DATA Mining (Phase 5)

Team Name: Trio-Chargers

Team Members:

1. Sumanth Reddy (Team Head) (sdesi4@unh.newhaven.edu)
2. Lakshmi Kara Gupta (lchan3@unh.newhaven.edu)
3. Uday Kiran Karumburi Arumugam (ukaru1@unh.newhaven.edu)

Research Question:

Account Security: Detecting spam or legit, unusual login behavior and unauthorized access attempts through Machine Learning Algorithms.

Selected Data Set:

1. https://www.kaggle.com/datasets/khajahussainsk/facebook-spam-dataset

   The dataset can be used for building machine learning models. To collect the dataset, Facebook API and Facebook Graph API are used and the data is collected from public profiles by Kaggle.

List of Data Mining Techniques used:

- K nearest neighbors
- Support vector machines
- Decision trees
- Logistic regression

K-Nearest Neighbors (K-NN):

K-Nearest Neighbors is a type of instance-based learning, or lazy learning, where the function is only approximated locally, and all computation is deferred until function evaluation. It is a non-parametric method used for classification and regression.

Support Vector Machines (SVM):

Support Vector Machines are a set of supervised learning methods used for classification, regression, and outliers' detection. They are effective in high dimensional spaces and best suited for problems with complex domains where there are clear margins of separation in the data. SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable.

Decision Trees:

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Logistic Regression:

Logistic Regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables.

Description parameters and hyperparameters:

Model	Parameter	Hyperparameter
K-Nearest Neighbors	Points	n_neighbors (K), weights
Support Vector Machines	Support Vectors Weights (Coefficients)	Kernel gamma
Decision Trees	Tree Structure Feature Weights	max_depth min_samples_split
Logistic Regression	Weights/Coefficients Bias/Intercept	penalty solver

Brief description of hardware used:

- Processor: i7
- Hard Drive: 1TB
- RAM: 16GB
- OS: Windows 11
- Tools: Google Collab
- Language: Python

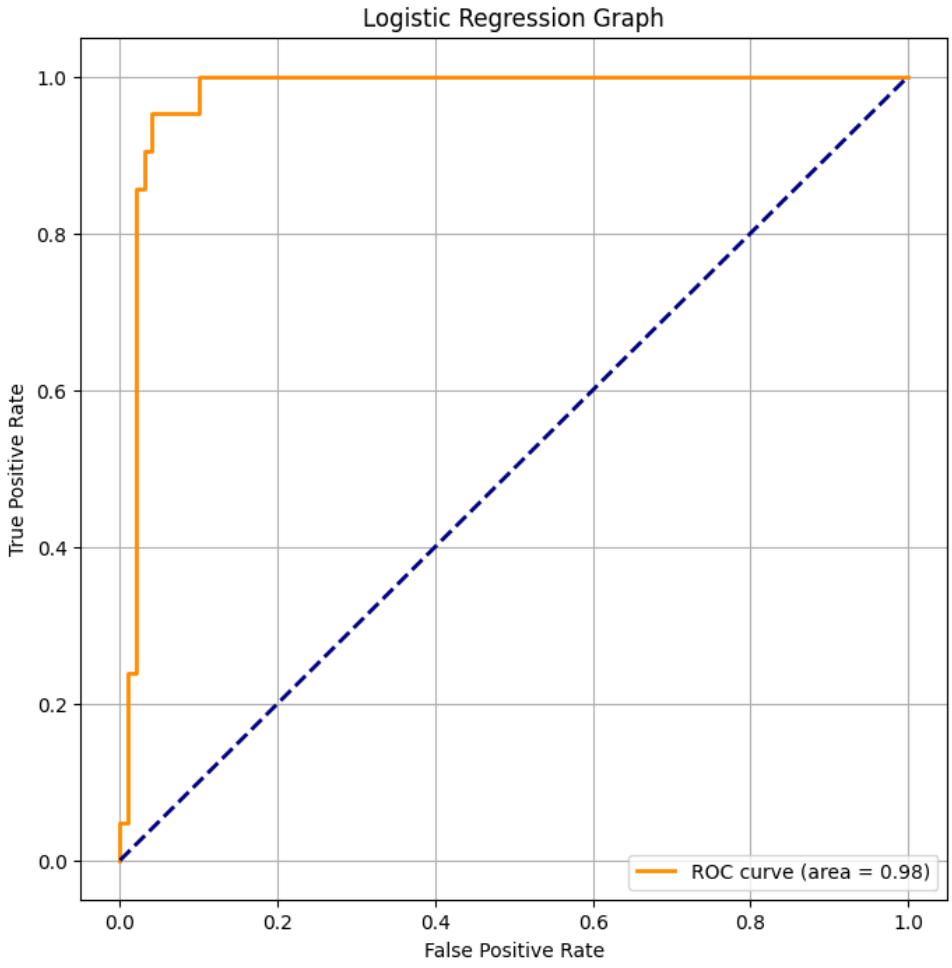
Logistic Regression

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
from sklearn.impute import SimpleImputer

df = pd.read_csv('/content/Facebook Spam Dataset.csv')

imputer = SimpleImputer(strategy='mean')
df['fpurls'] = imputer.fit_transform(df['fpurls'].values.reshape(-1, 1))

X = df.drop(['profile id', 'Label'], axis=1)
y = df['Label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

log_reg = LogisticRegression(random_state=42, max_iter=1000)

log_reg.fit(X_train_scaled, y_train)

y_pred = log_reg.predict(X_test_scaled)
print('Logistic Regression')
print('-----------------------------------')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('\nClassification Report:\n', classification_report(y_test, y_pred))
```

```
Logistic Regression
-----------------------------------
Accuracy: 0.9583333333333334

