Untitled

August 21, 2023

BTC Fee Prediction

In the realm of digital currencies and blockchain, accurately anticipating Bitcoin transaction fees is vital. This project leverages historical blockchain data to create a predictive model for Bitcoin fee rates. By applying machine learning to a comprehensive dataset, the model aims to predict how transaction fees evolve as network dynamics change.

The dataset encompasses diverse attributes, including block size, transaction count, network difficulty, and fee rate statistics. These features illuminate the intricate interplay within the Bitcoin network's fee ecosystem.

Through advanced techniques like time series cross-validation, the model's efficacy is evaluated. Evaluation metrics like Mean Absolute Error and R-squared provide insights into its alignment with actual fee rates.

The model's outcomes empower users to make well-informed transaction decisions and assist miners in optimizing their strategies during periods of fee variability. By merging machine learning and blockchain analysis, this project contributes not only a predictive tool but also fresh insights into the evolving cryptocurrency transaction landscape.

Library Imports & Data Preprocessing

[1]:
```
from os import chdir, getcwd
wd=getcwd()
chdir(wd)

import pandas as pd
```

[2]:
```
##### Data Preprocessing #####

# read init csv in
df = pd.read_csv("dataset.csv")

# Convert timestamp from milliseconds to seconds
df['timestamp'] = df['timestamp'] / 1000

# Convert the Unix timestamp to a human-readable datetime
df['datetime'] = pd.to_datetime(df['timestamp'], unit='s')

# clean up the datetime variable and make it more readable
```

```python
df['formatted_datetime'] = df['datetime'].dt.strftime('%Y-%m-%d %H:%M:%S')

# Create Test/Training sets for data.
# Do this based on a Temportal Dependency (timestamp or datetime)
# That way I can predict future BTC fee rate

column_names = df.columns.tolist()

print(column_names)
```

```
['height', 'timestamp', 'size', 'tx_count', 'difficulty', 'median_fee_rate',
'avg_fee_rate', 'total_fees', 'fee_range_min', 'fee_range_max', 'input_count',
'output_count', 'output_amount', 'datetime', 'formatted_datetime']
```

Feature Selection, Data Splitting, Feature Scaling

[3]:
```python
from sklearn.model_selection import TimeSeriesSplit
from sklearn.preprocessing import StandardScaler



# Create X/Ys & Target column naming

X = df[['height', 'size', 'tx_count', 'difficulty',
        'total_fees', 'fee_range_min', 'fee_range_max']]

y = df['avg_fee_rate']  # wish to predict the average fee rate

# Create the time series object
n_splits = 5
tscv = TimeSeriesSplit(n_splits)

# init scaler
scaler = StandardScaler()



# Iterate through the splits
for train_index, test_index in tscv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
```

