Predicting Social Media Engagement Rate

Javier Corpus

Bellevue University

DSC630-T302: Predictive Analytics

Dr. Andrew Hua

August 10, 2025

Introduction of Topic/Problem

Social media engagement is a critical metric for marketing success, serving as a key performance indicator (KPI) for brand visibility and customer reach. The ability to predict engagement rates—defined as the ratio of likes, shares, and comments to impressions—can significantly enhance marketing strategies by optimizing content, timing, and platform selection. This predictive capability allows companies to allocate advertising budgets more effectively, maximizing return on investment (ROI). The challenge lies in the dynamic and unpredictable nature of social media engagement. Factors such as platform algorithms, user behavior trends, and content virality create a complex ecosystem where traditional marketing intuition often falls short.

The primary objective of this project is to develop a predictive model that estimates engagement rates based on various features, like platform, day of week, location, language, emotion type, toxicity score, count of likes, shares and comments, number of impressions, and sentiment.

Key research questions include:

- Can engagement rate be reliably predicted using metadata (e.g., sentiment, platform, number of likes)?
- Which features have the strongest influence on engagement?
- Does the relationship between features and engagement vary by platform (e.g., Instagram vs. X/Twitter)?

Understanding these questions can help social media managers and marketing teams make informed decisions about when and where to post content, ultimately improving brand visibility and customer reach. By identifying the factors that contribute to high or low engagement, marketers can adjust their strategies to optimize engagement and ultimately drive business results.

Overview of Data Used

The dataset used for this project contains various features that are potential predictors for the engagement rate target variable. Key features include:

Platform (e.g., YouTube, Facebook, Twitter, Reddit, Instagram), Day of week, Location, Language, Sentiment Score, Sentiment Label, Emotion Type, Toxicity Score, Number of Likes, Number of Shares, Number of Comments, Number of Impressions, Topic Category, Campaign Phase.

The dataset was downloaded from Kaggle and contains a total of 12,000 posts across five major social media platforms:

- YouTube (2,436 posts)
- Facebook (2,431 posts)
- Twitter (2,406 posts)
- Reddit (2,372 posts)
- Instagram (2,355 posts)

This diversity allows for a robust analysis of platform-specific engagement patterns.

Preliminary analysis revealed posts are evenly split between positive and negative sentiment, with fewer neutral posts. This is consistent across platforms. For example, Facebook had 977 positive, 985 negative, and 469 neutral posts, while Twitter had 1,003 positive, 933 negative, and 470 neutral posts. There are no missing values in critical engagement metrics.

Scope and Limitations

This study focuses on five major social media platforms and uses a dataset of 12,000 posts. While comprehensive, the analysis has certain limitations:

- The data represents a snapshot in time and may not account for evolving platform algorithms.

- Cultural and demographic factors are not explicitly included in the dataset.
- The models predict engagement rates but do not account for long-term brand building effects.
- The dataset doesn't include video-specific metrics (crucial for YouTube/Instagram Reels).

Methods of Analysis

The methodology employed in this project involves several steps. First, exploratory data analysis was conducted to understand the distribution of features and their relationships. This involved visualizing the data using scatter plots, bar charts, and box plots to identify correlations and trends.

Next, the dataset was split into training (80%) and testing (20%) sets to simulate a real-world scenario and prevent data leakage. Feature engineering techniques were applied to create new features that could potentially improve model performance.

Model Selection

Three machine learning models were employed to predict engagement rates:

1. **Linear Regression**: Used as a simple baseline model to measure performance against more complex models. This model has the advantage of having simple, interpretable coefficients, but has the limitation of assuming linear relationships between variables.
2. **Random Forest**: Effective with data containing a mix of numerical and categorical features, capable of capturing complex, non-linear relationships. This model has the advantage of handling non-linear relationships, and it's robust to outliers, but has a risk of potential overfit.
3. **XGBoost**: Also effective with mixed data types, but with additional features like sequential correction of errors and L1/L2 regularization to help avoid overfitting. The disadvantage of this model is that is more computationally intensive.

Evaluation Metrics

Model performance was evaluated using the following metrics:

- Root Mean Squared Error (RMSE): Measures the average magnitude of prediction errors.
- Mean Absolute Error (MAE): Provides the average absolute prediction error.
- R-squared (R^2): Indicates the proportion of variance in engagement rates explained by the model.

Data Preprocessing – Linear Regression Model

To build and test the predictive models, some features were removed or transformed. Features that are not relevant for this exercise, such as post id, timestamp, user id, text content, hashtags, mentions, keywords, brand name, and product name, were excluded from the analysis. Also, dummy variables were created for the categorical features

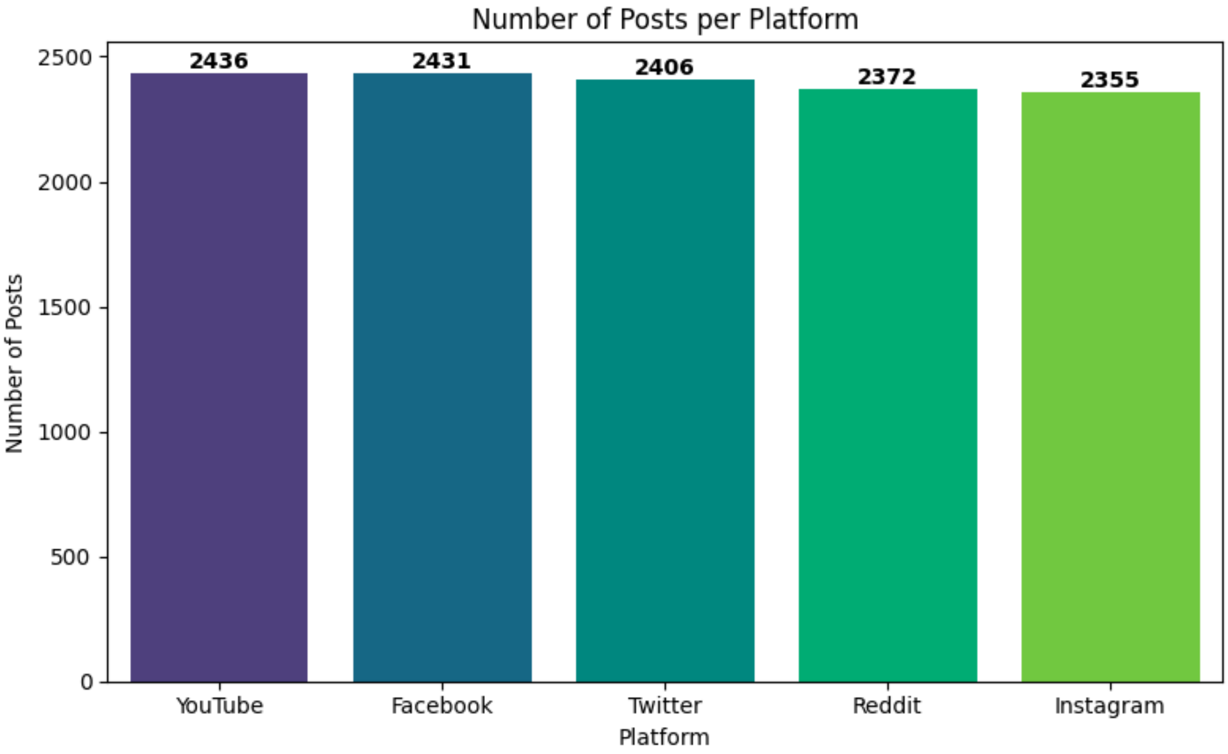
Data Preprocessing – Random Forest and XGBoost Models

Just like with the linear regression model, the irrelevant features were removed. All the categorical features were converted to the "category" type. New features were engineered, like log-transformed counts, total interactions, engagement per impressions, comment to likes ratio and shares to likes ratio. Numerical features were scaled using StandardScaler to ensure uniformity. Additionally, skewed numerical features were log-transformed to normalize their distributions.

Results & Findings Explained

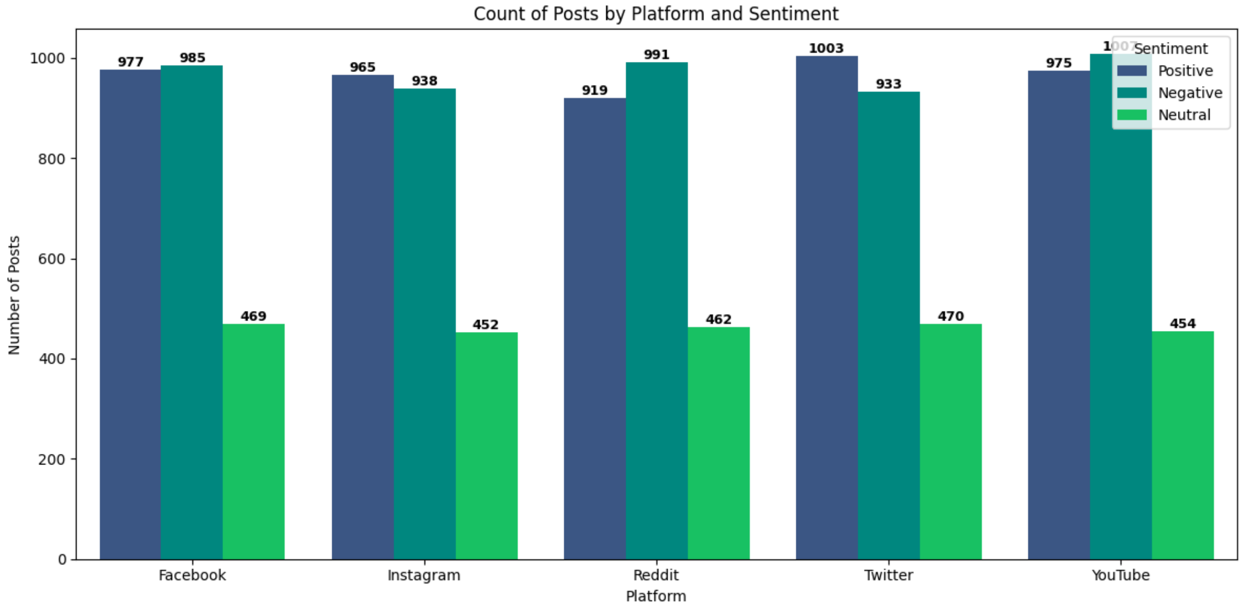
Exploratory Data Analysis

The dataset used has a uniform number of posts per platform, which is great to ensure that models are not biased towards a particular platform. Also, having a similar number or posts per platform allows for a more reliable statistical analysis.