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.97        99
           1       0.83      0.95      0.89        21

    accuracy                           0.96       120
   macro avg       0.91      0.96      0.93       120
weighted avg       0.96      0.96      0.96       120
```

Graph

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)
y_scores = log_reg.decision_function(X_test_scaled)

fpr, tpr, thresholds = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)


plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression Graph')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



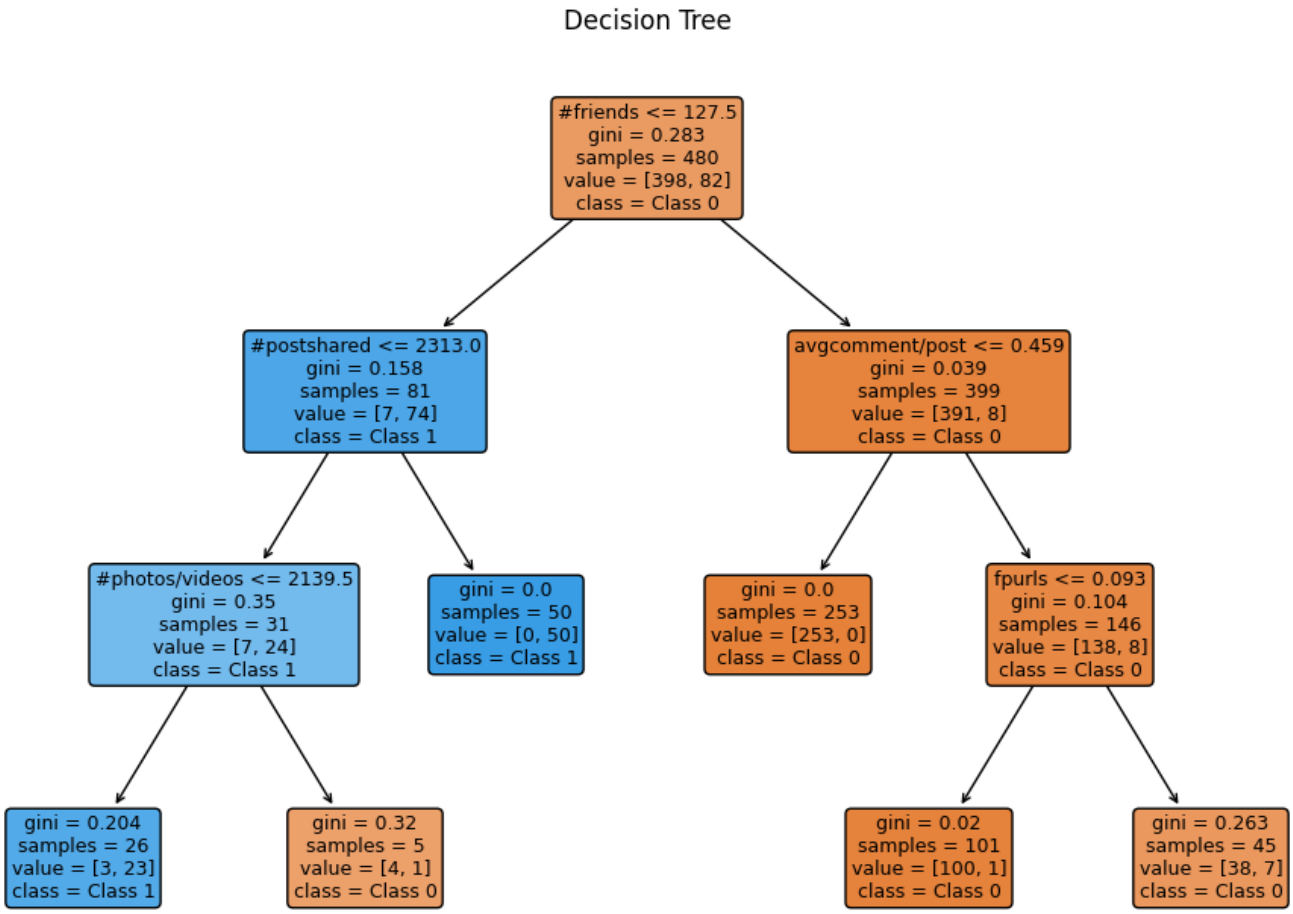
Decision Tree

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.impute import SimpleImputer

df = pd.read_csv('/content/Facebook Spam Dataset.csv')

imputer = SimpleImputer(strategy='mean')
df['fpurls'] = imputer.fit_transform(df['fpurls'].values.reshape(-1, 1))

X = df.drop(['profile id', 'Label'], axis=1)
y = df['Label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

decision_tree = DecisionTreeClassifier(random_state=42)

decision_tree.fit(X_train_scaled, y_train)

y_pred = decision_tree.predict(X_test_scaled)

print('Decision Tree')
print('-----------------------------------')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('\nClassification Report:\n', classification_report(y_test, y_pred))
```

```
Decision Tree
-----------------------------------
Accuracy: 0.9333333333333333

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.94      0.96        99
           1       0.76      0.90      0.83        21

    accuracy                           0.93       120
   macro avg       0.87      0.92      0.89       120
weighted avg       0.94      0.93      0.94       120
```

Graph

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Assume X_train and y_train are already defined
dt = DecisionTreeClassifier(max_depth=3)
dt.fit(X_train, y_train)