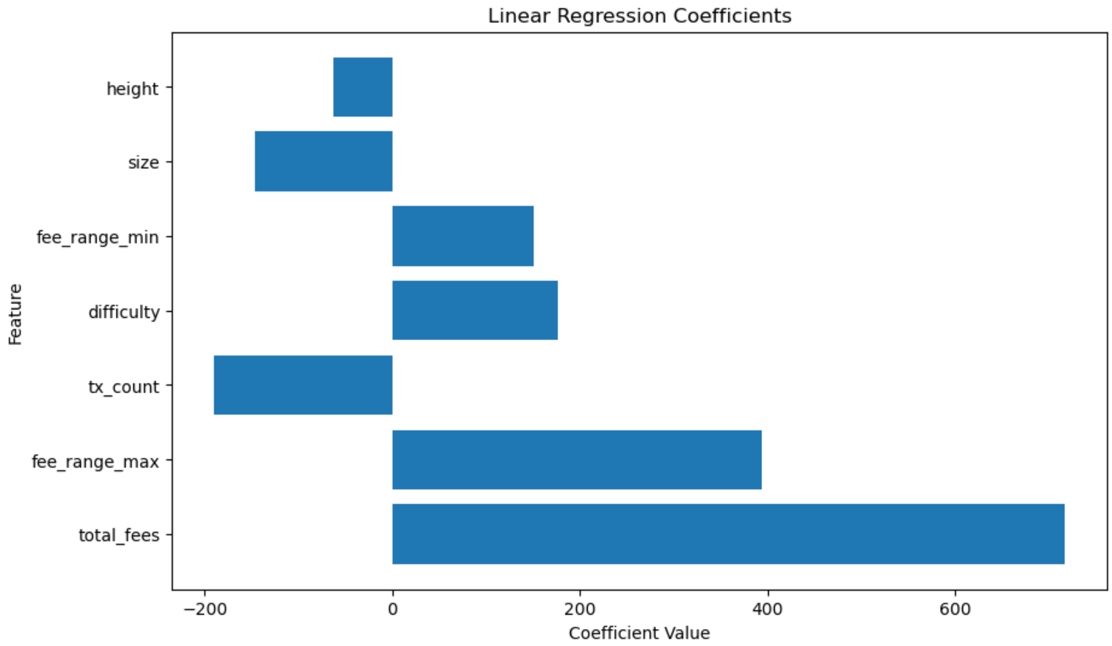
Model Selection

[5]:
```python
### Linear Regression ###

import numpy as np
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# init Linear Regression
linear_reg_model = LinearRegression()

# Fit the model
linear_reg_model.fit(X_train_scaled, y_train)

## Predictions and Evaluations

# Predict on the scaled test data
y_pred = linear_reg_model.predict(X_test_scaled)

# Evaluate the model's performance
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R-squared:", r2)

coef = linear_reg_model.coef_
intercept = linear_reg_model.intercept_

# Understanding the outcome

import matplotlib.pyplot as plt

# Create a DataFrame to store feature names and their corresponding coefficients
coef_df = pd.DataFrame({'Feature': X.columns, 'Coefficient': coef})

# Sort the DataFrame by coefficient magnitude
coef_df = coef_df.reindex(coef_df['Coefficient'].abs().
 ↪sort_values(ascending=False).index)

# Plot the coefficients
plt.figure(figsize=(10, 6))
plt.barh(coef_df['Feature'], coef_df['Coefficient'])
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.title('Linear Regression Coefficients')
plt.show()
```

```
Mean Absolute Error: 422.13656008982633
Mean Squared Error: 294508.64865168754
```

```
R-squared: -186.73623966888712
```