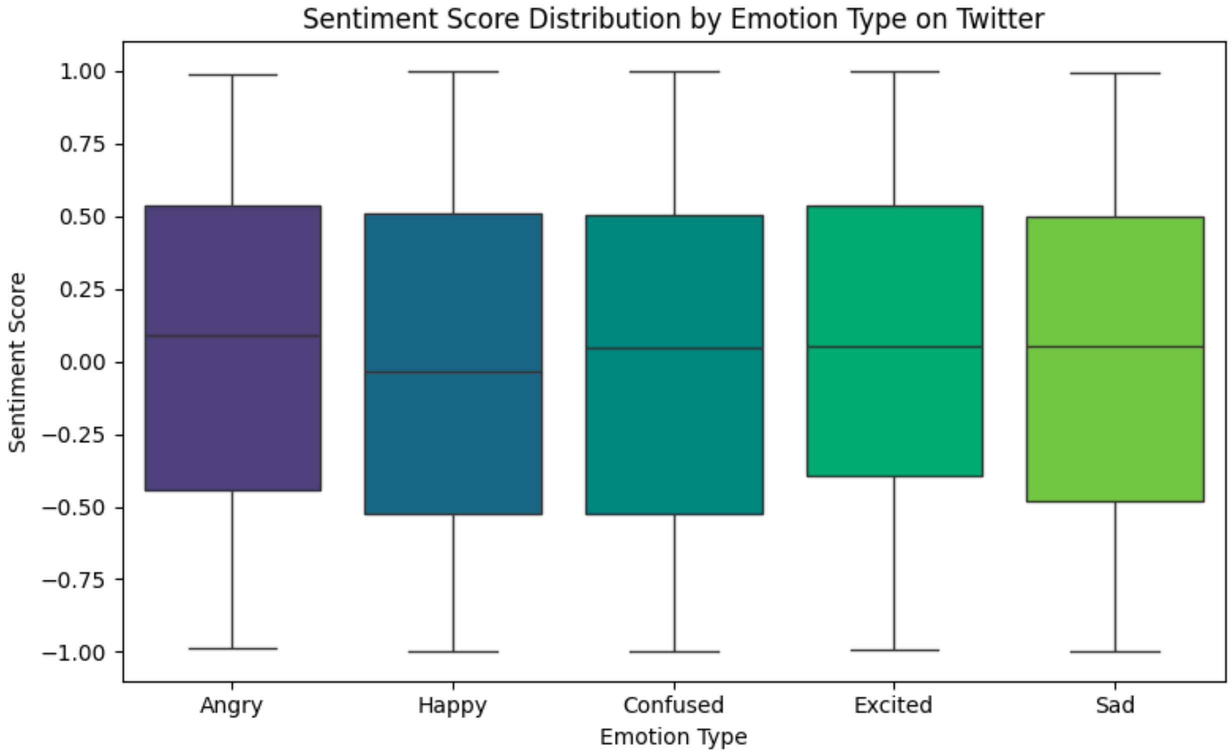


The sentiment analysis per platform showed varying distributions:

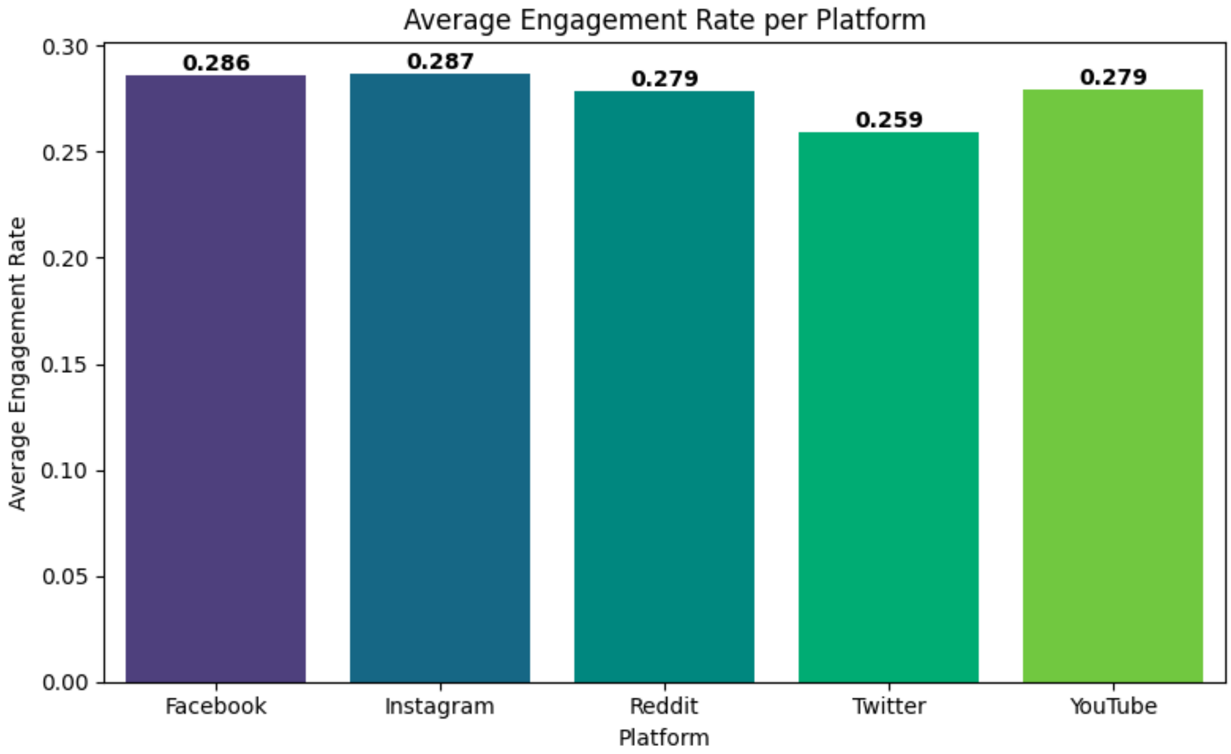
Platform	Positive	Negative	Neutral
Facebook	977	985	469
Instagram	965	938	452
Reddit	919	991	462
Twitter	1003	933	470
YouTube	975	1007	454



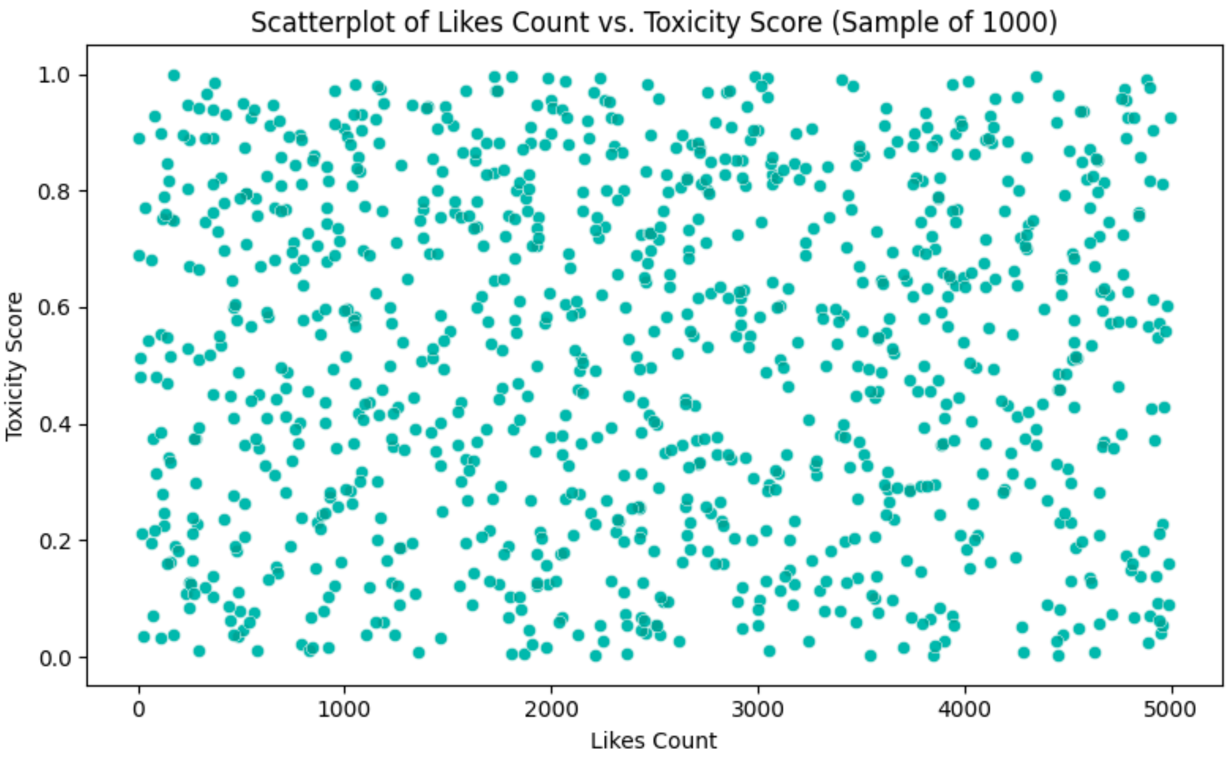
Looking at the distribution of sentiment score by emotion type, all the platforms have very similar score distribution by emotion type. The following box plot is an example of Twitter, but all the other platforms have a similar distribution:



When considering the average engagement rate, there are small differences per platform, but these differences are minimal.



We randomly took 1000 samples to look for a correlation between the number of likes and the toxicity score. There doesn’t seem to be any, which could indicate that users are engaging with content for reasons unrelated to its toxicity, such as humor, relatability, or other factors. If posts with high toxicity scores are still receiving a significant number of likes, it may reflect a tolerance or even preference for controversial or provocative content among users. This could be a point of concern if the goal is to promote healthier online interactions.



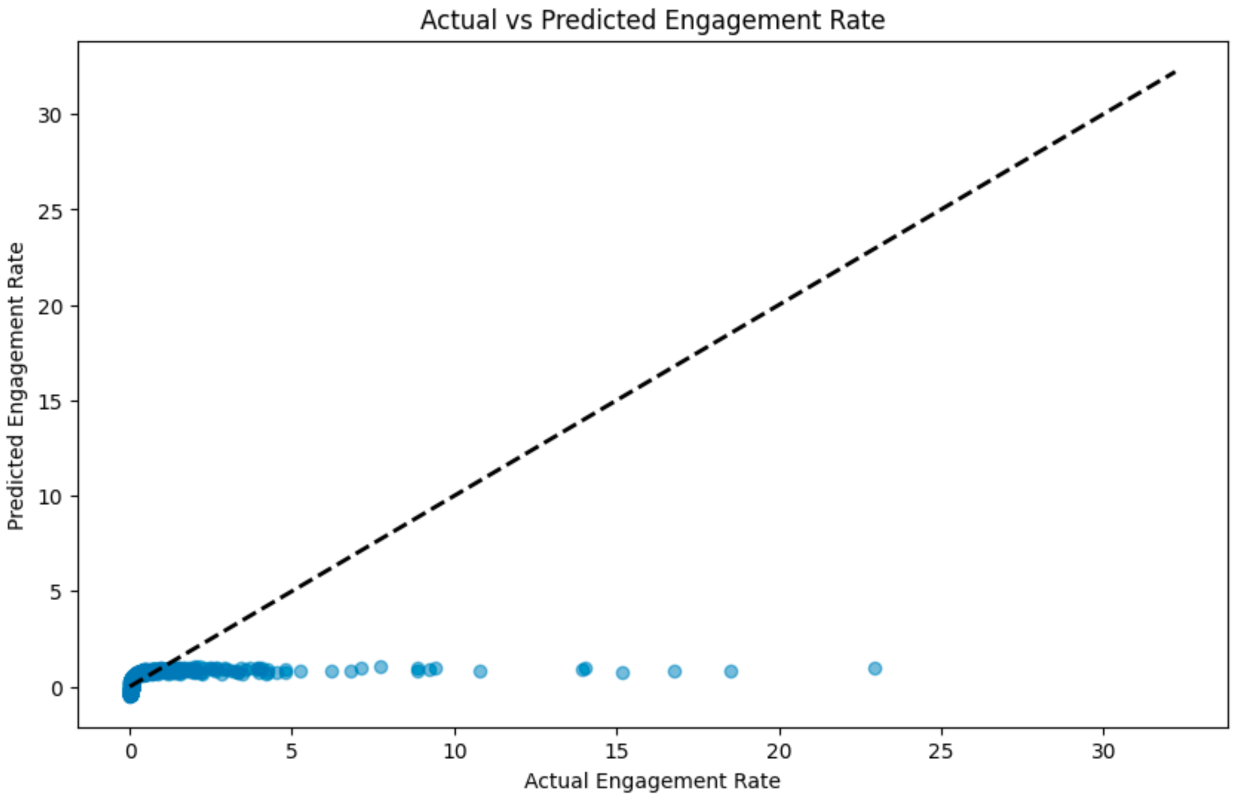
Linear Regression Model

When this model was evaluated, it showed the following results:

- RMSE: 1.0644
- R-squared (R²): 0.1032
- MAE: 0.3485

These results indicate that the model explained only 10.32% of the variance in engagement rates, suggesting weak explanatory power. The high RMSE and low R² values highlight the model's inability to capture the complex relationships within the data. This confirms that the social media engagement prediction requires a more robust model with the ability to handle non-linear relationships.