plt.figure(figsize=(12, 8))
plot_tree(dt, filled=True, feature_names=X_train.columns, class_names=['Class 0', 'Class 1'], rounded=True)
plt.title('Decision Tree')
plt.show()
```



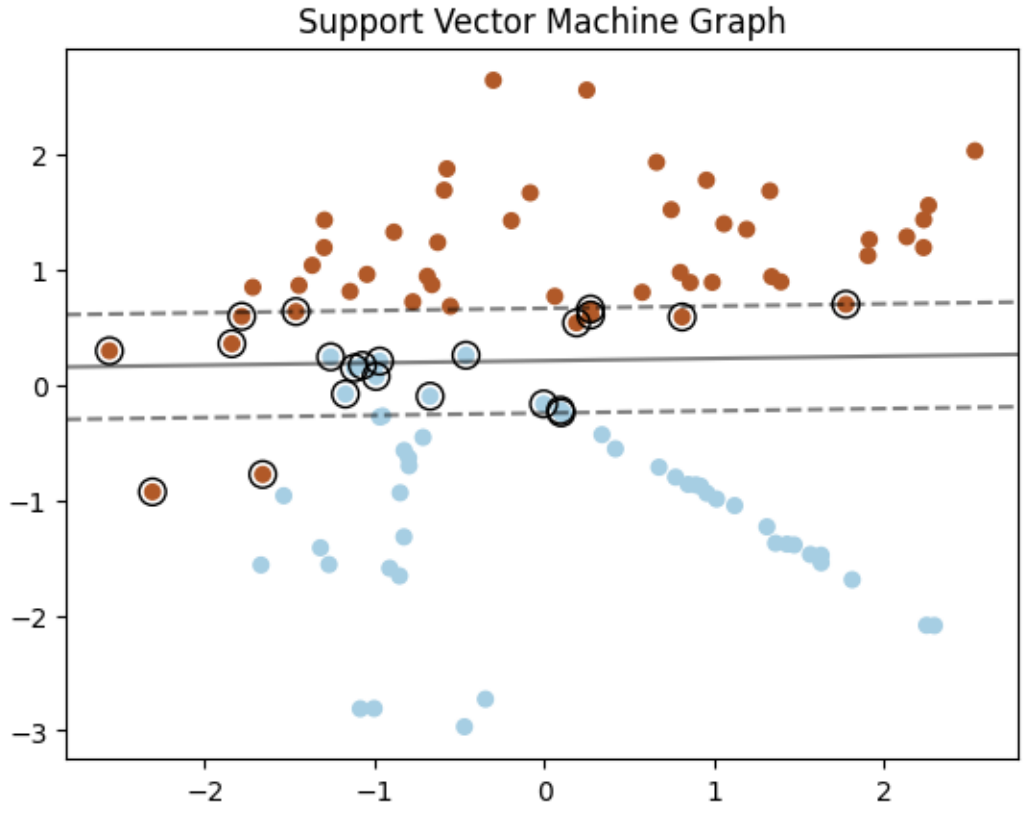
Support Vector Machine

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.impute import SimpleImputer

df = pd.read_csv('/content/Facebook Spam Dataset.csv')

imputer = SimpleImputer(strategy='mean')
df['fpurls'] = imputer.fit_transform(df['fpurls'].values.reshape(-1, 1))

X = df.drop(['profile id', 'Label'], axis=1)
y = df['Label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

svc = SVC(random_state=42)

svc.fit(X_train_scaled, y_train)

y_pred = svc.predict(X_test_scaled)

print('Support Vector Machine')
print('-----------------------------------')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('\nClassification Report:\n', classification_report(y_test, y_pred))
```

```
Support Vector Machine
-----------------------------------
Accuracy: 0.9583333333333334

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.97        99
           1       0.83      0.95      0.89        21

    accuracy                           0.96       120
   macro avg       0.91      0.96      0.93       120
weighted avg       0.96      0.96      0.96       120
```

Graph

```
import seaborn as sns
from sklearn.svm import SVC
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0, random_state=4)
clf = SVC(kernel='linear')
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 50),
                     np.linspace(ylim[0], ylim[1], 50))
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

ax.contour(xx, yy, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1, facecolors='none', edgecolors='k')
plt.show()
```



