```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import pearsonr
import statsmodels.api as sm
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score, explained_variance_score

# Loading the dataset from my drive
url = r"/content/drive/MyDrive/AAA_Empirical_Analysis/train_data.csv"

#Storing dataset in a datafram and displaying it
df = pd.read_csv(url)
df.head()
```

	campaign_id	sender	subject_len	body_len	mean_paragraph_len	day_of_week	is_we
0	1	3	76	10439	39	5	
1	2	3	54	2570	256	5	
2	3	3	59	12801	16	5	
3	4	3	74	11037	30	4	
4	5	3	80	10011	27	5	

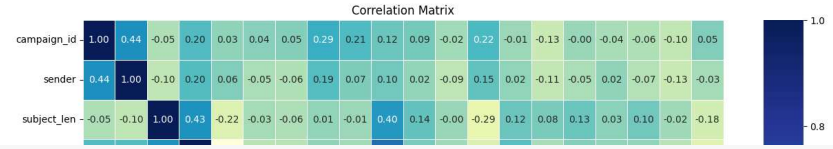
5 rows × 22 columns

```
df.describe().transpose()
```

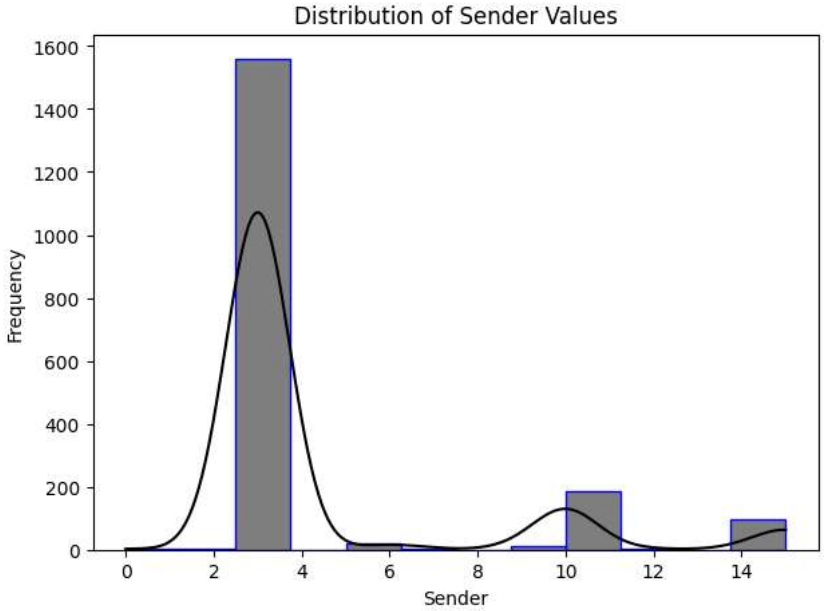
	count	mean	std	min	25%	50%
campaign_id	1888.0	944.500000	545.162973	1.0	472.750000	944.500000
sender	1888.0	4.395657	3.277927	0.0	3.000000	3.000000
subject_len	1888.0	86.246292	30.143206	9.0	69.000000	83.000000
body_len	1888.0	14185.780191	7327.615307	23.0	9554.500000	12689.000000
mean_paragraph_len	1888.0	35.239407	28.139498	4.0	21.000000	29.000000
day_of_week	1888.0	2.828390	1.763193	0.0	1.000000	3.000000
is_weekend	1888.0	0.199682	0.399867	0.0	0.000000	0.000000
category	1888.0	9.949682	5.300719	0.0	6.000000	10.000000
product	1888.0	17.525424	12.369526	0.0	9.000000	12.000000
no_of_CTA	1888.0	4.222458	4.628348	0.0	2.000000	3.000000
mean_CTA_len	1888.0	30.233051	11.848663	0.0	23.000000	29.000000
is_image	1888.0	0.909958	0.866467	0.0	0.000000	1.000000
is_personalised	1888.0	0.056674	0.231279	0.0	0.000000	0.000000
is_quote	1888.0	0.834216	1.033901	0.0	0.000000	1.000000
is_timer	1888.0	0.000000	0.000000	0.0	0.000000	0.000000
is_emoticons	1888.0	0.210805	0.613442	0.0	0.000000	0.000000
is_discount	1888.0	0.039725	0.195363	0.0	0.000000	0.000000
is_price	1888.0	40.197034	553.957470	0.0	0.000000	0.000000
is_urgency	1888.0	0.112288	0.315804	0.0	0.000000	0.000000
target_audience	1888.0	11.634534	2.949121	0.0	12.000000	12.000000
click_rate	1888.0	0.041888	0.084223	0.0	0.005413	0.010686

```
import seaborn as sns
import matplotlib.pyplot as plt

# Selecting relevant columns for correlation analysis
correlation_data = df.drop('is_timer', axis=1)

# Computing correlation matrix
correlation_matrix = correlation_data.corr()
plt.figure(figsize=(15, 15))

# Plotting correlation matrix using heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='YlGnBu', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```

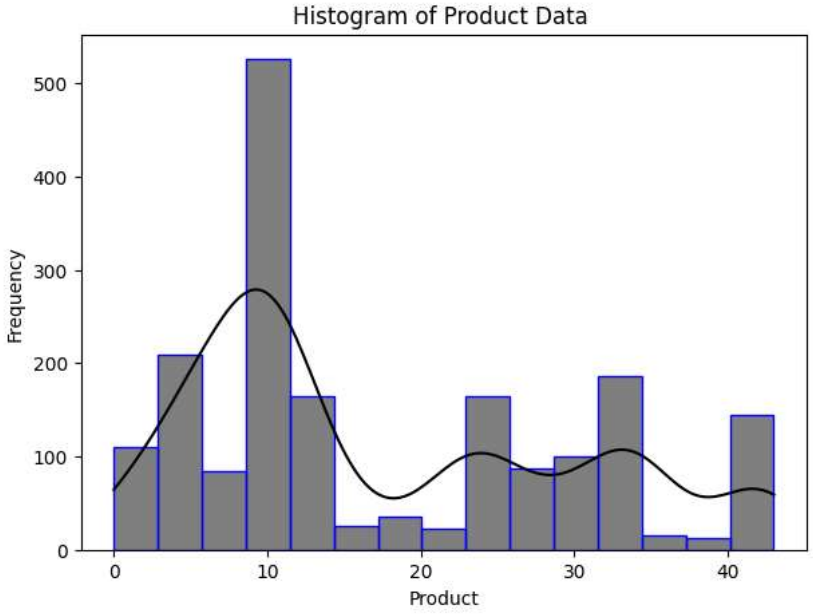
```
<ipython-input-4-521e89cdd9f1>:8: FutureWarning: The default value of numeric_only in
  correlation_matrix = correlation_data.corr()
```



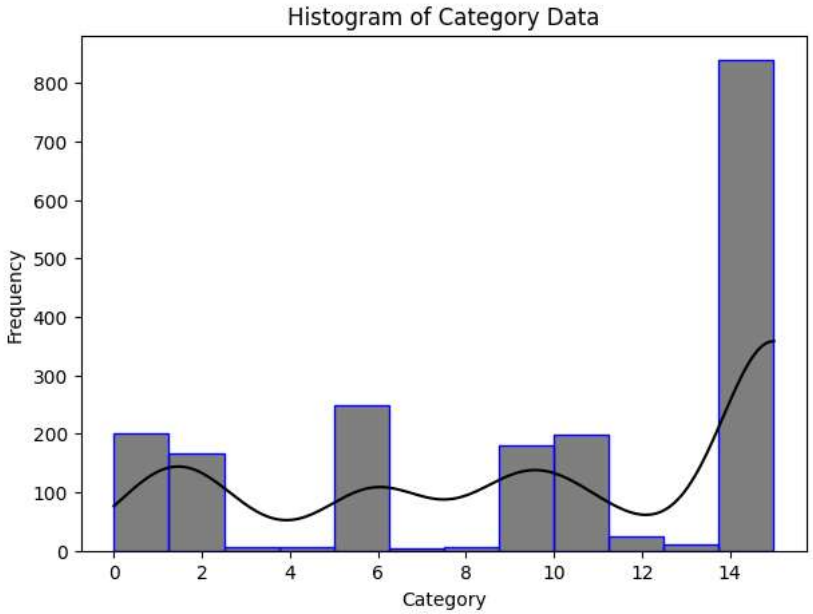

```
#Histogram of sender data
plt.figure(figsize=(7, 5))
sns.histplot(x='sender', data=df, bins='auto', kde=True, color='black', edgecolor='blue')

plt.xlabel('Sender')
plt.ylabel('Frequency')
plt.title('Distribution of Sender Values')
plt.show()
```




