19CSE304 - FOUNDATIONS OF DATA SCIENCE

BLOG WRITING

SOCIAL MEDIA ENGAGEMENT METRICS SYNTHETIC DATASET GENERATION(EDA)

Prepared by :

PURNIMA RANGAVAJJULA

AM.EN.U4CSE21046

In this blog

Data analysis involves different processes of cleaning, transforming, analyzing the data, and building models to extract specific, relevant insights.

Importance of EDA in Data Science

The Data Science field is now very important in the business world as it provides many opportunities to make vital business decisions by analyzing hugely gathered data. Understanding the data thoroughly needs its exploration from every aspect. The impactful features enable making meaningful and beneficial decisions; therefore, EDA occupies an invaluable place in Data science.

Role of EDA in Data Science

The role of data exploration analysis is based on the use of objectives achieved as above. After formatting the data, the performed analysis indicates patterns and trends that help to take the proper actions required to meet the expected goals of the business. Therefore, carrying out the right EDA with the correct tool based on befitting data will help achieve the expected goal.

Steps Involved in Exploratory Data Analysis (EDA)

The key components in an EDA are the main steps undertaken to perform the EDA. These are as follows:

1. Data Collection- Social Media Engagement Metrics

2. Finding all Variables and Understanding Them

When the analysis process starts, the first focus is on the available data that gives a lot of information.

- user_id (int): User identifier.
- post_type (object): Type of post (text, image, video).
- post_length (float): Length of the post in characters (with some NaN values).
- likes (int): Number of likes received.
- comments (int): Number of comments received.
- shares (int): Number of shares received.
- engagement_rate (float): Overall engagement rate for the post (with some NaN values).
- user_followers (int): Number of followers the user has.
- post_category (object): Category of the post (sports, technology, fashion, food).
- post_hour (int): Hour of the day when the post was made.
- is_weekend (int): Binary indicator if the post was made on the weekend (0 or 1).
- user_verified (int): Binary indicator if the user is verified (0 or 1).
- spam_flag (int): Binary indicator if the post is flagged as spam (0 or 1).

3. Cleaning the Dataset

The next step is to clean the data set, which may contain null values and irrelevant information. These are to be removed so that data contains only those values that are relevant and important from the target point of view. Preprocessing takes care of all issues, such as identifying null values, outliers, anomaly detection, etc.

4. Identify Correlated Variables

Finding a correlation between variables helps to know how a particular variable is related to another. The correlation matrix method gives a clear picture of how different variables correlate, which further helps in understanding vital relationships among them.

5. Choosing the Right Statistical Methods

Statistical formulae applied for numerical outputs give fair information, but graphical visuals are more appealing and easier to interpret.

6. Visualizing and Analyzing Results

Once the analysis is over, the findings are to be observed cautiously and carefully so that proper interpretation can be made. The trends in the spread of data and correlation between variables give good insights for making suitable changes in the data parameters.

Exploratory Data Analysis Tools

1. Python

Python is used for different tasks in EDA, such as finding missing values in data collection, data description, handling outliers, obtaining insights through charts, etc. The syntax for EDA libraries like Matplotlib, Pandas, Seaborn, NumPy, Altair, and more in Python is fairly simple and easy to use for beginners.

2. MATLAB

MATLAB is a well-known commercial tool among engineers since it has a very strong mathematical calculation ability. Due to this, it is possible to use MATLAB for EDA but it requires some basic knowledge of the MATLAB programming language.

Advantages of Using EDA

Here are a few advantages of using Exploratory Data Analysis -

1. Gain Insights Into Underlying Trends and Patterns

EDA assists data analysts in identifying crucial trends quickly through data visualizations using various graphs, such as box plots and histograms.

2. Improved Understanding of Variables

Using EDA, they can extract various information such as averages, means, minimum and maximum, and more such information is required for preprocessing the data appropriately.

3. Better Preprocess Data to Save Time

EDA can assist data analysts in identifying significant mistakes, abnormalities, or missing values in the existing dataset..

4. Make Data-driven Decisions

The most significant advantage of employing EDA in an organization is that it helps businesses to improve their understanding of data. With EDA, they can use the available tools to extract critical insights and make conclusions, which assist in making decisions based on the insights from the EDA.

Example of Exploratory Data Analysis

EDA in Social Media Engagement

The provided code generates a synthetic dataset for social media engagement metrics and saves it to a CSV file. The features include both categorical and numerical variables, and it's suitable for creating a supervised learning model where you can predict a specific label or target variable. In this case, you can consider the following:

Target Variable:
1. Regresion: Predicting Engagement rate
Label: engagement_rate
Objective: Predict based on various features.

2. Claasification: Identifying Spam posts
Label: spam_flag
Objective: Classify whether a social media post is spam or not based on its features.

3. Multi class Classification
Label: post_category
Objective:Classify the category of a post based on its features.

Feature Ranges:

Categorical Features:

'post_type': ['text', 'image', 'video']
'post_category': ['sports', 'technology', 'fashion', 'food']
'is_weekend': [0, 1]
'user_verified': [0, 1]
'spam_flag': [0, 1]

Numerical Features:

'post_length': [50, 300]
'likes': [0, 500]
'comments': [0, 100]
'shares': [0, 50]
'engagement_rate': [0.5, 5.0]
'user_followers': [100, 10000]
'post_hour': [0, 24]

Handling NaN Values:

Some records have NaN values in 'post_length' and 'engagement_rate'. You may need to handle these missing values appropriately, depending on the chosen machine learning algorithm. Options include removing those records, filling them with mean/median values, or using more advanced imputation techniques.

Data Exploration:

Before applying supervised learning, it's advisable to explore and visualize the dataset to understand the distribution of features, relationships, and potential patterns.

“Synthesizing Insights: Exploratory Analysis of Social Media Engagement Metrics"

#EDA #Datavisualisation #Python #Datascience #Pandas #Numpy #Matplot #Seaborn

Overview

- **Step by Step approach to Perform EDA**
- **Getting familiar with various Data Visualization techniques, charts, plots**
- **Demonstration of some steps with Python Code Snippet**

1.What Is Exploratory Data Analysis

Exploratory Data Analysis is an approach in analyzing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods.

Note: The dataset in this blog is being opted as Social Media Engagement Metrics dataset

2. Checking Introductory Details About Data

The first and foremost step of any data analysis, after loading the data file, should be about checking few introductory details like, no. Of columns, no. of rows, types of features(categorical or Numerical), data types of column entries.

```
In [8]: df=pd.read_excel('/content/mediarates.xlsx')

In [9]: #Introductory Details About Data

In [10]: df.shape
Out[10]: (2000, 13)

In [11]: df.size
Out[11]: 26000
```

3. Statistical Insights

This step should be performed for getting details about various statistical data like Mean, Standard Deviation, Median, Max Value, Min Value

```
In [34]: #summary statistics

In [35]: df.mean()
<ipython-input-35-c61f0c8f89b5>:1: FutureWarning: The default value of numeric_only in DataFrame.mean is deprec
ic_only=None' is deprecated. Select only valid columns or specify the value of numeric_only to silence this war
  df.mean()
Out[35]: user_id            1000.500000
post_length         174.162632
likes               247.492000
comments             49.600500
shares               24.164500
engagement_rate       2.693385
user_followers     5026.938000
post_hour            11.745000
is_weekend            0.486500
user_verified         0.518000
spam_flag             0.499500
dtype: float64
```

```
In [39]: df.min()

Out[39]: user_id                 1
         post_type             image
         post_length            50.0
         likes                     0
         comments                  0
         shares                    0
         engagement_rate    0.501085
         user_followers          107
         post_category       fashion
         post_hour                 0
         is_weekend                0
         user_verified             0
         spam_flag                 0
         dtype: object

In [40]: df.max()

Out[40]: user_id                 2000
         post_type              video
         post_length            299.0
         likes                    499
         comments                  99
         shares                    49
         engagement_rate     4.997573
         user_followers          9995
         post_category     technology
         post_hour                 23
         is_weekend                 1
         user_verified              1
         spam_flag                  1
         dtype: object
```

```
In [17]: df.describe(percentiles=[0.3,0.5,0.7])
```

Out[17]:

	user_id	post_length	likes	comments	shares	engagement_rate	user_followers	post_hour	is_weekend	user_verified	spam_flag
count	2000.000000	1900.000000	2000.000000	2000.000000	2000.000000	1900.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	1000.500000	174.162632	247.492000	49.600500	24.164500	2.693385	5026.938000	11.745000	0.486500	0.518000	0.499500
std	577.494589	71.251606	144.747846	28.628844	14.419862	1.289315	2879.222375	6.840349	0.499943	0.499801	0.500125
min	1.000000	50.000000	0.000000	0.000000	0.000000	0.501085	107.000000	0.000000	0.000000	0.000000	0.000000
30%	600.700000	127.000000	145.000000	29.000000	14.000000	1.793116	3008.600000	7.000000	0.000000	0.000000	0.000000
50%	1000.500000	176.000000	246.500000	50.000000	24.000000	2.683675	5096.500000	12.000000	0.000000	1.000000	0.000000
70%	1400.300000	221.000000	351.000000	70.000000	34.000000	3.535114	7029.600000	17.000000	1.000000	1.000000	1.000000
max	2000.000000	299.000000	499.000000	99.000000	49.000000	4.997573	9995.000000	23.000000	1.000000	1.000000	1.000000

4. Data cleaning

This is the most important step in EDA involving removing duplicate rows/columns, filling the void entries with values like mean/median of the data, dropping various values, removing null entries

```
In [25]: df_copy.isnull().head(7)
```

Out[25]:

	user_id	post_type	post_length	likes	comments	shares	engagement_rate	user_followers	post_category	post_hour	is_weekend	user_verified	spam_flag
0	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False	False	False	False

In [26]:
```
df_copy.isnull().sum()
```

Out[26]:
```
user_id              0
post_type            0
post_length        100
likes                0
comments             0
shares               0
engagement_rate    100
user_followers       0
post_category        0
post_hour            0
is_weekend           0
user_verified        0
spam_flag            0
dtype: int64
```

