INTRODUCTION

This is the dataset has been taken from kaggle and the full interpretation and visualization has been done by me, though I had used chatgpt for some parts where some codes are beyond my reach.

Data source: - https://www.kaggle.com/datasets/mczielinski/bitcoin-historical-data

Blueprint of the plan

```
def generate_hierarchy(node, level=0):
    bubble = f" [{node}] "
    print("   " * level + "┌" + "─" * len(bubble) + "┐")
    print("   " * level + "│{:^{}}│".format(bubble, len(bubble) + 2))
    print("   " * level + "└" + "─" * len(bubble) + "┘")

    if node == "Blueprint":
        generate_hierarchy("Load and Preprocess Data", level + 1)
        generate_hierarchy("Train Linear Regression Model", level + 1)
        generate_hierarchy("Evaluate Model Performance", level + 1)
        generate_hierarchy("Predict on New Data", level + 1)

    elif node == "Load and Preprocess Data":
        generate_hierarchy("Load dataset into DataFrame", level + 1)
        generate_hierarchy("Drop rows with NaN values", level + 1)
        generate_hierarchy("Define features (X) and target (y)", level
+ 1)
        generate_hierarchy("Apply feature scaling", level + 1)
        generate_hierarchy("Split data into training and testing
sets", level + 1)

    elif node == "Train Linear Regression Model":
        generate_hierarchy("Initialize Linear Regression model
(model)", level + 1)
        generate_hierarchy("Train the model on the training data",
level + 1)

    elif node == "Evaluate Model Performance":
        generate_hierarchy("Make predictions on testing data", level +
1)
        generate_hierarchy("Calculate Mean Squared Error (MSE)", level
+ 1)

    elif node == "Predict on New Data":
        generate_hierarchy("Prompt user for new data input", level +
1)
        generate_hierarchy("Create DataFrame with user-input values",
level + 1)
        generate_hierarchy("Scale and preprocess new data", level + 1)
```

```python
        generate_hierarchy("Use trained model to predict target value for new data", level + 1)

def main():
    generate_hierarchy("Blueprint")

if __name__ == "__main__":
    main()
```

```
┌─────────────┐
│ [Blueprint] │
└─────────────┘
    ┌──────────────────────────────┐
    │ [Load and Preprocess Data]   │
    └──────────────────────────────┘
        ┌──────────────────────────────────┐
        │ [Load dataset into DataFrame]    │
        └──────────────────────────────────┘
        ┌──────────────────────────────┐
        │ [Drop rows with NaN values]  │
        └──────────────────────────────┘
        ┌────────────────────────────────────────┐
        │ [Define features (X) and target (y)]   │
        └────────────────────────────────────────┘
        ┌──────────────────────────┐
        │ [Apply feature scaling]  │
        └──────────────────────────┘
        ┌─────────────────────────────────────────────┐
        │ [Split data into training and testing sets] │
        └─────────────────────────────────────────────┘
    ┌──────────────────────────────────┐
    │ [Train Linear Regression Model]  │
    └──────────────────────────────────┘
        ┌──────────────────────────────────────────────┐
        │ [Initialize Linear Regression model (model)] │
        └──────────────────────────────────────────────┘
        ┌────────────────────────────────────────┐
        │ [Train the model on the training data] │
        └────────────────────────────────────────┘
    ┌──────────────────────────────┐
    │ [Evaluate Model Performance] │
    └──────────────────────────────┘
        ┌────────────────────────────────────┐
        │ [Make predictions on testing data] │
        └────────────────────────────────────┘
        ┌────────────────────────────────────────┐
        │ [Calculate Mean Squared Error (MSE)]   │
        └────────────────────────────────────────┘
    ┌──────────────────────────┐
    │ [Predict on New Data]    │
    └──────────────────────────────┘

        ┌───────────────────────────────────┐
        │  [Prompt user for new data input]  │
        └───────────────────────────────────┘

        ┌──────────────────────────────────────────┐
        │  [Create DataFrame with user-input values]  │
        └──────────────────────────────────────────┘

        ┌──────────────────────────────────┐
        │  [Scale and preprocess new data]  │
        └──────────────────────────────────┘

        ┌──────────────────────────────────────────────────────┐
        │  [Use trained model to predict target value for new data]  │
        └──────────────────────────────────────────────────────┘
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv(r'C:\Users\Anonymous\Documents\bitstampUSD_1-min_data_2012-01-01_to_2021-03-31.csv')

df.info
```

```
<bound method DataFrame.info of            Timestamp      Open
High       Low     Close  Volume_(BTC)  \
0        1325317920      4.39      4.39      4.39      4.39
0.455581
1        1325317980       NaN       NaN       NaN       NaN
NaN
2        1325318040       NaN       NaN       NaN       NaN
NaN
3        1325318100       NaN       NaN       NaN       NaN
NaN
4        1325318160       NaN       NaN       NaN       NaN
NaN
...             ...       ...       ...       ...       ...
...
4857372  1617148560  58714.31  58714.31  58686.00  58686.00
1.384487
4857373  1617148620  58683.97  58693.43  58683.97  58685.81
7.294848
4857374  1617148680  58693.43  58723.84  58693.43  58723.84
1.705682
4857375  1617148740  58742.18  58770.38  58742.18  58760.59
0.720415
4857376  1617148800  58767.75  58778.18  58755.97  58778.18
2.712831

         Volume_(Currency)  Weighted_Price
```

```
0                   2.000000        4.390000
1                        NaN             NaN
2                        NaN             NaN
3                        NaN             NaN
4                        NaN             NaN
...                      ...             ...
4857372         81259.372187    58692.753339
4857373        428158.146640    58693.226508
4857374        100117.070370    58696.198496
4857375         42332.958633    58761.866202
4857376        159417.751000    58764.349363

[4857377 rows x 8 columns]>
```

```
df.isnull().mean()*100
```

```
Timestamp             0.00000
Open                 25.60246
High                 25.60246
Low                  25.60246
Close                25.60246
Volume_(BTC)         25.60246
Volume_(Currency)    25.60246
Weighted_Price       25.60246
dtype: float64
```

```
df.columns
```

```
Index(['Timestamp', 'Open', 'High', 'Low', 'Close', 'Volume_(BTC)',
       'Volume_(Currency)', 'Weighted_Price'],
      dtype='object')
```

