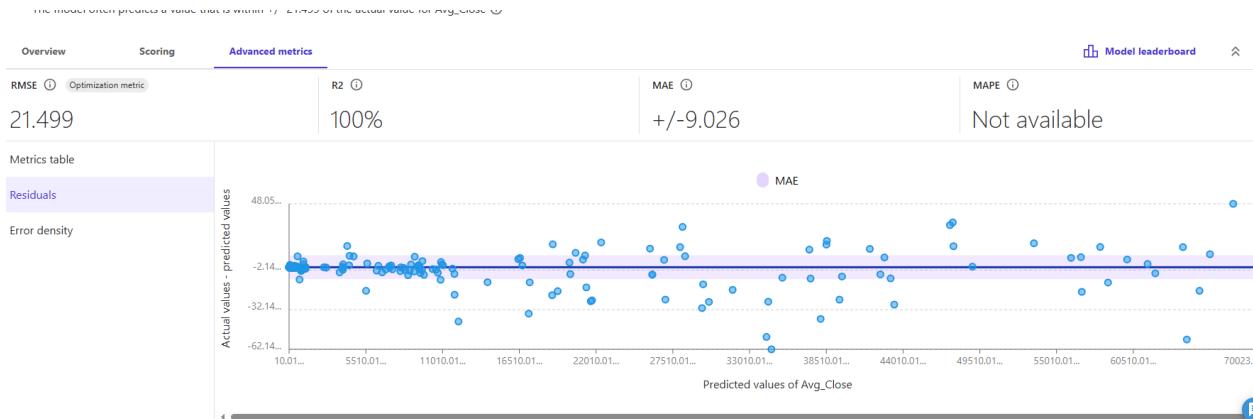
212.046

## Exploring Model Leaderboard:

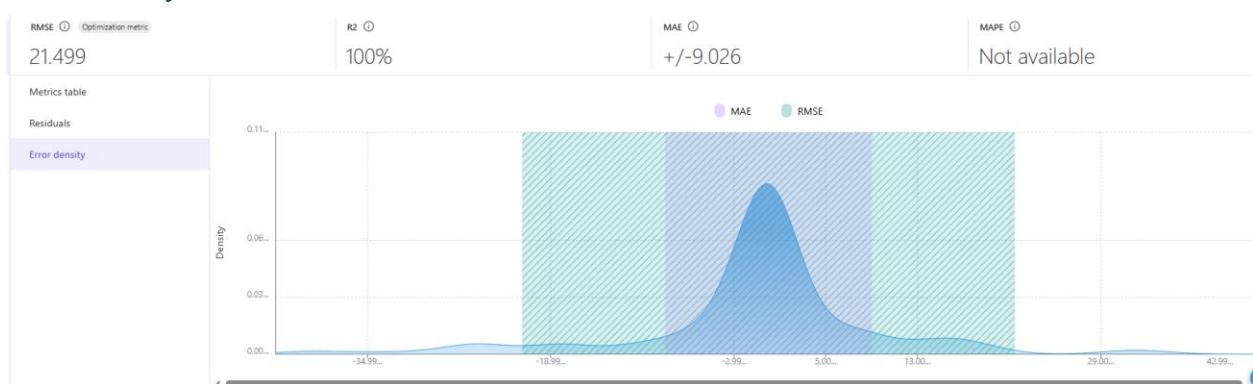
My models > bitcoin > Version 2

Model name ↓	RMSE	Optimization	MAE	MSE	R2	Inference latency (seconds)
FULL-t1503561425936Canvas1734224041990	21.499	Default model	9.026	462.217	100.000%	0.313
FULL-t9503561425936Canvas1734224041990	25.732		10.871	662.145	100.000%	0.190
FULL-t8503561425936Canvas1734224041990	23.969		10.166	574.531	100.000%	0.203
FULL-t7503561425936Canvas1734224041990	23.969		10.166	574.531	100.000%	0.219
FULL-t6503561425936Canvas1734224041990	65.751		30.576	4323.188	99.999%	0.103
FULL-t5503561425936Canvas1734224041990	65.751		30.576	4323.191	99.999%	0.102
FULL-t4503561425936Canvas1734224041990	23.969		10.166	574.532	100.000%	0.205
FULL-t3503561425936Canvas1734224041990	65.655		30.865	4310.609	99.999%	0.125
FULL-t2503561425936Canvas1734224041990	23.969		10.166	574.531	100.000%	0.221
FULL-t10503561425936Canvas1734224041990	25.732		10.871	662.145	100.000%	0.184