In [27]:
```
df_copy.isnull().sum().sum()
```

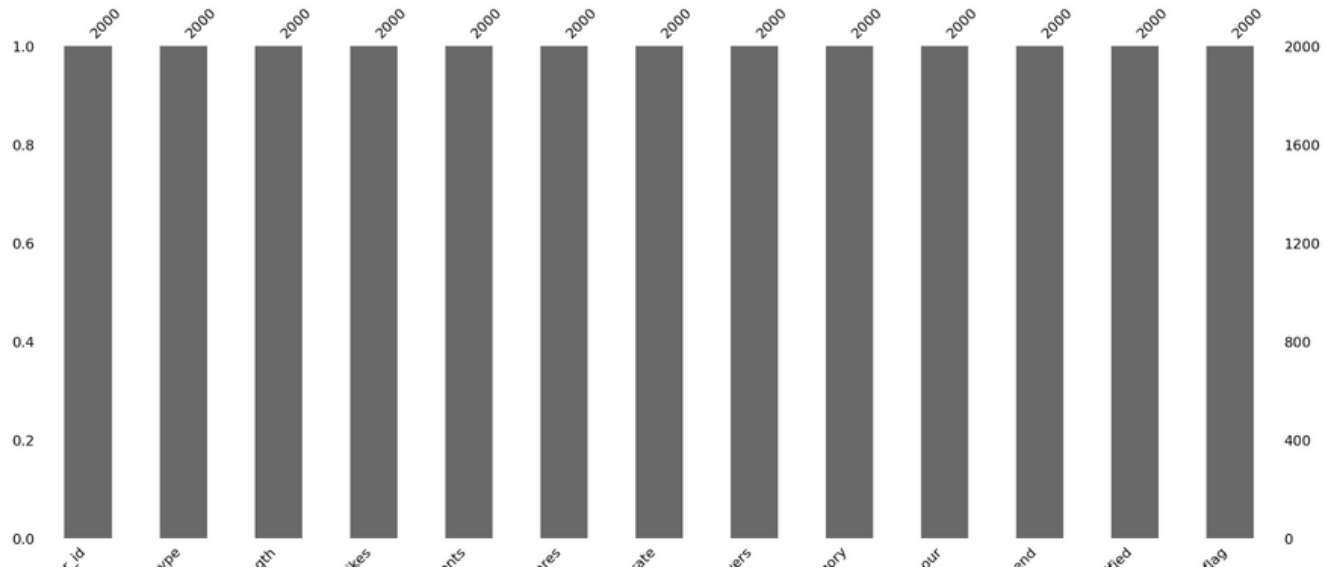
Out[27]:
```
200
```

Removing Null Entries

In [58]:
```
# Count of missing values in each column
missing_count = df.isnull().sum()
# Percentage of missing values in each column
missing_percentage = (missing_count / len(df)) * 100
# Create a DataFrame to display the results
missing_info = pd.DataFrame({
    'Missing Count': missing_count,
    'Missing Percentage': missing_percentage
})
# Display the missing values information
print(missing_info)
```

```
                Missing Count  Missing Percentage
user_id                     0                 0.0
post_type                   0                 0.0
post_length               100                 5.0
likes                       0                 0.0
comments                    0                 0.0
shares                      0                 0.0
engagement_rate           100                 5.0
user_followers              0                 0.0
post_category               0                 0.0
post_hour                   0                 0.0
is_weekend                  0                 0.0
user_verified               0                 0.0
spam_flag                   0                 0.0
```

```
# Impute missing values (using mean in this example)
df['post_length'].fillna(df['post_length'].mean(), inplace=True)
df['engagement_rate'].fillna(df['engagement_rate'].mean(), inplace=True)
# Visualize again after imputation

msno.bar(df)
plt.show()
```

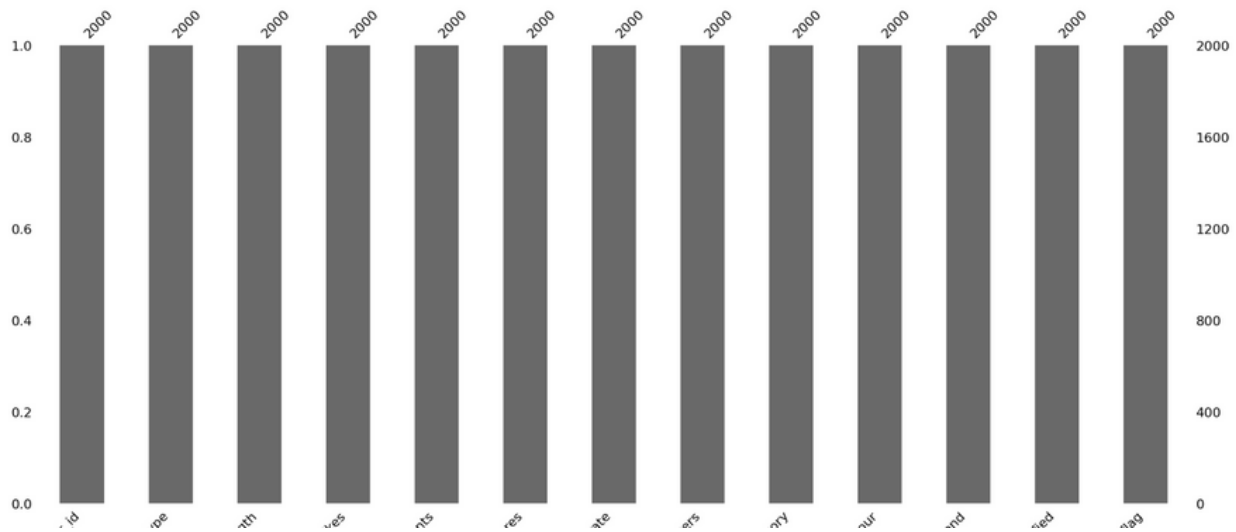


Imputation by Mean

In [60]:
```
import missingno as msno
# Impute missing values (using mean in this example)
df['post_length'].fillna(df['post_length'].mean(), inplace=True)
df['engagement_rate'].fillna(df['engagement_rate'].mean(), inplace=True)
# Visualize again after imputation
msno.matrix(df)
plt.show()
```

In [63]:
```
'''Imputation Method1: most common class'''
most_common_post_length = df['post_length'].mode()[0]
df['post_length'].fillna(most_common_post_length, inplace=True)
msno.bar(df)
```

Out[63]: <Axes: >



In [73]:
```
'''Imputation method 2: Unknown class'''
df['post_category'].fillna("Unknown", inplace=True)
# Display the first few rows of the dataset with imputed values
print(df.head())
```

```
   user_id post_type  post_length  likes  comments  shares  engagement_rate  \
0        1     video        287.0    306        52      35         4.457269
1        2      text        149.0    330        78      36         4.845244
2        3     video        222.0    460        78      40         2.037067
3        4     video        185.0    265         3      16         2.023281
4        5      text        270.0    397        95      34         0.531913

   user_followers post_category  post_hour  is_weekend  user_verified  \
0            9204       fashion          3           0              0
1             596        sports          7           0              1
2            1563          food         12           1              1
3            6928        sports          9           1              0
4            4353          food         17           1              1

   spam_flag
0          0
1          0
2          1
3          0
4          1
```

n [82]:
```
# One-hot encode categorical variables
df_cleaned = pd.get_dummies(df_cleaned, columns=['post_type', 'post_category'])
```

Removing Duplicates

```
duplicate_entries = df[df.duplicated()]

# Display any duplicate entries
print("Duplicate Entries:")
print(duplicate_entries)

# Check if there are any duplicates based on the 'user_id' column
are_there_duplicates = df['user_id'].duplicated().any()

# Display the result
print("\nAre there any duplicates based on 'user_id' column? ", are_there_duplicates)
```

```
Duplicate Entries:
Empty DataFrame
Columns: [user_id, post_type, post_length, likes, comments, shares, engagement_rate, user_followers, post_category, post_hour, is_weekend, user_verified, spa
Index: []

Are there any duplicates based on 'user_id' column?  False
```

5. Data Visualization

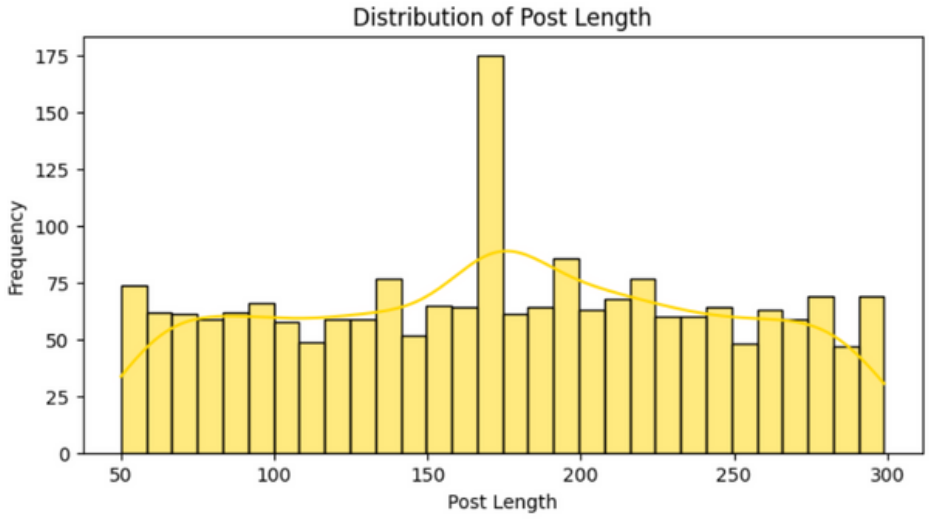
Data visualization is the method of converting raw data into a visual form, such as a map or graph, to make data easier for us to understand and extract useful insights.

The main goal of data visualization is to put large datasets into a visual representation. It is one of the important steps and simple steps when it comes to data science

HISTOGRAM ANALYSIS

In [111…
```
#Data Visualisation
```

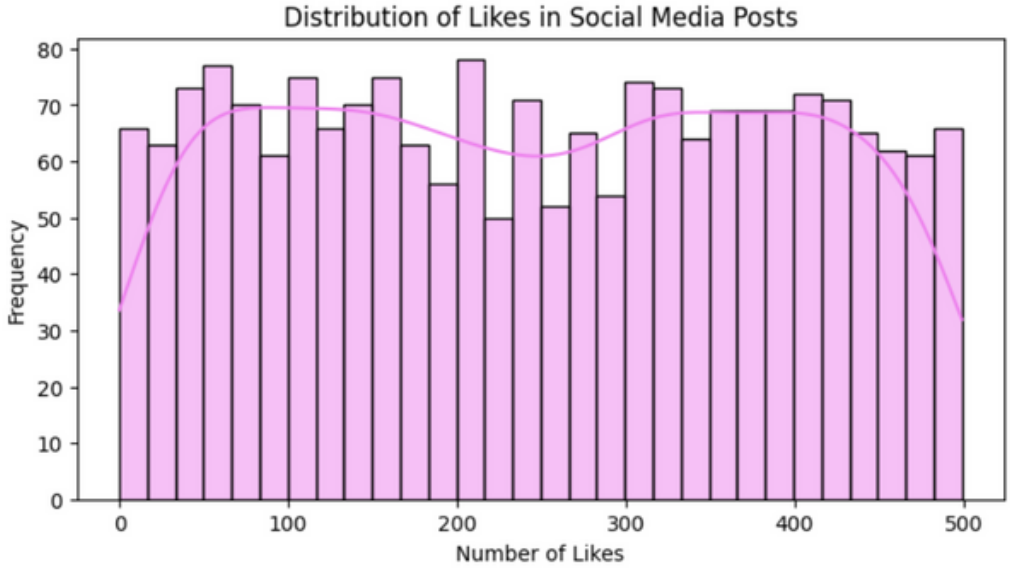
In [112…
```
''' distribution plot for 'post_length' using Seaborn'''
plt.figure(figsize=(8, 4))
sns.histplot(data=df, x='post_length', kde=True, bins=30, color='gold')
# Add title and labels
plt.title('Distribution of Post Length')
plt.xlabel('Post Length')
plt.ylabel('Frequency')
# Show the plot
plt.show()
```