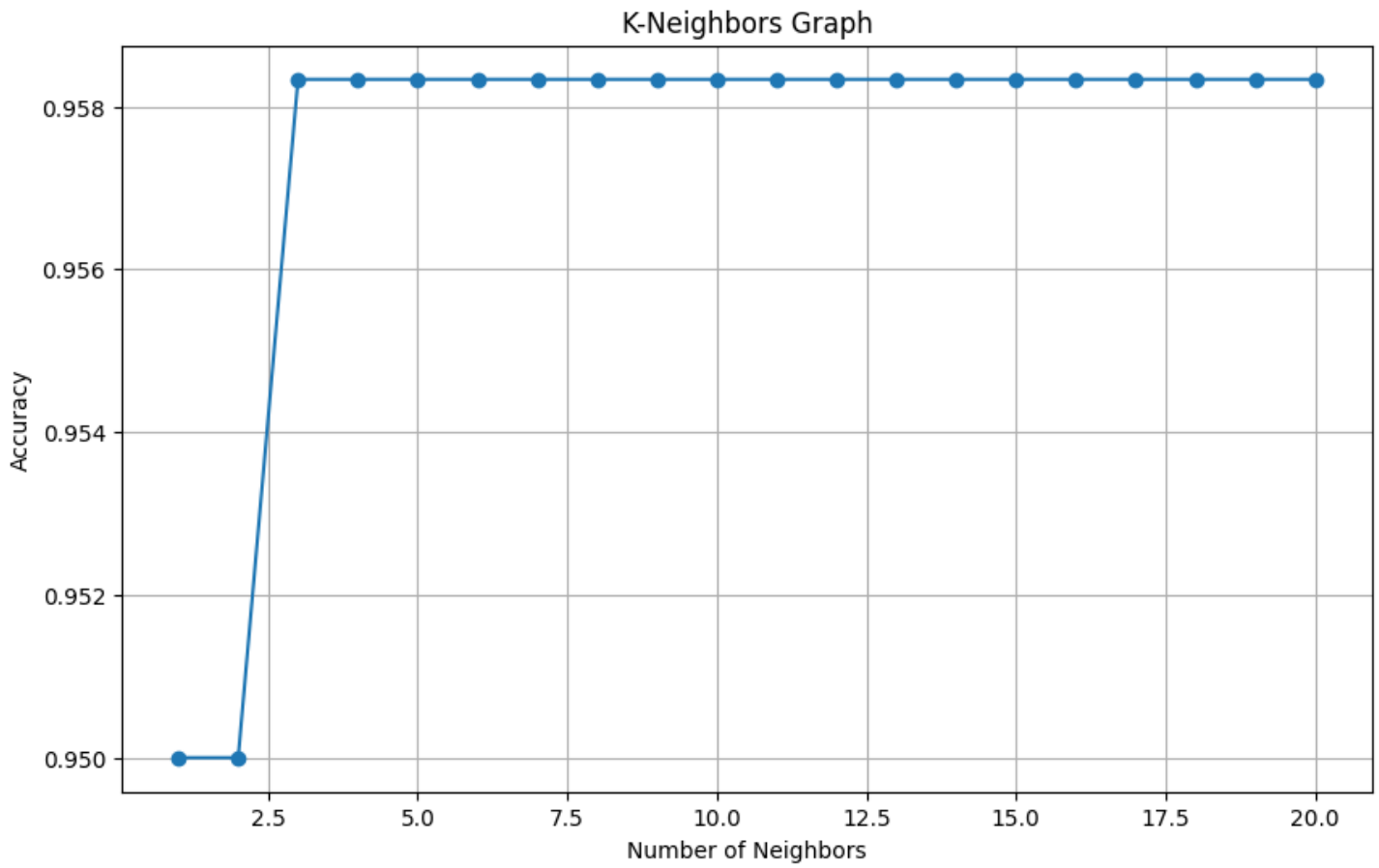
K-Neighbor

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.impute import SimpleImputer

df = pd.read_csv('/content/Facebook Spam Dataset.csv')

imputer = SimpleImputer(strategy='mean')
df['fpurls'] = imputer.fit_transform(df['fpurls'].values.reshape(-1, 1))

X = df.drop(['profile id', 'Label'], axis=1)  # Dropping 'profile id' as it seems to be an identifier
y = df['Label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train_scaled, y_train)

y_pred = knn.predict(X_test_scaled)

print('K-Nearest Neighbors')
print('-----------------------------------')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('\nClassification Report:\n', classification_report(y_test, y_pred))
```

```
K-Nearest Neighbors
-----------------------------------
Accuracy: 0.9583333333333334

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.97        99
           1       0.83      0.95      0.89        21

    accuracy                           0.96       120
   macro avg       0.91      0.96      0.93       120
weighted avg       0.96      0.96      0.96       120
```

Graph

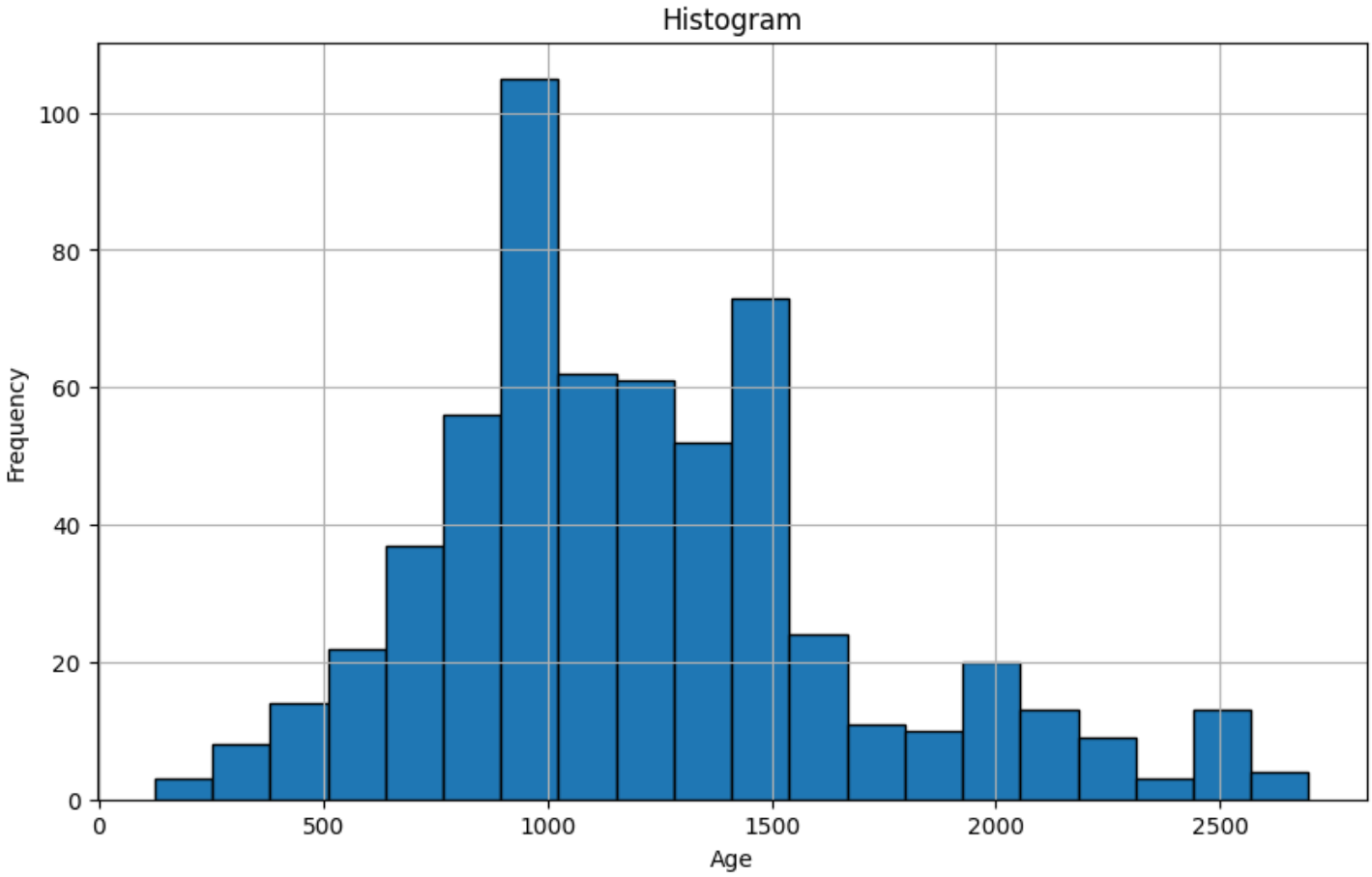
```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

accuracy_scores = []
neighbors = range(1, 21)

for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    accuracy_scores.append(knn.score(X_test_scaled, y_test))

plt.figure(figsize=(10, 6))
plt.plot(neighbors, accuracy_scores, marker='o')
plt.title('KNN: Accuracy vs. Number of Neighbors')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```



