ITD 105 - BIG DATA ANALYTICS

BITCOIN HISTORICAL DATA

ANALYSIS ON HIGH AND LOW PRICES OF BITCOIN

SEANNE CAÑETE

Introduction

Bitcoin, the first decentralized cryptocurrency, has revolutionized the world of finance since its inception in 2009. As a digital currency, Bitcoin operates without a central authority or intermediaries, utilizing a technology called blockchain to secure transactions and manage the issuance of new units. Its decentralized nature, scarcity, and pseudonymous transactions have sparked interest among investors, technologists, and economists alike.

One of the key aspects of Bitcoin's appeal is its price volatility, with its value often experiencing significant fluctuations over short periods. This volatility has made Bitcoin both a speculative asset and a subject of intense market analysis.

In this analysis, we will focus on these objectives that will be discussed below.

Objectives:

- Develop a comprehensive analysis of Bitcoin price data from 2019, including visualization of trends and correlations between key variables (Open, High, Low, Close, Weighted Price).
- Calculate statistical measures (mean, median, standard deviation) for numerical columns to understand the data distribution.
- Perform linear regression analysis to predict future price movements based on historical data.
- Compare the results of the analysis with those obtained using Weka to validate the findings and ensure consistency across platforms.

Importance:

Through this analysis, we seek to deepen our understanding of Bitcoin's market dynamics and explore the implications for investors, traders, and the broader financial ecosystem.

The Dataset

Describing the Dataset

The provided CSV files contain minute-by-minute updates of Bitcoin price data from select exchanges for the period of January 2012 to December 2021. The data includes Open, High, Low, Close prices, Volume in BTC and indicated currency, and weighted Bitcoin price. Timestamps are in Unix time format. In total, it has 8 Columns. Moreover, the data was sourced from Kaggle.

The focus of this analysis will be the period from 2019 to 2021 due to the data size being too large to be analyzed, and the current technology of mine can't manage to process the entire dataset efficiently. In addition, to use the other years' data, the file was spliced into every year.

It's important to note that timestamps without any trades or activity are filled with NaNs, and missing timestamps or jumps in data may be due to technical issues with the exchange or its API. Efforts have been made to deduplicate entries and verify the data's correctness and completeness, but users should exercise caution and trust the data at their own risk.

These are the columns and their descriptions:

- **Timestamp:** The Unix timestamp representing the specific time interval for which the data is recorded. Each row in the dataset corresponds to a minute-by-minute update of Bitcoin price data, with the timestamp indicating the start of the minute interval.
- **Open:** The price of Bitcoin at the beginning of the minute interval.
- **High:** The highest price of Bitcoin reached during the minute interval.
- **Low:** The lowest price of Bitcoin reached during the minute interval.
- **Close:** The price of Bitcoin at the end of the minute interval.
- **Volume (BTC):** The amount of Bitcoin traded during the minute interval, measured in BTC.
- **Volume (Currency):** The total value of Bitcoin traded during the minute interval, measured in the indicated currency.

- **Weighted Price:** The volume-weighted average price of Bitcoin during the minute interval, calculated as the sum of (Price * Volume) / Total Volume.

Predicting

In this dataset, we have the potential to predict various aspects related to Bitcoin price movements and market trends. Here are some predictions we could explore:

Short-Term Price Movements:

Using techniques such as time series forecasting, you could predict the short-term (e.g., daily or weekly) movements of Bitcoin prices based on historical price data.

Columns: 'Open', 'High', 'Low', 'Close'

Technique: Time series forecasting

Description: Predicting the short-term (e.g., daily or weekly) movements of Bitcoin prices based on historical price data. This can help traders make informed decisions about when to buy or sell Bitcoin.

Volatility:

You could predict the future volatility of Bitcoin prices, which could be useful for risk management and trading strategies.

Columns: 'High', 'Low'

Technique: Statistical modeling or machine learning

Description: Predicting the future volatility of Bitcoin prices, which can be useful for risk management and trading strategies. Higher volatility may indicate greater potential for price fluctuations.

Trading Volume:

Predicting future trading volume of Bitcoin could help in understanding market liquidity and investor interest.

Columns: 'Volume_(BTC)'

Technique: Time series forecasting or regression analysis

Description: Predicting future trading volume of Bitcoin can help in understanding market liquidity and investor interest. Higher trading volume may indicate greater market activity.

Trend Reversals:

Using trend analysis techniques, you could predict potential trend reversals in Bitcoin prices, which could be valuable for traders looking to capitalize on market shifts.

Columns: 'Close'

Technique: Trend analysis or machine learning

Description: Predicting potential trend reversals in Bitcoin prices can be valuable for traders looking to capitalize on market shifts. This can help traders identify opportunities to enter or exit trades.

It's important to note that while predictions can provide valuable insights, they are inherently uncertain and subject to various factors that may influence Bitcoin prices. Therefore, it's crucial to use appropriate models and methods and to consider the limitations and risks associated with predictions.

Benefits of Analyzing the Data

Analyzing the overall trend, volatility, trading volume, and future outlook of Bitcoin prices can significantly benefit the business process of trading Bitcoin in several ways:

Informed Decision Making

By understanding the price trend and volatility, traders can make more informed decisions about when to buy or sell Bitcoin. This can help maximize profits and minimize losses.

Risk Management:

Analyzing trading volume and volatility can help traders manage risk effectively. By understanding market liquidity and potential price fluctuations, traders can implement risk management strategies such as stop-loss orders to protect their investments.

Market Timing:

By analyzing historical data and future outlook, traders can better time their trades to take advantage of potential price movements. This can help traders enter and exit the market at optimal times, maximizing returns.

Strategy Development:

The analysis can help traders develop and refine their trading strategies. By understanding market dynamics and trends, traders can tailor their strategies to better suit current market conditions.

Competitive Advantage:

Traders who use data-driven analysis are better equipped to compete in the market. By staying informed about market trends and developments, traders can stay ahead of the competition and capitalize on opportunities.

Compliance:

For businesses involved in trading Bitcoin, staying compliant with regulations is crucial. Analyzing market data can help businesses ensure they are following relevant regulations and mitigate regulatory risks.

