I am doing a project — Fraud Detection in Blockchain using Machine Learning
For this project I have obtained a dataset from kaggle
I have attached the sample of the dataset to you
Right now I just want to analyze the dataset into various aspects and visualize it.
This is basically pre data analysis where I want to learn more about dataset from various aspects and dimensions using graphs, text, info visuals etc.

What I want form you is to give me python code to analyze and visualize the dataset from various aspects (Write the code cell by cell with short description of why we did this)

I want the analysis and visualization to be very deep and meaningful and intense.

After this intense analysis I will see what all feature extraction and feature engineering and preprocessing I have to do and based on that I will shortlist models and will train them and check accuracy and enhance them and later deploy them (This is later on)

Blockchain Transactions Fraud Detec...

Python

```python
# Cell 1: Import necessary libraries
# We import pandas for data manipulation, matplotlib and seaborn for visu
# These libraries allow us to load, analyze, and plot the data effectivel
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import numpy as np  # For numerical operations if needed
```

```
# Set seaborn style for better-looking plots
sns.set(style="whitegrid")
```

Python

```
# Cell 2: Load the dataset
# We load the CSV file into a pandas DataFrame. The filename is as provid
# This step is crucial to bring the data into memory for further analysis
df = pd.read_csv('Blockchain Transactions Fraud Detection – Fraud Detecti

# Display the first few rows to get an initial look at the data
print("First 5 rows of the dataset:")
display(df.head())
```

Python

```
# Cell 3: Basic dataset information
# Use info() to check data types, non-null counts, and memory usage.
# This helps identify if columns need type conversion (e.g., TimeStamp is
print("Dataset Information:")
df.info()

# Check the shape of the dataset (rows, columns)
print(f"\nDataset Shape: {df.shape}")
```

Python

```
# Cell 4: Summary statistics for numerical columns
# describe() provides count, mean, std, min, max, and quartiles for numer
# This gives a quick overview of distributions, potential outliers (e.g.,
print("Summary Statistics:")
display(df.describe())

# For categorical/string columns like From, To, TxHash, get unique counts
print("\nUnique Values Count:")
print(f"Unique TxHash: {df['TxHash'].nunique()}")
print(f"Unique From Addresses: {df['From'].nunique()}")
print(f"Unique To Addresses: {df['To'].nunique()}")
print(f"Unique isError Values: {df['isError'].unique()}")
```

Python

```python
# Cell 5: Check for missing values
# Identify any null or missing data in each column.
# This is important for data quality; if missing, we might need imputatio
print("Missing Values per Column:")
display(df.isnull().sum())

# Visualize missing values if any (heatmap for visual inspection)
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.show()
```

Python

```python
# Cell 6: Convert TimeStamp to datetime
# TimeStamp appears to be Unix timestamps (e.g., 1.53E+09). Convert to re
# This enables grouping by date, hour, etc., for temporal patterns in tra
df['TimeStamp'] = pd.to_datetime(df['TimeStamp'], unit='s')  # Assuming s

# Display updated head to verify conversion
print("Dataset after TimeStamp Conversion:")
display(df.head())
```

Python

```python
# Cell 7: Analyze isError distribution (potential fraud label)
# isError is binary (0/1). Use a pie chart to visualize the proportion of
# This is key for imbalance check in fraud detection datasets, where frau
error_counts = df['isError'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(error_counts, labels=error_counts.index, autopct='%1.1f%%', start
plt.title('Distribution of isError (0: No Error, 1: Error)')
plt.show()

# Text summary
print(f"Non-Error Transactions: {error_counts.get(0, 0)}")
print(f"Error Transactions: {error_counts.get(1, 0)}")
print(f"Error Rate: {error_counts.get(1, 0) / len(df) * 100:.2f}%")
```

Python

```python
# Cell 8: Histogram of Transaction Values
# Plot the distribution of 'Value' to see how transaction amounts are spr
# Use log scale if skewed (common in crypto transactions) to handle outli
plt.figure(figsize=(12, 6))
sns.histplot(df['Value'], bins=50, kde=True, color='blue')
plt.title('Distribution of Transaction Values')
plt.xlabel('Value (ETH?)')
plt.ylabel('Frequency')
plt.xscale('log')  # Log scale for better visibility if values vary widel
plt.show()

# Text insights: Skewness and kurtosis for depth
print(f"Value Skewness: {df['Value'].skew():.2f} (Positive skew indicates
print(f"Value Kurtosis: {df['Value'].kurt():.2f} (High kurtosis means hea
```