The following plot illustrates the poor performance of the model when comparing the Actual vs. Predicted engagement rates.



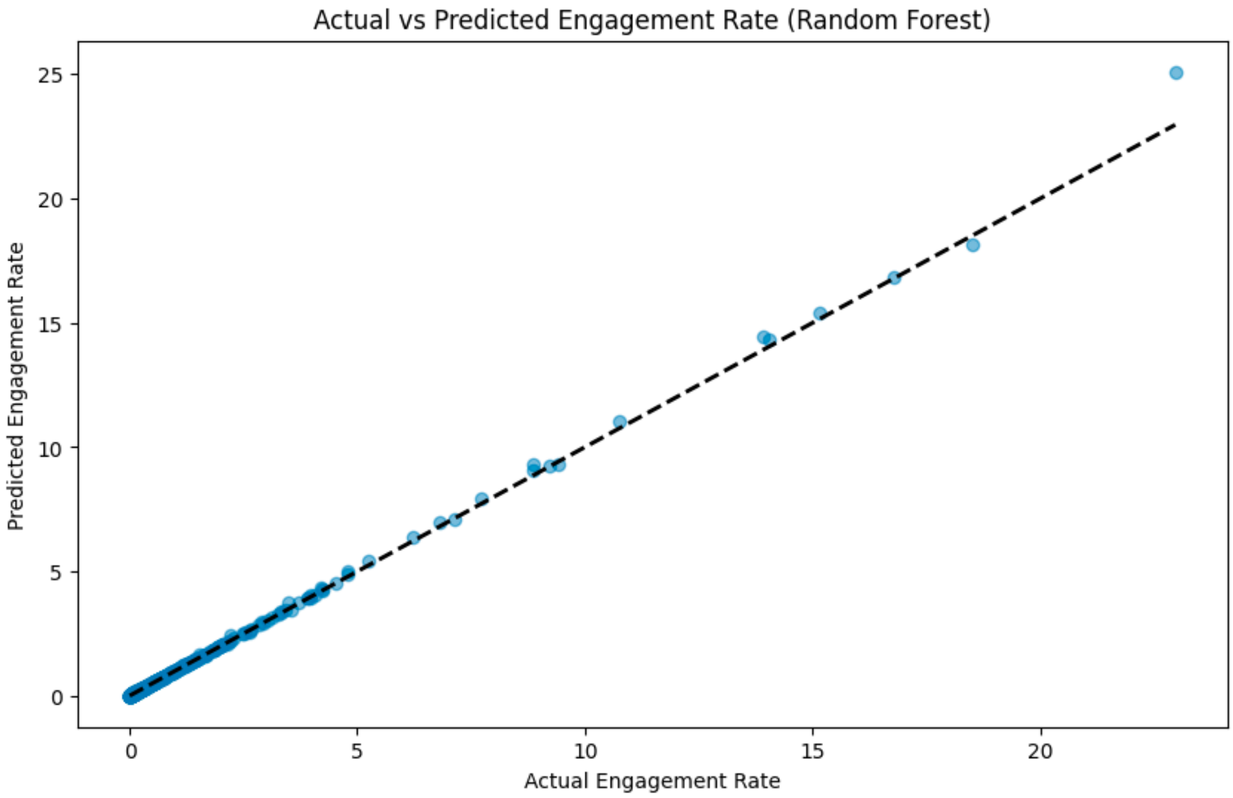
Random Forest Model

This model performed exceptionally well, explaining 99.78% of the variance. It had the following results:

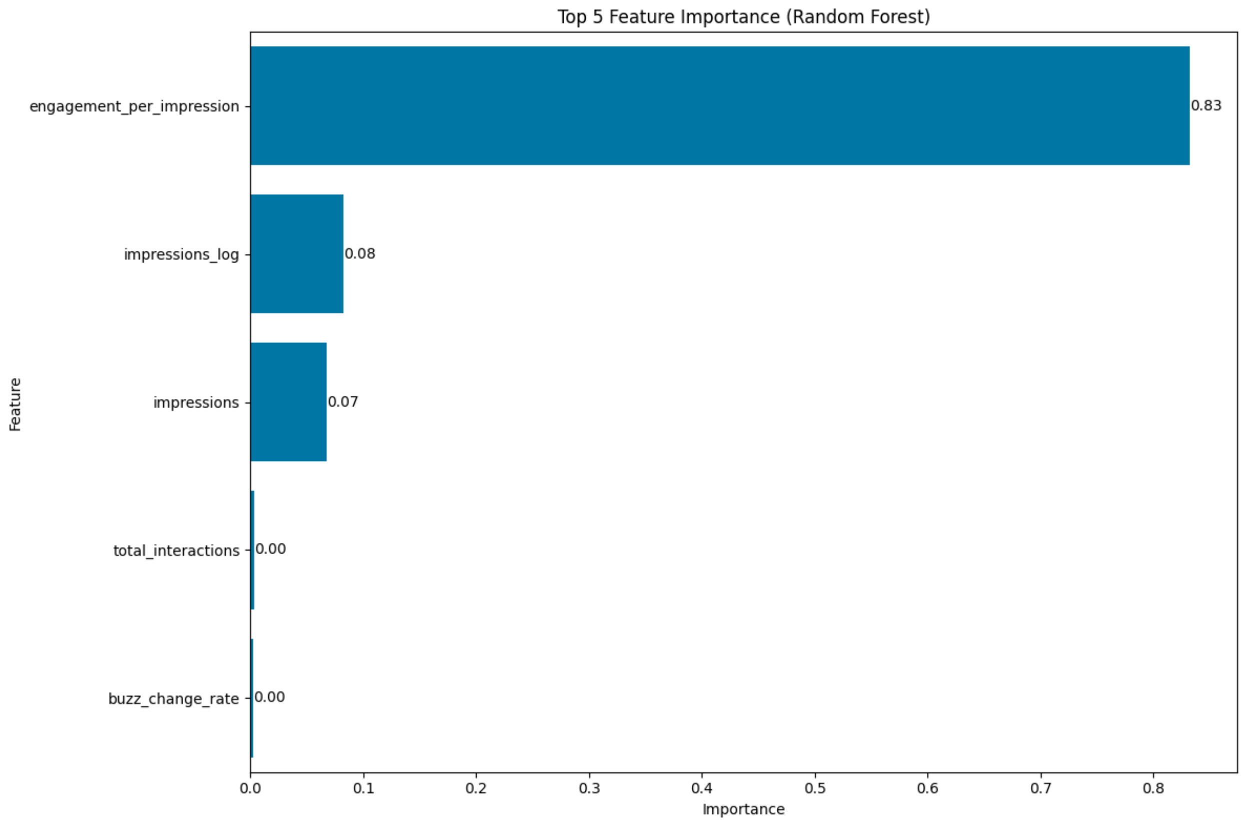
- RMSE: 0.0532
- R-squared (R^2): 0.9978
- MAE: 0.0038

While these metrics suggest near-perfect predictive accuracy, the high R^2 value raises concerns about overfitting.

The following visualization shows the suspiciously high accuracy of the model when comparing Actual vs. Predicted engagement rates:



The model's feature importance analysis identified "engagement_per_impression" as the most influential feature, followed by other interaction metrics.



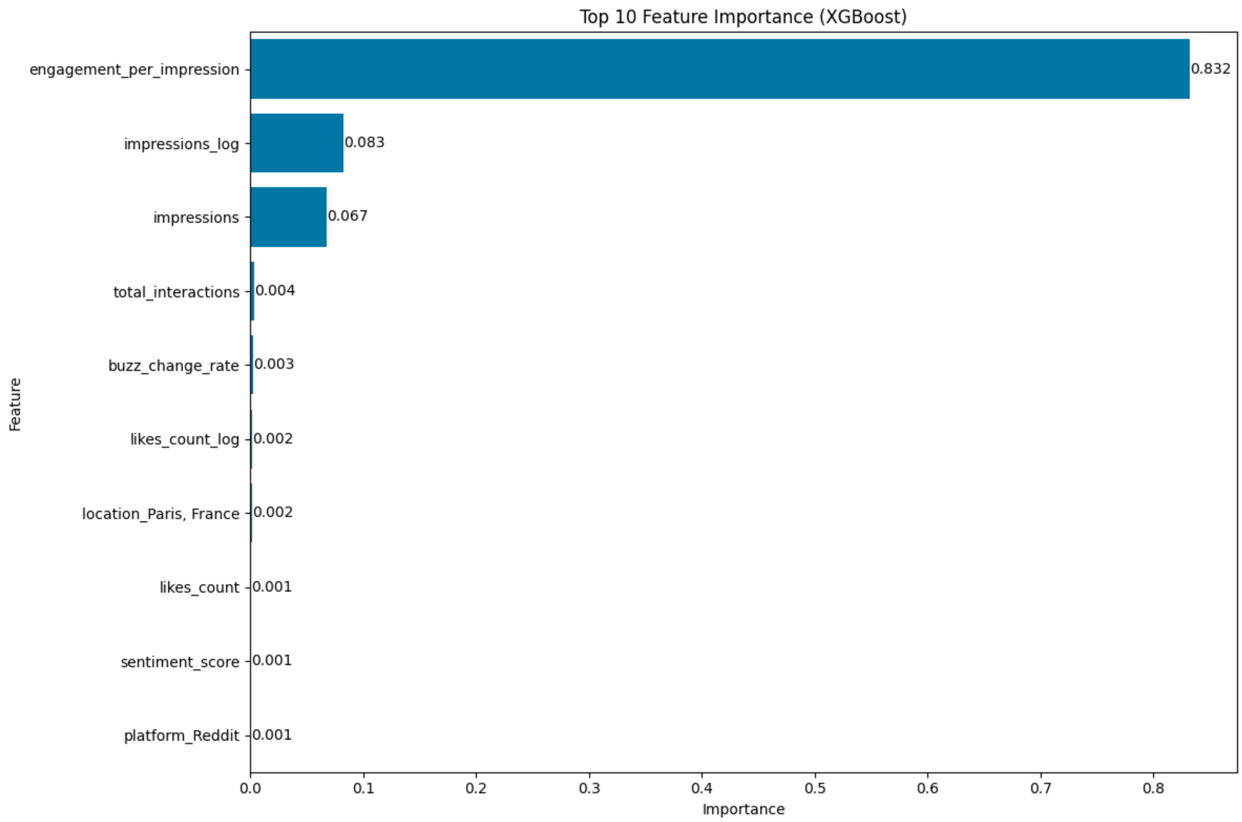
XGBoost Model

When this model was evaluated, it showed the following results:

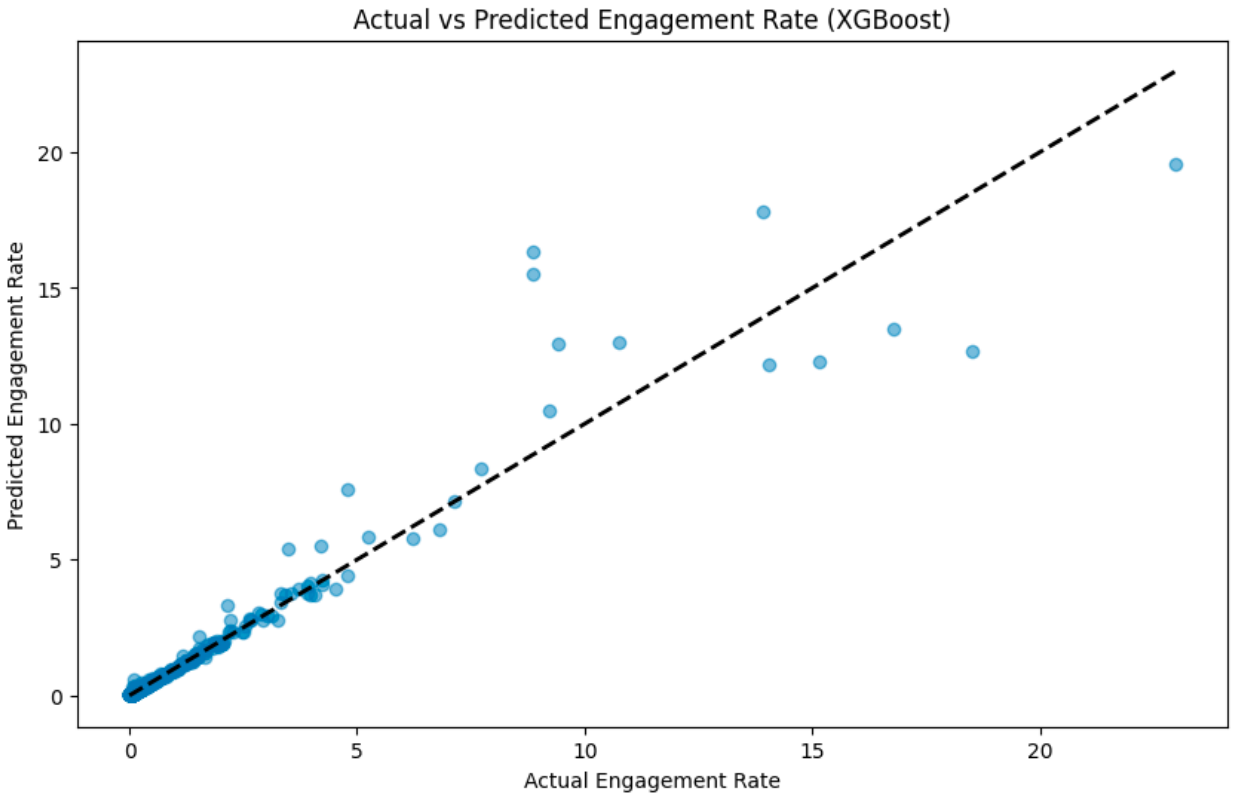
- RMSE: 0.3853
- R-squared (R^2): 0.8825
- MAE: 0.0336

Although slightly less accurate than the Random Forest model, XGBoost's performance is more realistic and less prone to overfitting, explaining 88.25% of the variance.

The feature importance analysis revealed that "engagement_per_impression " was the most critical feature, with "impressions_log" ranking second. The consistency of engagement_per_impression as a top feature across models validates its predictive power.



When plotting the Actual vs. Predicted engagement rate for XGBoost, we see the strong results.


Actual vs Predicted Engagement Rate (XGBoost)
Predicted Engagement Rate
Actual Engagement Rate
0
5
10
15
20
0
5
10
15
20

Conclusion

This study explored the predictive capabilities of three machine learning models (Linear Regression, Random Forest, and XGBoost) in estimating social media engagement rates. The findings indicate that while Linear Regression is inadequate for this task, both Random Forest and XGBoost models demonstrate strong predictive power. However, the Random Forest model's near-perfect accuracy raises concerns about overfitting, making the XGBoost model a more suitable choice for deployment.

Based on the analysis, the following recommendations are proposed:

1. Adopt XGBoost for Predictive Modeling for its balanced performance and reliability.
2. Optimize the XGBoost model using techniques such as GridSearchCV or RandomizedSearchCV to fine-tune parameters like learning rate and max depth.
3. Validate the model's performance using different data splits to ensure robustness.
4. Leverage feature importance insights to guide content strategy and platform selection.
5. Establishing a feedback loop to continuously improve model accuracy.
6. Expand data to include newer trends or platforms, such as TikTok.

Future work could explore temporal trends (e.g., hour of posting) and incorporate real-time data to enhance predictive power.

By implementing these recommendations, social media managers and marketing teams can enhance their predictive analytics capabilities, leading to more effective marketing strategies and improved ROI.

It's important to note that while the models perform well on the training data, further validation is needed to ensure they generalize well to new data. Hyperparameter tuning and cross-validation should be conducted to optimize the XGBoost model for deployment.

References

Subash Shanmugam, Social Media Engagement Dataset. Retrieved on June 2025, from Kaggle:

https://www.kaggle.com/datasets/subashmaster0411/social-media-engagement-dataset

Appendix - Python Code – Jupyter Notebook

Week 10, assignment 10.2: Course Project, Milestone 5

Javier Corpus

DSC630-T302 Predictive Analytics

Professor Andrew Hua

Visualizations

In [ ]:

```
# Importing the required libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('../Social Media Engagement Dataset.csv')
```

Dropping columns that are not necessary

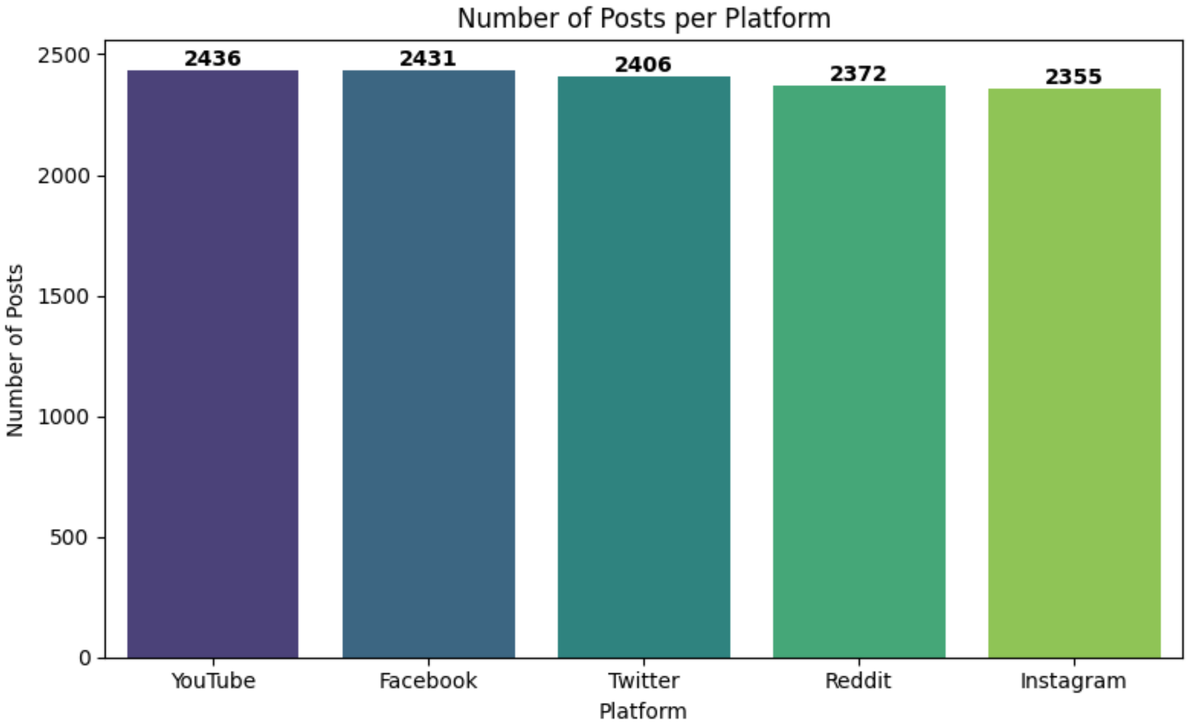
In [ ]:

```
df = df.drop(columns=[
    'post_id', 'user_id', 'text_content', 'hashtags', 'mentions',
    'keywords', 'campaign_phase', 'user_past_sentiment_avg',
    'user_engagement_growth', 'buzz_change_rate'
])
```

Number of posts per Platform

In [ ]:

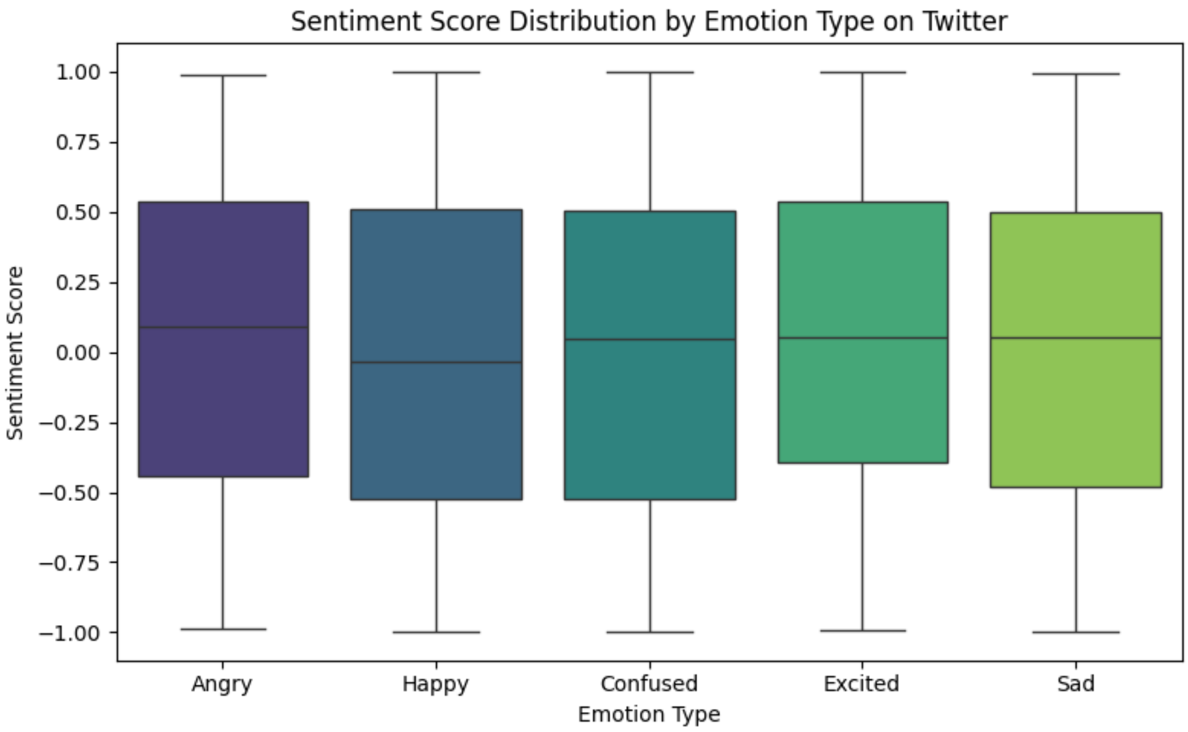
```
# Comparing number of posts per platform.
platform_counts = df['platform'].value_counts().reset_index()
platform_counts.columns = ['platform', 'count']

plt.figure(figsize=(8, 5))
ax = sns.barplot(data=platform_counts,
x='platform',
y='count',
palette='viridis',
hue='platform')

for i, row in platform_counts.iterrows():
    ax.text(i, row['count'], row['count'], ha='center', va='bottom',
            fontweight='bold')

plt.title('Number of Posts per Platform')
plt.xlabel('Platform')
plt.ylabel('Number of Posts')
plt.tight_layout()
plt.show()
```



Sentiment Score Distribution by Emotion - Twitter

In [ ]:

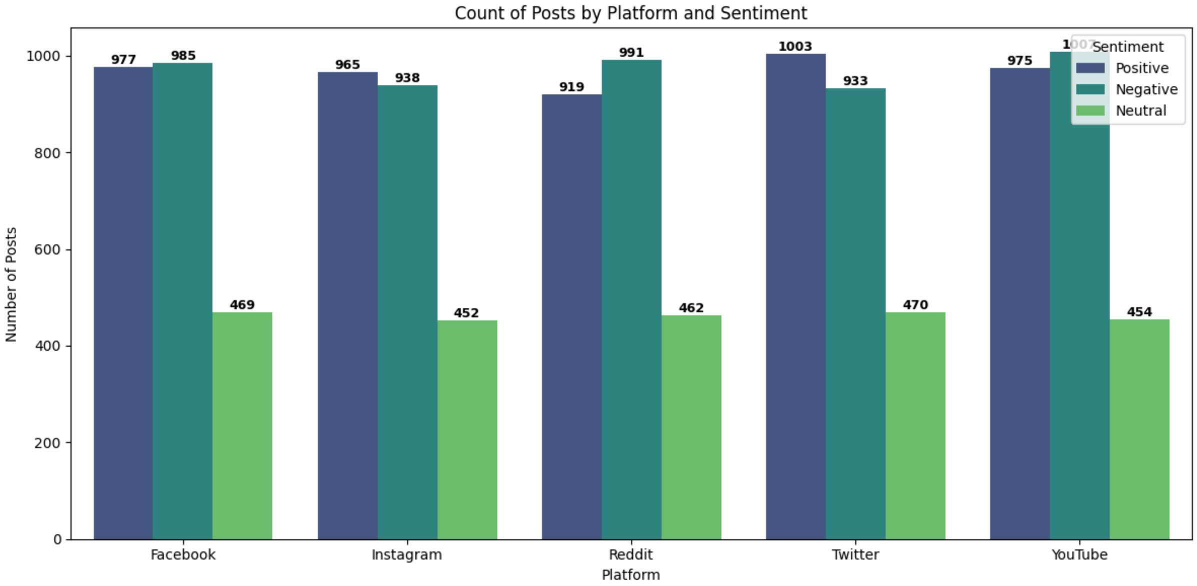
```
# Sentiment Score Distribution by Emotion – Twitter
plt.figure(figsize=(8, 5))
sns.boxplot(
    data=df[df['platform'] == 'Twitter'],
    x='emotion_type',
    y='sentiment_score',
    palette='viridis',
    hue='emotion_type',
)
plt.title('Sentiment Score Distribution by Emotion Type on Twitter')
plt.xlabel('Emotion Type')
plt.ylabel('Sentiment Score')
plt.tight_layout()
plt.show()
```



Count of posts by Platform and Sentiment

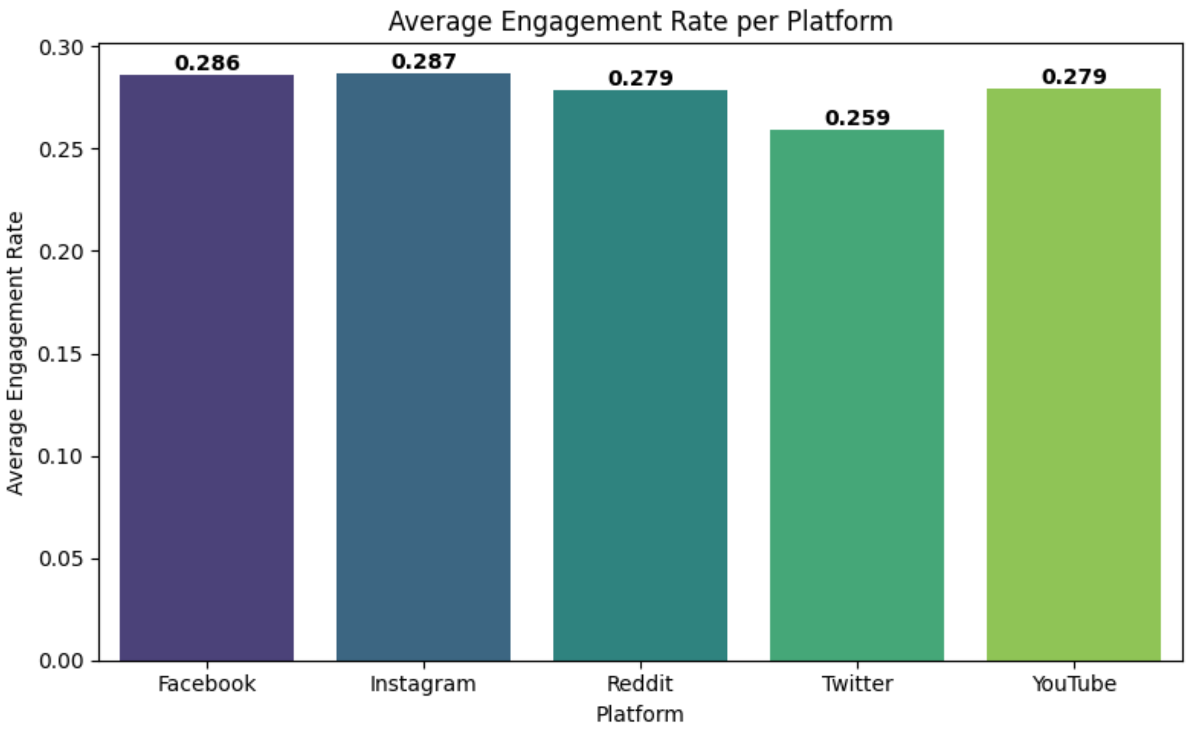
In [ ]:

```
# Count posts by brand_name and sentiment_label
platform_sentiment_counts = df.groupby(['platform',
        'sentiment_label']).size().reset_index(name='count')

# Sentiment order and color mapping
sentiment_order = ['Positive', 'Negative', 'Neutral']

plt.figure(figsize=(12, 6))
ax = sns.barplot(
    data=platform_sentiment_counts,
    x='platform',
    y='count',
    hue='sentiment_label',
    order=sorted(df['platform'].unique()),
    hue_order=sentiment_order,
    palette='viridis'
)

# Showing the value of each bar
for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.annotate(f'{int(height)}',
                    (p.get_x() + p.get_width() / 2, height),
                    ha='center', va='bottom',
                    fontweight='bold', fontsize=9)

plt.title('Count of Posts by Platform and Sentiment')
plt.xlabel('Platform')
plt.ylabel('Number of Posts')
plt.legend(title='Sentiment')
plt.tight_layout()
plt.show()
```



Average Engagement per Platform

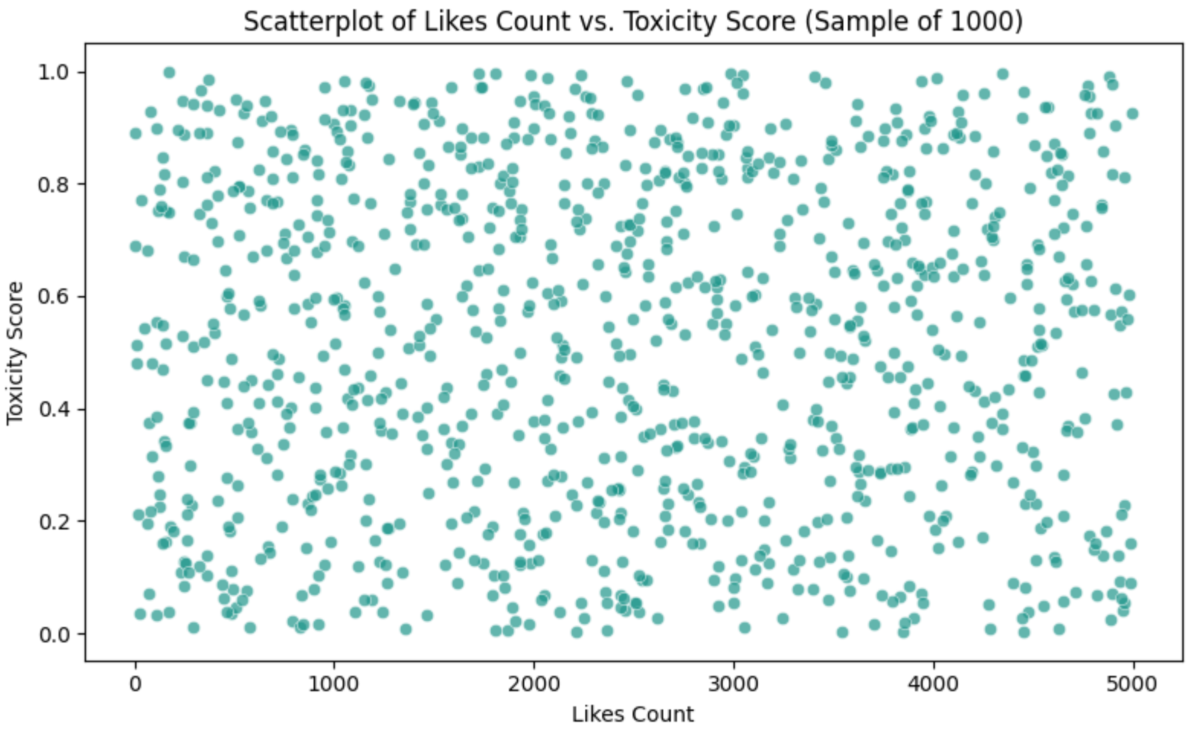
In [ ]:
```
# calculating average engagement rate per platform
avg_engagement = (
    df.groupby('platform')['engagement_rate'].mean().reset_index())

plt.figure(figsize=(8, 5))
ax = sns.barplot(data=avg_engagement,
x='platform',
y='engagement_rate',
palette='viridis',
hue='platform')

for i, row in avg_engagement.iterrows():
    ax.text(i, row['engagement_rate'],
            f"{row['engagement_rate']:.3f}", ha='center', va='bottom',
            fontweight='bold')

plt.title('Average Engagement Rate per Platform')
plt.xlabel('Platform')
plt.ylabel('Average Engagement Rate')
plt.tight_layout()
plt.show()
```