Overall, analyzing Bitcoin price data can help businesses trading Bitcoin make more informed decisions, manage risk effectively, and stay competitive in the market.

Data Pre-Processing

In the pre-processing phase, I cleaned the data using a jupyter notebook and also using a python module called pandas. The process was composed of removing data with rows composed of N/As and also removing duplicate rows as we don't want data of the same timestamp as it will conflict. In addition, the data is already in numerical form so there is no need to convert strings into nominal values. Moreover, the timestamp is in unicode form and was processed into a readable date when it was displayed in html.

```python
import pandas as pd

# Load dataset
df = pd.read_csv('bigdata/dataset/bitcoin_cleaned_1.csv')

# Drop rows with null values
df.dropna(inplace=True)

# Removing duplicates
df.drop_duplicates(inplace=True)

# Data transformation (if necessary)

# Save cleaned data to a new file
df.to_csv('bigdata/dataset/bitcoin_cleaned_2.csv', index=False)

print(df.columns)
```

Python

Figure 1. Data Cleaning in Jupyter Notebook

Splicing the Data

To reduce the time of reading millions of columns for one CSV file, I decided to split the data into per year. Moreover, there are a total of 10 CSV files composed of years ranging from 2011 up to 2021. Moreover, I also splitted the data of 2019 to monthly data to be used for time series analysis.

To reduce the time for generating the charts, I decided to split the bitcoin historical data and count it by day using the average values of the day instead of by minute, composing 365 rows which is easier to do time series analysis and reduce the time for it to be completed.

Handling Missing Values and Outliers

The missing values were removed during the initial process of pre-processing by removing all rows with null values which clears more than 60% of the data during 2011 and 2012. Due to missing values during the time of 2011 and 2012, I will not be able to include it in my analysis as there are missing values between these dates.

Python Data Analysis for checking relationships between data

To implement the analysis using Python, we will first load the dataset into a pandas DataFrame and then use matplotlib to visualize the data. We can calculate various statistical measures such as mean, median, and standard deviation for the numerical columns.

The one I used during this analysis is using a linear regression to analyze the data and their correlation. I was able to compare it with the result of my linear regression in weka also, the results look quite similar.