## Residuals:



## Error Density:



Explanation for the above pictures:

*Metrics (RMSE-Optimized Model):*

RMSE: 21.499

MAE: 9.026

MSE: 462.217

R<sup>2</sup> : 100%

Inference Latency: 0.313

*Model Leaderboard:*

Model Name	RMSE	MAE	MSE	R <sup>2</sup>	Latency (s)
FULL... (Default Mode)	21.499	9.026	462.217	100%	0.313
Optimized Version 2	25.732	10.871	662.145	100%	0.190
FULL... (Other versions)	Varying ~23-30	~10.166	574.531-4321.19	~100%	0.102-0.313

It shows multiple models with almost very similar RMSE.

The residuals and predicted vs actual plot display perfect almost perfect diagonal showing that the model is reliable.

The error range is also concentrated within a narrow band, aligning with the RMSE and MAE value.

Coming to bias, I think it's very important to understand the distribution of the 0s and 1s signals that essentially tell you whether to buy or sell based on 30, 10, 5 day moving average. If the signal is unbalanced, then the model might overfit on the past trends and fail to capture the actual distribution. Maybe perhaps, a better model that is fit for such use cases should be used.

Since the dataset is not of an individual person, company, university, and is open source, there is no risk of exposing any personal or proprietary information. Additionally, all the columns are anonymized, so aggregation or usage is no problem.

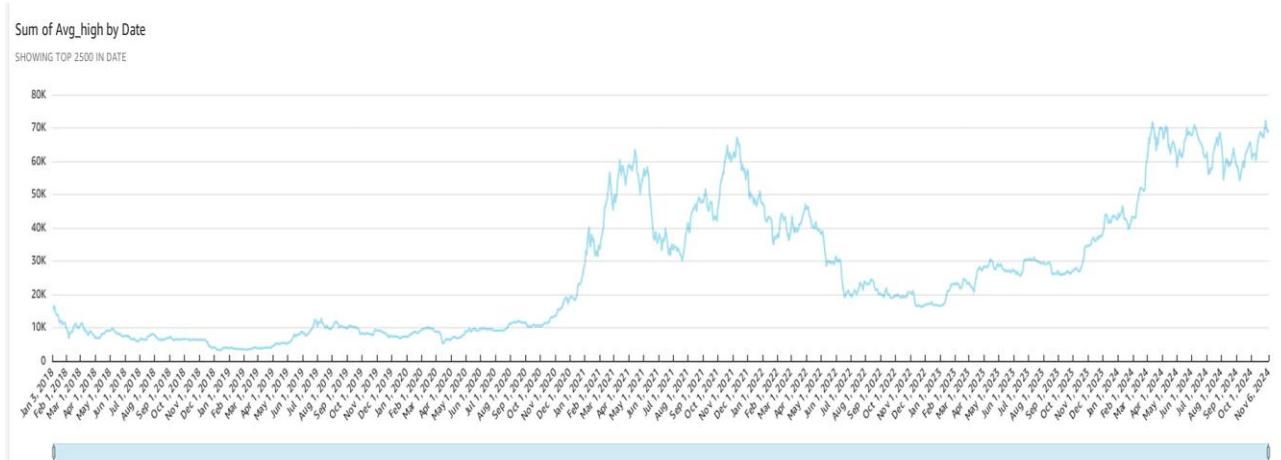
One of the things about ethical responsibility is that the model output should not be manipulated at no cost and should be avoided for overreliance. The model's predictions are only recommendations, not guaranteed and it should inherently be used like that.

## Visualizations –

1. The only major issue that I faced with QuickSight was creating a manifest file through which I could load the data from the s3 bucket onto memory in QuickSight. Other than that, everything else was pretty straightforward and here is the viz that I created along with the explanations behind creating them.