Visualization techniques used:

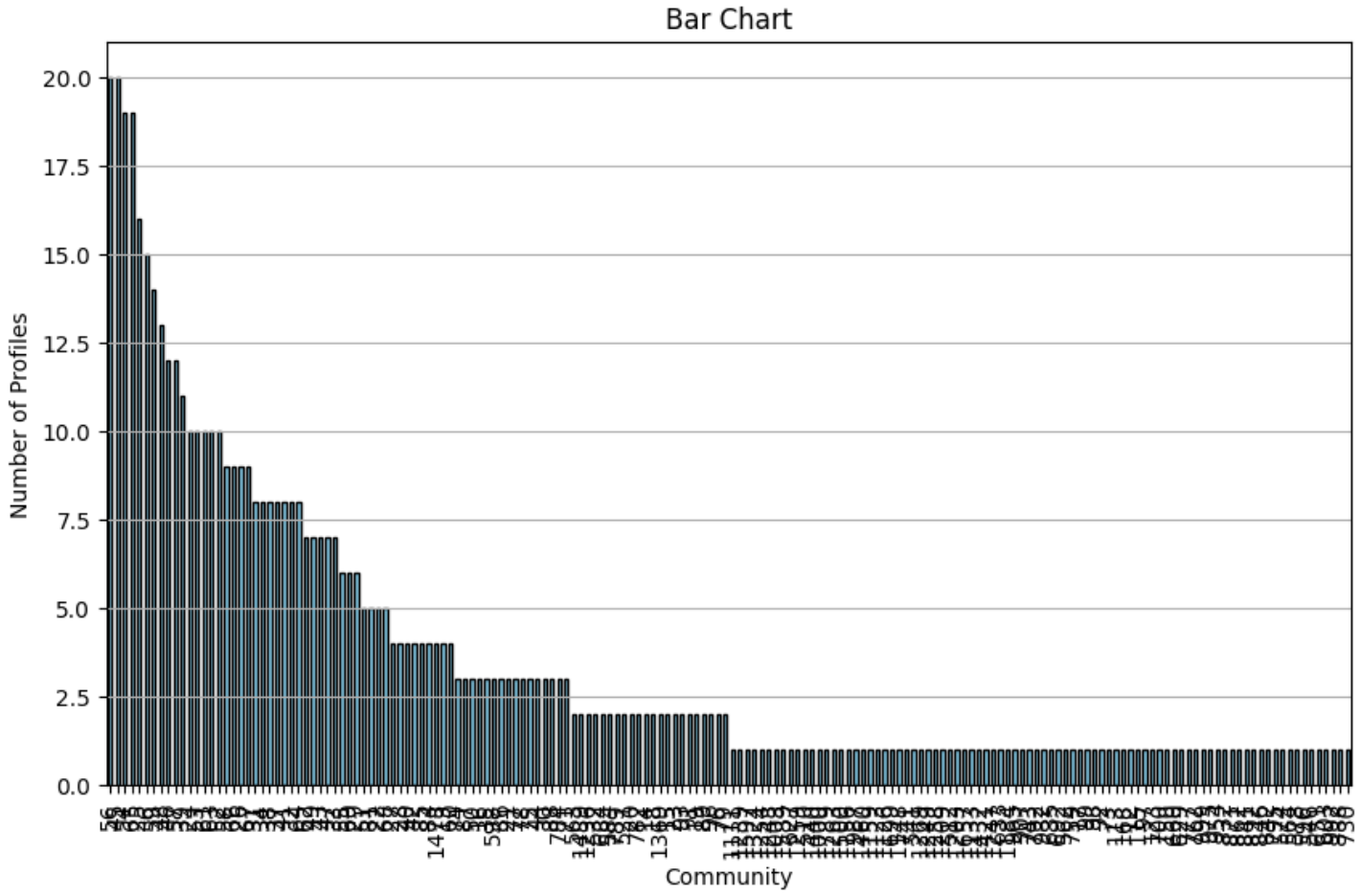
Histogram Graph

```
# Histogram for Age Distribution
plt.figure(figsize=(10, 6))
plt.hist(df['age'], bins=20, edgecolor='black')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



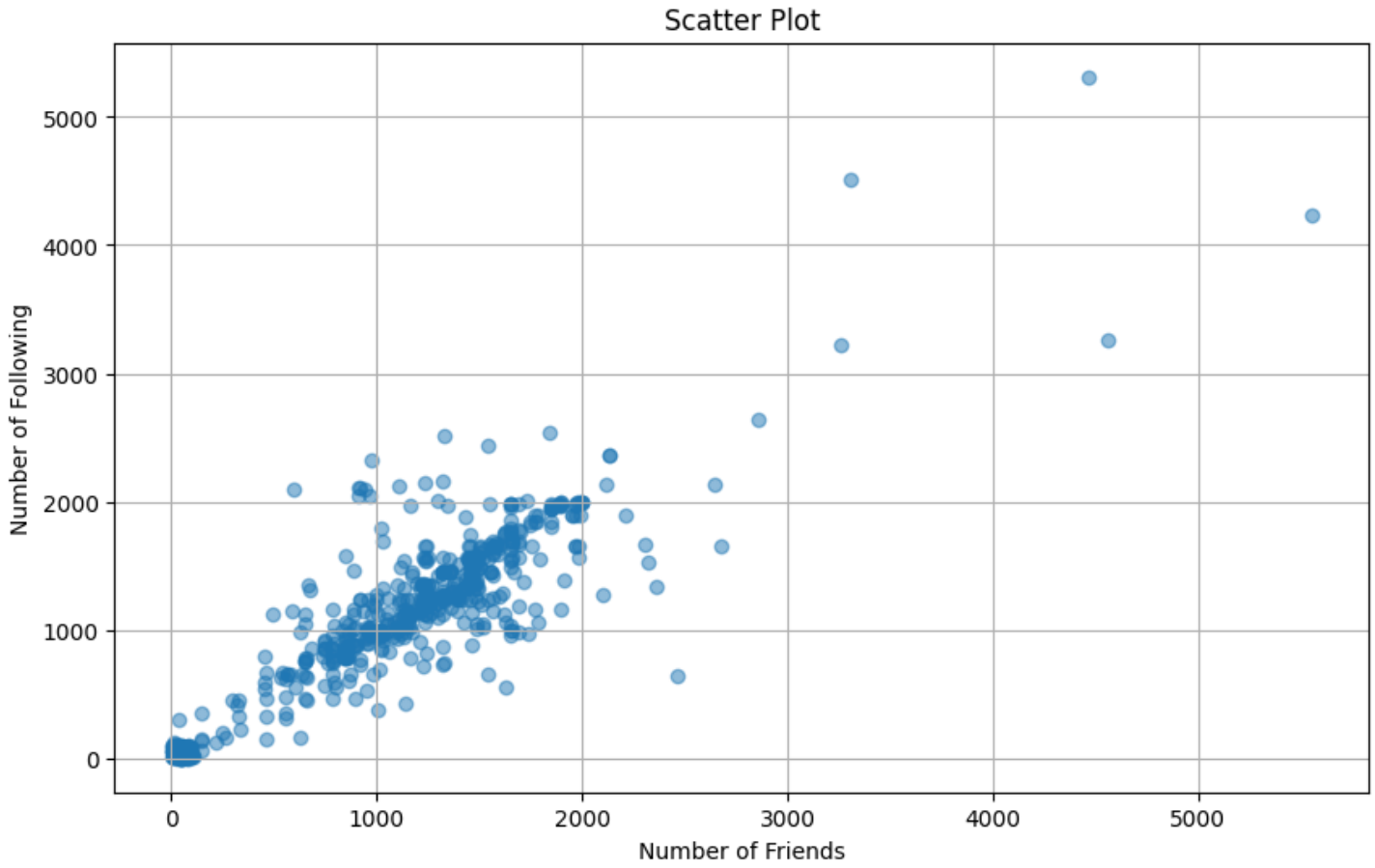
Bar Chart Graph