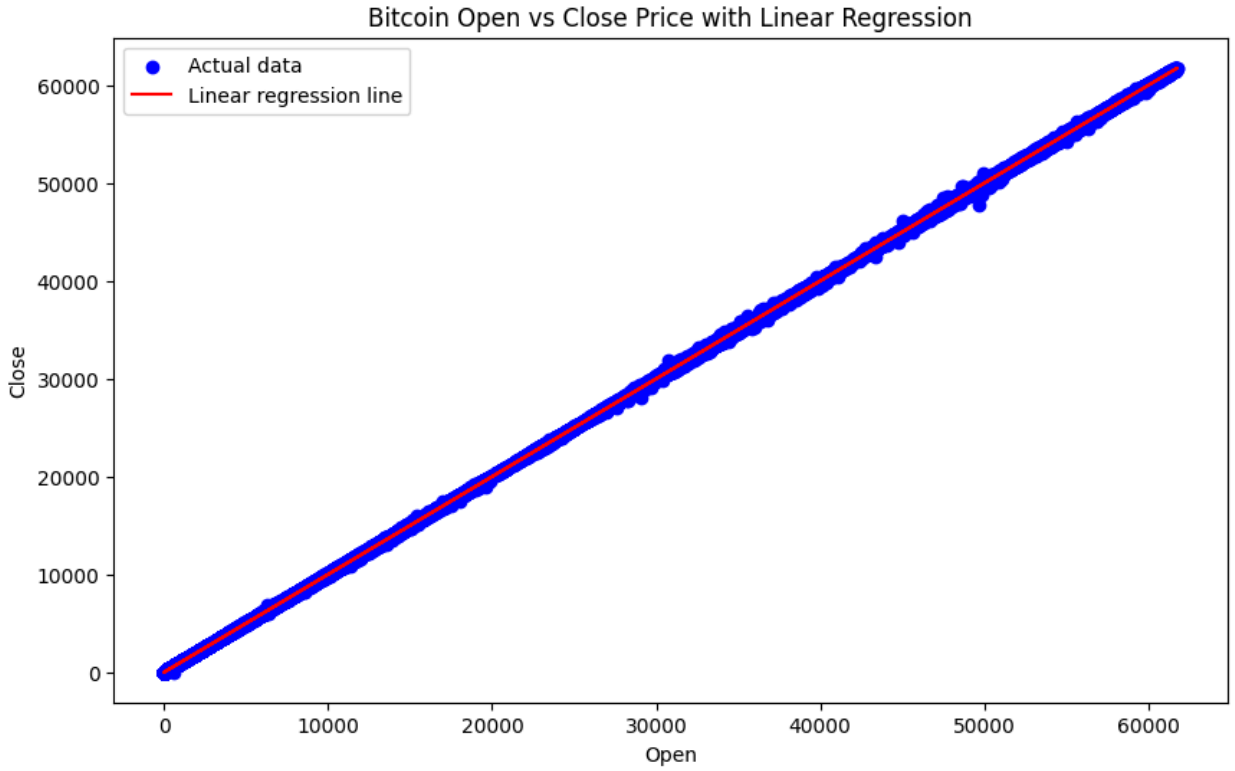


Figure 2. Linear regression using Python

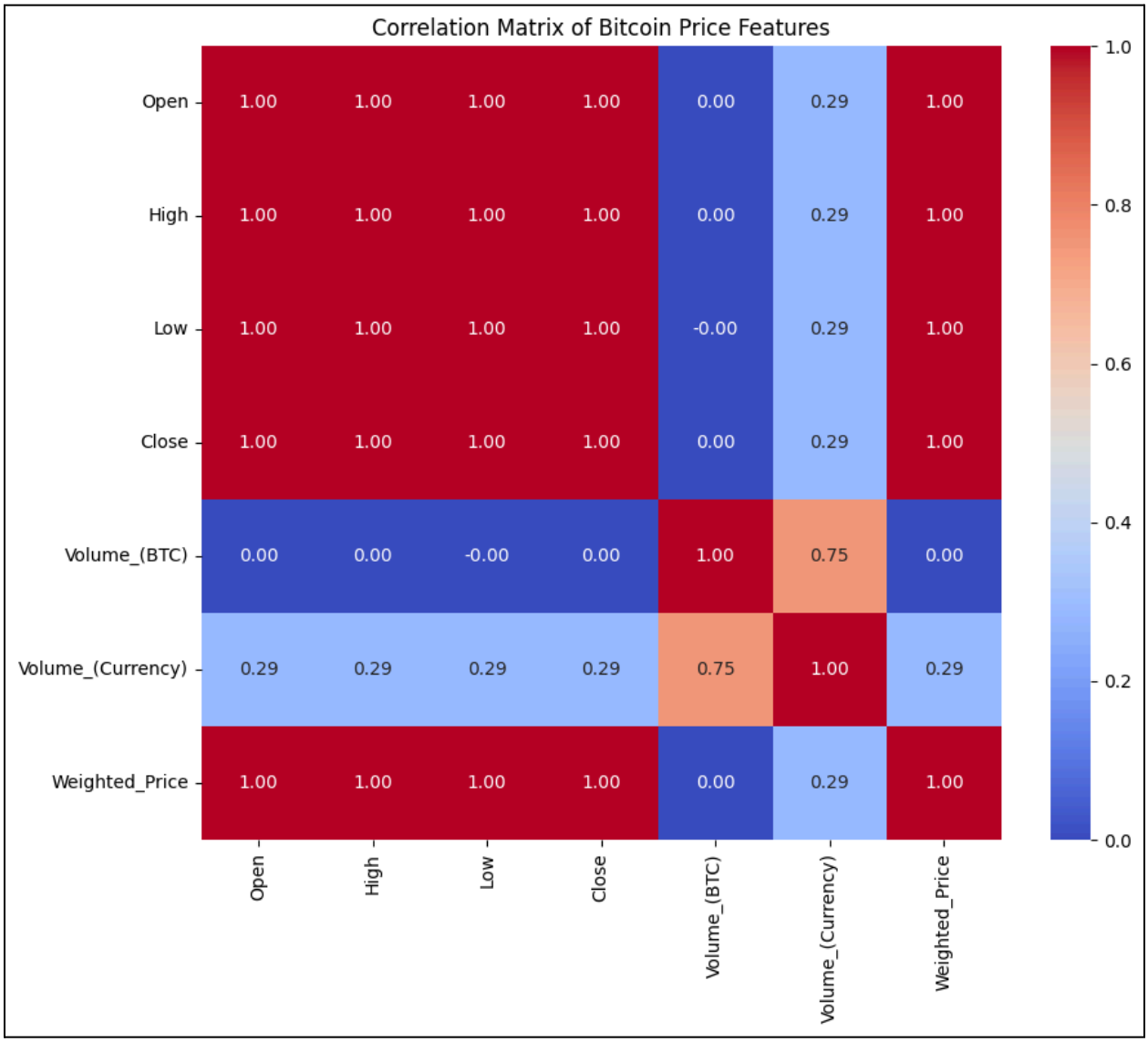


Figure 3. Correlation Matrix

The fact that 'Open', 'High', 'Low', 'Close', and 'Weighted_Price' columns have a correlation coefficient of 1.0 indicates a perfect positive linear relationship between these variables. In the context of Bitcoin price data, this perfect correlation is expected, as these columns are all related to the pricing of Bitcoin and are derived from the same underlying data.

Prediction Model Creation

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np
from joblib import dump

df = pd.read_csv('bigdata/dataset/bitcoin_cleaned_4.csv')

# Select columns to predict
columns_to_predict = ['Open', 'High', 'Low', 'Close', 'Volume_(BTC)', 'Volume_(Currency)', 'Weighted_Price']

for target_column in columns_to_predict:
    # Split data into features and target
    X = df[['Timestamp']]
    y = df[target_column]