Python

```python
# Cell 9: Boxplot of Values by isError
# Compare Value distributions between error and non-error transactions.
# This can reveal if fraudulent transactions have different value pattern
plt.figure(figsize=(10, 6))
sns.boxplot(x='isError', y='Value', data=df, palette='Set2')
plt.title('Boxplot of Transaction Values by isError')
plt.yscale('log')  # Log scale for outliers
plt.show()

# Statistical test for difference (e.g., t-test if applicable, but for de
from scipy.stats import ttest_ind
non_error_values = df[df['isError'] == 0]['Value']
error_values = df[df['isError'] == 1]['Value']
if len(error_values) > 0:
    t_stat, p_val = ttest_ind(non_error_values, error_values)
    print(f"T-test for Value difference: t-stat={t_stat:.2f}, p-value={p_
else:
    print("No error transactions for comparison.")
```

Python

```python
# Cell 10: Time series plot of transaction counts over time
# Group by date and count transactions to see volume trends.
# This helps identify peaks, trends, or seasonality in blockchain activit
df['Date'] = df['TimeStamp'].dt.date  # Extract date for grouping
daily_counts = df.groupby('Date').size()
```

```python
plt.figure(figsize=(14, 6))
daily_counts.plot(kind='line', marker='o', color='green')
plt.title('Daily Transaction Counts')
plt.xlabel('Date')
plt.ylabel('Number of Transactions')
plt.grid(True)
plt.show()

# Text: Trend analysis
print(f"Total Days Covered: {len(daily_counts)}")
print(f"Average Daily Transactions: {daily_counts.mean():.2f}")
print(f"Max Daily Transactions: {daily_counts.max()} on {daily_counts.idx
```

Python

```python
# Cell 11: Transaction counts by hour of day
# Extract hour from TimeStamp and plot to find intraday patterns.
# Useful for detecting if fraud occurs at specific times (e.g., off-hours
df['Hour'] = df['TimeStamp'].dt.hour
hourly_counts = df.groupby('Hour').size()

plt.figure(figsize=(12, 6))
sns.barplot(x=hourly_counts.index, y=hourly_counts.values, color='purple'
plt.title('Transaction Counts by Hour of Day')
plt.xlabel('Hour (0-23)')
plt.ylabel('Count')
plt.show()

# Cross-tab with isError for deeper insight
hourly_error = pd.crosstab(df['Hour'], df['isError'], normalize='index')
print("Percentage of Errors by Hour:")
display(hourly_error)
```

Python

```python
# Cell 12: Top 10 From and To addresses by frequency
# Count occurrences of sender/receiver addresses to identify frequent use
# This can highlight wallets involved in many transactions, potential hot
top_from = df['From'].value_counts().head(10)
top_to = df['To'].value_counts().head(10)

fig, axes = plt.subplots(1, 2, figsize=(16, 6))
```

```
sns.barplot(x=top_from.values, y=top_from.index, ax=axes[0], palette='Blu
axes[0].set_title('Top 10 From Addresses by Frequency')
axes[0].set_xlabel('Count')

sns.barplot(x=top_to.values, y=top_to.index, ax=axes[1], palette='Greens_
axes[1].set_title('Top 10 To Addresses by Frequency')
axes[1].set_xlabel('Count')
plt.tight_layout()
plt.show()

# Text: Check if any address is both top sender and receiver
common_addresses = set(top_from.index) & set(top_to.index)
print(f"Common Addresses in Top 10 From and To: {common_addresses}")
```

Python

```
# Cell 13: Top addresses by total Value sent/received
# Sum Values for each From/To to see high-value players.
# This complements frequency by showing economic impact, useful for anoma
from_value_sum = df.groupby('From')['Value'].sum().sort_values(ascending=
to_value_sum = df.groupby('To')['Value'].sum().sort_values(ascending=Fals

fig, axes = plt.subplots(1, 2, figsize=(16, 6))
sns.barplot(x=from_value_sum.values, y=from_value_sum.index, ax=axes[0],
axes[0].set_title('Top 10 From Addresses by Total Value Sent')
axes[0].set_xlabel('Total Value')

sns.barplot(x=to_value_sum.values, y=to_value_sum.index, ax=axes[1], pale
axes[1].set_title('Top 10 To Addresses by Total Value Received')
axes[1].set_xlabel('Total Value')
plt.tight_layout()
plt.show()

# Correlation with errors: Average Value for error vs non-error
print(f"Avg Value for Non-Error: {non_error_values.mean():.4f}")
if len(error_values) > 0:
    print(f"Avg Value for Error: {error_values.mean():.4f}")
```

Python

```
# Cell 14: Scatter plot of BlockHeight vs Value
# Plot to see if transaction values change with block height (proxy for t
# Add color by isError for patterns in errors over blockchain progression
```

```python
plt.figure(figsize=(12, 6))
sns.scatterplot(x='BlockHeight', y='Value', hue='isError', data=df, palet
plt.title('Scatter Plot: BlockHeight vs Transaction Value (Colored by isE
plt.yscale('log')
plt.show()

# Correlation matrix for numerical columns
numerical_cols = ['BlockHeight', 'Value', 'isError']  # TimeStamp exclude
corr_matrix = df[numerical_cols].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```

Python