```
# Bar Chart for Community Counts
community_counts = df['#community'].value_counts()
community_counts.plot(kind='bar', figsize=(10, 6), color='skyblue', edgecolor='black')
plt.title('Community Counts')
plt.xlabel('Community')
plt.ylabel('Number of Profiles')
plt.grid(axis='y')
plt.show()
```



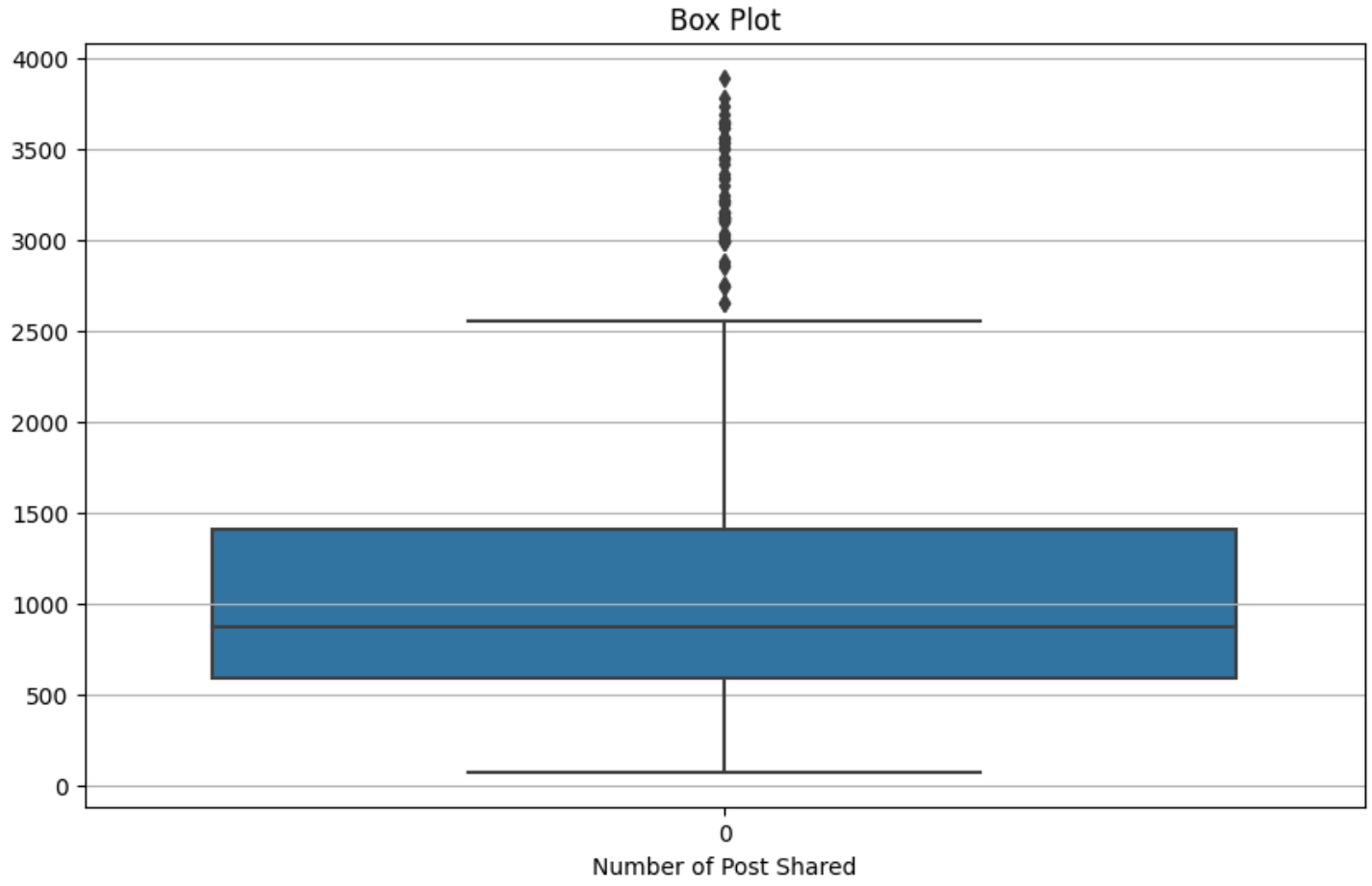
Scatter Plot Graph