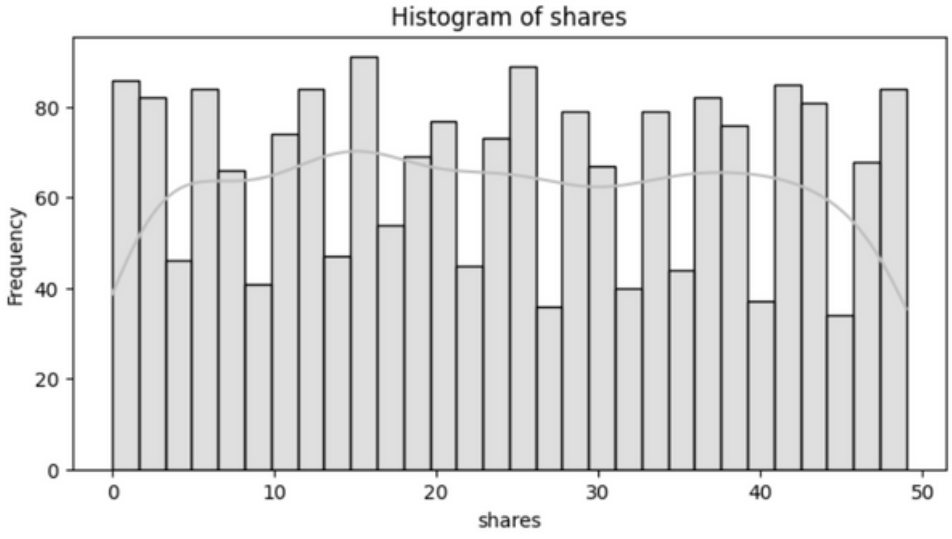
The resulting plot will show a histogram of the distribution of post lengths in your dataset. The x-axis represents the post lengths, the y-axis represents the frequency of each length, and the gold-colored bars represent the distribution. The kernel density estimate (kde) provides a smooth curve over the histogram, giving you an idea of the underlying probability density function.

In [114…

```
'''Create a histogram using Seaborn for the 'likes' column'''
plt.figure(figsize=(8, 4))
sns.histplot(data=df, x='likes', bins=30, kde=True, color='violet')
plt.title('Distribution of Likes in Social Media Posts')
plt.xlabel('Number of Likes')
plt.ylabel('Frequency')
plt.show()
```



The violet bars and the kde curve give an overview of how likes are distributed across the social media posts, helping to identify trends, concentration areas, and potential outliers. This visualization can be useful for understanding the engagement patterns in the dataset.



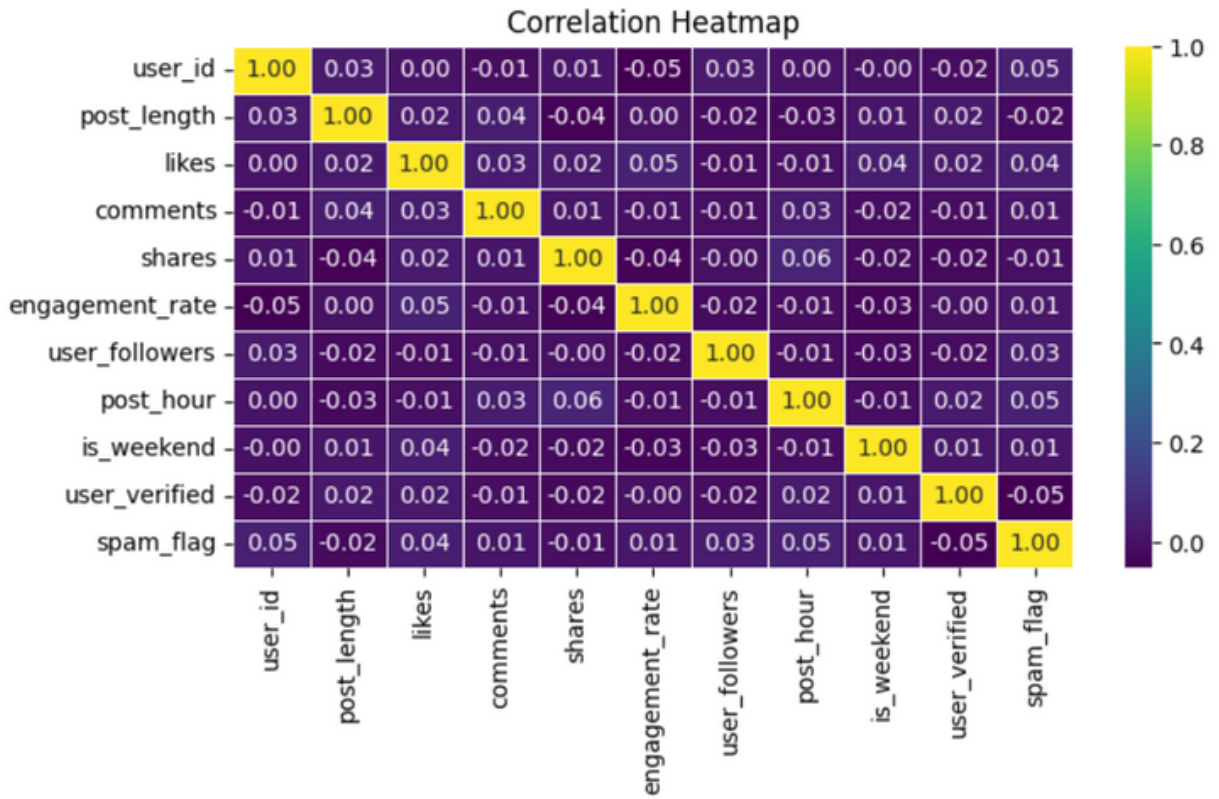
Overall, this code generates histograms for numerical columns in the DataFrame, providing visual representations of the distribution of each numerical feature. It helps in understanding the data's distribution and identifying potential patterns or outliers.

CORRELATION MATRIX

In [116…

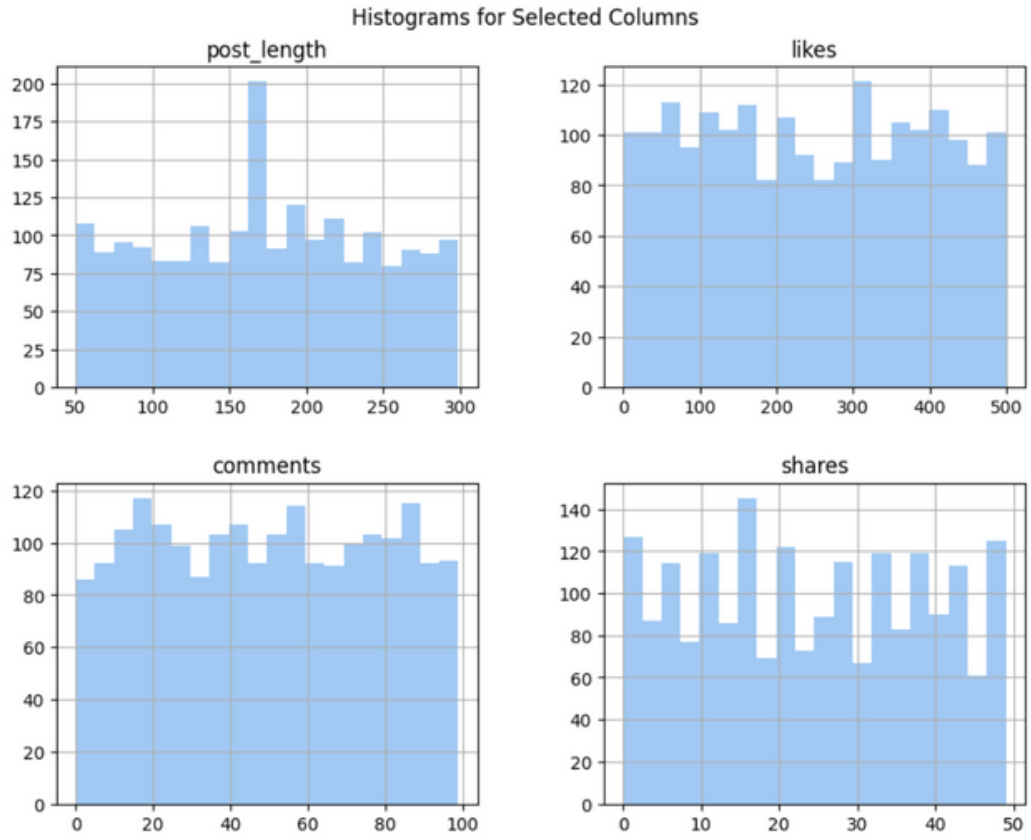
```
''' correlation matrix'''
correlation_matrix = df.corr()
print(correlation_matrix)
```

```
                  user_id  post_length     likes  comments    shares  \
user_id          1.000000     0.025051  0.003965 -0.014835  0.010452
post_length      0.025051     1.000000  0.021944  0.036424 -0.039131
likes            0.003965     0.021944  1.000000  0.031627  0.023215
comments        -0.014835     0.036424  0.031627  1.000000  0.010526
shares           0.010452    -0.039131  0.023215  0.010526  1.000000
engagement_rate -0.047880     0.000211  0.051246 -0.005097 -0.038686
user_followers   0.025697    -0.018255 -0.014305 -0.009352 -0.004852
post_hour        0.002892    -0.026320 -0.007245  0.031319  0.056979
is_weekend      -0.002593     0.007859  0.036591 -0.016682 -0.020752
user_verified   -0.016855     0.022812  0.021611 -0.011471 -0.022380
spam_flag        0.045393    -0.021084  0.035564  0.008808 -0.006197

                engagement_rate  user_followers  post_hour  is_weekend  \
user_id               -0.047880        0.025697   0.002892   -0.002593
post_length            0.000211       -0.018255  -0.026320    0.007859
likes                  0.051246       -0.014305  -0.007245    0.036591
comments              -0.005097       -0.009352   0.031319   -0.016682
shares                -0.038686       -0.004852   0.056979   -0.020752
engagement_rate        1.000000       -0.020561  -0.011509   -0.034704
user_followers        -0.020561        1.000000  -0.011823   -0.027729
post_hour             -0.011509       -0.011823   1.000000   -0.014026
is_weekend            -0.034704       -0.027729  -0.014026    1.000000
user_verified         -0.004024       -0.016374   0.024901    0.013986
spam_flag              0.014375        0.031458   0.045147    0.013978

                user_verified  spam_flag
user_id             -0.016855   0.045393
post_length          0.022812  -0.021084
likes                0.021611   0.035564
comments            -0.011471   0.008808
shares              -0.022380  -0.006197
engagement_rate     -0.004024   0.014375
user_followers      -0.016374   0.031458
post_hour            0.024901   0.045147
is_weekend           0.013986   0.013978
user_verified        1.000000  -0.050997
spam_flag           -0.050997   1.000000
```



This code creates a heatmap of the correlation matrix using Seaborn's heatmap function. It visually represents the correlation between different numeric features. The parameter annot=True displays the correlation values on the heatmap.