```python
# Cell 15: Pairplot for multivariate analysis
# Visualize relationships between all numerical pairs, colored by isError
# This provides a deep overview of interactions, clusters, and separation
sns.pairplot(df[numerical_cols + ['TimeStamp'].copy()], hue='isError', pa
plt.suptitle('Pairplot of Key Features by isError', y=1.02)
plt.show()
```

Python

```python
# Cell 16: Analyze transaction chains or repeated From-To pairs
# Group by From-To pairs to find frequent interactions.
# This can indicate patterns like loops or suspicious repeated transfers.
pair_counts = df.groupby(['From', 'To']).size().sort_values(ascending=Fal
print("Top 10 From-To Pairs by Frequency:")
display(pair_counts)

# Visualize as a network graph if small (using networkx for depth)
import networkx as nx
G = nx.from_pandas_edgelist(df.head(100), 'From', 'To', edge_attr='Value'
plt.figure(figsize=(12, 12))
nx.draw(G, with_labels=False, node_size=50, edge_color='gray', node_color
plt.title('Network Graph of Transactions (Sample)')
plt.show()
```

Python

```python
# Cell 17: Outlier detection in Value using IQR
# Identify outliers which might be fraudulent or high-risk transactions.
# Boxplot already showed, but here quantify and list them for intensity.
Q1 = df['Value'].quantile(0.25)
Q3 = df['Value'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['Value'] < (Q1 - 1.5 * IQR)) | (df['Value'] > (Q3 + 1.5

print(f"Number of Outliers in Value: {len(outliers)}")
print("Outlier Transactions:")
display(outliers[['TxHash', 'Value', 'isError']])

# Percentage of outliers that are errors
if len(outliers) > 0:
    outlier_error_rate = outliers['isError'].mean() * 100
    print(f"Error Rate in Outliers: {outlier_error_rate:.2f}%")
```

Python

```python
# Cell 18: Time differences between consecutive transactions
# Calculate time deltas to analyze transaction speed/frequency.
# Sort by TimeStamp, compute diffs, and plot distribution.
df_sorted = df.sort_values('TimeStamp')
df_sorted['TimeDelta'] = df_sorted['TimeStamp'].diff().dt.total_seconds()

plt.figure(figsize=(12, 6))
sns.histplot(df_sorted['TimeDelta'].dropna(), bins=50, kde=True, color='r
plt.title('Distribution of Time Differences Between Consecutive Transacti
plt.xscale('log')
plt.show()

# Average time delta by isError
print("Average Time Delta (minutes) by isError:")
display(df_sorted.groupby('isError')['TimeDelta'].mean())
```

Python

```python
# Cell 19: Cumulative sum of Values over time
# Plot cumulative transaction values to see growth patterns.
# Separate by isError for comparison if errors exist.
plt.figure(figsize=(14, 6))
df_sorted['CumValue'] = df_sorted['Value'].cumsum()
df_sorted.plot(x='TimeStamp', y='CumValue', kind='line', color='orange')
```

```
plt.title('Cumulative Transaction Value Over Time')
plt.ylabel('Cumulative Value')
plt.show()

# If errors, plot separate cumulatives
if df['isError'].sum() > 0:
    error_cum = df_sorted[df_sorted['isError'] == 1]['Value'].cumsum()
    non_error_cum = df_sorted[df_sorted['isError'] == 0]['Value'].cumsum(
    plt.figure(figsize=(14, 6))
    plt.plot(df_sorted['TimeStamp'], non_error_cum, label='Non-Error', co
    plt.plot(df_sorted['TimeStamp'], error_cum, label='Error', color='red
    plt.title('Cumulative Value by isError Over Time')
    plt.ylabel('Cumulative Value')
    plt.legend()
    plt.show()
```

Python

```
# Cell 20: Word cloud for addresses (if meaningful) or summary stats
# Addresses are hashes, so not textual, but we can check length or patter
# For depth, check address lengths or if they follow patterns.
df['From_Len'] = df['From'].str.len()
df['To_Len'] = df['To'].str.len()

print("Address Length Statistics:")
print(df[['From_Len', 'To_Len']].describe())

# Plot if variations exist
plt.figure(figsize=(10, 6))
sns.countplot(x='From_Len', data=df, color='cyan')
plt.title('Distribution of From Address Lengths')
plt.show()

# Final text summary of key insights
print("\nKey Insights from Analysis:")
print("- Dataset is small; consider full Kaggle data for more robust patt
print("- Check for class imbalance in isError.")
print("- Values are skewed; log-transform may help in modeling.")
print("- Temporal patterns: Look for anomalies in off-peak hours.")
print("- Frequent addresses: Monitor for suspicious activity.")
print("- Outliers: High values may correlate with risks.")
print("- Next steps: Feature engineering like address embeddings, time fe
```

↳ Feature Engineering Ideas

↳ Ethereum Fraud Patterns