Scatterplot - Likes Count vs. Toxicity Score

In [ ]:

```
sample_df = df.sample(n=1000, random_state=100)
plt.figure(figsize=(8, 5))
sns.scatterplot(data=sample_df,
x='likes_count',
y='toxicity_score',
color='#1F968BFF',
alpha=0.7)
plt.title('Scatterplot of Likes Count vs.\
 Toxicity Score (Sample of 1000)')
plt.xlabel('Likes Count')
plt.ylabel('Toxicity Score')
plt.tight_layout()
plt.show()
```



Linear Regression Model

Importing required libraries

- pandas: for data manipulation and analysis.
- numpy: to calculate the square root of the Mean Squared Error.
- train_test_split: to split the dataset into training and test sets.
- LinearRegression: to create a Linear Regression model.
- mean_squared_error, mean_absolute_error, r2_score: to calculate the performance of the model.
- OneHotEncoder: to convert categorical variables into zeroes and ones.
- ColumnTransformer: to preprocess hot-encoded categorical variables.
- pyplot: To create visualizations.

In [ ]:

```
# Importing the required library
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,\
r2_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
import matplotlib.pyplot as plt

# Loading the dataset
```

```
df = pd.read_csv('../Social Media Engagement Dataset.csv')
```

Removing unnecessary features

We are going to remove features that are not required to predict Social Media engagement, such as unique identifiers, fields that would require NLP (like `text_content`), fields with high cardinality (`hashtags`, `mentions`), and other non-predictive columns.

In [ ]:

```
features = ['day_of_week', 'platform', 'location', 'language',
            'sentiment_score', 'emotion_type', 'toxicity_score',
            'likes_count', 'shares_count', 'comments_count',
            'impressions', 'topic_category', 'campaign_phase',
            'user_past_sentiment_avg', 'user_engagement_growth',
            'buzz_change_rate']

target = 'engagement_rate'

# Separating features and target
X = df[features]
y = df[target]

# Categorical columns
categorical_cols = ['day_of_week', 'platform', 'location', 'language',
                    'emotion_type', 'topic_category',
                    'campaign_phase']

# Numerical columns (all the remaining non-categorical columns)
numerical_cols = [col for col in features
                  if col not in categorical_cols]
```

Preprocessing the data, splitting it into Train (80%) and Test (20%) sets.

In [ ]:

```
# Preprocess the data with one-hot encoding for categorical variables
preprocessor = ColumnTransformer(
    transformers=[
        ('num', 'passthrough', numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'),
         categorical_cols)
    ])

# Fit and transform the data
X_processed = preprocessor.fit_transform(X)

# Splitting the data into training (80%) and test (20%) sets
X_train, X_test, y_train, y_test = train_test_split(
    X_processed, y, test_size=0.2, random_state=100)
```

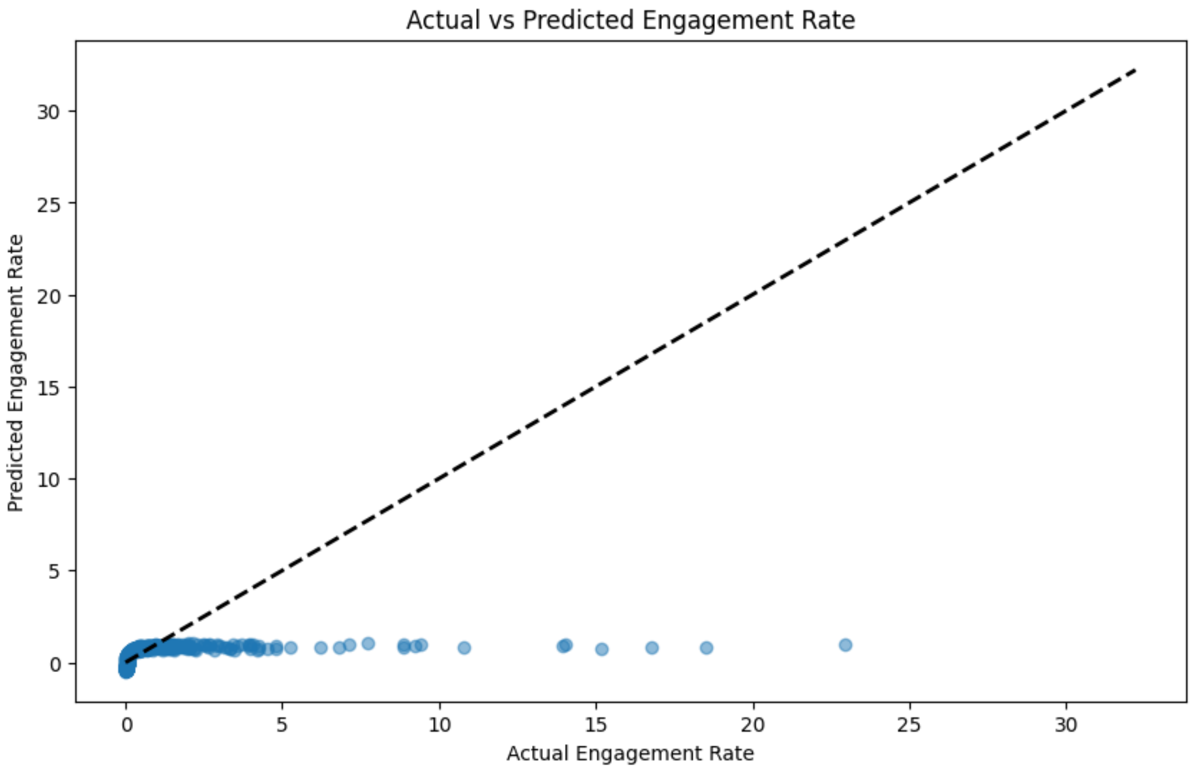
Creating and evaluating the model performance - Linear Regression

In [ ]:
```
# Creating and training the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Evaluating the model
y_pred = model.predict(X_test)

# Calculating metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("\nModel Performance")
print('------------------')
print(f'Root Mean Squared Error (RMSE): {rmse:.4f}')
print(f'R-squared (R²): {r2:.4f}')
print(f'Mean Absolute Error: {mean_absolute_error(y_test,
                                                  y_pred):.4f}')
```

```
Model Performance
-----------------
Root Mean Squared Error (RMSE): 1.0323
R-squared (R²): 0.1118
Mean Absolute Error: 0.3496
```

Predicted vs. Actual values - Linear Regression

In [ ]:
```
# Visualizing predictions vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
plt.xlabel('Actual Engagement Rate')
plt.ylabel('Predicted Engagement Rate')
plt.title('Actual vs Predicted Engagement Rate')
plt.show()
```



Feature Importance - Linear Regression

In [ ]:

```
# Examining feature importance (coefficients)
cat_encoder = preprocessor.named_transformers_['cat']
cat_feature_names = (
    cat_encoder.get_feature_names_out(categorical_cols))
all_feature_names = numerical_cols + list(cat_feature_names)

# Creating a data frame of coefficients
coefficients = pd.DataFrame({
    'Feature': all_feature_names,
    'Coefficient': model.coef_
}).sort_values(by='Coefficient', ascending=False)

print('\nTop 10 Positive Impact Features:')
print(coefficients.head(10))
```

```
Top 10 Positive Impact Features:
                       Feature   Coefficient
3                 shares_count  7.026817e-05
2                  likes_count  6.464678e-05
4               comments_count  5.284060e-05
69      topic_category_Delivery  3.460225e-06
9           day_of_week_Friday  2.468052e-06
77  campaign_phase_Pre-Launch  1.442138e-06
71       topic_category_Pricing  1.215408e-06
59                 language_hi  9.864609e-07
43        location_Paris, France  9.830177e-07
66        emotion_type_Excited  9.123405e-07
```

Random Forest

Importing required libraries

- pandas: for data manipulation and analysis.
- numpy: to calculate the square root of the Mean Squared Error.
- pyplot/seaborn: To create visualizations.
- train_test_split: to split the dataset into training and test sets.
- RandomizedSearchCV: for random search with cross-validation.
- RandomForestRegressor: to create a Random Forest model.
- mean_squared_error, mean_absolute_error, r2_score: to calculate the performance of the model.
- OneHotEncoder: to convert categorical variables into zeroes and ones.
- ColumnTransformer: to preprocess hot-encoded categorical variables.
- Pipeline: to chain together multiple data processing steps and model training into one object.
- SimpleImputer: to handle missing data (in this exercise, with median and most frequent values).
- randint: to generate random numbers for hyperparameters.