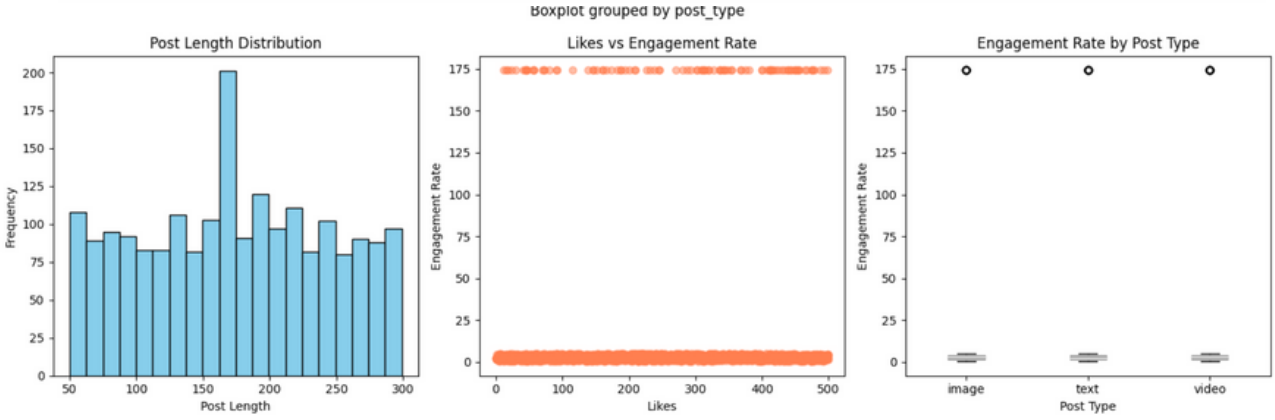
Positive correlation (values close to 1) implies that as one variable increases, the other also tends to increase. Negative correlation (values close to -1) implies that as one variable increases, the other tends to decrease. Values close to 0 indicate weak or no linear correlation.



Correlation Heatmap: The heatmap provides a visual representation of how different numeric features in the dataset are correlated. You can observe which features have a strong positive or negative correlation.

Histograms: The histograms give insights into the distribution of values in selected columns. For example, you can see the distribution of post length, likes, comments, shares, engagement rate, and user followers.

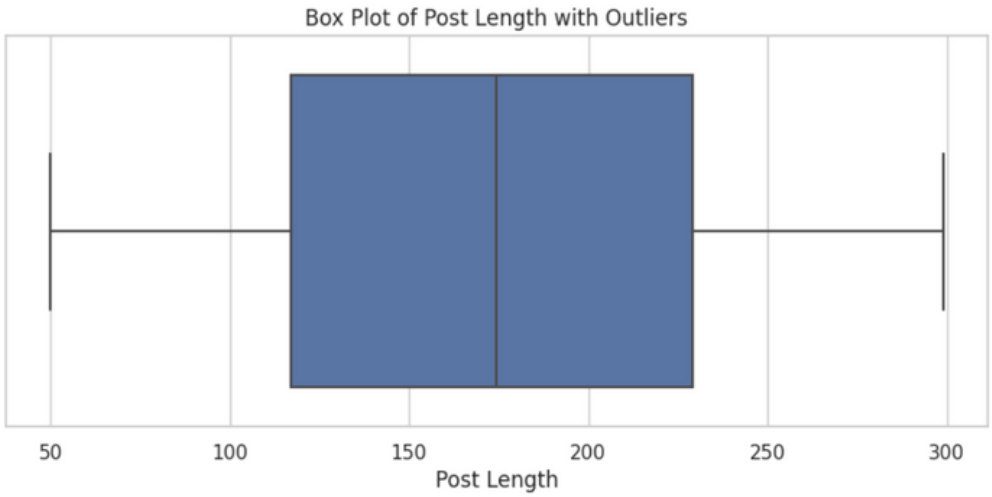
```
In [118…
''' subplots'''
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15,5))
# Plot 1: post_length distribution
axes[0].hist(df['post_length'].dropna(), bins=20, color='skyblue', edgecolor='black')
axes[0].set_title('Post Length Distribution')
axes[0].set_xlabel('Post Length')
axes[0].set_ylabel('Frequency')
# Plot 2: likes vs engagement_rate
axes[1].scatter(df['likes'], df['engagement_rate'], alpha=0.5, color='coral')
axes[1].set_title('Likes vs Engagement Rate')
axes[1].set_xlabel('Likes')
axes[1].set_ylabel('Engagement Rate')
# Plot 3: Boxplot of engagement_rate by post_type
df.boxplot(column='engagement_rate', by='post_type', ax=axes[2], grid=False, patch_artist=True)
axes[2].set_title('Engagement Rate by Post Type')
axes[2].set_xlabel('Post Type')
axes[2].set_ylabel('Engagement Rate')
plt.tight_layout()
plt.show()
```



In summary, the code generates a box plot to visualize the distribution of post lengths in the social media engagement metrics dataset. Outliers are shown in the plot, providing insights into the variability and potential extreme values in the 'post_length' variable. Adjustments to the figure size, titles, and labels can be made based on specific preferences.

In [119…

```
sns.set(style="whitegrid")
# Create a box plot with outliers
plt.figure(figsize=(10, 4))
sns.boxplot(x='post_length', data=df, showfliers=True)
# Add labels and title
plt.title('Box Plot of Post Length with Outliers')
plt.xlabel('Post Length')
# Show the plot
plt.show()
```



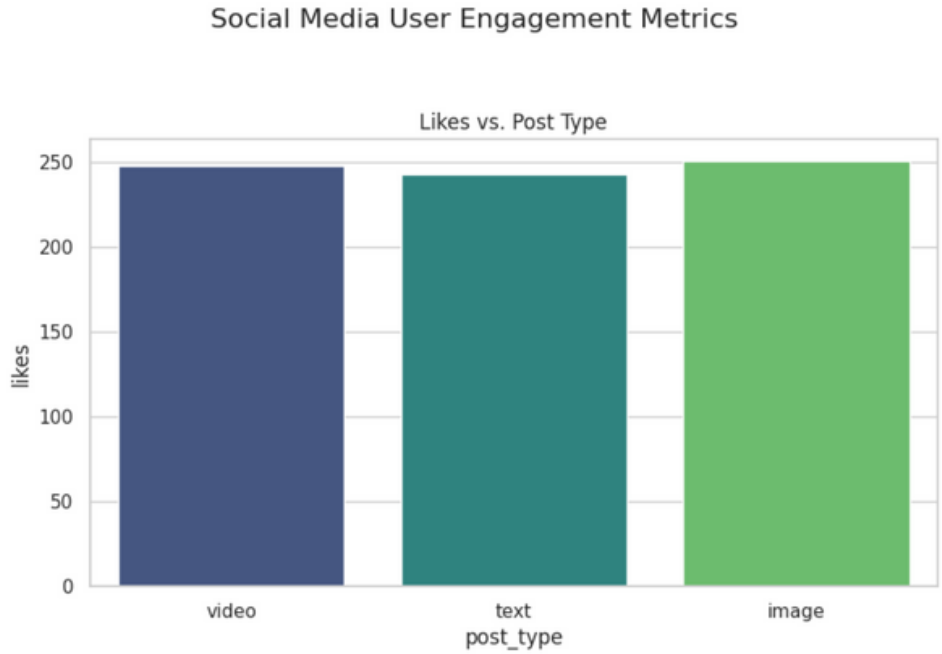
EDA- EXPLORATORY DATA ANALYSIS

#Univariate and Bivariate analysis

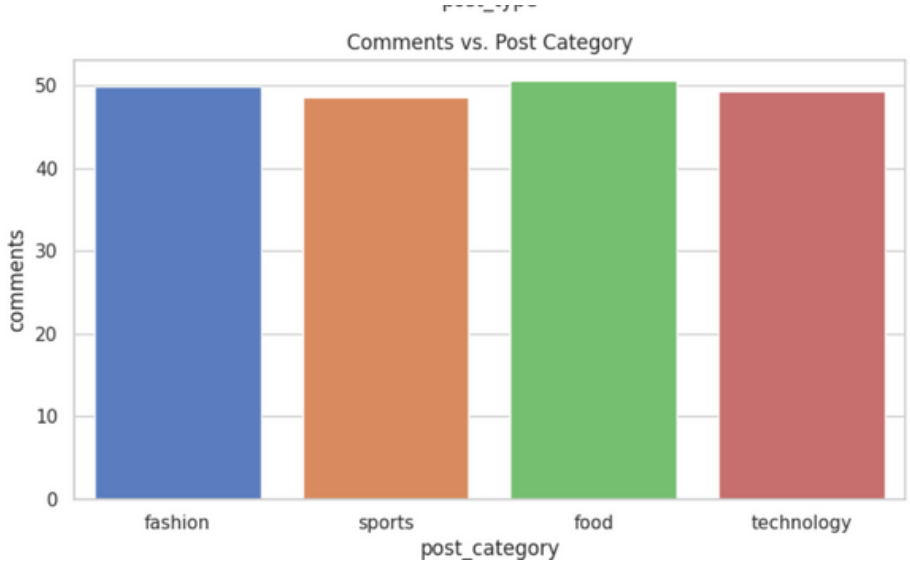
SUB PLOTS

In [130…

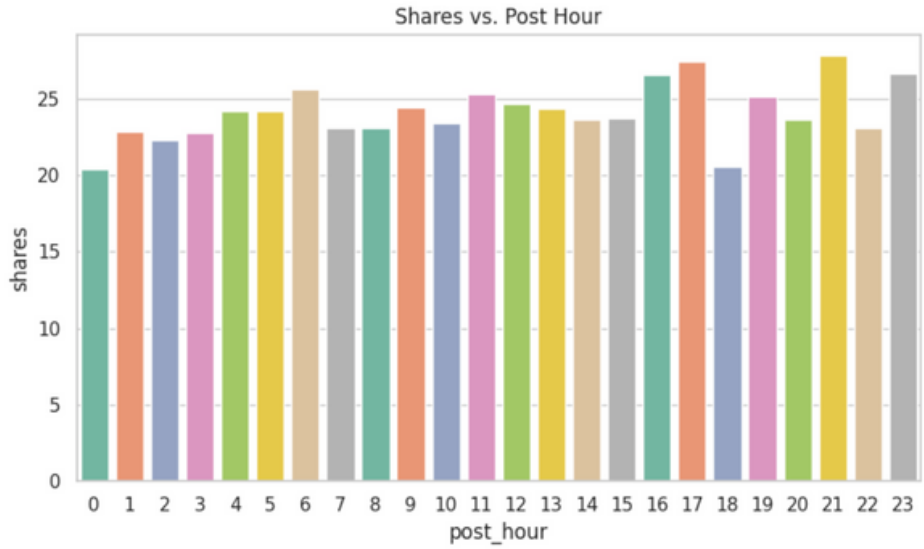
```
import matplotlib.pyplot as plt
import seaborn as sns
# Set the style for seaborn
sns.set(style="whitegrid")
'''Create subplots'''
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(8, 15))
fig.suptitle('Social Media User Engagement Metrics', fontsize=16)
# Subplot 1: Likes vs. Post Type
sns.barplot(x='post_type', y='likes', data=df, ax=axes[0], ci=None, palette='viridis')
axes[0].set_title('Likes vs. Post Type')
# Subplot 2: Comments vs. Post Category
sns.barplot(x='post_category', y='comments', data=df, ax=axes[1], ci=None, palette='muted')
axes[1].set_title('Comments vs. Post Category')
# Subplot 3: Shares vs. Post Hour
sns.barplot(x='post_hour', y='shares', data=df, ax=axes[2], ci=None, palette='Set2')
axes[2].set_title('Shares vs. Post Hour')
# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.96])
# Show the plot
plt.show()
```



It provides insights into the popularity of different post types based on user likes.