```
#histogram of product data
plt.figure(figsize=(7, 5))
sns.histplot(data=df, x='product', bins='auto', kde=True, color='black', edgecolor='blue')
plt.xlabel('Product')
plt.ylabel('Frequency')
plt.title('Histogram of Product Data ')
plt.show()
```

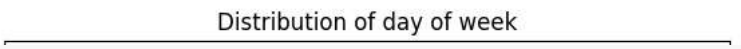


```
#Histogram of category data
plt.figure(figsize=(7, 5))
sns.histplot(data=df, x='category', bins='auto', kde=True, color='black', edgecolor='blue')
plt.xlabel('Category')
plt.ylabel('Frequency')
plt.title('Histogram of Category Data')
plt.show()
```

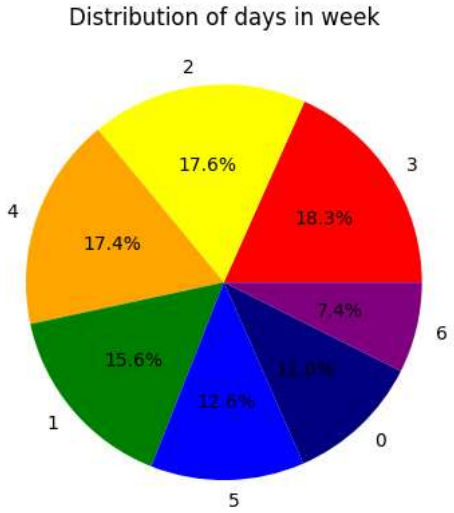



```
# independent variable 'is_weekend', 'days_of_week', and 'times_of_day'

weekend = np.array(df['is_weekend'])
daysOfWeek = np.array(df['day_of_week'])

# encode 'times_of_day' into numerical order
times = df["times_of_day"].unique()
time_mapping = {'Morning': 1, 'Noon': 2, 'Evening': 3}
times_encoded = df['times_of_day'].map(time_mapping)

# dependent variable 'click_rate'
click_rate = df['click_rate']
```

```
# Visualize each data

# visualize day_of_week in histogram
plt.figure(figsize=(7, 5))
sns.histplot(x='day_of_week', data=df, bins='auto', kde=False, color='black', edgecolor='blue')

plt.xlabel('Day_of_Week')
plt.ylabel('Frequency')
plt.title('Distribution of day of week')
plt.show()
```

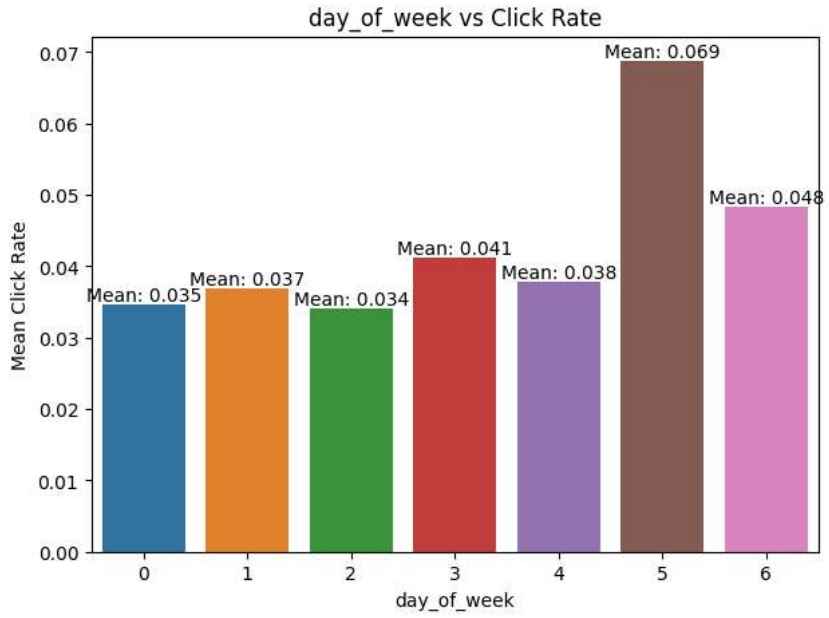



```
# count each element
value_counts = df['day_of_week'].value_counts()
# count total
total_count = len(df['day_of_week'])

# visualize day_of_week in pie chart
plt.pie(value_counts, labels=value_counts.index, autopct='%1.1f%%', colors=['red', 'yellow', 'orange', 'green', 'blue', 'navy', 'purple
plt.title('Distribution of days in week')
plt.show()
```

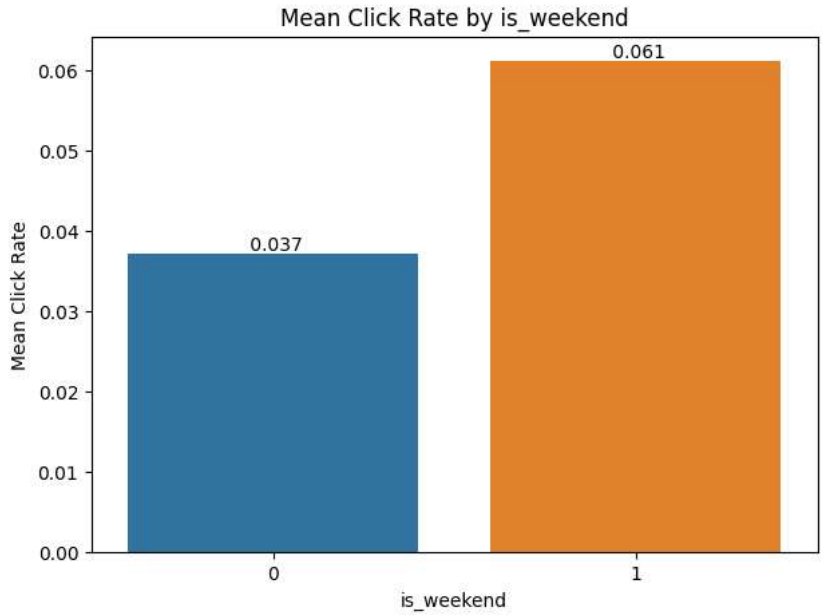



```
# Bar plot for 'day_of_week' vs 'click_rate'
plt.figure(figsize=(7, 5))
sns.barplot(x='day_of_week', y='click_rate', data=df, ci=None)
mean_click_rate_image = df.groupby('day_of_week')['click_rate'].mean()
for i, mean_rate in enumerate(mean_click_rate_image):
    plt.text(i, mean_rate, f'Mean: {mean_rate:.3f}', ha='center', va='bottom')
plt.title('day_of_week vs Click Rate')
plt.xlabel('day_of_week')
plt.ylabel('Mean Click Rate')
plt.show()
```

```
<ipython-input-11-6d3ecafb432f>:3: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='day_of_week', y='click_rate', data=df, ci=None)
```



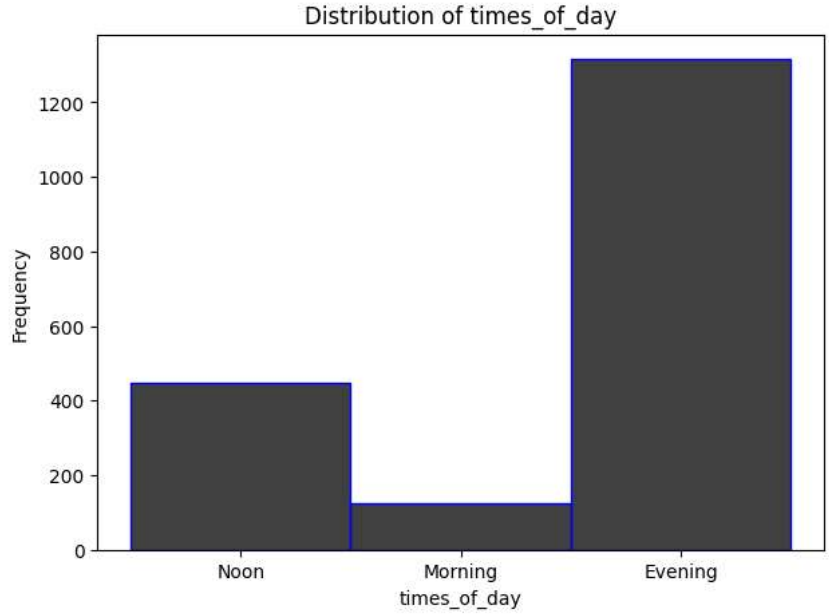
```
# Visualize 'is_weekend' vs 'click_rate'
plt.figure(figsize=(7, 5))
sns.barplot(x='is_weekend', y='click_rate', data=df, errorbar=None)
mean_click_rate_urgency = df.groupby('is_weekend')['click_rate'].mean()
for i, mean_rate in enumerate(mean_click_rate_urgency):
    plt.text(i, mean_rate, f' {mean_rate:.3f}', ha='center', va='bottom')
plt.title('is_weekend vs Click Rate')
plt.title('Mean Click Rate by is_weekend')
plt.xlabel('is_weekend')
plt.ylabel('Mean Click Rate')