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, \
    RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score,\
      mean_absolute_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from scipy.stats import randint

# Loading the dataset
df = pd.read_csv('../Social Media Engagement Dataset.csv')

# Dropping columns with high cardinality or not useful for prediction
cols_to_drop = ['post_id', 'timestamp', 'user_id', 'text_content',
'hashtags', 'mentions', 'keywords', 'brand_name', 'product_name']
df = df.drop(cols_to_drop, axis=1)

# Converting categorical columns to appropriate types
categorical_cols = ['day_of_week', 'platform', 'location', 'language',
                   'sentiment_label', 'emotion_type',
                   'campaign_name', 'campaign_phase']
df[categorical_cols] = df[categorical_cols].astype('category')
```

Feature engineering

In [ ]:

```
# Feature Engineering
# Creating interaction features
df['total_interactions'] = (df['likes_count'] + df['shares_count'] +
                           df['comments_count'])
df['engagement_per_impression'] = (df['total_interactions'] /
                                 (df['impressions'] + 1))
df['comments_to_likes_ratio'] = (df['comments_count'] /
                               (df['likes_count'] + 1))
df['shares_to_likes_ratio'] = (df['shares_count'] /
                             (df['likes_count'] + 1))

# Log transform skewed numerical features
skewed_features = ['likes_count', 'shares_count',
                   'comments_count', 'impressions']

for feature in skewed_features:
    df[feature + '_log'] = np.log1p(df[feature])

# Preparing data for modeling
X = df.drop('engagement_rate', axis=1)
y = df['engagement_rate']
```

Splitting the data into Training (80%) and Test (20%) sets.

In [ ]:

```
# Splitting data into train (80%) and test (20%) sets
X_train, X_test, y_train, y_test = (
train_test_split(X, y, test_size=0.2, random_state=100))
```

Preprocessing data, hyperparameter tuning.

In [ ]:

```
# Preprocessing
# Identifying numeric and categorical features
numeric_features = (
    X.select_dtypes(include=['int64', 'float64']).columns)
categorical_features = (
    X.select_dtypes(include=['category']).columns)

# Creating transformers
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median'))
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore',
                             sparse_output=False))
])

# Combining preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Creating a Random Forest model pipeline
rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(random_state=100))
])

# Hyperparameter tuning setup
param_dist = {
    'regressor__n_estimators': randint(100, 500),
    'regressor__max_depth': [None, 10, 20, 30, 50],
    'regressor__min_samples_split': randint(2, 20),
    'regressor__min_samples_leaf': randint(1, 10),
    'regressor__max_features': ['sqrt', 'log2', 0.5, 0.8]
}

# Randomized search with cross-validation
random_search = RandomizedSearchCV(
    rf_pipeline,
    param_distributions=param_dist,
    n_iter=50,
    cv=5,
    scoring='neg_mean_squared_error',
    random_state=100,
    n_jobs=-1,
    verbose=1
)

print('Starting hyperparameter tuning...')
random_search.fit(X_train, y_train)
print('Hyperparameter tuning completed!')
```

```
Starting hyperparameter tuning...
Fitting 5 folds for each of 50 candidates, totalling 250 fits
Hyperparameter tuning completed!
```

Evaluating the model - Random Forest

In [ ]:

```
# Getting the best model
best_rf = random_search.best_estimator_

# Making predictions
```

```python
y_pred = best_rf.predict(X_test)

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print('\nModel Evaluation Metrics')
print('------------------------')
print(f'Best Parameters: {random_search.best_params_}')
print(f'Mean Squared Error: {mse:.4f}')
print(f'Root Mean Squared Error: {rmse:.4f}')
print(f'Mean Absolute Error: {mae:.4f}')
print(f'R-squared: {r2:.4f}')
```

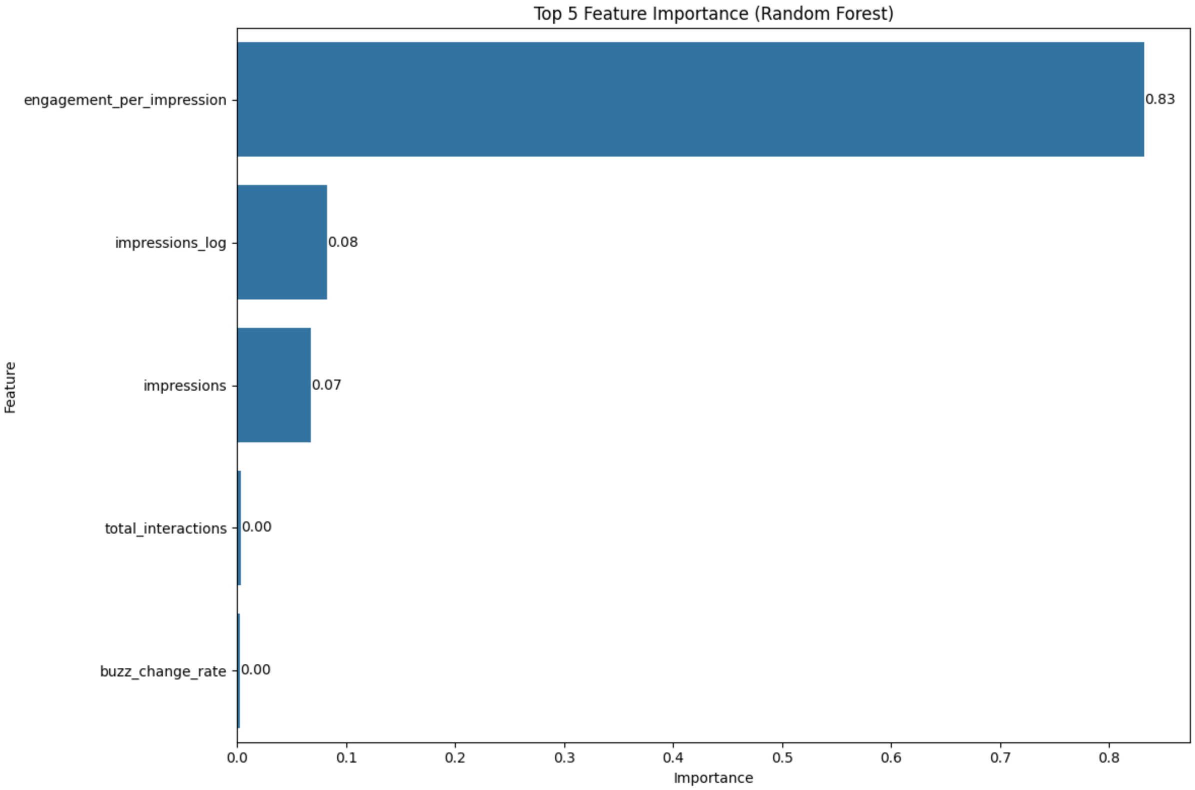
```
Model Evaluation Metrics
------------------------
Best Parameters: {'regressor__max_depth': 50, 'regressor__max_feature
s': 0.8, 'regressor__min_samples_leaf': 1, 'regressor__min_samples_spli
t': 6, 'regressor__n_estimators': 142}
Mean Squared Error: 0.0024
Root Mean Squared Error: 0.0495
Mean Absolute Error: 0.0036
R-squared: 0.9980
```

Feature Importance - Random Forest

In [ ]:

```python
# Feature Importance
# Getting feature names after preprocessing
numeric_feature_names = numeric_features.tolist()
categorical_feature_names = (
    best_rf.named_steps['preprocessor'].named_transformers_['cat'].nam
all_feature_names = (
    numeric_feature_names + list(categorical_feature_names))

# Getting feature importances
importances = best_rf.named_steps['regressor'].feature_importances_

# Creating a data frame for feature importance
feature_importance = pd.DataFrame({
    'Feature': all_feature_names,
    'Importance': importances
}).sort_values('Importance', ascending=False)

# Plot top 10 features
plt.figure(figsize=(12, 8))
bar_plot = sns.barplot(x='Importance',
                       y='Feature',
                       data=feature_importance.head(5))

# Adding the values at the end of each bar
```

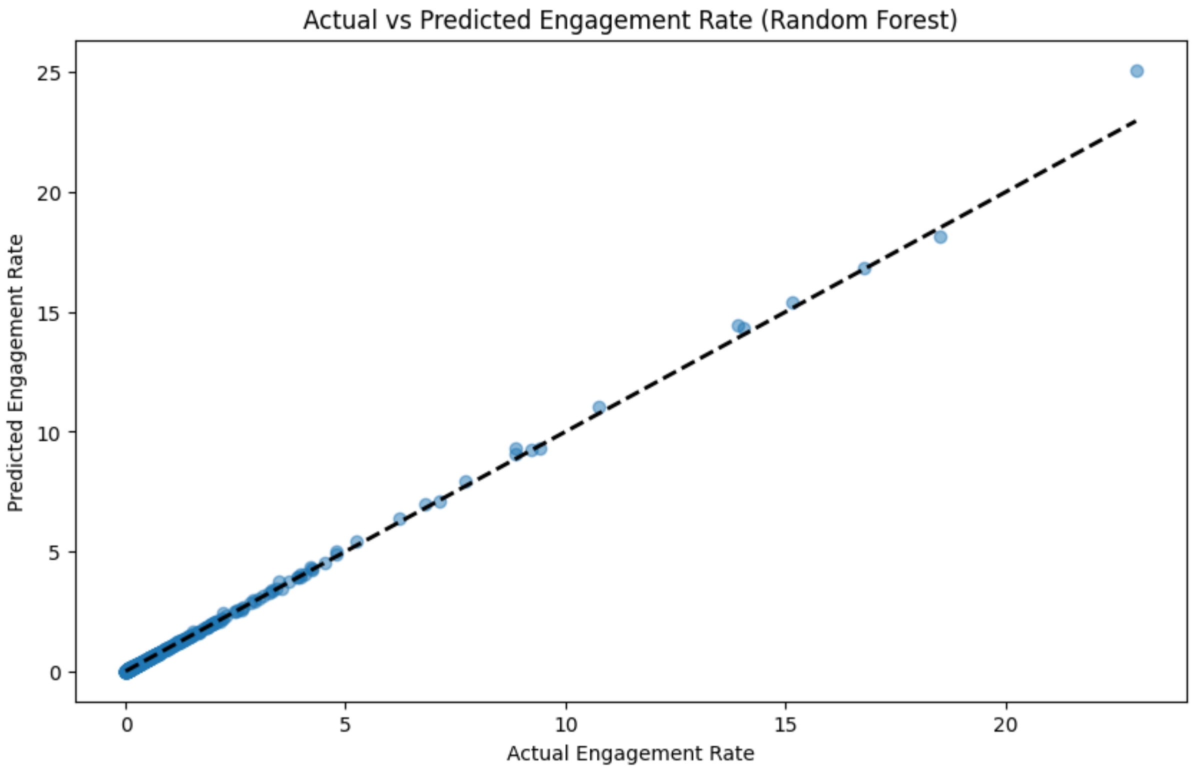
```
for p in bar_plot.patches:
    bar_plot.annotate(f'{p.get_width():.2f}',
                      (p.get_width(), p.get_y() + p.get_height() / 2),
                      ha='left', va='center')

plt.title('Top 5 Feature Importance (Random Forest)')
plt.tight_layout()
plt.show()
```



Actual vs. Predicted - Random Forest

In [ ]:

```
# Actual vs Predicted plot
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel('Actual Engagement Rate')
plt.ylabel('Predicted Engagement Rate')
plt.title('Actual vs Predicted Engagement Rate (Random Forest)')
plt.show()
```



XGBoost

Importing required libraries

- pandas: for data manipulation and analysis.
- numpy: to calculate the square root of the Mean Squared Error.
- pyplot/seaborn: To create visualizations.
- train_test_split: to split the dataset into training and test sets.
- RandomizedSearchCV: for random search with cross-validation.
- XGBRegressor: to create the XGBoost model pipeline
- mean_squared_error, mean_absolute_error, r2_score: to calculate the performance of the model.
- OneHotEncoder: to convert categorical variables into zeroes and ones.
- ColumnTransformer: to preprocess hot-encoded categorical variables.
- Pipeline: to chain together multiple data processing steps and model training into one object.
- SimpleImputer: to handle missing data (in this exercise, with median and most frequent values).
- randint: to generate random numbers for hyperparameters.
- uniform: to generate random numbers with a uniform distribution

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, \
    RandomizedSearchCV
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score,\
      mean_absolute_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from scipy.stats import randint, uniform

# Loading the dataset
df = pd.read_csv('../Social Media Engagement Dataset.csv')
```