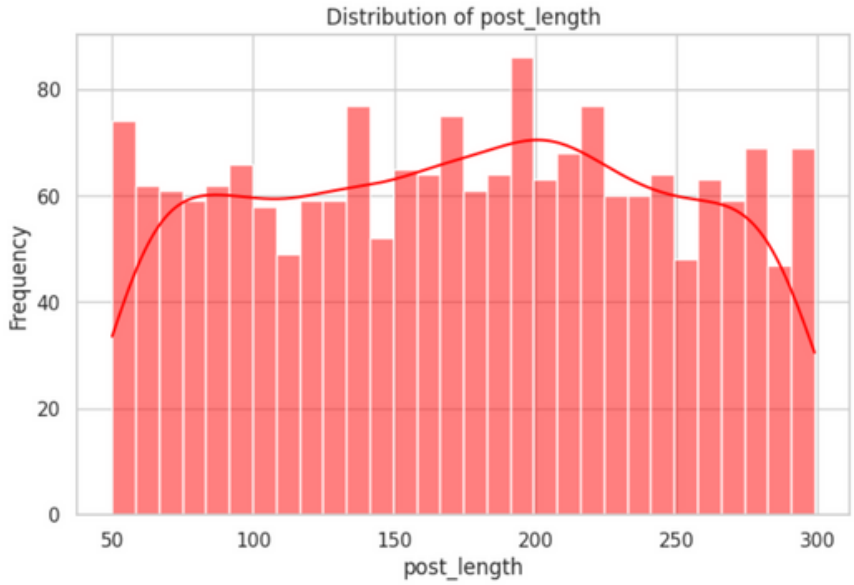
It gives an indication of the level of engagement across various content categories.

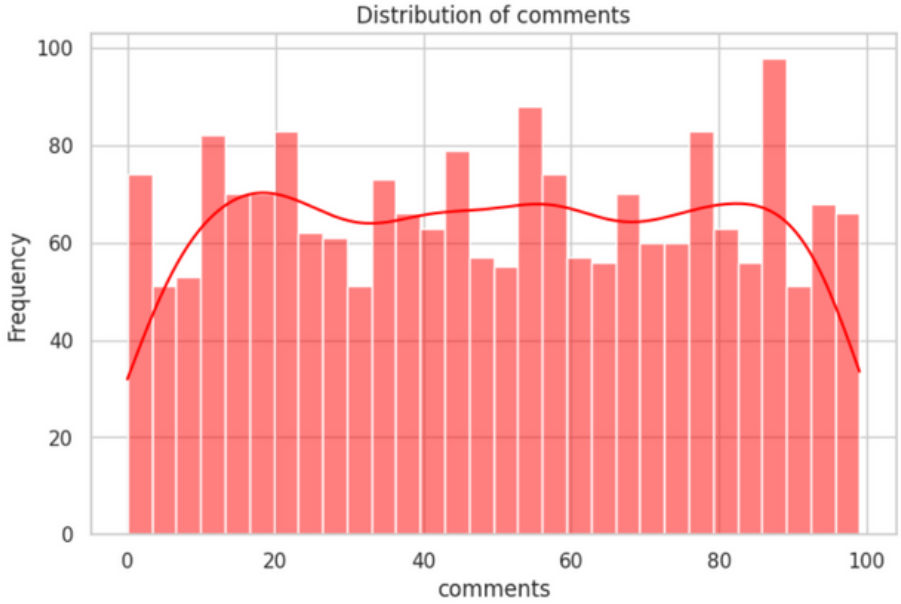


HISTOGRAM

This bar plot shows the average number of shares for posts published at different hours of the day. It helps in understanding when users are more likely to share content, indicating potential peak engagement hours.

```
In [131…
'''Select numerical columns for distribution analysis'''
numerical_columns = ['post_length', 'likes', 'comments', 'shares', 'engagement_rate', 'user_followers']
'''Plot histograms for numerical features'''
for column in numerical_columns:
    plt.figure(figsize=(8, 5))
    sns.histplot(df[column].dropna(), kde=True, bins=30, color='red')
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
```

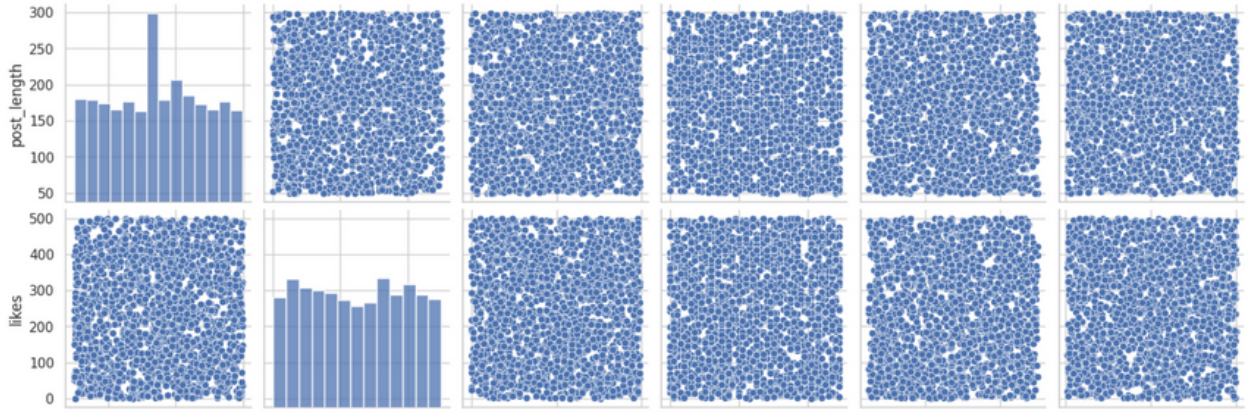




This code provides a quick visual analysis of the distribution of numerical features in your social media engagement metrics dataset. You can gain insights into the spread and concentration of values for each selected numerical variable. Adjust the parameters (e.g., bins, kde) based on your preferences and the characteristics of your data.

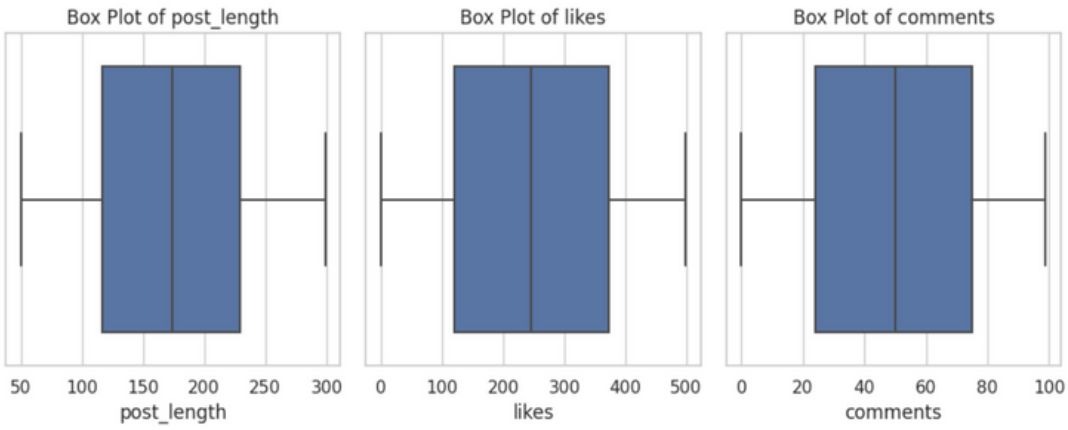
PAIR PLOT

```
In [136…
numerical_cols = ['post_length', 'likes', 'comments', 'shares', 'engagement_rate', 'user_followers']
# Drop NaN values for pairplot
df_pairplot = df[numerical_cols].dropna()
'''Create pairplot'''
sns.pairplot(df_pairplot)
plt.show()
```



By examining the scatterplots, you can visually assess the relationships and correlations between different pairs of numerical features.
Clusters or patterns in the scatterplots may indicate interesting relationships or trends in your data.
The histograms along the diagonal show the distribution of each individual numerical variable.

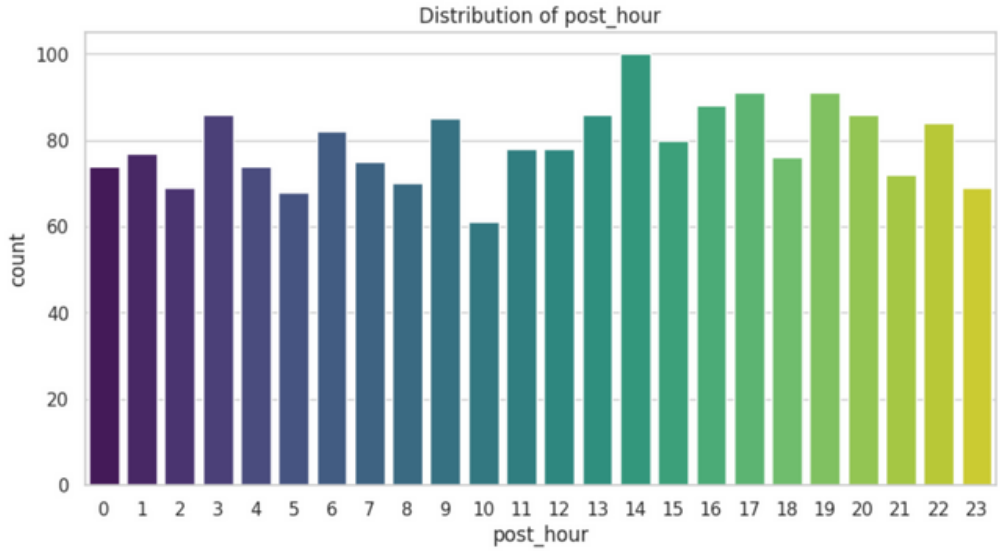
```
In [137…
sns.set(style="whitegrid")
# Define the columns for which you want to create box plots
numeric_columns = ['post_length', 'likes', 'comments', 'shares', 'engagement_rate', 'user_followers']
''' Create box plots'''
plt.figure(figsize=(10, 8))
for i, column in enumerate(numeric_columns, 1):
    plt.subplot(2, 3, i)
    sns.boxplot(x=df[column])
    plt.title(f'Box Plot of {column}')
plt.tight_layout()
plt.show()
```