plt.show()
```



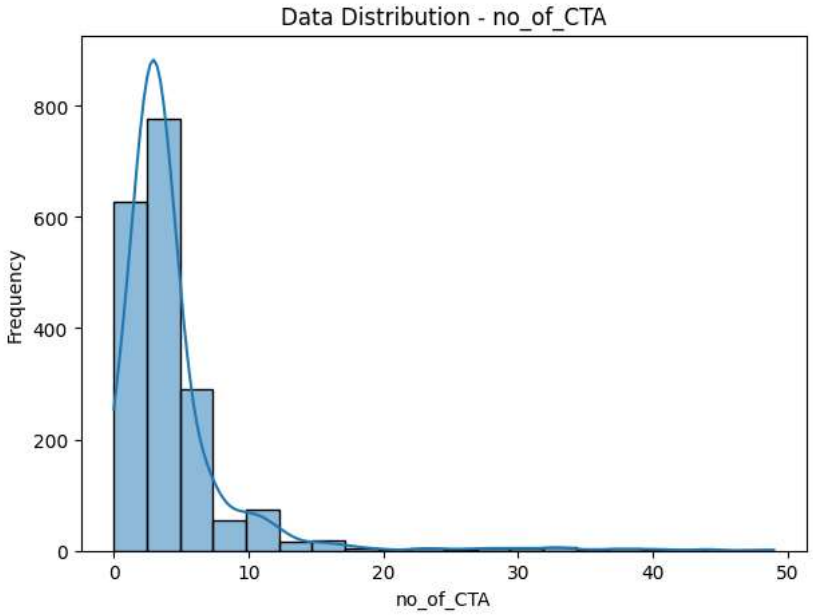

```
# check every kinds of element in 'times_of_day' column
unique_elements = df['times_of_day'].unique()
unique_elements
```

```
array(['Noon', 'Morning', 'Evening'], dtype=object)
```

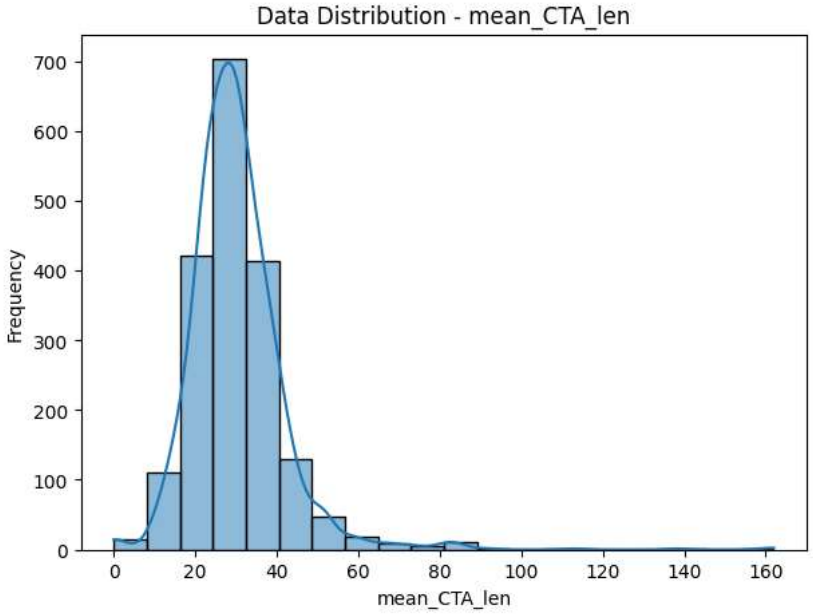
```
# visualize 'times_of_day' in histogram
plt.figure(figsize=(7, 5))
sns.histplot(x='times_of_day', data=df, bins='auto', kde=False, color='black', edgecolor='blue')

plt.xlabel('times_of_day')
plt.ylabel('Frequency')
plt.title('Distribution of times_of_day')
plt.show()
```



```
# Plotting a histogram to understand the distribution of subject lenth data
feature = 'no_of_CTA'
plt.figure(figsize=(7, 5))
sns.histplot(df[feature], bins=20, kde=True)
plt.title(f'Data Distribution - {feature}')
plt.xlabel(feature)
plt.ylabel('Frequency')
plt.show()
```

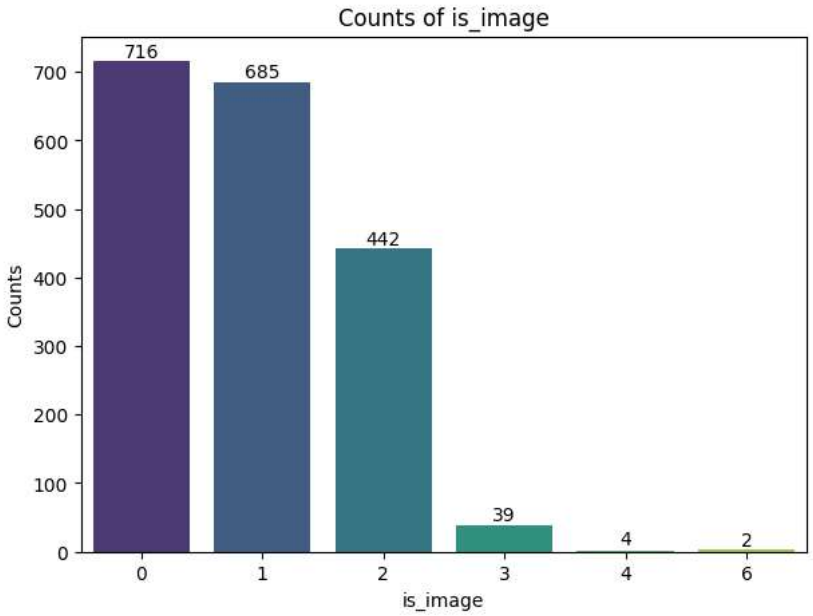



```
# Plotting a histogram to understand the distribution of subject lenth data
feature = 'mean_CTA_len'
plt.figure(figsize=(7, 5))
sns.histplot(df[feature], bins=20, kde=True)
plt.title(f'Data Distribution - {feature}')
plt.xlabel(feature)
plt.ylabel('Frequency')
plt.show()
```



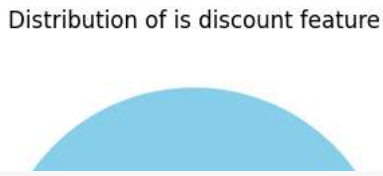
```
#is_image count and representation in bar plot

# Count values
value_counts = df['is_image'].value_counts()

# Display total count
total_count = len(df['is_image'])
print(f'Total Count: {total_count} counts\n')

plt.figure(figsize=(7, 5))
barplot = sns.barplot(x=value_counts.index, y=value_counts.values, palette='viridis')

# Displaying value counts on bars
for i, count in enumerate(value_counts.values):
    barplot.text(i, count + 0.1, str(count), ha='center', va='bottom')

plt.title('Counts of is_image')
plt.xlabel('is_image')
plt.ylabel('Counts')
plt.show()
```

```
Total Count: 1888 counts
```



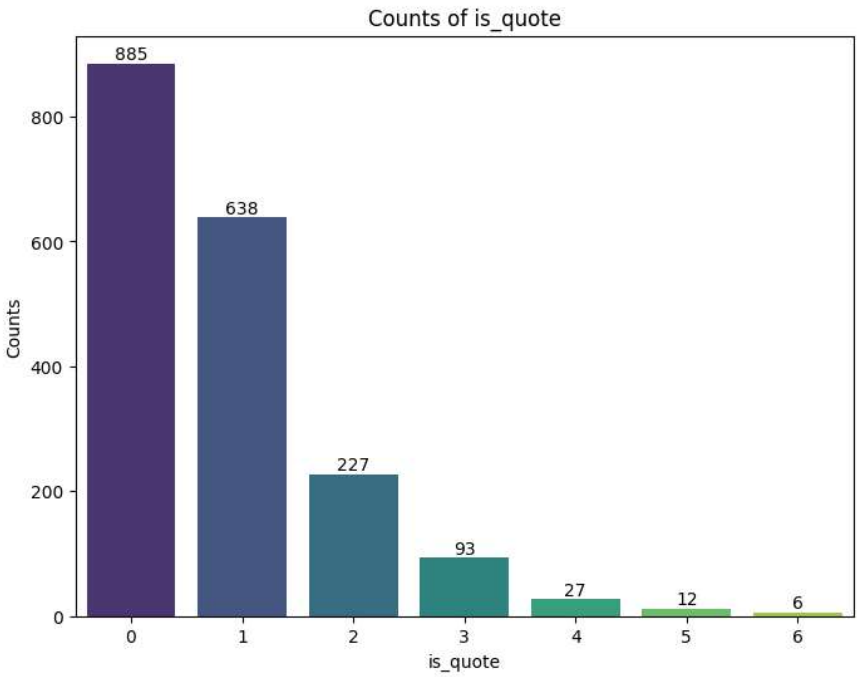

```
#is_personalised count and representation as piechart

# Count values
value_counts = df['is_personalised'].value_counts()

# Display counts
for index, count in value_counts.items():
    print(f'is_personalised {index}: {count} counts')

# Display total count
total_count = len(df['is_personalised'])
print(f'Total Count: {total_count} counts\n')

# Pie chart
plt.pie(value_counts, labels=value_counts.index, autopct='%1.1f%%', colors=['skyblue', 'lightgreen'])
plt.title('Distribution of is discount feature')
plt.show()
```