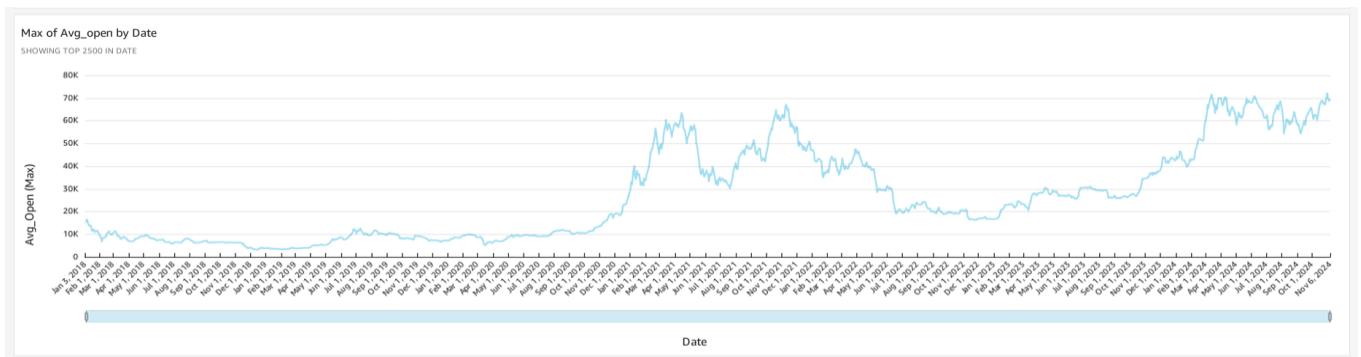
### 1. Sum of Average High Prices vs Date:

- Analyzes price peaks and periods of high market activity.
- Insights: Peak activity between **Jan 2021 to Jan 2022** followed by a slowdown and regained momentum in February 2023.



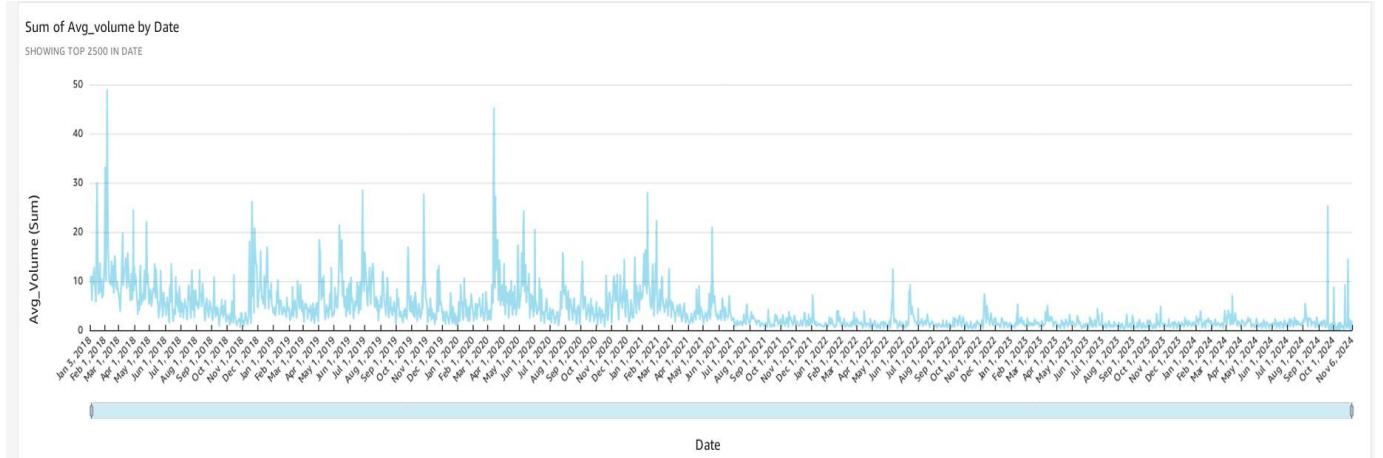
### 2. Maximum of Average Open Price vs Date

- Highlights trends in opening prices, indicating bullish days and demand patterns.
- Help identify optimal entry points for trading strategies.