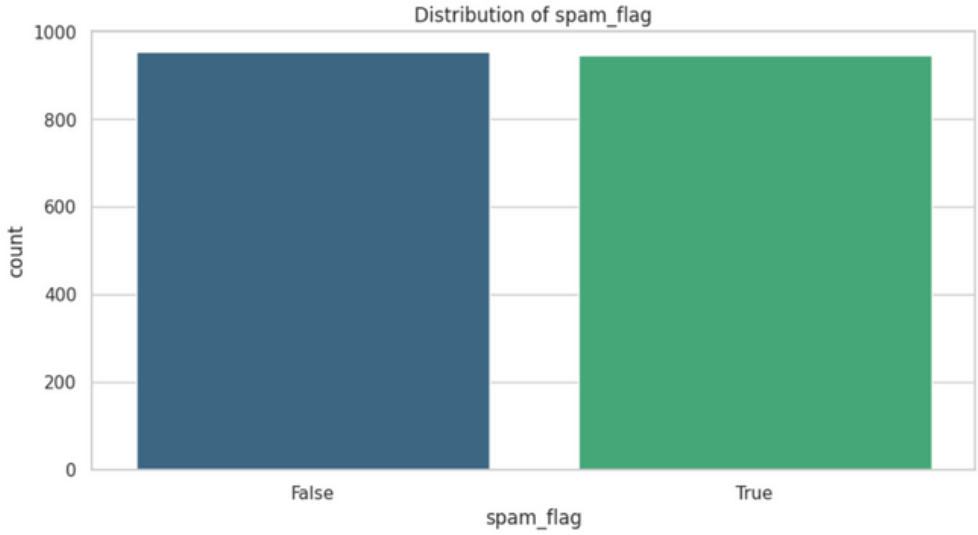
The code generates a 2x3 grid of box plots, each representing the distribution of values for a specific numeric column in your dataset. These plots help visualize the central tendency and spread of each variable and identify potential outliers. Adjust the subplot grid and figure size as needed for your preferences.

In [140…

```
'''discrete variables'''
# Set style for better visualization
sns.set(style="whitegrid")
# Define the discrete variables for analysis
discrete_vars = [ 'post_hour', 'is_weekend', 'user_verified', 'spam_flag']
# Plot histograms for each discrete variable
for var in discrete_vars:
    plt.figure(figsize=(10, 5))
    sns.countplot(x=var, data=df, palette="viridis")
    plt.title(f'Distribution of {var}')
    plt.show()
```



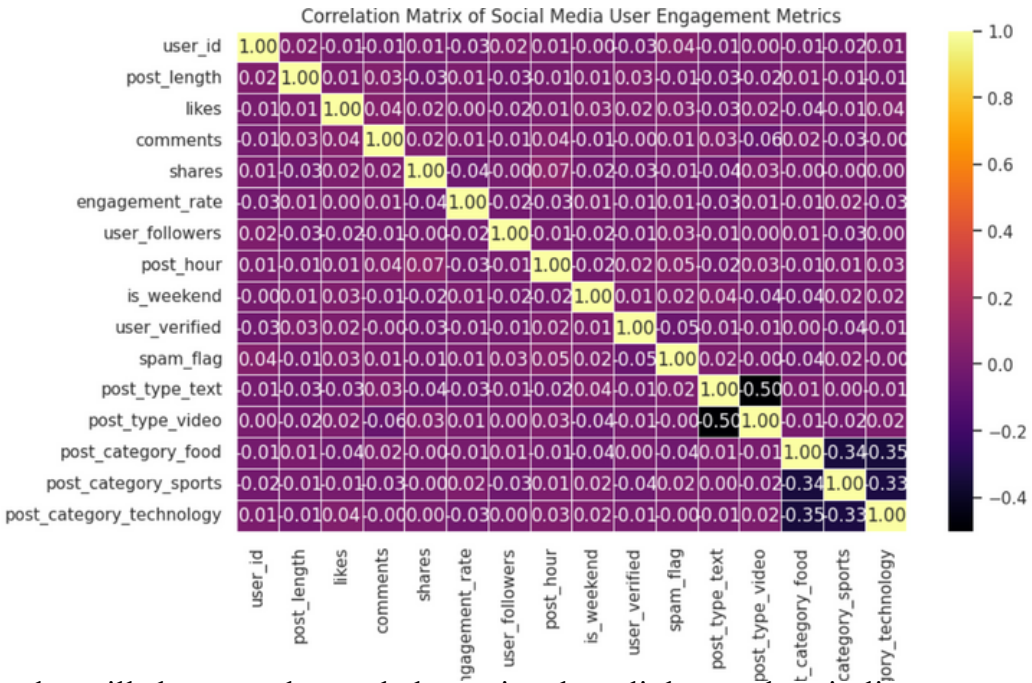
These histograms provide insights into the distribution of values for each discrete variable, helping you understand how often each value occurs in your dataset. The plots show the count (frequency) of each unique value in the discrete variables. Adjustments to the code can be made based on specific visualization preferences or additional details you want to include in the plots.



COORELATION MATRIX-HEATMAP

In [142…

```
''' Calculate the correlation matrix'''
correlation_matrix = df.corr()
# Plot the heatmap
plt.figure(figsize=(10, 6))
print(correlation_matrix)
sns.heatmap(correlation_matrix, annot=True, cmap='inferno', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix of Social Media User Engagement Metrics')
plt.show()
```



he resulting plot will show a color-coded matrix where lighter colors indicate stronger correlations, and darker colors indicate weaker or negative correlations. The diagonal of the matrix will be all 1's as it represents the correlation of each variable with itself.

STRONGLY CORRELATED PAIRS

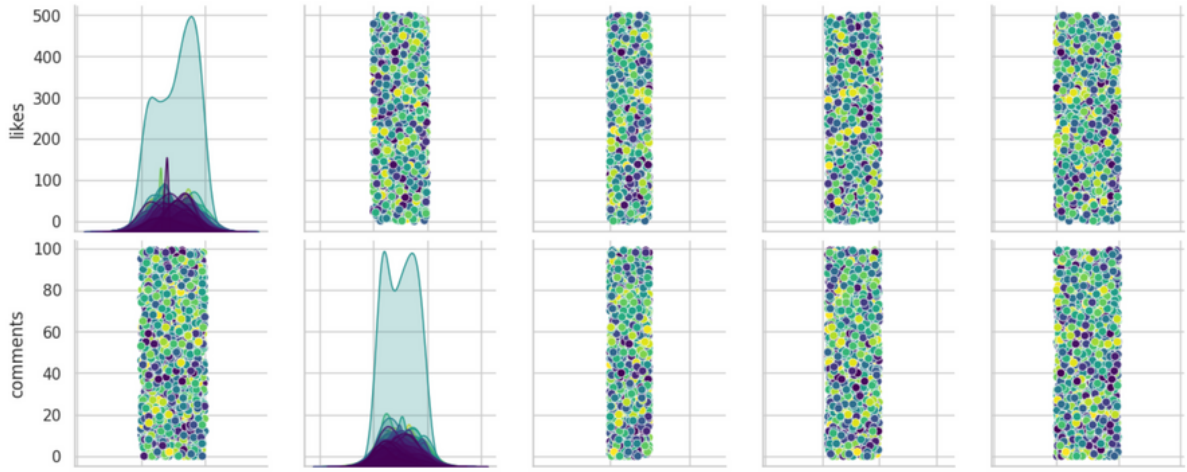
In [143…

```
'''Display the correlation matrix'''
print("Correlation Matrix:")
print(correlation_matrix)
# Extract strongly correlated pairs
strongly_correlated_pairs = correlation_matrix.unstack().sort_values(ascending=False).drop_duplicates()
strongly_correlated_pairs = strongly_correlated_pairs[strongly_correlated_pairs != 1]

# Display strongly correlated pairs
print("\nStrongly Correlated Feature Pairs:")
print(strongly_correlated_pairs)
```

```
Strongly Correlated Feature Pairs:
shares                post_hour                  0.069194
spam_flag             post_hour                  0.045101
comments              likes                      0.041552
user_id               spam_flag                  0.039932
comments              post_hour                  0.038247
                                                    ...
                      post_type_video           -0.064539
post_category_sports  post_category_technology  -0.328134
                      post_category_food        -0.337284
post_category_food    post_category_technology  -0.347454
post_type_video       post_type_text            -0.500621
Length: 120, dtype: float64
```

PAIR PLOT

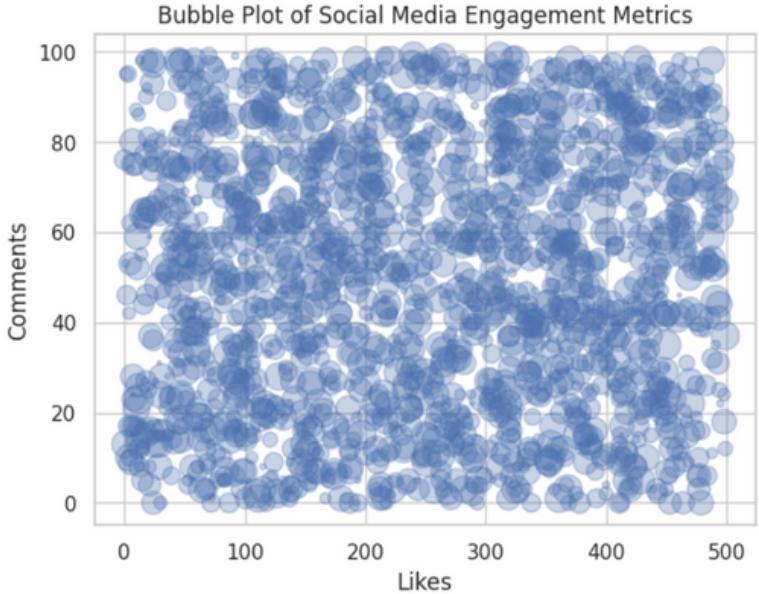
```
In [144…
'''Select features for pairplot'''
selected_features = ['post_length', 'likes', 'comments', 'shares', 'engagement_rate', 'user_followers']

# Create a pairplot
sns.pairplot(df[selected_features], diag_kind='kde', markers='o', hue='post_length', palette='viridis')

# Show the plot
plt.show()
```



The pairplot provides a visual representation of the relationships between pairs of selected features. Scatter plots in the lower triangle show bivariate relationships, while kernel density plots on the diagonal show the univariate distribution of each selected feature.
Points in the scatter plots are colored based on the 'post_length' feature.