```
is_personalised 0: 1781 counts
is_personalised 1: 107 counts
Total Count: 1888 counts
```



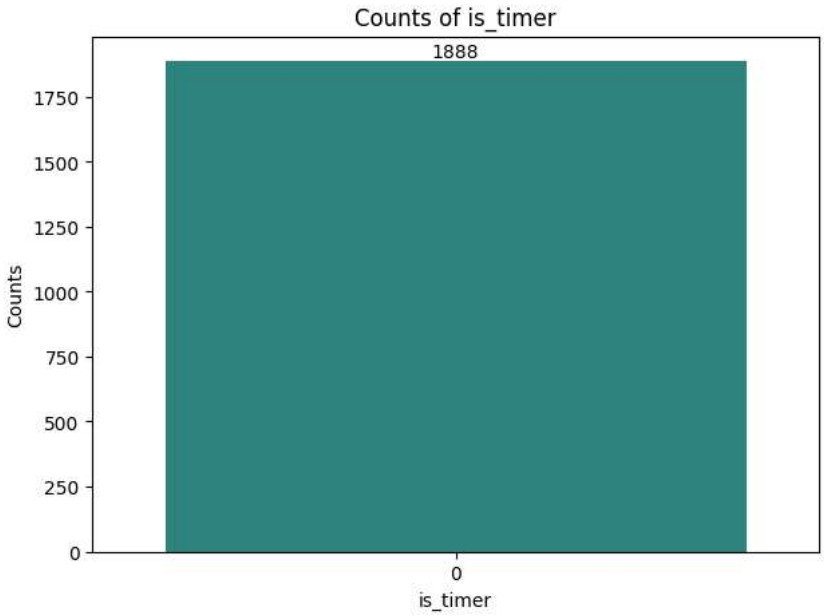

```
#is_quote count and representation in bar plot

# Count values
value_counts = df['is_quote'].value_counts()

# Display total count
total_count = len(df['is_quote'])
print(f'Total Count: {total_count} counts\n')

plt.figure(figsize=(8, 6))
barplot = sns.barplot(x=value_counts.index, y=value_counts.values, palette='viridis')

# Displaying value counts on bars
for i, count in enumerate(value_counts.values):
    barplot.text(i, count + 0.1, str(count), ha='center', va='bottom')

plt.title('Counts of is_quote')
plt.xlabel('is_quote')
plt.ylabel('Counts')
plt.show()
```

```
Total Count: 1888 counts
```



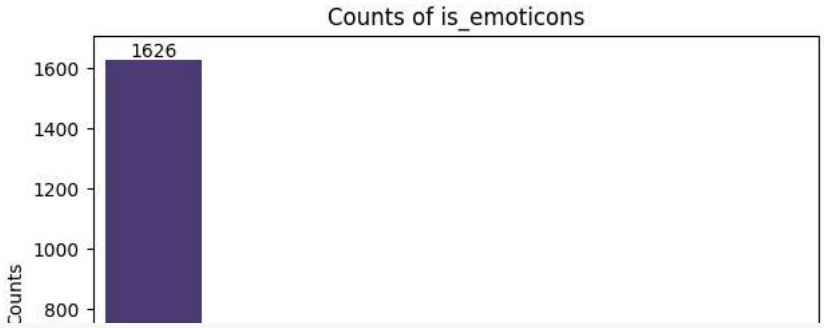
```
#is_timer count and representation in bar plot

# Count values
value_counts = df['is_timer'].value_counts()

# Display total count
total_count = len(df['is_timer'])
print(f'Total Count: {total_count} counts\n')

plt.figure(figsize=(7, 5))
barplot = sns.barplot(x=value_counts.index, y=value_counts.values, palette='viridis')

# Displaying value counts on bars
for i, count in enumerate(value_counts.values):
    barplot.text(i, count + 0.1, str(count), ha='center', va='bottom')

plt.title('Counts of is_timer')
plt.xlabel('is_timer')
plt.ylabel('Counts')
plt.show()
```

```
Total Count: 1888 counts
```



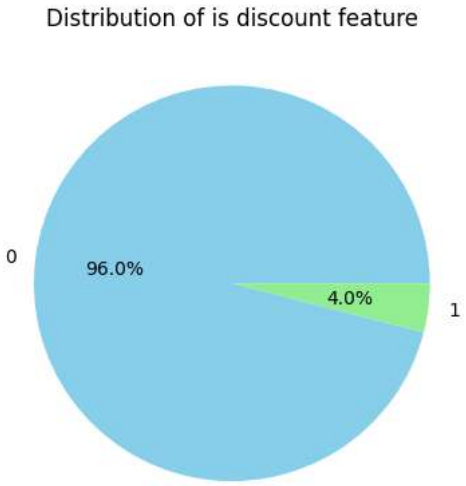

```
#Is_emoticons count and representation in bar plot

# Count values
value_counts = df['is_emoticons'].value_counts()

# Display total count
total_count = len(df['is_emoticons'])
print(f'Total Count: {total_count} counts\n')

plt.figure(figsize=(7, 5))
barplot = sns.barplot(x=value_counts.index, y=value_counts.values, palette='viridis')

# Displaying value counts on bars
for i, count in enumerate(value_counts.values):
    barplot.text(i, count + 0.1, str(count), ha='center', va='bottom')

plt.title('Counts of is_emoticons')
plt.xlabel('is_emoticons')
plt.ylabel('Counts')
plt.show()
```

```
Total Count: 1888 counts
```



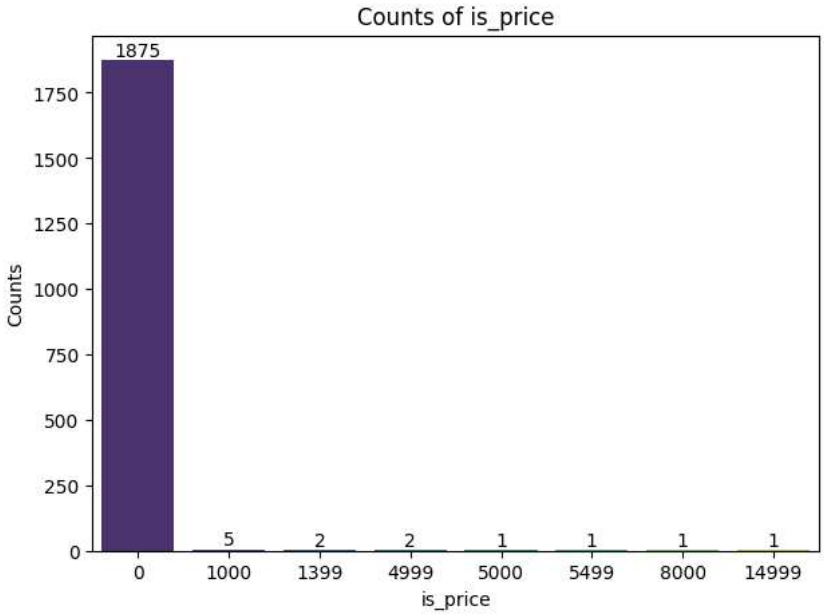

```
#Is_discount count and representation as piechart

# Count values
value_counts = df['is_discount'].value_counts()

# Display counts
for index, count in value_counts.items():
    print(f'is_discount {index}: {count} counts')

# Display total count
total_count = len(df['is_discount'])
print(f'Total Count: {total_count} counts\n')

# Pie chart
plt.pie(value_counts, labels=value_counts.index, autopct='%1.1f%%', colors=['skyblue', 'lightgreen'])
plt.title('Distribution of is discount feature')
plt.show()
```