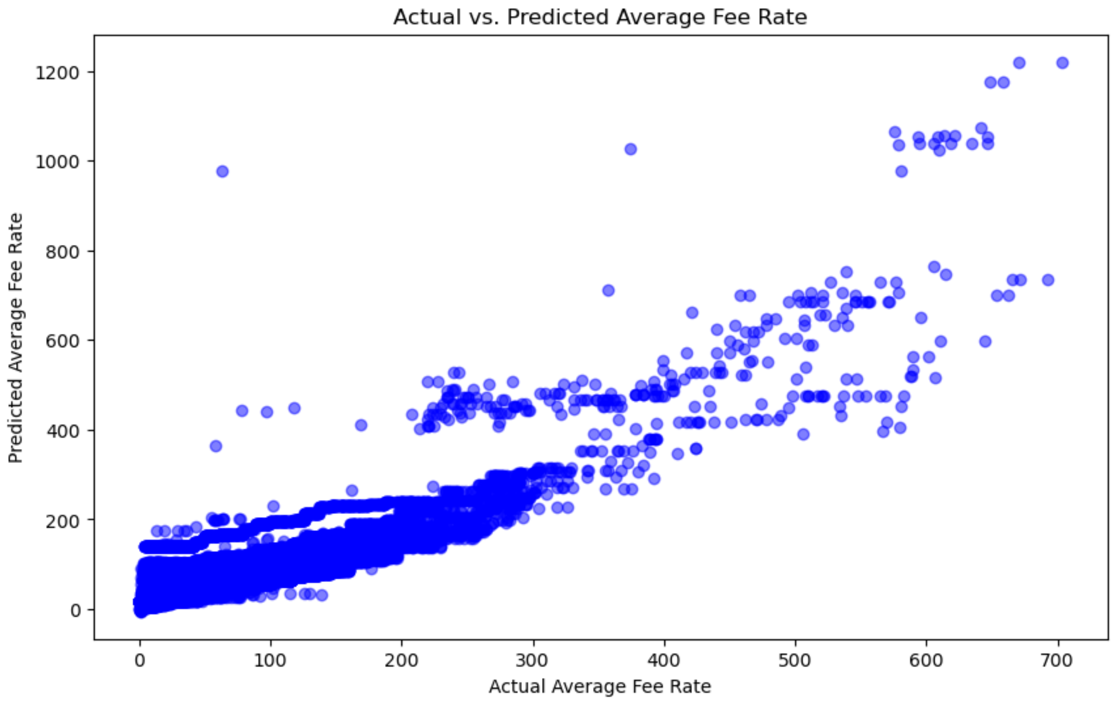


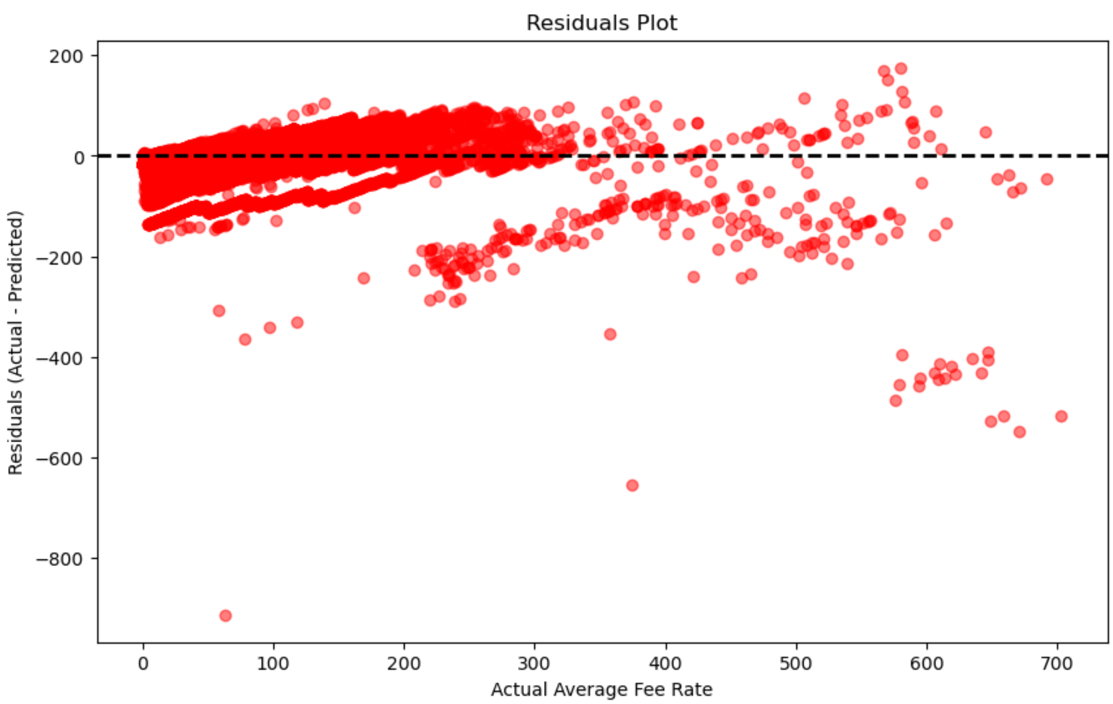
[7]:
```python
### Gradient Boosting ###

from sklearn.ensemble import GradientBoostingRegressor
# Initialize the Gradient Boosting Regressor
gb_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,␣
↪random_state=42)

# Initialize lists to store evaluation metrics
mae_scores = []
mse_scores = []
r2_scores = []

# Iterate through the splits
for train_index, test_index in tscv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Train the Gradient Boosting model on the scaled training data
    gb_model.fit(X_train_scaled, y_train)

    # Predict the target variable for the test set
```

```
    y_pred = gb_model.predict(X_test_scaled)

    # Calculate evaluation metrics
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Append scores to lists
    mae_scores.append(mae)
    mse_scores.append(mse)
    r2_scores.append(r2)

# Calculate average scores across folds
average_mae = sum(mae_scores) / len(mae_scores)
average_mse = sum(mse_scores) / len(mse_scores)
average_r2 = sum(r2_scores) / len(r2_scores)

# Print or display the average evaluation scores
print(f"Mean Absolute Error: {average_mae}")
print(f"Mean Squared Error: {average_mse}")
print(f"R-squared: {average_r2}")
```

