you can fully understand what's happening at each step, along with explanations of the parameters used.

📁 Phase 1: Data Collection and Loading

```
import pandas as pd

# Read the CSV file
df = pd.read_csv('/kaggle/input/ugransome-dataset/final(2).csv')

# Convert to a DataFrame and rename columns for clarity
df2 = pd.DataFrame(df)
df2.columns = ['Time', 'Protocol', 'Flag', 'Family', 'Clusters', 'SeedAddress',
               'ExpAddress', 'BTC', 'USD', 'Netflow_Bytes', 'IPaddress',
               'Threats', 'Port', 'Prediction']
```

Explanation:

- `pd.read_csv()`: Loads the dataset from the specified path.
    - `/kaggle/input/ugransome-dataset/final(2).csv` is the path to your dataset.
- **Renaming Columns**: Assigns readable column names to improve clarity.

🧹 Data Cleaning

Step 1: Correcting Misspelled Values

```
# Renaming the attack "Bonet" to "Botnet"
df2['Threats'] = df2['Threats'].str.replace('Bonet', 'Botnet')
```

Explanation:

- `str.replace()`: Corrects spelling mistakes in the 'Threats' column.

Step 2: Dropping Duplicate Rows

```
df2 = df2.drop_duplicates()
```

Explanation:

- `drop_duplicates()`: Removes rows that are identical.

🛠 Data Transformation

Handling the 'Time' Column

```
# Adjusting 'Time' values by adding 11
df2['Time'] = df2['Time'] + 11
```

Explanation:

- This is used to shift all time entries by 11 units, possibly for correcting time zone differences or alignment.

Applying Mathematical Transformations to Reduce Skewness

- Transformations help normalize data distribution, making it easier for models to learn patterns.

Log Transformation on `Netflow_Bytes`

```
df2['Netflow_Bytes'] = np.log(df2['Netflow_Bytes'] + 1)
```

- `np.log()` : Compresses the range of data, especially useful for right-skewed distributions.
- `+1` : Ensures no issues with log(0), as the logarithm of zero is undefined.

Square Root Transformation on `USD`

```
df2['USD'] = np.sqrt(df2['USD'])
```

- `np.sqrt()` : Reduces skewness but is less aggressive than log transformations.

Yeo-Johnson Transformation on `BTC`

```
from scipy import stats
df2['BTC'], _ = stats.yeojohnson(df2['BTC'])
```

- **Yeo-Johnson**: Handles both positive and negative values, making it more versatile than Box-Cox.

📊 Data Normalization and Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df2_normalized = df2.copy()
df2_normalized[['USD', 'BTC', 'Netflow_Bytes']] = scaler.fit_transform(df2[['USD',
'BTC', 'Netflow_Bytes']])
```

Explanation:

- **StandardScaler**: Scales features to have a mean of 0 and standard deviation of 1.
- Normalization is essential for models sensitive to feature scaling (e.g., SVM, KNN).

📈 Data Visualization

Plotting Transformed Data Distributions

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.hist(df2['USD'], bins=50, alpha=0.5, color='blue', label='USD (Square Root)')
ax.hist(df2['BTC'], bins=50, alpha=0.5, color='green', label='BTC (Yeo-Johnson)')
ax.hist(df2['Netflow_Bytes'], bins=50, alpha=0.5, color='red', label='Netflow_Bytes
(Log)')
ax.set_xlabel('Transformed Values')
ax.set_ylabel('Frequency')
ax.set_title('Distribution of Transformed Columns')
ax.legend()
plt.show()
```

Explanation:

- `plt.hist()`: Plots histograms for visualizing the distribution of transformed features.
- Multiple histograms are overlaid to compare distributions.

Density Plot of Normalized Features

```
sns.kdeplot(df2_normalized['USD'], color='blue', label='USD', ax=ax)
sns.kdeplot(df2_normalized['BTC'], color='green', label='BTC', ax=ax)
sns.kdeplot(df2_normalized['Netflow_Bytes'], color='red', label='Netflow_Bytes',
ax=ax)
ax.set_title('Density Plot of Normalized Columns')
plt.show()
```

🏷️ Encoding Categorical Variables

```
from sklearn import preprocessing

lab_encoder = preprocessing.LabelEncoder()
df2['Protocol'] = lab_encoder.fit_transform(df2['Protocol'])
df2['Flag'] = lab_encoder.fit_transform(df2['Flag'])
df2['Family'] = lab_encoder.fit_transform(df2['Family'])
df2['SeedAddress'] = lab_encoder.fit_transform(df2['SeedAddress'])
df2['ExpAddress'] = lab_encoder.fit_transform(df2['ExpAddress'])
df2['IPaddress'] = lab_encoder.fit_transform(df2['IPaddress'])
```

```
df2['Threats'] = lab_encoder.fit_transform(df2['Threats'])
df2['Prediction'] = lab_encoder.fit_transform(df2['Prediction'])
```

Explanation:

- **LabelEncoder**: Converts categorical values into numeric labels.

📊 Model Training and Evaluation

Splitting Data into Training and Testing Sets

```
from sklearn.model_selection import train_test_split

X = df2.iloc[:, :-1]
y = df2.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
random_state=42)
```

Explanation:

- **train_test_split()**: Splits the data into training (80%) and testing (20%) sets.

🔄 Ensemble Models: Random Forest, SVM, Naive Bayes

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB

# Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
rf_accuracy = accuracy_score(rf_pred, y_test)

# SVM
svr = LinearSVC()
svr.fit(X_train, y_train)
svr_pred = svr.predict(X_test)
svr_accuracy = accuracy_score(svr_pred, y_test)

# Naive Bayes
nb = GaussianNB()
nb.fit(X_train, y_train)
```

```
nb_pred = nb.predict(X_test)
nb_accuracy = accuracy_score(nb_pred, y_test)
```

Explanation:

- **RandomForestClassifier**: Uses an ensemble of decision trees for classification.
- **LinearSVC**: A linear support vector classifier.
- **GaussianNB**: Naive Bayes classifier for normally distributed data.

🛠️ Evaluating Model Performance

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

print('Accuracy of Random Forest:', rf_accuracy)
print('Accuracy of SVM:', svr_accuracy)
print('Accuracy of Naive Bayes:', nb_accuracy)

print('Classification Report (Random Forest):\n', classification_report(rf_pred,
y_test))
print('Confusion Matrix (Random Forest):\n', confusion_matrix(rf_pred, y_test))
```

🏆 Ensemble Learning with Stacking Classifier

```
from sklearn.ensemble import StackingClassifier

estimators = [
    ('rf', RandomForestClassifier(n_estimators=1000, random_state=42)),
    ('svr', LinearSVC(random_state=42))
]
clf = StackingClassifier(estimators=estimators, final_estimator=GaussianNB())
clf.fit(X_train, y_train)
pred = clf.predict(X_test)

eb_accuracy = accuracy_score(pred, y_test)
print('Accuracy of Ensemble Model:', eb_accuracy)
```

🔍 Visualization of Model Metrics

```
models = ['Random Forest', 'SVM', 'Naive Bayes', 'Ensemble Learning']
accuracies = [rf_accuracy, svr_accuracy, nb_accuracy, eb_accuracy]

plt.figure(figsize=(10, 6))
```

```python
plt.bar(models, accuracies)
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.show()
```

This comprehensive breakdown should help you understand each part of the process. Let me know if you have further questions or need clarification on any specific section!