BUBBLE PLOT

```
In [145…
'''Bubble plot example'''
plt.scatter(df['likes'], df['comments'], s=df['shares']*5, alpha=0.3)
plt.title('Bubble Plot of Social Media Engagement Metrics')
plt.xlabel('Likes')
plt.ylabel('Comments')
plt.show()
```



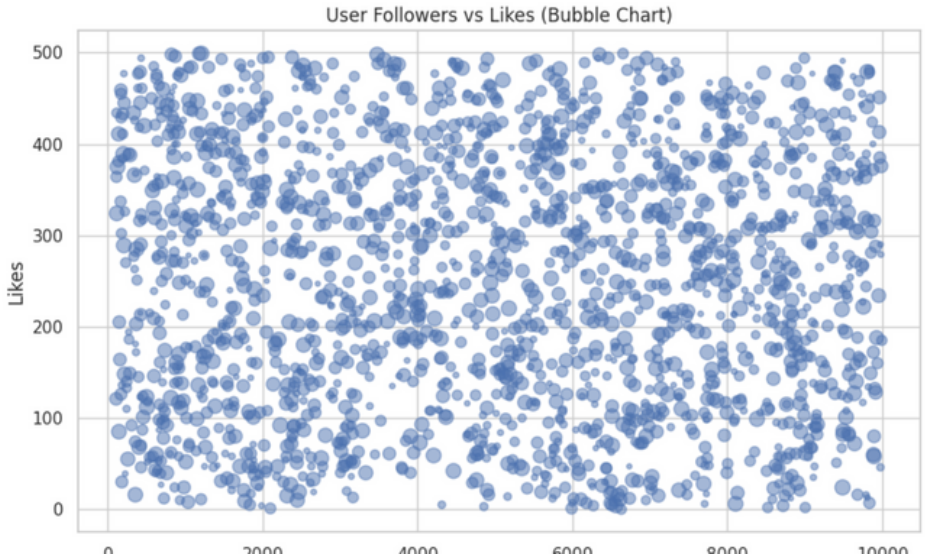
Each point in the scatter plot is a post, and the size of the point indicates the level of engagement through shares.

Larger bubbles represent posts with more shares, while smaller bubbles represent posts with fewer shares.

The transparency (alpha) is set to 0.3, allowing overlapping bubbles to be partially visible.

BUBBLE CHART

```
# Plotting bubble chart
plt.figure(figsize=(10, 6))
plt.scatter(df['user_followers'], df['likes'], s=df['engagement_rate']*20, alpha=0.5)

# Adding labels and title
plt.xlabel('User Followers')
plt.ylabel('Likes')
plt.title('User Followers vs Likes (Bubble Chart)')

# Show plot
plt.show()
```

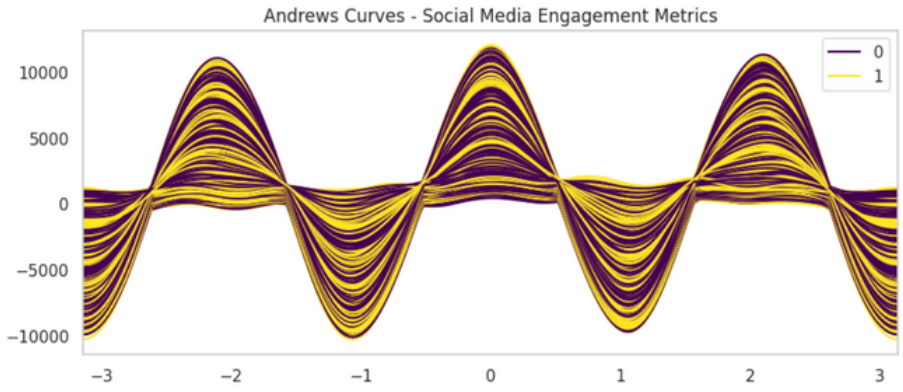


In summary, the bubble chart visualizes the relationship between 'User Followers' and 'Likes,' with the size of each bubble indicating the 'engagement_rate' for each data point. This type of chart can be useful for exploring patterns and correlations between these variables in the dataset. Adjustments can be made to customize the chart further based on specific preferences or insights you're seeking.

'''Multivariate analysis'''

ANDREW CURVES

```
In [148...
from pandas.plotting import andrews_curves
df['label'] = np.random.choice([0, 1], len(df))

# Perform Andrews Curves
plt.figure(figsize=(10, 4))
andrews_curves(df, 'label', colormap='viridis')
plt.title('Andrews Curves - Social Media Engagement Metrics')
plt.show()
```

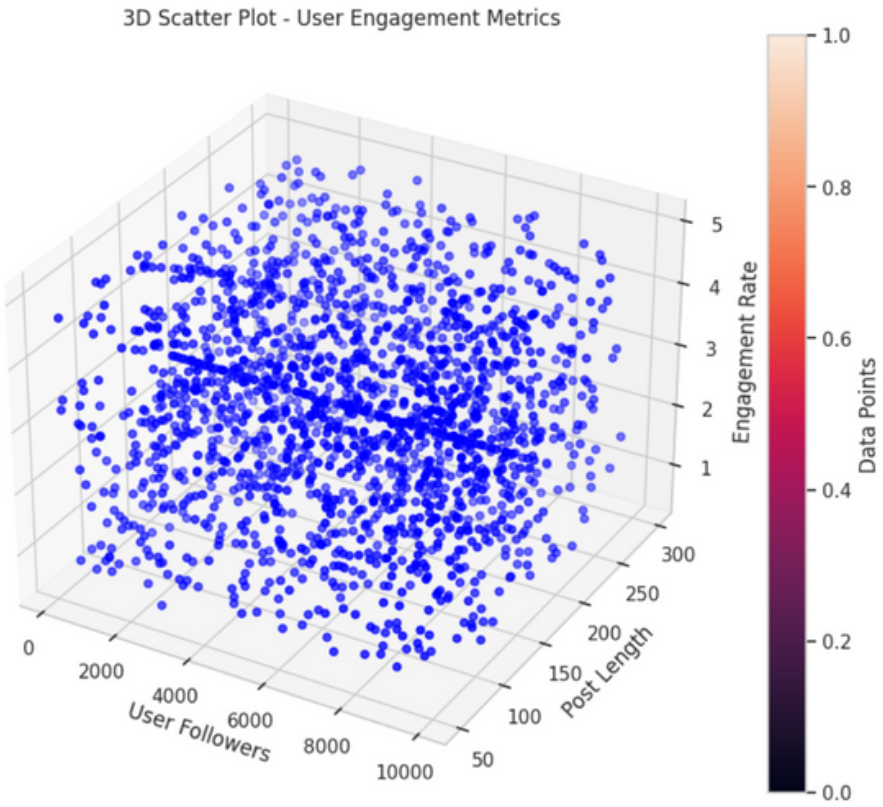


The resulting plot will show curves representing each row in the dataset, and the color of the curve is determined by the randomly assigned 'label' (0 or 1). Andrews Curves are useful for detecting patterns and differences between classes in high-dimensional datasets.

3D SCATTER PLOT

In [149…

```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

''' Create a 3D scatter plot'''
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot
scatter = ax.scatter(df['user_followers'], df['post_length'], df['engagement_rate'], c='blue', marker='o')

# Set labels
ax.set_xlabel('User Followers')
ax.set_ylabel('Post Length')
ax.set_zlabel('Engagement Rate')

# Set title
ax.set_title('3D Scatter Plot - User Engagement Metrics')

# Add color bar
fig.colorbar(scatter, ax=ax, label='Data Points')

# Show the plot
plt.show()
```

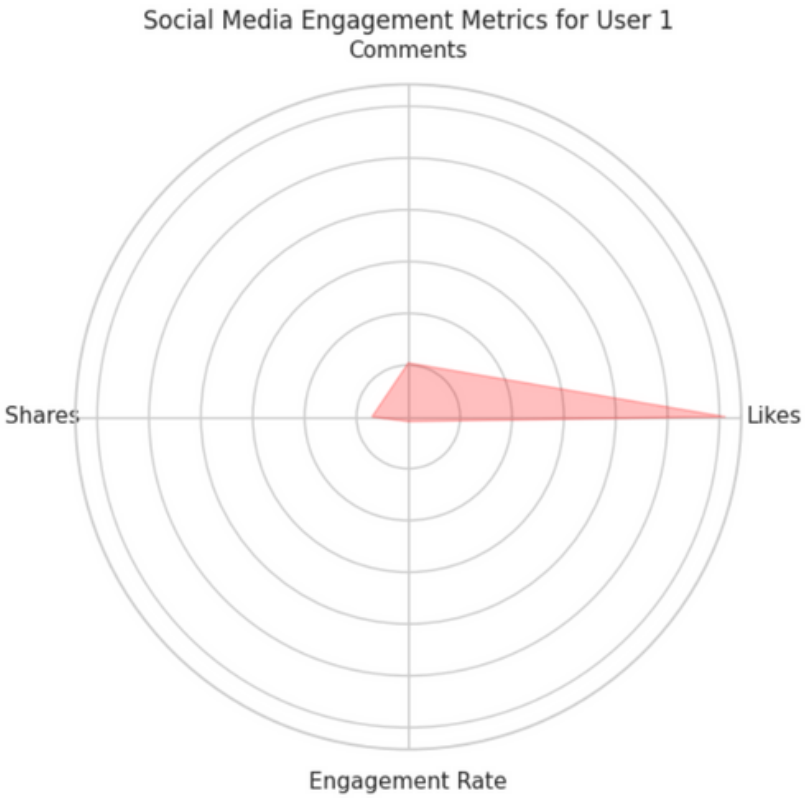


The 3D scatter plot provides a visual representation of how 'User Followers', 'Post Length', and 'Engagement Rate' are distributed and if there are any patterns or relationships between these variables. Each point in the plot corresponds to a data entry in your social media engagement metrics dataset.