```
# Scatter Plot for Friends vs Following
plt.figure(figsize=(10, 6))
plt.scatter(df['#friends'], df['#following'], alpha=0.5)
plt.title('Friends vs Following')
plt.xlabel('Number of Friends')
plt.ylabel('Number of Following')
plt.grid(True)
plt.show()
```



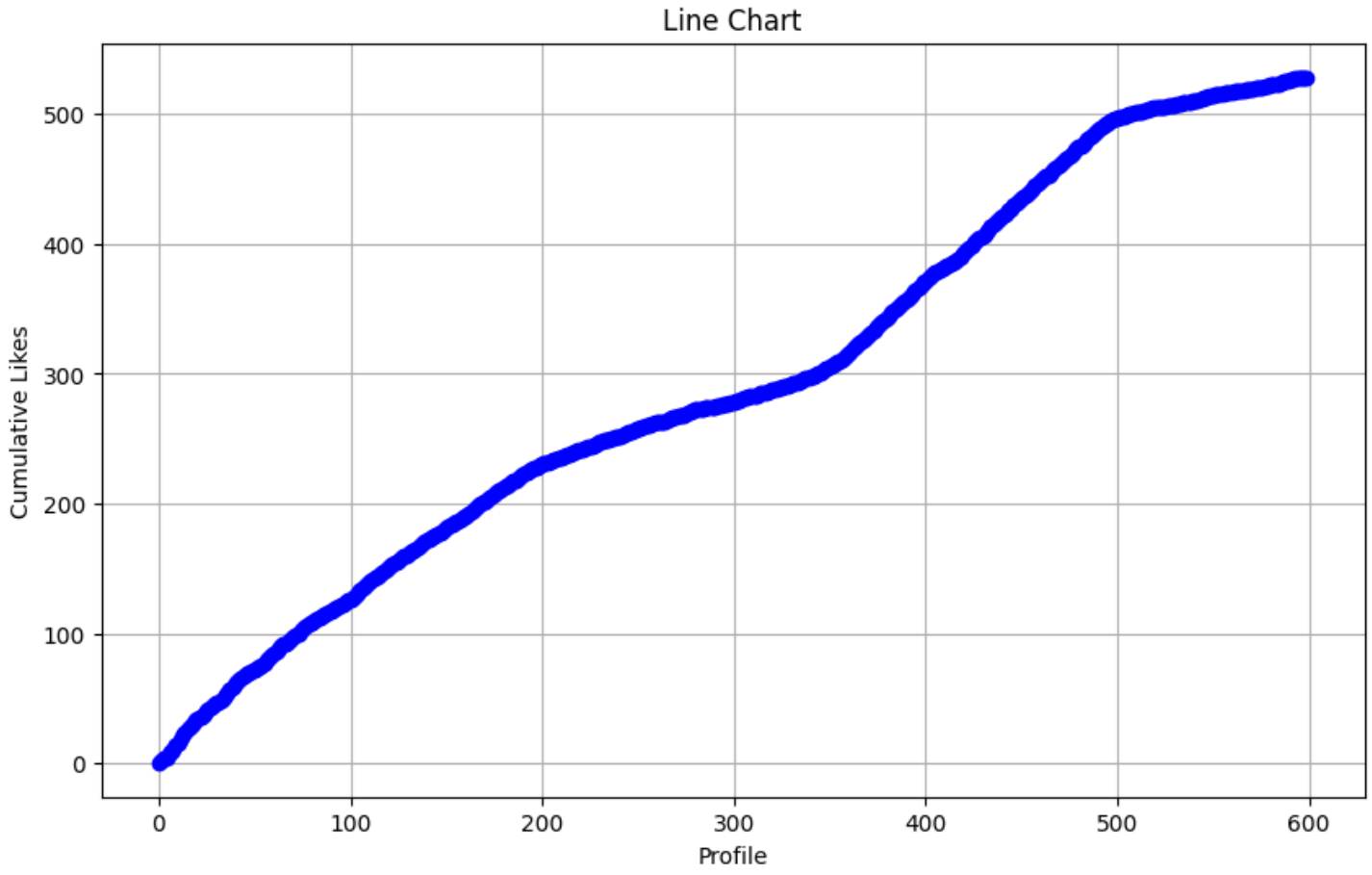
Box Plot Graph