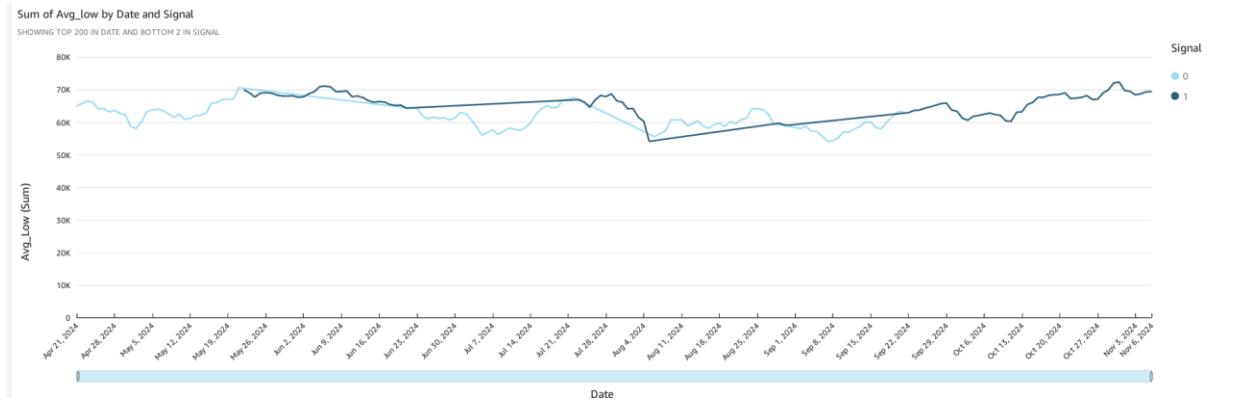
### 3. Sum of Average Volume of Bitcoins Traded vs Date: -

- Tracks volume spikes to identify buy/sell events and market movements.
- Insights: High volume confirms the validity of price breakouts.



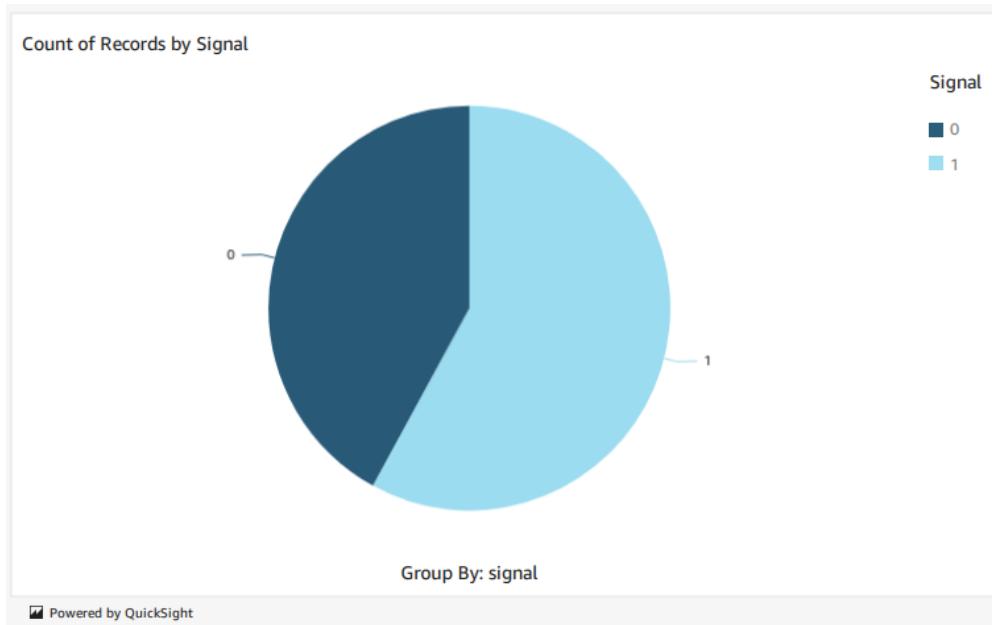
### 4. Sum of Average Low price by date and Signal.

- Analyzes the relationship between low prices and trading signals (0 = Sell, 1 = Buy).
- Helps validate trading strategies by identifying undervalued periods.