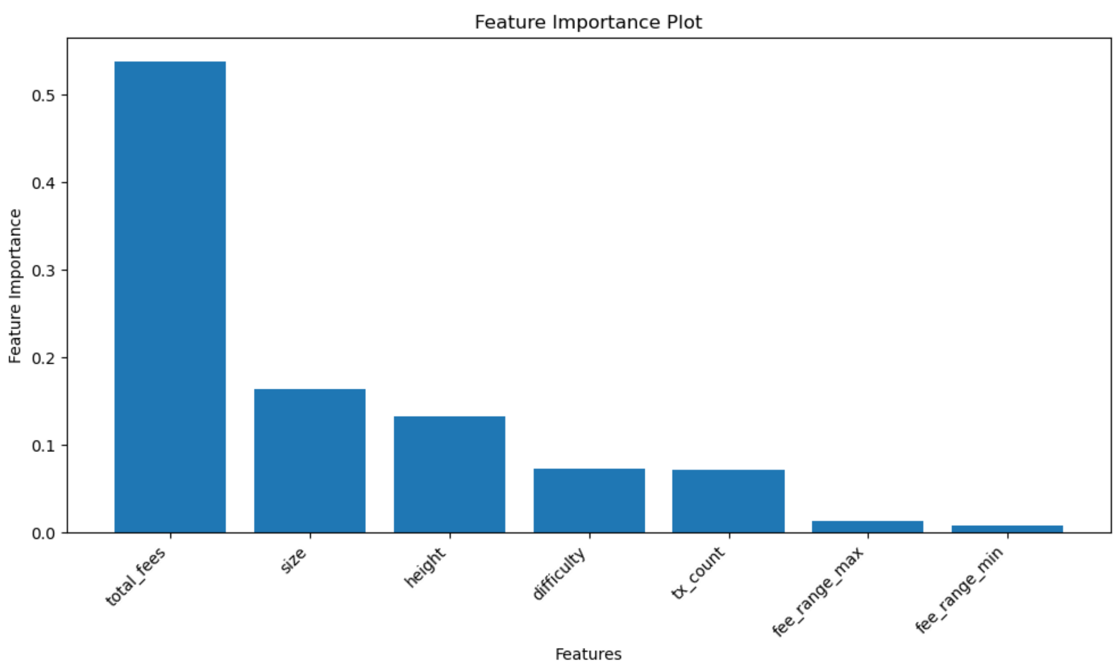
```
Mean Absolute Error: 113.69047985607911
Mean Squared Error: 3971191.199111941
R-squared: -1.6548790252816759
```

[8]:
```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.xlabel('Actual Average Fee Rate')
plt.ylabel('Predicted Average Fee Rate')
plt.title('Actual vs. Predicted Average Fee Rate')
plt.show()

# Calculate and plot residuals
residuals = y_test - y_pred
plt.figure(figsize=(10, 6))
plt.scatter(y_test, residuals, color='red', alpha=0.5)
plt.axhline(y=0, color='black', linestyle='--', linewidth=2)
plt.xlabel('Actual Average Fee Rate')
plt.ylabel('Residuals (Actual - Predicted)')
plt.title('Residuals Plot')
plt.show()
```


Actual vs. Predicted Average Fee Rate
Predicted Average Fee Rate
Actual Average Fee Rate
0
200
400
600
800
1000
1200
0
100
200
300
400
500
600
700


Residuals Plot
Residuals (Actual - Predicted)
Actual Average Fee Rate
200
0
−200
−400
−600
−800
0
100
200
300
400
500
600
700

[9]:
```
from sklearn.ensemble import GradientBoostingRegressor


# Fit the Gradient Boosting model on the entire dataset
gb_model.fit(X, y)

# Get feature importances from the model
feature_importances = gb_model.feature_importances_

# Get the names of the features
feature_names = X.columns

# Sort feature importances in descending order
sorted_idx = np.argsort(feature_importances)[::-1]

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.bar(range(X.shape[1]), feature_importances[sorted_idx], align='center')
plt.xticks(range(X.shape[1]), np.array(feature_names)[sorted_idx], rotation=45,␣
 ↪ha='right')
plt.xlabel('Features')
plt.ylabel('Feature Importance')
plt.title('Feature Importance Plot')
plt.tight_layout()
plt.show()
```



Conclusion

During this project, I attempted to use Multiple regression and gradient boosting to predict the potential networking fee of bitcoin. Based on the results received, I must return to the drawing board. It seems as though size, height, difficulty, and transaction count are not able to explain the fluctuations in bitcoin networking fees as much as I expected.

Since the bitcoin market is so volatile and transactions pick up and drop so quickly, it is likely that transactions per second and current bitcoin price would be better features.