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train linear regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Make predictions
    date_input = '2021-01-01'
    date_input_timestamp = pd.to_datetime(date_input).timestamp()
    prediction = model.predict(pd.DataFrame({'Timestamp': [date_input_timestamp]}))
    print(f'Predicted {target_column} value for {date_input}: {prediction[0]}')
    dump(model, f'linear_regression_model_{target_column}.pkl')
    # Evaluate model
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f'Mean Squared Error for {target_column}: {mse}')
```

✓ 10.3s

Figure 4. Model Creation

The trained linear regression models for each column in the dataset will be valuable for future development in predicting the values of those columns.

In this code snippet, we load a dataset containing Bitcoin price data and train a separate linear regression model for each column of interest, such as 'Open', 'High', 'Low', 'Close', 'Volume_(BTC)', 'Volume_(Currency)', and 'Weighted_Price'. For each target column, we split the data into training and testing sets, train the model using the training data, and then make predictions for a specific date ('2021-01-01' in this case). We save each trained model using the joblib.dump function with a filename that includes the target column name. Additionally, we evaluate each model using the Mean Squared Error (MSE) metric to assess

its performance on the test data. This approach allows us to create individual models for each column of interest, enabling us to predict future values based on a given date input.

Analysis

We will analyze the Bitcoin dataset using Python libraries such as pandas for data manipulation, matplotlib for data visualization, and possibly other libraries for specific analysis tasks.

The data is processed using pandas to read the columns and calculate various descriptive statistics. This includes the mean, median, mode, standard deviation, range, number of rows, maximum value, maximum date, minimum value, and minimum date, providing a comprehensive overview of the dataset's characteristics.

The substantial range between the minimum and maximum values for 'Open', 'High', 'Low', 'Close', and 'Weighted_Price' reflects the volatile nature of Bitcoin prices. Bitcoin's volatility is a well-known characteristic, often attributed to factors such as market demand, regulatory developments, macroeconomic trends, and investor sentiment. The high standard deviations further highlight the significant variability and potential for rapid price changes in the Bitcoin market. This volatility is a key aspect of Bitcoin's appeal to traders

and investors seeking opportunities for profit, but it also poses risks due to the potential for large and sudden price swings.

	Mean	Median	Mode	Std Deviation	Min	Max
Open	13230.10	9372.99	9600.00	12210.63	3334.00	61763.56
High	13239.07	9377.51	9700.00	12221.04	3346.12	61781.83
Low	13221.09	9369.00	9600.00	12200.00	3322.19	61673.55
Close	13230.11	9373.25	9700.00	12210.89	3334.00	61781.80
Volume_(BTC)	5.99	1.61	1.00	15.85	0.00	1098.35
Volume_(Currency)	79262.35	16562.14	5.85	230258.45	0.00	13900672.41
Weighted_Price	13230.17	9373.41	3975.00	12210.09	3335.26	61716.21

Figure 5. Descriptive statistics

System Development

The system is a web application developed using Django Framework, it is able to support running on mobile devices and desktop browsers. However, it is currently developed for desktop applications and running on mobile devices is for future development.

Architecture

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It follows the Model-View-Template (MVT) architectural pattern, which is a variation of the popular Model-View-Controller (MVC) pattern. In addition, HTML, CSS and Javascript were used for UI and frontend development. Moreover, Python and python modules such as seaborn, sklearn, matplotlib and pandas were used in the backend and analytics.

Input Processing and Analytics

CSV Reading and Input Processing

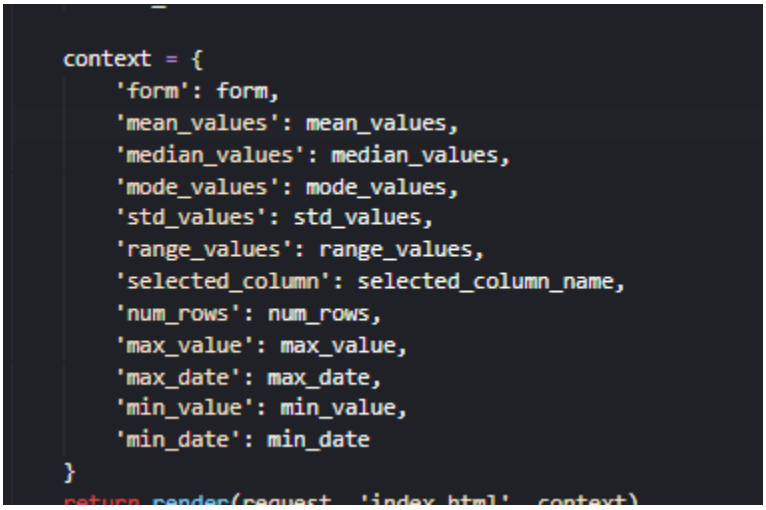
The CSV is inserted in the system using a file input available in the UI of the system. It is then processed using pandas to read the columns and get its mean, median, mode, standard deviation, range, number of rows, max value, max date, min value and min date. All of these are descriptive statistics of the dataset. In addition, The system also features comparison to previous csv data from the same selected column and displays differences.

I selected these descriptive statistics as it is more common in data analytics of crypto and also that it is applicable in this use case.

```
def index(request):
            if request.method == 'POST' and request.FILES.get('csv_file'):
                csv_file = request.FILES['csv_file'] #Receive CSV file
                df = pd.read_csv(csv_file) #  Read CSV File
                # Store DataFrame in session
                request.session['csv_data'] = df.to_dict(orient='records')
                columns = df.columns.tolist()
                messages.success(request, 'CSV file loaded successfully. Select a column.')


            elif 'csv_data' in request.session:
                # Load DataFrame from session if available
                df = pd.DataFrame(request.session['csv_data'])
                columns = df.columns.tolist()
            else:
                # If no file is uploaded and no session data, load a default CSV file
                csv_file_path = 'static/csv/fixed.csv'
                df = pd.read_csv(csv_file_path)

            if request.method == 'POST':
                columns = df.columns.tolist()
                selected_column = request.POST.get('column') #Select Column
                messages.success(request, f'Successfully selected column "{selected_column}". Displaying Data...')


                if selected_column in df.columns:
                    selected_columns = df[selected_column]
                    selected_column_name = selected_column
                    timestamp = df['Timestamp']  # Assuming 'Timestamp' is the column name for your timestamp data
                    mean_values = selected_columns.mean()
                    median_values = selected_columns.median()
                    mode_values = selected_columns.mode().iloc[0]  # Mode can have multiple values, so take the first
                    std_values = selected_columns.std()
                    range_values = selected_columns.max() - selected_columns.min()
                    num_rows = len(selected_column)  # Number of rows in the selected column
                    max_value, max_date, min_value, min_date = get_max_min_dates(df, selected_column)

                else:
```

Figure 6. CSV Reading and Input Processing

After processing the dataset, it is stored in variables to be displayed in the system user interface as context.



```
context = {
    'form': form,
    'mean_values': mean_values,
    'median_values': median_values,
    'mode_values': mode_values,
    'std_values': std_values,
    'range_values': range_values,
    'selected_column': selected_column_name,
    'num_rows': num_rows,
    'max_value': max_value,
    'max_date': max_date,
    'min_value': min_value,
    'min_date': min_date
}
return render(request, 'index.html', context)
```

Figure 7. Passing the context onto the HTML

Clearing of Values

The system has a button to clear the data presented in the user interface. It only clears the variables where the data are assigned and the actual CSV files were not involved during this clearing process.