Data Preprocessing

In [ ]:

```
# Dropping high-cardinality or non-predictive columns
cols_to_drop = ['post_id', 'timestamp', 'user_id', 'text_content',
                'hashtags', 'mentions', 'keywords']
df = df.drop(cols_to_drop, axis=1)

# Converting to categorical
categorical_cols = ['day_of_week', 'platform', 'location', 'language',
                    'topic_category', 'sentiment_label',
                    'emotion_type', 'brand_name', 'product_name',
                   'campaign_name', 'campaign_phase']

df[categorical_cols] = df[categorical_cols].astype('category')
```

Feature Engineering

In [ ]:

```
# Interaction features
df['total_interactions'] = (
    df['likes_count'] + df['shares_count'] + df['comments_count'])
df['engagement_per_impression'] = (
    df['total_interactions'] / (df['impressions'] + 1))
df['comments_to_likes_ratio'] = (
    df['comments_count'] / (df['likes_count'] + 1))
df['shares_to_likes_ratio'] = (
    df['shares_count'] / (df['likes_count'] + 1))

# Log-transform skewed features
skewed_features = ['likes_count', 'shares_count',
                   'comments_count', 'impressions']

for feature in skewed_features:
    df[f'log_{feature}'] = np.log1p(df[feature])
```

Creating the pipeline

In [ ]:

```
# Defining features and target
X = df.drop('engagement_rate', axis=1)
y = df['engagement_rate']

# Splitting the data into Training (80%) and Test (20%) sets.
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=100)


# Preprocessing features
numeric_features = (
    X.select_dtypes(include=['int64', 'float64']).columns)
categorical_features = (
    X.select_dtypes(include=['category']).columns)

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median'))
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore',
                             sparse_output=False))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# XGBoost model pipeline
xgb_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', XGBRegressor(random_state=100, n_jobs=-1))
])
```

Hyperparameter tuning

In [ ]:

```
param_dist = {
    'regressor__n_estimators': randint(100, 500),
    'regressor__max_depth': randint(3, 10),
    'regressor__learning_rate': uniform(0.01, 0.3),
    'regressor__subsample': uniform(0.6, 0.4),
    'regressor__colsample_bytree': uniform(0.6, 0.4),
    'regressor__gamma': uniform(0, 0.5),
    'regressor__reg_alpha': uniform(0, 1),
    'regressor__reg_lambda': uniform(0, 1)
}

random_search = RandomizedSearchCV(
    xgb_pipeline,
    param_distributions=param_dist,
    n_iter=100,
    cv=5,
    scoring='neg_mean_squared_error',
    random_state=100,
    n_jobs=-1,
    verbose=1,
)

print('Starting hyperparameter tuning...')
random_search.fit(X_train, y_train)
print('Tuning completed!')
```

```
Starting hyperparameter tuning...
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Tuning completed!
```

Evaluating the model - XGBoost

In [ ]:

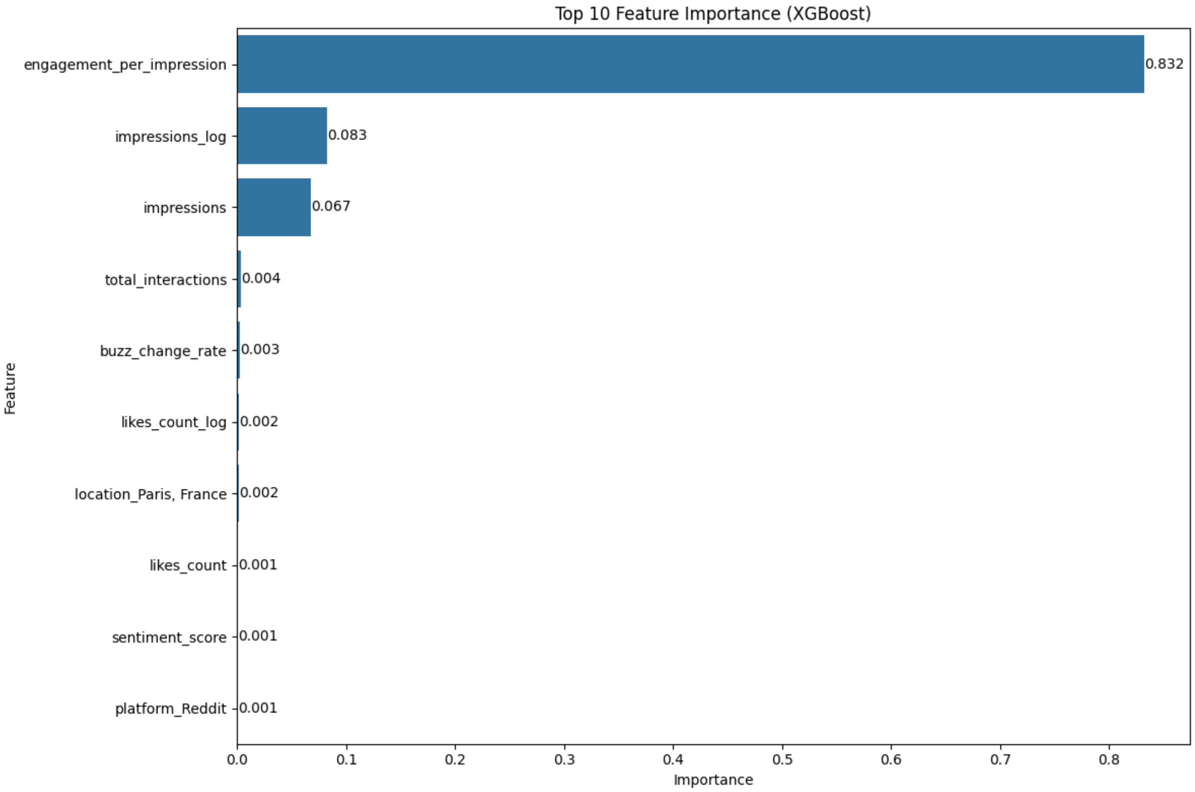
```
best_xgb = random_search.best_estimator_
y_pred = best_xgb.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print('\n=== Best Model Evaluation ===')
print(f'Best Parameters: {random_search.best_params_}')
print(f'Mean Squared Error (MSE): {mse:.4f}')
print(f'Root Mean Squared Error (RMSE): {rmse:.4f}')
print(f'Mean Absolute Error (MAE): {mae:.4f}')
print(f'R-squared (R²): {r2:.4f}')
```

```
=== Best Model Evaluation ===
Best Parameters: {'regressor__colsample_bytree': np.float64(0.611364742
6853007), 'regressor__gamma': np.float64(0.35066325704675805), 'regress
or__learning_rate': np.float64(0.01755146916544357), 'regressor__max_de
pth': 3, 'regressor__n_estimators': 403, 'regressor__reg_alpha': np.flo
at64(0.07352706186557412), 'regressor__reg_lambda': np.float64(0.060884
56434663547), 'regressor__subsample': np.float64(0.6445625266816213)}
Mean Squared Error (MSE): 0.0927
Root Mean Squared Error (RMSE): 0.3045
Mean Absolute Error (MAE): 0.0371
R-squared (R²): 0.9227
```

Feature Importance - XGBoost

In [ ]:

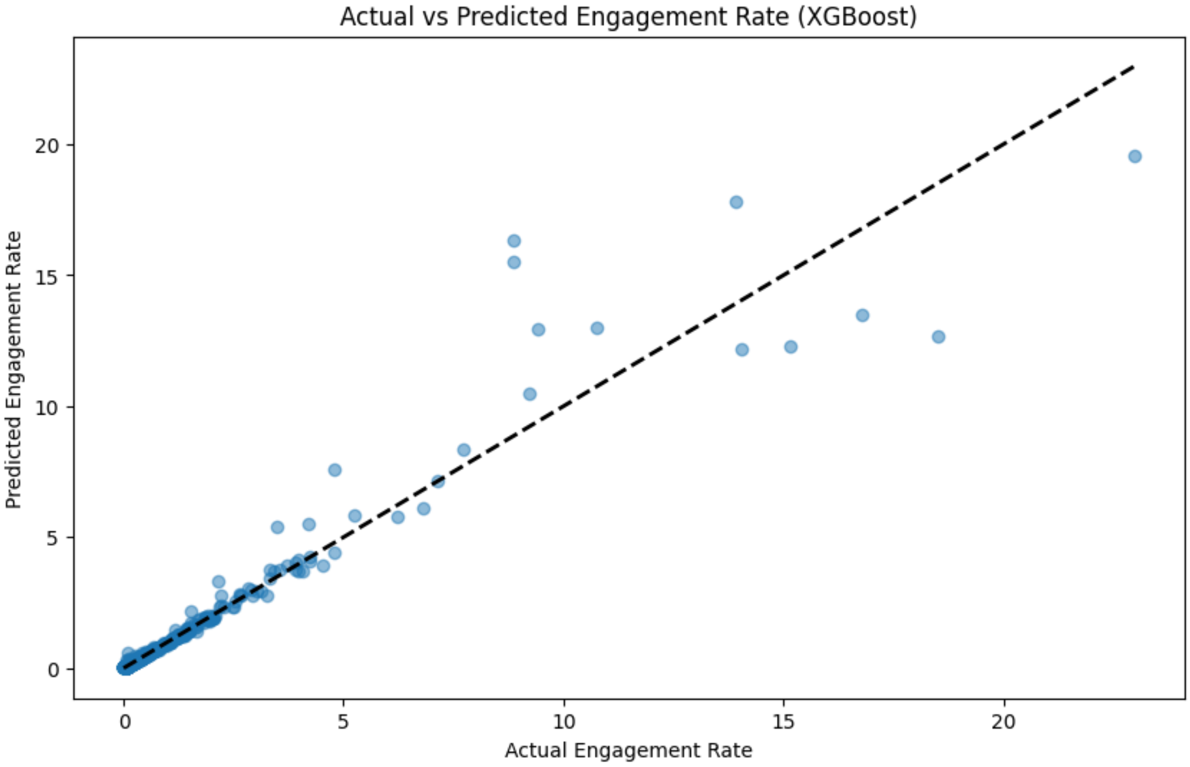
```
# Creating importance data frame
feature_importance = pd.DataFrame({
    'Feature': all_feature_names,
    'Importance': importances
}).sort_values('Importance', ascending=False)

# Plot top 10 features
plt.figure(figsize=(12, 8))
bar_plot = sns.barplot(x='Importance',
                       y='Feature',
                       data=feature_importance.head(10)
                       )

# Adding the values at the end of each bar
for p in bar_plot.patches:
    bar_plot.annotate(f'{p.get_width():.3f}',
                      (p.get_width(), p.get_y() + p.get_height() / 2),
                      ha='left', va='center')

plt.title('Top 10 Feature Importance (XGBoost)')
plt.tight_layout()
plt.show()
```



Actual vs Predicted values - XGBoost

In [ ]:

```
# Actual vs Predicted
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(),
                                        y_test.max()],
                                        'k--', lw=2)
plt.xlabel('Actual Engagement Rate')
plt.ylabel('Predicted Engagement Rate')
plt.title('Actual vs Predicted Engagement Rate (XGBoost)')
plt.show()
```


Actual vs Predicted Engagement Rate (XGBoost)
Predicted Engagement Rate
Actual Engagement Rate