```
is_discount 0: 1813 counts
is_discount 1: 75 counts
Total Count: 1888 counts
```



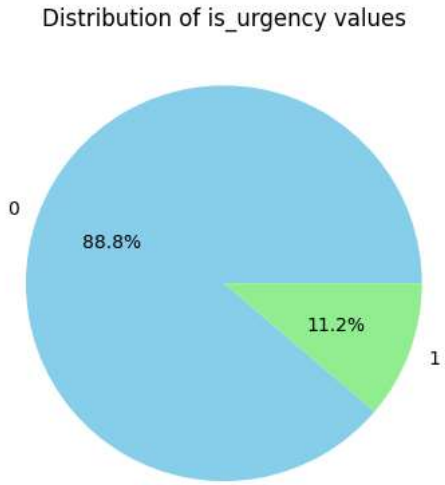

```
#is_price count and representation in bar plot

# Count values for is_price
value_counts_is_price = df['is_price'].value_counts()

# Display total count
total_count = len(df['is_price'])
print(f'Total Count: {total_count} counts\n')

plt.figure(figsize=(7, 5))
barplot = sns.barplot(x=value_counts_is_price.index, y=value_counts_is_price.values, palette='viridis')

# Displaying value counts on bars
for i, count in enumerate(value_counts_is_price.values):
    barplot.text(i, count + 0.1, str(count), ha='center', va='bottom')

plt.title('Counts of is_price')
plt.xlabel('is_price')
plt.ylabel('Counts')
plt.show()
```

```
Total Count: 1888 counts
```



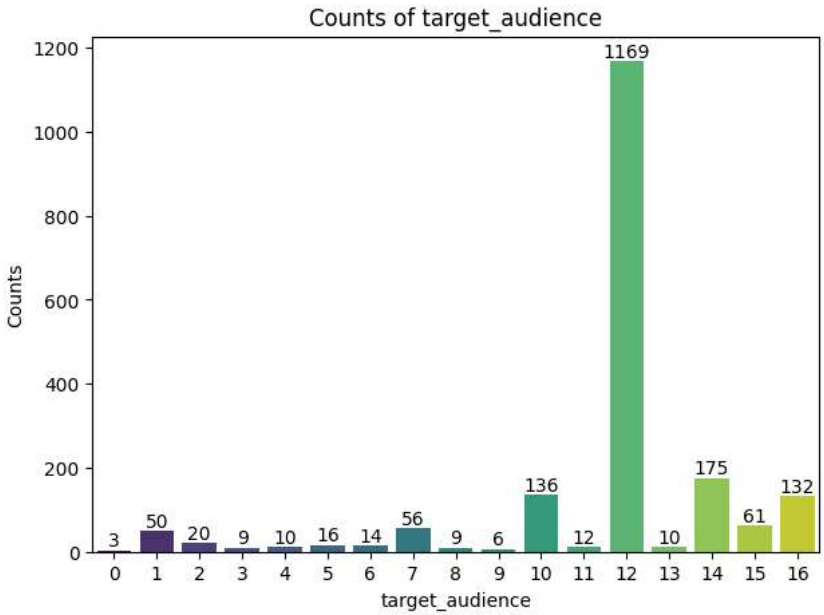

```
#Is_urgency count and representation as piechart

# Count values
value_counts = df['is_urgency'].value_counts()

# Display counts
for index, count in value_counts.items():
    print(f'is_urgency {index}: {count} counts')

# Display total count
total_count = len(df['is_urgency'])
print(f'Total Count: {total_count} counts\n')

# Pie chart
plt.pie(value_counts, labels=value_counts.index, autopct='%1.1f%%', colors=['skyblue', 'lightgreen'])
plt.title('Distribution of is_urgency values')
plt.show()
```

```
is_urgency 0: 1676 counts
is_urgency 1: 212 counts
Total Count: 1888 counts
```



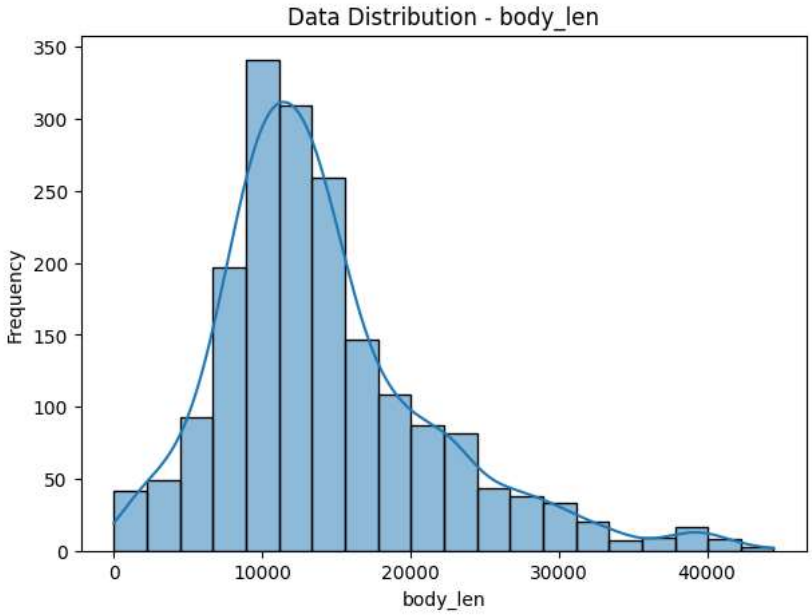
```
#target_audience count and representation as Histogram

# Count values for target_audience
value_counts_target_audience = df['target_audience'].value_counts().sort_index()

# Display total count
total_count = len(df['target_audience'])
print(f'Total Count: {total_count} counts\n')

plt.figure(figsize=(7, 5))
barplot = sns.barplot(x=value_counts_target_audience.index, y=value_counts_target_audience.values, palette='viridis')

# Displaying value counts on bars
for i, count in enumerate(value_counts_target_audience.values):
    barplot.text(i, count + 0.1, str(count), ha='center', va='bottom')

plt.title('Counts of target_audience')
plt.xlabel('target_audience')
plt.ylabel('Counts')
plt.show()
```

```
Total Count: 1888 counts
```

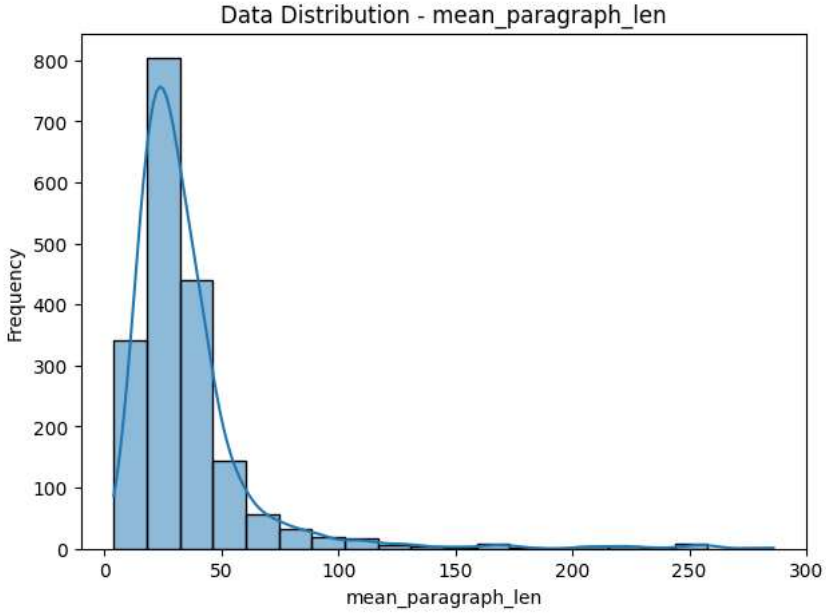



```
numerical_features = ['subject_len', 'body_len', 'mean_paragraph_len']
# Plotting a histogram to understand the distribution of subject lenth data
feature = 'subject_len'
plt.figure(figsize=(7, 5))
sns.histplot(df[feature], bins=20, kde=True)
plt.title(f'Data Distribution - {feature}')
plt.xlabel(feature)
plt.ylabel('Frequency')
plt.show()
```


```
# Plotting a histogram to understand the distribution of length of the email body
feature = 'body_len'
plt.figure(figsize=(7, 5))
sns.histplot(df[feature], bins=20, kde=True)
plt.title(f'Data Distribution - {feature}')
plt.xlabel(feature)
plt.ylabel('Frequency')
plt.show()
```

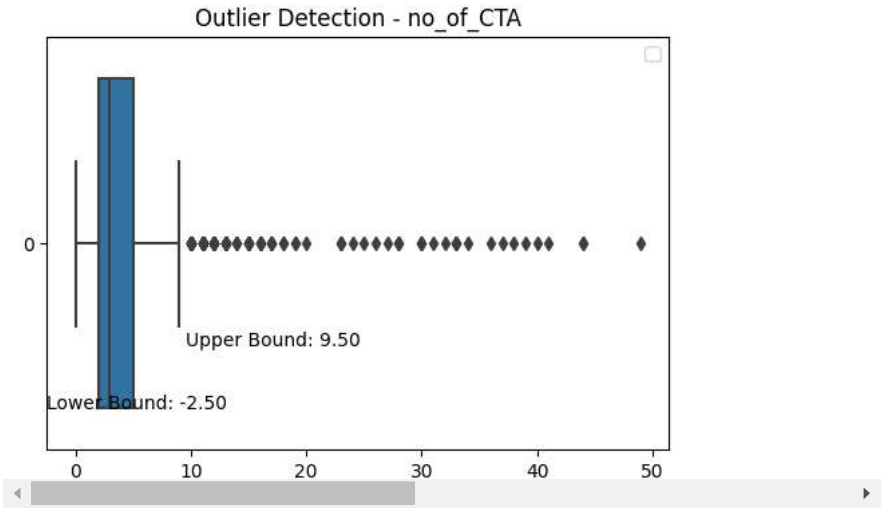