```python
def index(request):
    if request.method == 'POST':
        if 'clear_values' in request.POST:
            # Clear the values
            form = ColumnSelectionForm()
            selected_columns = None
            mean_values = None
            median_values = None
            mode_values = None
            std_values = None
            range_values = None
            max_value = None
            max_date = None
            min_value = None
            min_date = None
        else:
            form = ColumnSelectionForm(request.POST)
            if form.is_valid():
                selected_column = form.cleaned_data['columns'].strip()  # Get the selected column name
                if 'csv_file' in request.FILES:
                    csv_file = request.FILES['csv_file']
                else:
                    # Select a default CSV file
                    default_csv_path = './path/to/default.csv'
                    csv_file = open(default_csv_path, 'rb')
                df = pd.read_csv(csv_file)
                print("Selected column:", selected_column)
                print("Columns in DataFrame:", df.columns)
```

Figure 8. Clearing of Data Function

Prediction

```python
def prediction(request):
    columns_to_predict = ['Open', 'High', 'Low', 'Close', 'Volume_(BTC)', 'Volume_(Currency)', 'Weighted_Price']
    if request.method == 'POST':
        date_input = request.POST.get('date_input')  # Assuming the input field in the form is named 'date_input'

        # Convert date input to Unix timestamp in seconds
        date_input_timestamp = pd.to_datetime(date_input).timestamp()

        predictions = {}
        for target_column in columns_to_predict:
            # Make predictions
            model = joblib.load(f'dataset/models/linear_regression_model_{target_column}.pkl')
            prediction = model.predict(pd.DataFrame({'Timestamp': [date_input_timestamp]}))
            predictions[target_column] = prediction[0]

        # Render the predictions in a template
        return render(request, 'prediction.html', {'predictions': predictions, 'date': date_input})


    # Render the form to input date
    return render(request, 'prediction.html')
```

Figure 9. Prediction Code Snippet

The prediction function is designed to handle a POST request containing a date input from a form. Upon receiving the date input, the function converts it to a Unix timestamp and uses it to make predictions for various columns ('Open', 'High', 'Low', 'Close', 'Volume_(BTC)', 'Volume_(Currency)', 'Weighted_Price') using pre-trained linear regression models loaded from files. Each model is specific to a target column, and the predicted values are stored in a dictionary. Finally, the function renders a template ('prediction.html') with the predicted values displayed alongside the input date. If the function receives a GET request, it simply renders the form for inputting the date.

Visualization

```python
def visualization(request):
    # default_csv_path = './dataset/csv/2015_split_by_month/bitcoin_2015_01.csv'
    default_csv_path = './dataset/csv/bitcoin_2015_daily_average.csv'
    # Check if file exists
    if not os.path.exists(default_csv_path):
        return render(request, 'visual.html', {'error': 'CSV file not found.'})

    try:
        df = pd.read_csv(default_csv_path)
    except Exception as e:
        return render(request, 'visual.html', {'error': f'Error reading CSV file: {e}'})

    # Ensure 'Timestamp' column exists
    if 'Timestamp' in df.columns:
        # Check if the 'Timestamp' values are already in datetime format
        try:
            df['Timestamp'] = pd.to_datetime(df['Timestamp'])
        except ValueError:
            return render(request, 'visual.html', {'error': 'Timestamp column is not in a valid datetime format.'})

        dates = df['Timestamp'].dt.strftime('%Y-%m-%d').tolist()
        column_data = {col: df[col].tolist() for col in df.columns if col != 'Timestamp'}