RADAR CHART

In [151…

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Your code to generate the dataset (assuming it's already done)

# Assuming you want to compare 'likes', 'comments', 'shares', and 'engagement_rate' for the first user (user_id = 1)
user_data = df[df['user_id'] == 1][['likes', 'comments', 'shares', 'engagement_rate']].fillna(0).values.flatten()

labels = ['Likes', 'Comments', 'Shares', 'Engagement Rate']
num_vars = len(labels)

# Compute angle for each axis
angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()

# The plot is a circle, so we need to "complete the loop" and append the start value to the end
user_data = np.concatenate((user_data,[user_data[0]]))
angles += angles[:1]

fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))
ax.fill(angles, user_data, color='red', alpha=0.25)
ax.set_yticklabels([])
plt.xticks(angles[:-1], labels)
plt.title(f'Social Media Engagement Metrics for User {1}')
plt.show()
```



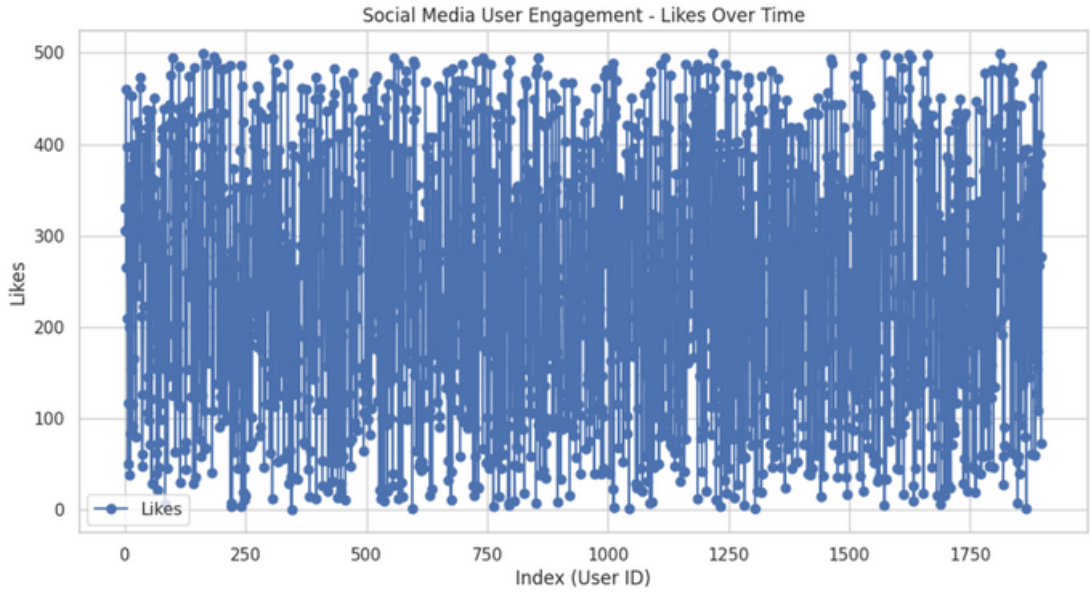
The radar chart provides a visual representation of how the user's engagement metrics (likes, comments, shares, and engagement rate) compare to each other.

The radial lines represent different metrics, and the distance from the center indicates the magnitude of each metric.

The filled area under the curve shows the overall pattern of engagement for the user.

#Time Series Analysis

```
# Set a numerical index
df.reset_index(drop=True, inplace=True)

# Sort the DataFrame based on the index
df.sort_index(inplace=True)

# Plotting time series for 'Likes'
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['likes'], label='Likes', marker='o')
plt.title('Social Media User Engagement - Likes Over Time')
plt.xlabel('Index (User ID)')
plt.ylabel('Likes')
plt.legend()
plt.show()
```

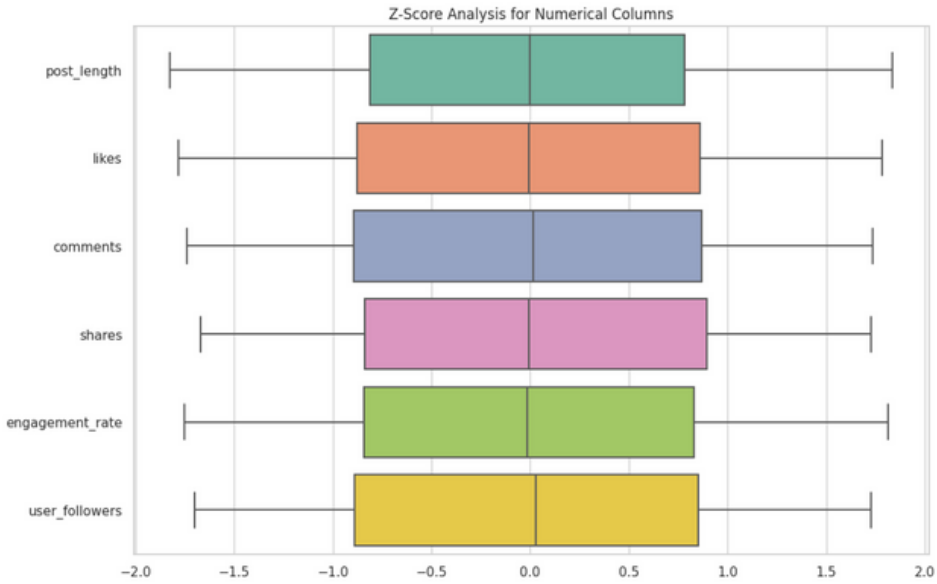


#Outlier Analysis

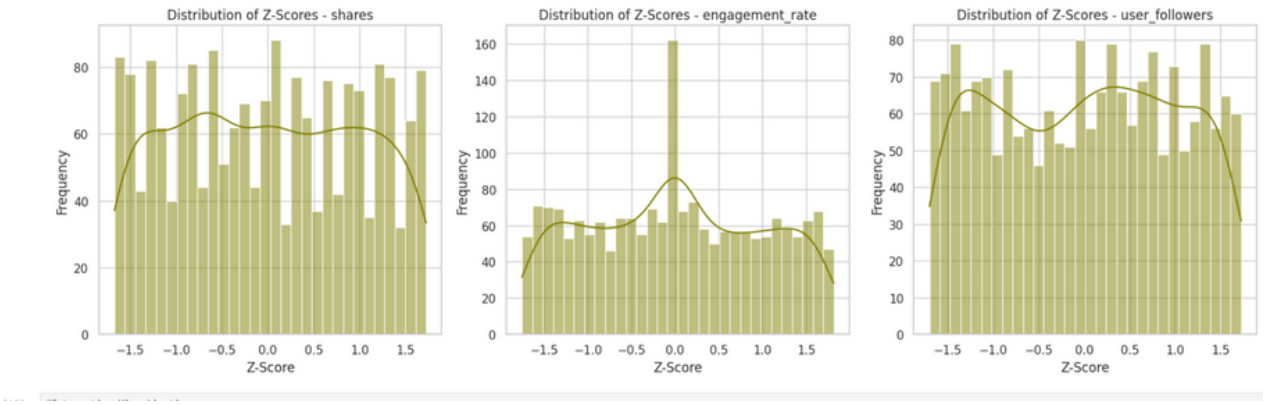
In [158...

```
# Columns to consider for Z-Score analysis
numerical_columns = ['post_length', 'likes', 'comments', 'shares', 'engagement_rate', 'user_followers']

# Calculate Z-Scores for selected columns
z_scores = (df[numerical_columns] - df[numerical_columns].mean()) / df[numerical_columns].std()

# Plot Z-Scores using a boxplot
plt.figure(figsize=(12, 8))
sns.boxplot(data=z_scores, orient='h', palette='Set2')
plt.title('Z-Score Analysis for Numerical Columns')
plt.show()

# Plot distribution of Z-Scores for each numerical column
plt.figure(figsize=(16, 10))
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(2, 3, i)
    sns.histplot(z_scores[column], kde=True, bins=30, color='olive')
    plt.title(f'Distribution of Z-Scores - {column}')
    plt.xlabel('Z-Score')
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

The resulting plot visualizes the 'likes' over time (or over the user IDs). Each point on the plot represents the number of 'likes' for a specific user or at a specific time point. The 'marker='o'' argument adds circular markers to each data point for better visibility. The legend provides information about the variable being plotted.



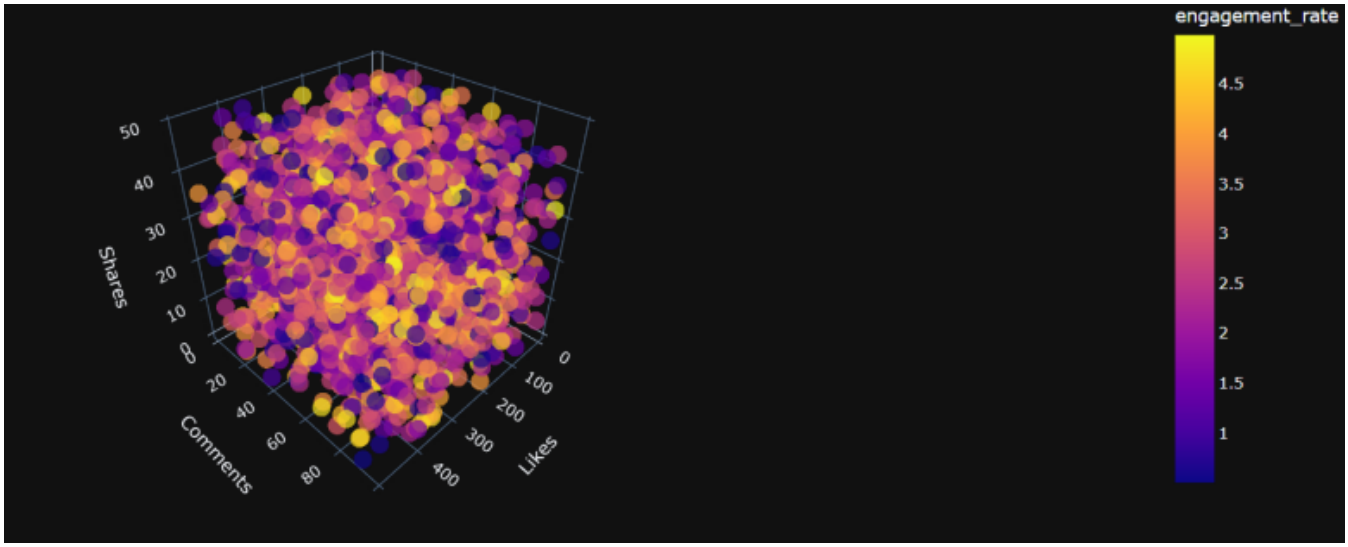
The boxplot provides an overview of the central tendency, spread, and presence of outliers in the Z-Scores of the selected numerical columns.
The distribution plots show how the Z-Scores are distributed for each column, helping to identify skewness or unusual patterns in the data.



INTERACTIVE ANALYSIS-3D

In [159…
```
#Interactive Visualization:
```

In [160…
```
import plotly.express as px

# Remove rows with NaN values for the plot
cleaned_df = df.dropna(subset=['likes', 'comments', 'shares', 'engagement_rate'])

# Create an interactive 3D scatter plot
fig = px.scatter_3d(cleaned_df, x='likes', y='comments', z='shares', color='engagement_rate',
                    opacity=0.7, size_max=10, template='plotly_dark',
                    labels={'likes': 'Likes', 'comments': 'Comments', 'shares': 'Shares'})

# Show the plot
fig.show()
```



Each point in the 3D scatter plot represents a social media post.
The position of a point along the x, y, and z axes corresponds to the number of likes, comments, and shares the post received, respectively.
The color of each point indicates the engagement rate, providing additional information about the post's overall engagement relative to others.
The plot allows for interactive exploration, such as rotating, zooming, and hovering over data points to view specific information.

End Notes

In this article, we understood the meaning of Exploratory Data Analysis (EDA) with the help of an example dataset. We looked at how we can analyze the dataset, draw conclusions from the same, and form a hypothesis based on that.

The author of this BLOG is PURNIMA RANGAVAJJULA.

THE END