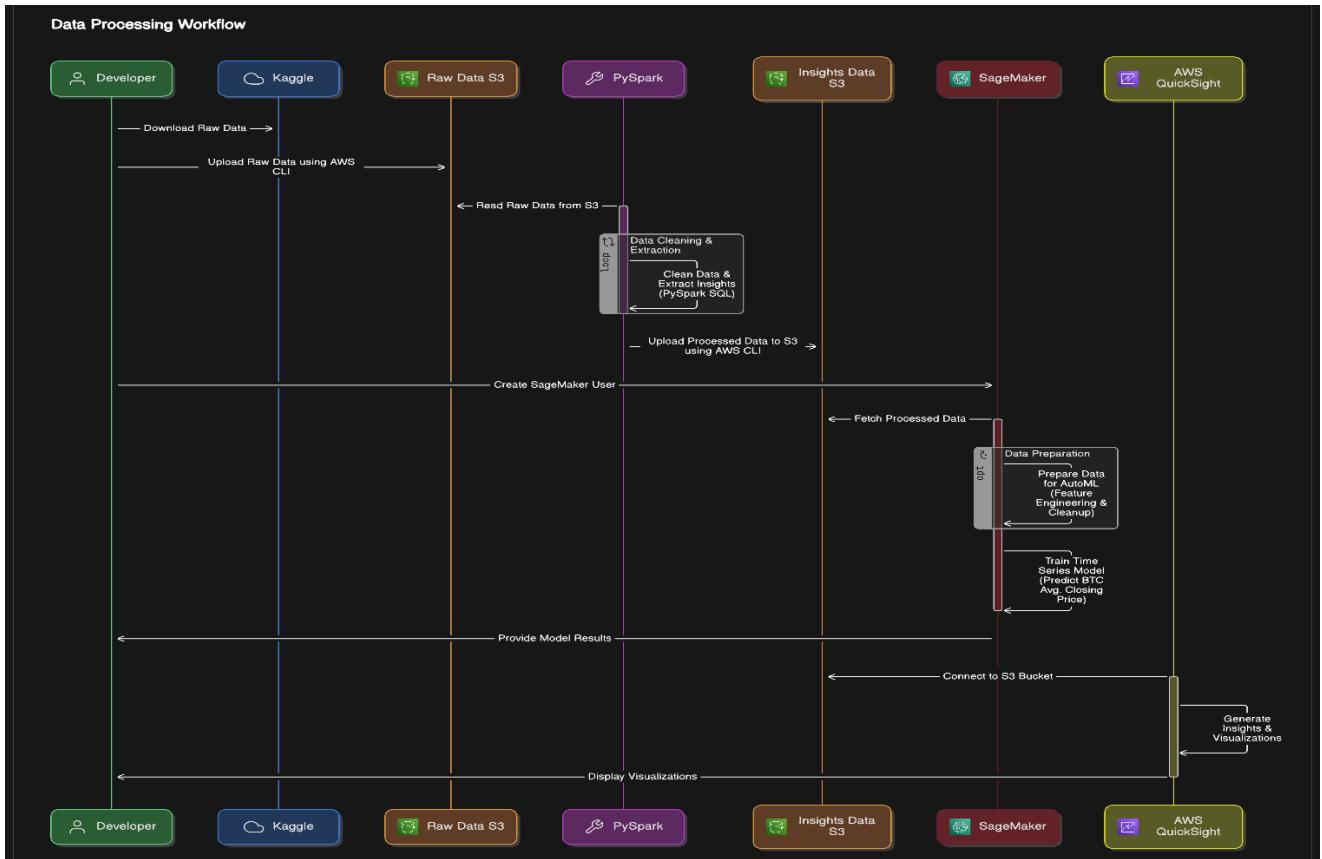
## 5. Count of records by Signal

- Shows the distribution of buy (1) and sell (0) signals.
- Insights: Imbalanced signals could indicate dataset bias, requiring preprocessing like oversampling or under sampling.



**Note:** You can view the entire Visualization rendered as pdf, provided as an attachment with this report.

## Architecture Diagram:



## Summary of the Workflow:

- Download Data from Kaggle → Upload Raw Data to S3
- Read Data from S3 → Process with PySpark Locally
- Upload Processed Data to S3 → Connect S3 to SageMaker
- Prepare Data in SageMaker → Train Time-Series Model in AutoML
- Connect S3 to QuickSight → Generate Visualizations in QuickSight

## Conclusion:

In summary, this mini project mainly tried to demonstrate how to leverage a comprehensive big data pipeline—from raw data ingestion and preprocessing with PySpark, to model building with AWS SageMaker Autopilot, and finally visualization in QuickSight—to generate actionable insights into Bitcoin's closing prices.

Exploring and transforming the dataset, using advanced machine learning techniques to predict future values, and validating the results through intuitive visual analytics all on cloud tells you how capable and advanced the cloud infra has turned into. Yes, there are challenges in configuring modules on clouds to handle huge datasets, but what do not?