```
# Plotting a histogram to understand the distribution of mean paragraph length
feature = 'mean_paragraph_len'
plt.figure(figsize=(7, 5))
sns.histplot(df[feature], bins=20, kde=True)
plt.title(f'Data Distribution - {feature}')
plt.xlabel(feature)
plt.ylabel('Frequency')
plt.show()
```



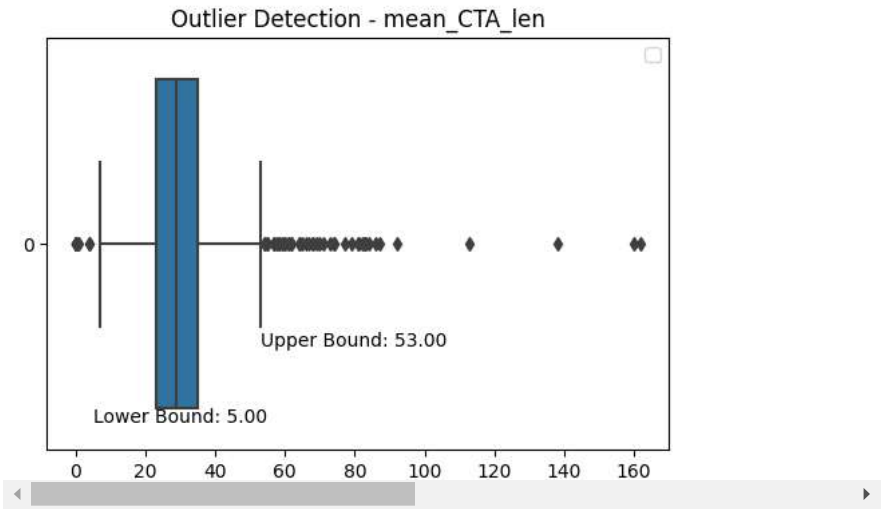
```
# Plotting a box plot to observe the outliers in no_of_CTA feature
# Calculating the upper and lower bounds
Q1 = df['no_of_CTA'].quantile(0.25)
Q3 = df['no_of_CTA'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

plt.figure(figsize=(6, 4))
sns.boxplot(data=df['no_of_CTA'], orient="h")
plt.title('Outlier Detection - no_of_CTA')

# Add lines for upper and lower bounds
plt.text(lower_bound, 0.4, f'Lower Bound: {lower_bound:.2f}', color='black', fontsize=10)
plt.text(upper_bound, 0.25, f'Upper Bound: {upper_bound:.2f}', color='black', fontsize=10)

plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that a
```



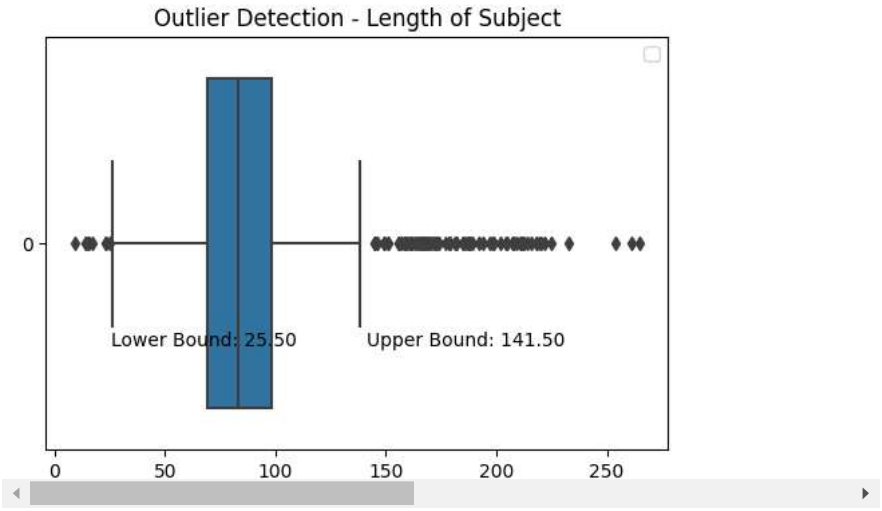

```
# Plotting a box plot to observe the outliers in mean_CTA_len feature
# Calculating the upper and lower bounds
Q1 = df['mean_CTA_len'].quantile(0.25)
Q3 = df['mean_CTA_len'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

plt.figure(figsize=(6, 4))
sns.boxplot(data=df['mean_CTA_len'], orient="h")
plt.title('Outlier Detection - mean_CTA_len')

# Add lines for upper and lower bounds
plt.text(lower_bound, 0.435, f'Lower Bound: {lower_bound:.2f}', color='black', fontsize=10)
plt.text(upper_bound, 0.25, f'Upper Bound: {upper_bound:.2f}', color='black', fontsize=10)

plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that a
```



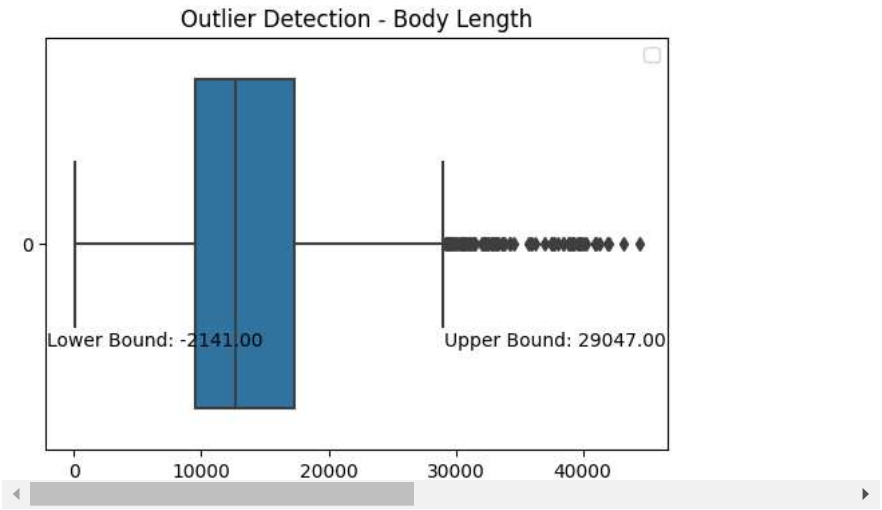
```
# Plotting a box plot to observe the outliers in subject_len feature
# Calculating the upper and lower bounds
Q1 = df['subject_len'].quantile(0.25)
Q3 = df['subject_len'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

plt.figure(figsize=(6, 4))
sns.boxplot(data=df['subject_len'], orient="h")
plt.title('Outlier Detection - Length of Subject')

# Add lines for upper and lower bounds
plt.text(lower_bound, 0.25, f'Lower Bound: {lower_bound:.2f}', color='black', fontsize=10)
plt.text(upper_bound, 0.25, f'Upper Bound: {upper_bound:.2f}', color='black', fontsize=10)

plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that a
```



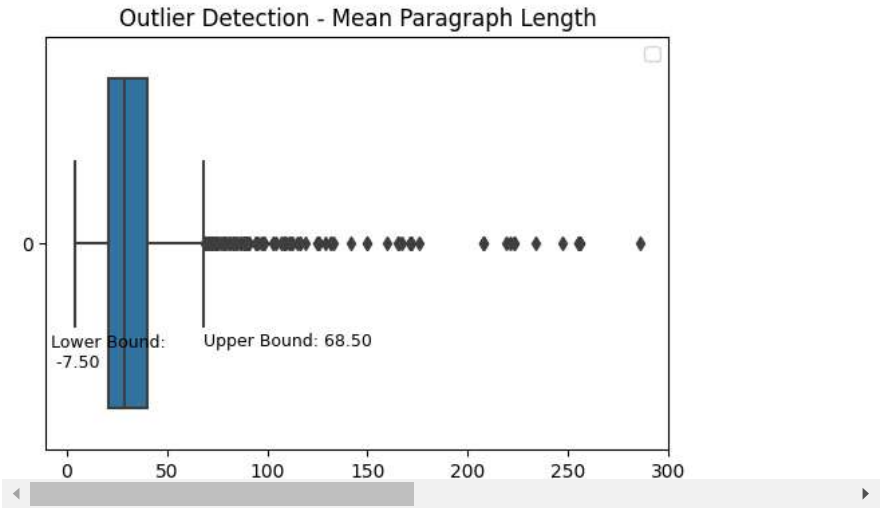