```
# Box Plot for Post Shared
plt.figure(figsize=(10, 6))
sns.boxplot(df['#postshared'])
plt.title('Distribution of Post Shared')
plt.xlabel('Number of Post Shared')
plt.grid(axis='y')
plt.show()
```



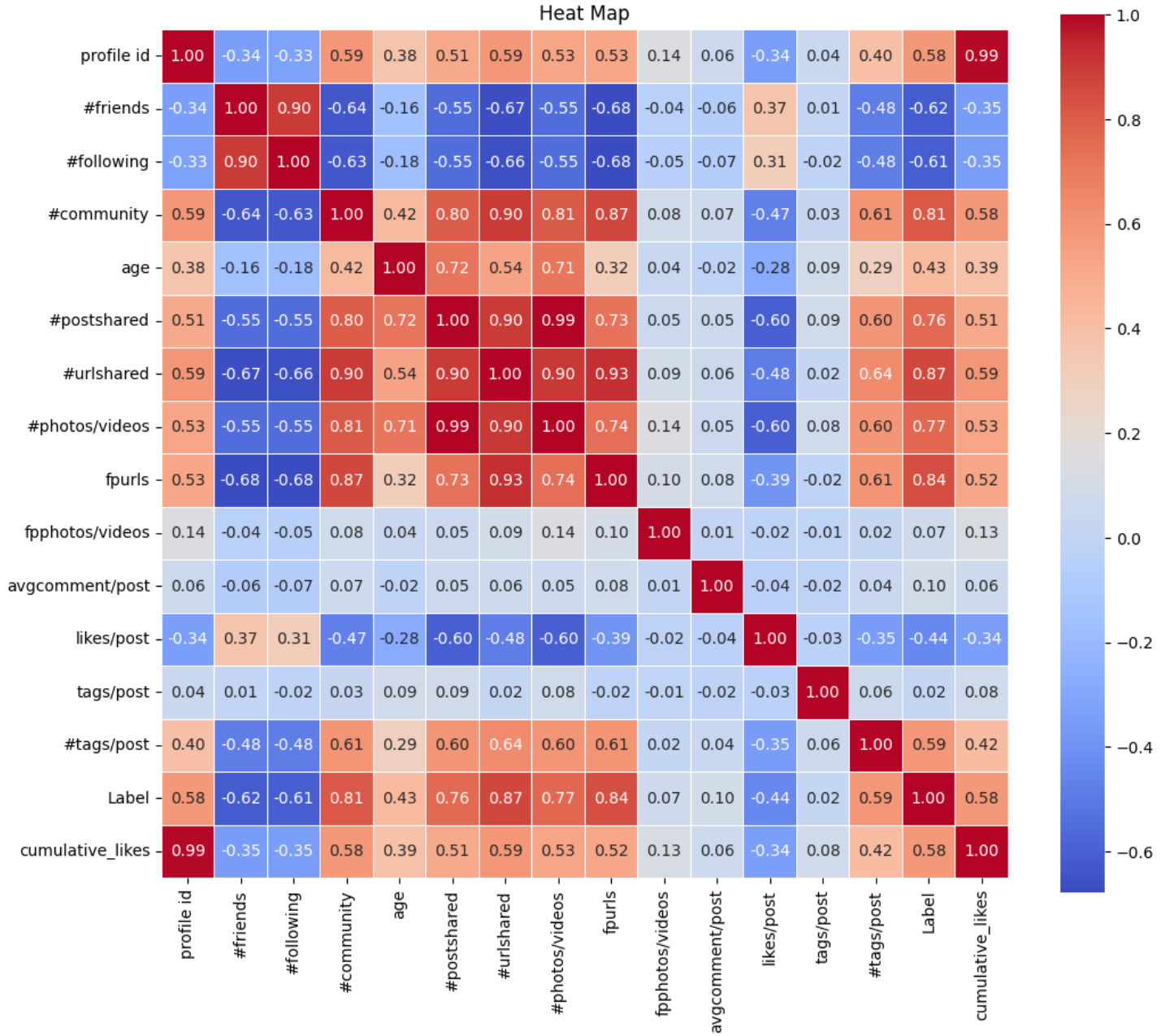
Line Chart Graph

```
# Line Chart for Cumulative Likes per Post
df['cumulative_likes'] = df['likes/post'].cumsum()
plt.figure(figsize=(10, 6))
plt.plot(df['cumulative_likes'], marker='o', linestyle='-', color='b')
plt.title('Cumulative Likes per Post')
plt.xlabel('Profile')
plt.ylabel('Cumulative Likes')
plt.grid(True)
plt.show()
```



Heat Map

```
import seaborn as sns

# Heatmap for Correlation Matrix
corr_matrix = df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True, linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```



Data Modeling Conclusion:

1. **Preprocessing:** The dataset has to be adequately preprocessed before any machine learning models are used. This involves dealing with missing values, which in this instance might be fixed by imputation, the median, or the mean. Additionally, feature scaling could be required, particularly for algorithms that depend on the size of the input features, such Support Vector Machines and K-Nearest Neighbors.

2. **Feature Selection:** A number of characteristics are present in the dataset, however not all of them could be equally important for the classification job. To save the most important characteristics, dimensionality reduction methods (like PCA) or feature selection might be used, which could enhance the model's functionality and lessen overfitting.

3. **Model Selection:** The choice of the model depends on the nature of the data and the specific requirements of the task. For instance:

    a. **Logistic Regression** could be a good starting point due to its simplicity and ease of interpretation. However, its linear nature might limit its performance if the relationship between features and the target variable is complex.
    b. **Decision Trees** and their ensembles (like Random Forest) can capture non-linear relationships and are also interpretable, but they can be prone to overfitting, especially with deep trees.
    c. **Support Vector Machines** can perform well for both linear and non-linear classification tasks, depending on the choice of the kernel.
    d. **K-Nearest Neighbors** is a lazy learner that makes predictions based on the majority class of its nearest neighbors. It's simple and intuitive but can become computationally expensive and performs poorly with high-dimensional data.

4. **Conclusion:** In the end, the model and its parameters should be selected after carefully examining the data, the issue at hand, and the trade-offs between predictability, interpretability, and simplicity. It can also be required to continuously monitor and update the model in order to preserve its accuracy and relevance over time.

GitHub Link: https://github.com/sumanthreddy8910/Phase_5.git