        return render(request, 'visual.html', {
            'column_data_json': json.dumps(column_data),
            'dates_json': json.dumps(dates),
            'column_data': column_data,
            'dates': dates
        })
    else:
        return render(request, 'visual.html', {'error': 'No "Timestamp" column found in CSV file.'})
```

Figure 10. Visualization Code Snippet

The provided Django view function visualization reads a CSV file containing Bitcoin data and processes it for visualization. It first checks if the CSV file exists and can be read without errors. It then ensures the 'Timestamp' column is present and converts it to a datetime format. The function extracts dates and other column data from the DataFrame and prepares them as JSON objects. These JSON objects are then passed to the visual.html template, which renders a grid of line charts using Chart.js, each chart representing a different data column over time. If any errors occur during these steps, appropriate error messages are displayed. The template uses Chart.js to dynamically create and display the charts, ensuring the data is visualized effectively.

User Interface

The interface is composed of an index page or the home page, this is where you can locate the descriptive statistics analysis of the CSV file which are displayed on the cards. In addition, there is also a page for about pages which covers the dataset information and sources. Lastly, there are plans for data visualization and prediction and currently the available pages are their initial views and functionality.

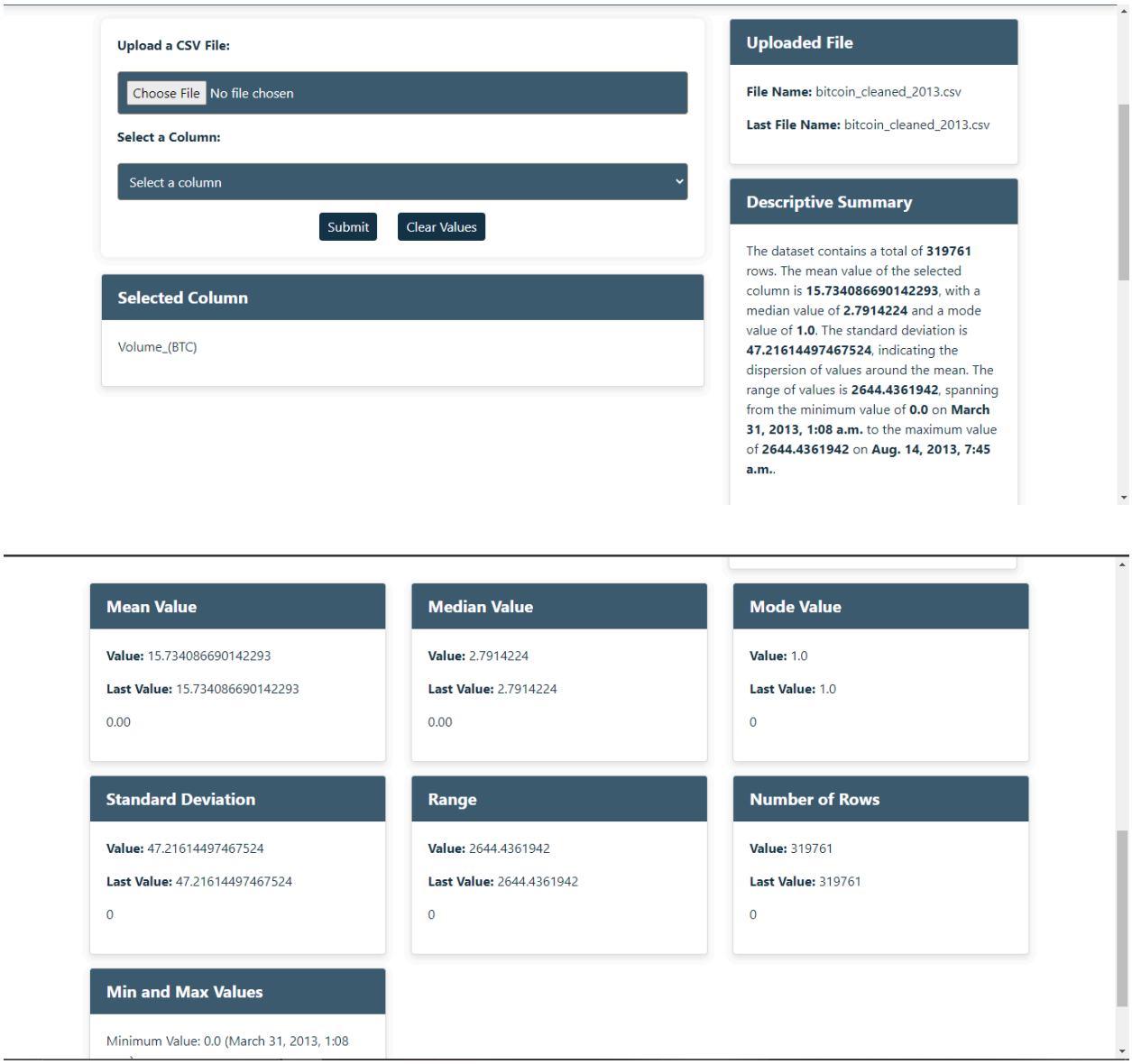


Figure 11. Homepage

their own risk.

Data Columns and Descriptions

Timestamp: The Unix timestamp representing the specific time interval for which the data is recorded. Each row in the dataset corresponds to a minute-by-minute update of Bitcoin price data, with the timestamp indicating the start of the minute interval.

Open: The price of Bitcoin at the beginning of the minute interval.

High: The highest price of Bitcoin reached during the minute interval.

Low: The lowest price of Bitcoin reached during the minute interval.

Close: The price of Bitcoin at the end of the minute interval.