Predictive Model

Here you can enter the values for the open, high, low, close, volume of BTC and weighted price and after that you will get the value of volume of currency traded in future.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

# Assuming you've already loaded the data into df
# Drop rows with NaN values
df.dropna(inplace=True)

# Select features (X) and target (y)
```

```python
features = ['Open', 'High', 'Low', 'Close', 'Volume_(BTC)',
'Weighted_Price']
target = 'Volume_(Currency)'

X = df[features]
y = df[target]

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

# Initialize the Linear Regression model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Take user input for new data values
new_open_value = float(input("Enter Open value: "))
new_high_value = float(input("Enter High value: "))
new_low_value = float(input("Enter Low value: "))
new_close_value = float(input("Enter Close value: "))
new_volume_btc_value = float(input("Enter Volume_(BTC) value: "))
new_weighted_price_value = float(input("Enter Weighted_Price value:
"))

# Create a DataFrame with the user-input values
new_data = pd.DataFrame({
    'Open': [new_open_value],
    'High': [new_high_value],
    'Low': [new_low_value],
    'Close': [new_close_value],
    'Volume_(BTC)': [new_volume_btc_value],
    'Weighted_Price': [new_weighted_price_value]
})

# Handle missing values and scale new data
new_data.dropna(inplace=True)
new_data_scaled = scaler.transform(new_data)
```

```
# Make predictions on the new data
predicted_volume = model.predict(new_data_scaled)
print(f"Predicted Volume_(Currency): {predicted_volume}")

Mean Squared Error: 11677888354.938972
```

Import Libraries:

Begin by bringing in the tools needed for your analysis. These libraries are like toolkits that contain functions for data manipulation, machine learning, and creating visualizations.

Load and Prepare Data:

Assume you have your financial data in a structured format. Start by tidying up the data, making sure there aren't any gaps or missing entries that could lead to confusion later.

Choose Features and Target:

Decide which aspects of the data are important for making predictions. For instance, you might pick variables like the starting price ('Open'), the highest price ('High'), the lowest price ('Low'), the final price ('Close'), the traded volume in Bitcoin ('Volume_(BTC)'), and the weighted price ('Weighted_Price'). The thing you're aiming to predict, like the currency volume ('Volume_(Currency)'), becomes the target.

Scale Features:

To make sure the different variables are playing on a level field, apply a transformation so that they're all in a similar range. This helps the prediction process work better.

Divide into Training and Testing Sets:

Divide the data into two parts. One part will be used to teach the algorithm how to make predictions ('training set'), and the other part will be used to test how well it learned ('testing set'). A common split is 80% for training and 20% for testing.

Prepare and Train the Model:

Set up the prediction tool, which in this case is a Linear Regression model. Use the training data to 'teach' the model how to predict the target based on the features you've chosen.

Predict and Evaluate:

Once the model is trained, it's time to see how well it does. Have it make predictions on the testing set and measure how close its predictions are to the actual results. The 'Mean Squared Error' gives you a sense of how accurate the predictions are.

Predict with New Data:

Imagine you have new data about a financial situation. Enter the relevant numbers, like the opening and closing prices, the trading volume, and the weighted price. These become the 'input' for the model.

Prepare New Data and Predict:

Just as you cleaned up and transformed the original data, do the same for the new data. This makes sure it's in a format the model understands. Then, use the model you trained earlier to predict the currency volume for this new scenario.