```
# Plotting a box plot to observe the outliers in body_len feature
# Calculating the upper and lower bounds
Q1 = df['body_len'].quantile(0.25)
Q3 = df['body_len'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

plt.figure(figsize=(6, 4))
sns.boxplot(data=df['body_len'], orient="h")
plt.title('Outlier Detection - Body Length')

# Add lines for upper and lower bounds
plt.text(lower_bound, 0.25, f'Lower Bound: {lower_bound:.2f}', color='black', fontsize=10)
plt.text(upper_bound, 0.25, f'Upper Bound: {upper_bound:.2f}', color='black', fontsize=10)

plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that a
```



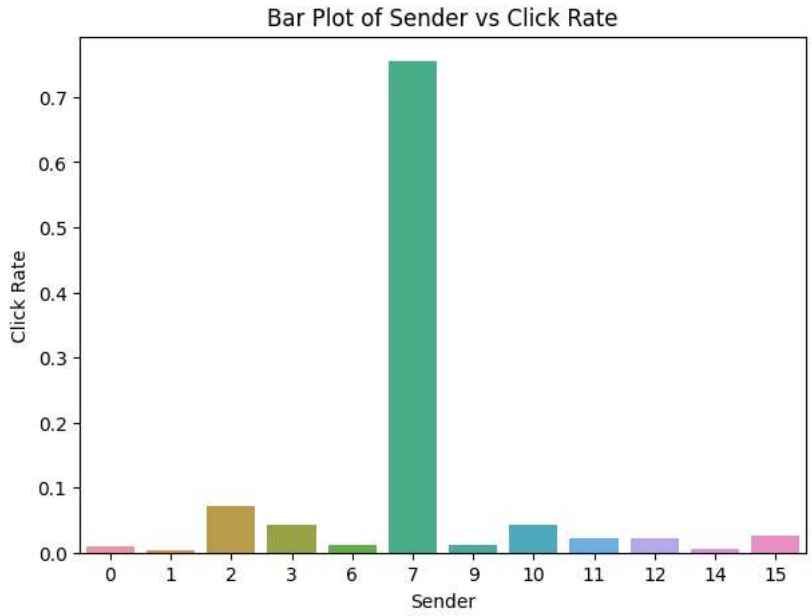
```
# Plotting a box plot to observe the outliers in mean_paragraph_len feature

# Calculating the upper and lower bounds
Q1 = df['mean_paragraph_len'].quantile(0.25)
Q3 = df['mean_paragraph_len'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

plt.figure(figsize=(6, 4))
sns.boxplot(data=df['mean_paragraph_len'], orient="h")
plt.title('Outlier Detection - Mean Paragraph Length')

# Add lines for upper and lower bounds
plt.text(lower_bound, 0.3, f'Lower Bound:\n {lower_bound:.2f}', color='black', fontsize=9)
plt.text(upper_bound, 0.25, f'Upper Bound: {upper_bound:.2f}', color='black', fontsize=9)

plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that a
```



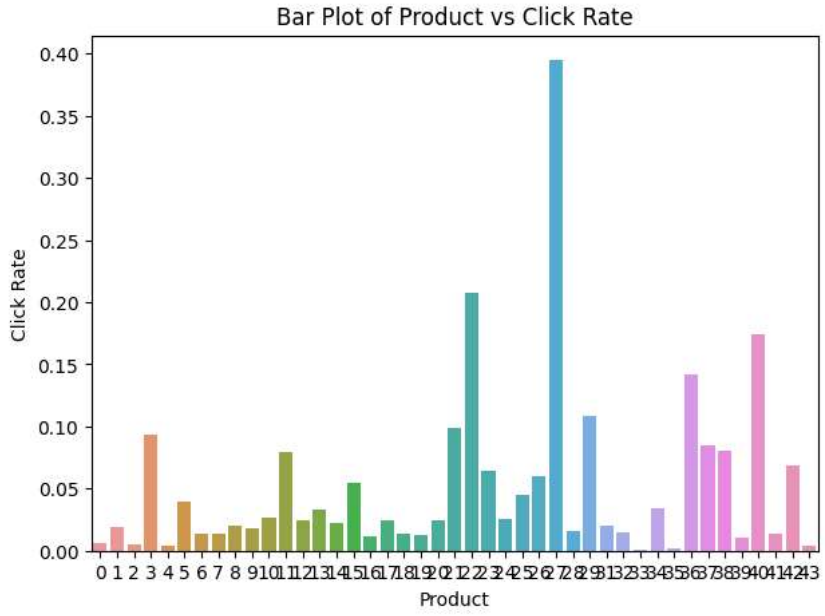

```
# Create a bar plot
plt.figure(figsize=(7, 5))
# Bar plot
sns.barplot(x='sender', y='click_rate', data=df, ci=None)
plt.title('Bar Plot of Sender vs Click Rate')
plt.xlabel('Sender')
plt.ylabel('Click Rate')
plt.show()
```

```
<ipython-input-34-bf60f3b49a39>:4: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='sender', y='click_rate', data=df, ci=None)
```



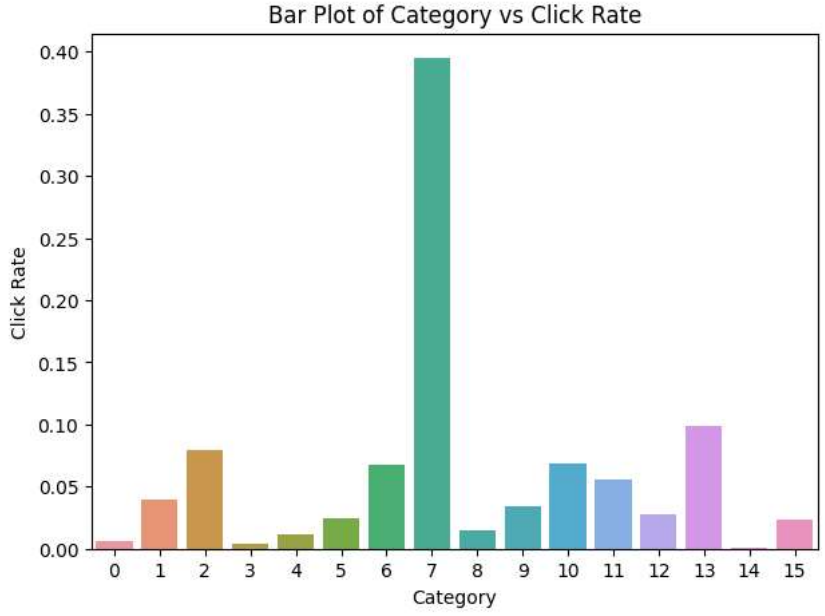
```
# Create a bar plot
plt.figure(figsize=(7, 5))
# Bar plot
sns.barplot(x='product', y='click_rate', data=df, ci=None)
plt.title('Bar Plot of Product vs Click Rate')
plt.xlabel('Product')
plt.ylabel('Click Rate')
plt.show()
```

```
<ipython-input-35-ceb776a56a4a>:4: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='product', y='click_rate', data=df, ci=None)
```

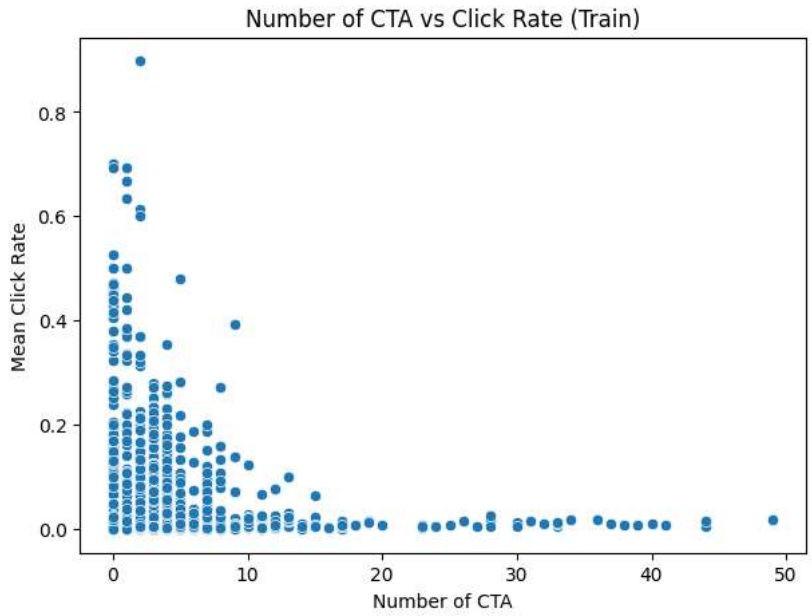