Volume (BTC): The amount of Bitcoin traded during the minute interval, measured in BTC.

Volume (Currency): The total value of Bitcoin traded during the minute interval, measured in the indicated currency.

Weighted Price: The volume-weighted average price of Bitcoin during the minute interval, calculated as the sum of (Price * Volume) / Total Volume.

The data was sourced from Kaggle.

Figure 12. About Page

Figure 13. Initial Visualization

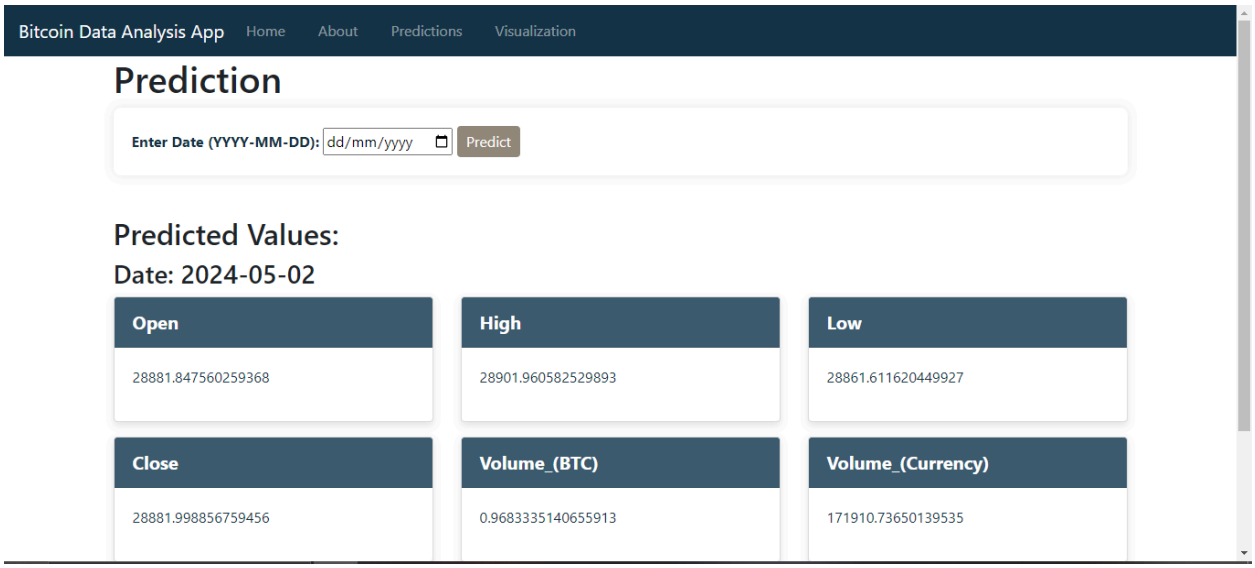


Figure 14. Initial Prediction Page

Data Flow Diagrams

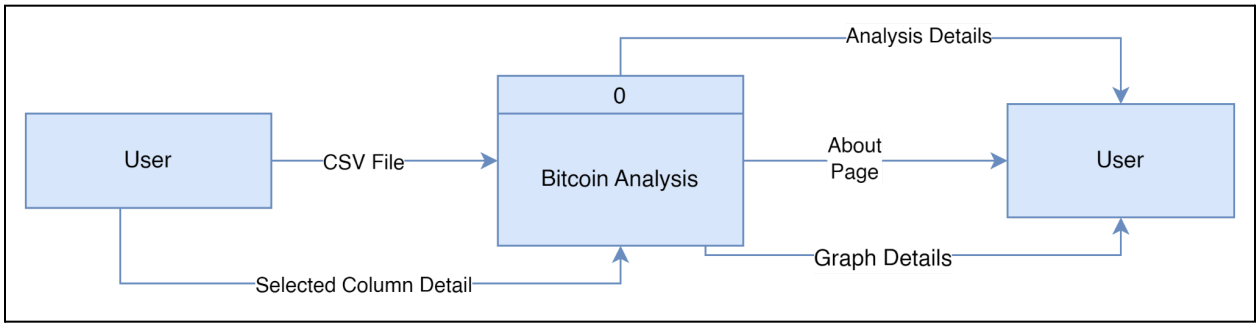


Figure 15. Level 0 Diagram

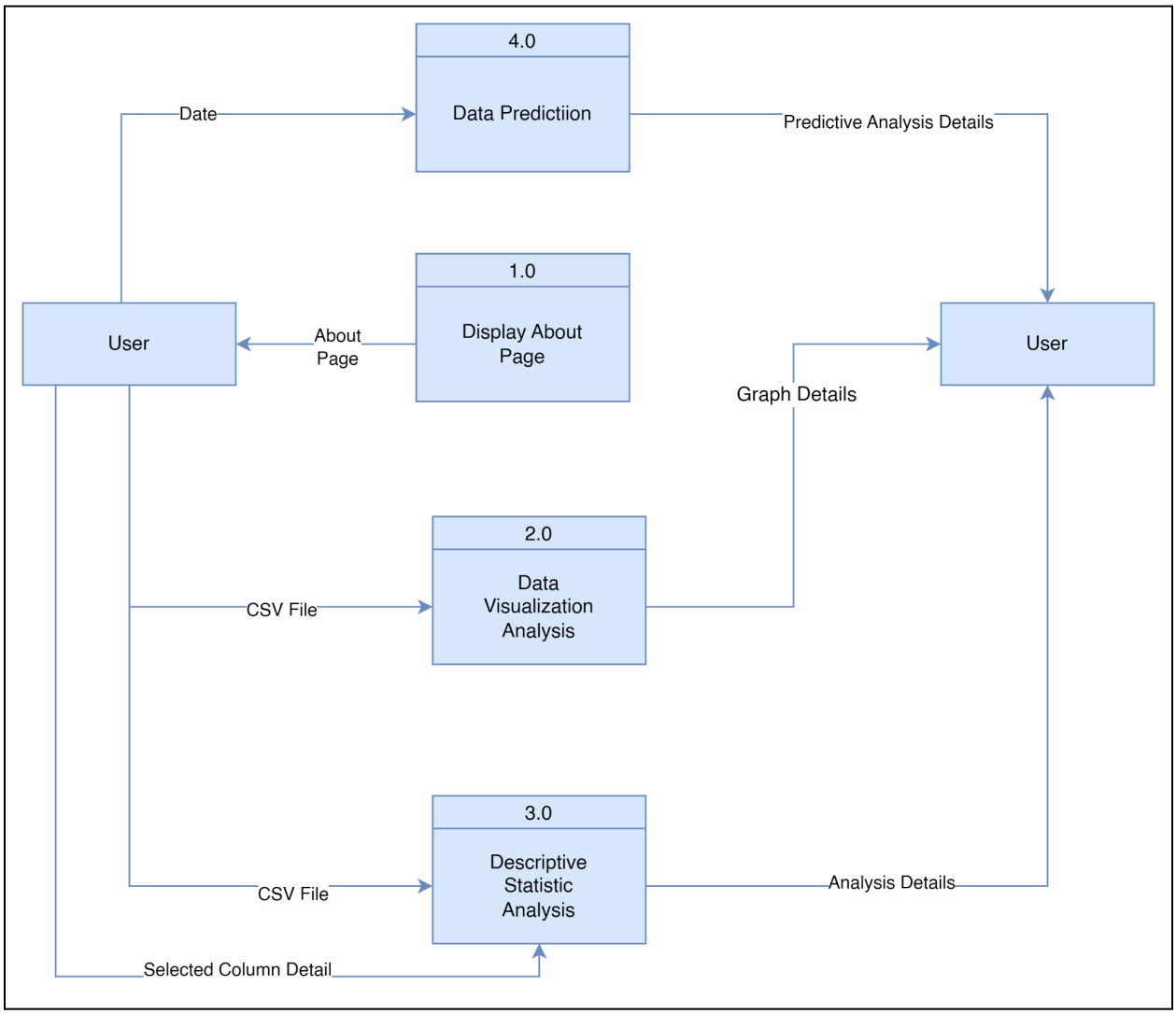


Figure 16. Level 1 Diagram

The system currently has 4 different pages, it has a about page which discusses the columns involved in this analysis, a data visualization analysis page which renders graphs for the user to see, and a description statistics page which displays the descriptive statistics of a selected column. In addition, there is the prediction page where we can predict values based on the date inputted.

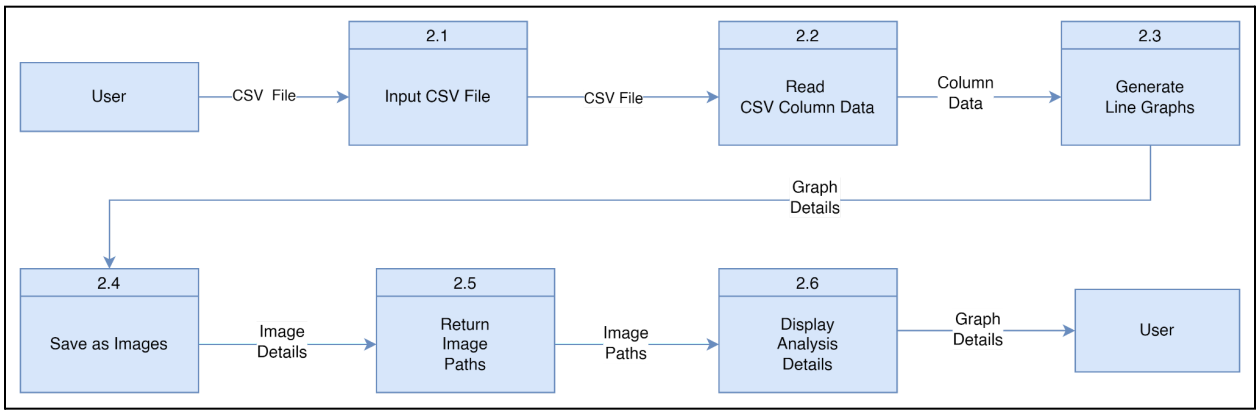


Figure 17. Level 2 - Data VIsualization Analysis

The process involves inputting a CSV file then the system will read it, it will use the column data gathered from the csv file to render bar graphs. After that, the image will save in the local storage of the host server, the path of the image will be returned to the HTML as context so that the images can be called. It is important to note that old images are overwritten during the save process.

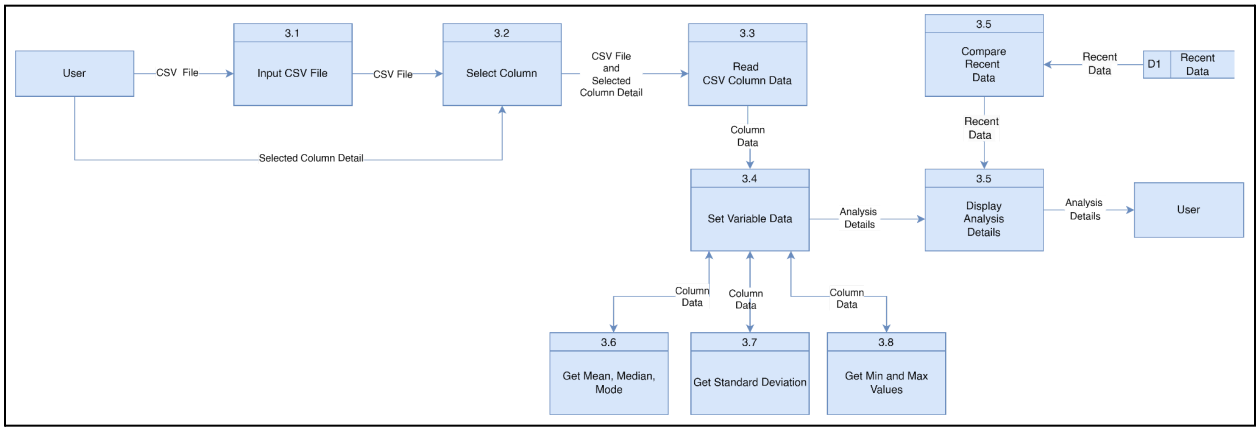


Figure 18. Level 2 - Descriptive Statistic Analysis

The process involves inputting a CSV file then the system will read it and selecting the column you want to see, it will use the column data gathered from the csv file to generate descriptive statistics such as mean, median, mode, standard deviation, min and max values. It will pass the analysis details to the HTML which will render it.

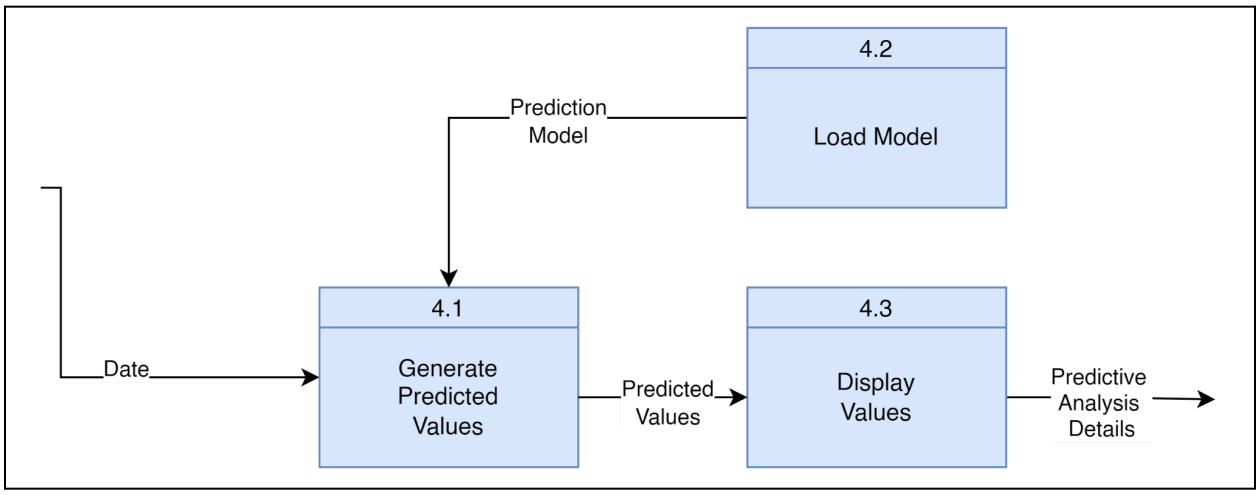


Figure 19. Level 2 - Predictive Statistic Analysis

The process involves inputting a date then the system will read it and generate predicted values for the selected date for all columns in the CSV.

Result and Conclusion

Summary of Project Outcomes

The initial findings from the analysis of Bitcoin price data from 2019 using Python and Weka are quite insightful.

The data was loaded into a pandas DataFrame, and matplotlib was used to visualize the data. Various statistical measures such as mean, median, and standard deviation were calculated for the numerical columns.

Linear regression was used to analyze the data and its correlation. The results were compared with the linear regression analysis in Weka, showing similar results. This indicates the consistency of the analysis across different platforms.

The fact that 'Open', 'High', 'Low', 'Close', and 'Weighted_Price' columns have a correlation coefficient of 1.0 indicates a perfect positive linear relationship between these variables. In

the context of Bitcoin price data, this perfect correlation is expected, as these columns are all related to the pricing of Bitcoin and are derived from the same underlying data.

The analysis suggests that the value of these columns is likely to rise in the next few years. This insight can be valuable for investors and traders looking to make informed decisions about Bitcoin investments.

Overall, the analysis provides valuable insights into the trends and correlations within the Bitcoin price data, highlighting the potential for future price movements based on historical data.

Lesson Learned

The lessons I learned from this project is that processing and visualizing data of large size can be challenging if you don't have access to advanced or good technology. In addition, to relate to data analysis I have learned that I should practice more in analyzing predictions as it will be helpful to me in my next project of visualization and prediction.

Future Improvement

I have determined that I need to improve the visualization page and also improve the prediction page. Moreover, In order to improve the visualization I need to research more on when to use the graphs and what is suitable for my dataset as it will help me analyze the data more effectively.

References:

mohamedelnahry. (2023, December 31). Bitcoin Linear Regression. Kaggle.com; Kaggle. https://www.kaggle.com/code/mohamedelnahry/bitcoin-linear-regression