```
# Create a bar plot
plt.figure(figsize=(7, 5))
# Bar plot
sns.barplot(x='category', y='click_rate', data=df, ci=None)
plt.title('Bar Plot of Category vs Click Rate')
plt.xlabel('Category')
plt.ylabel('Click Rate')
plt.show()
```

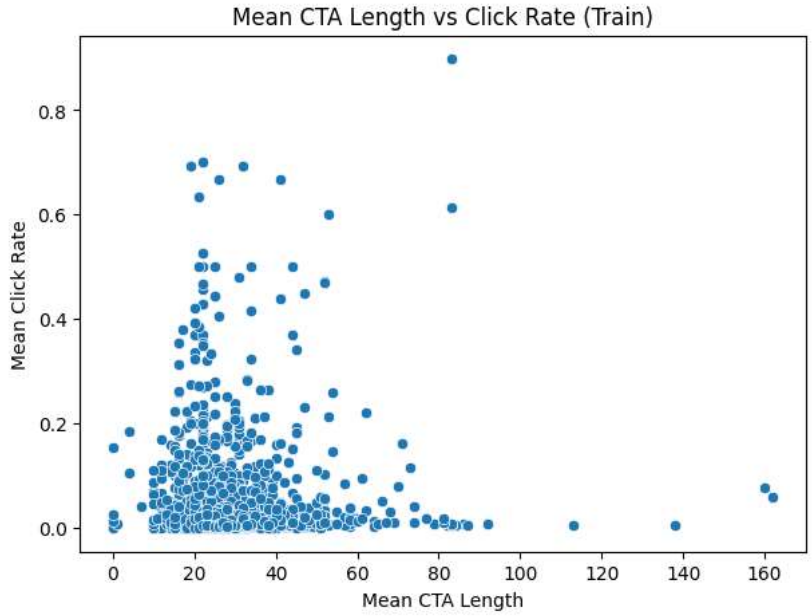
```
<ipython-input-36-bdbfd0cf8ef0>:4: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='category', y='click_rate', data=df, ci=None)
```



```
# Scatter plot for 'no_of_CTA' vs 'click_rate'
plt.figure(figsize=(7, 5))
sns.scatterplot(x='no_of_CTA', y='click_rate', data=df)
plt.title('Number of CTA vs Click Rate (Train)')
plt.xlabel('Number of CTA')
plt.ylabel('Mean Click Rate')
plt.show()
```

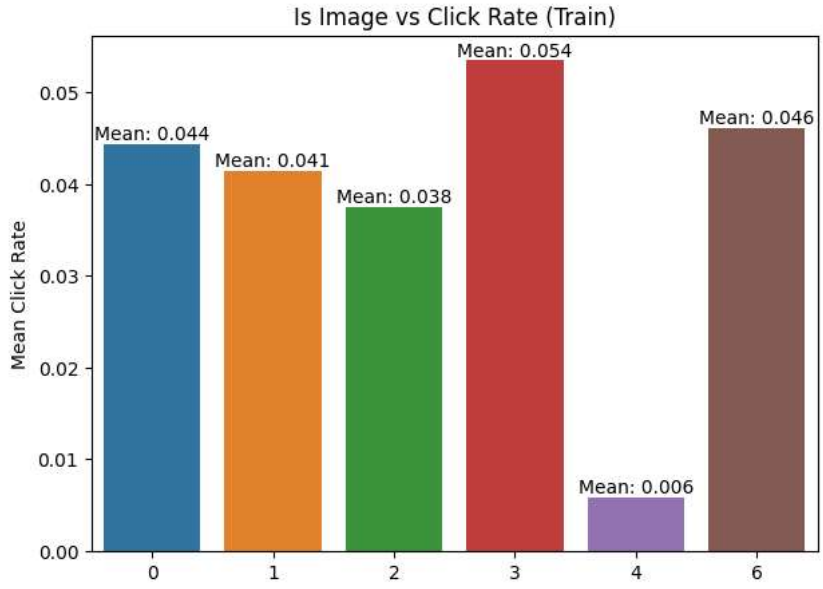



```
# Scatter plot for 'mean_CTA_len' vs 'click_rate'
plt.figure(figsize=(7, 5))
sns.scatterplot(x='mean_CTA_len', y='click_rate', data=df)
plt.title('Mean CTA Length vs Click Rate (Train)')
plt.xlabel('Mean CTA Length')
plt.ylabel('Mean Click Rate')
plt.show()
```



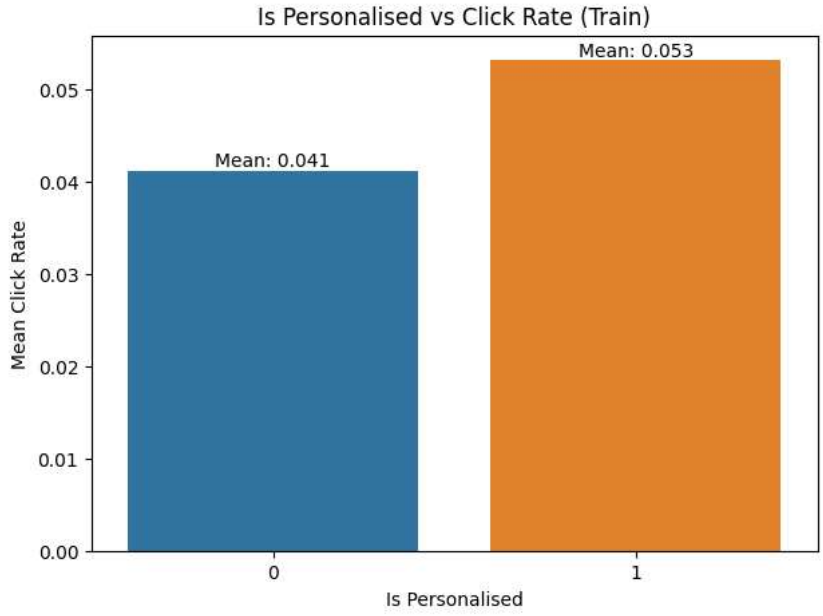

```
# Bar plot for 'is_image' vs 'click_rate'
plt.figure(figsize=(7, 5))
sns.barplot(x='is_image', y='click_rate', data=df, ci=None)
mean_click_rate_image = df.groupby('is_image')['click_rate'].mean()
for i, mean_rate in enumerate(mean_click_rate_image):
    plt.text(i, mean_rate, f'Mean: {mean_rate:.3f}', ha='center', va='bottom')
plt.title('Is Image vs Click Rate (Train)')
plt.xlabel('Is Image')
plt.ylabel('Mean Click Rate')
plt.show()
```

```
<ipython-input-39-869abce968e4>:3: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='is_image', y='click_rate', data=df, ci=None)
```



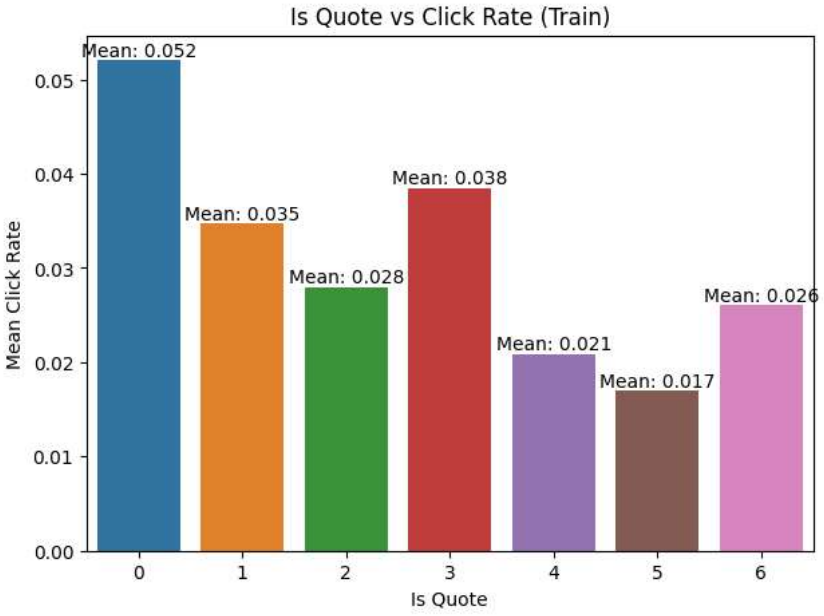

```
# Bar plot for 'is_personalised' vs 'click_rate'
plt.figure(figsize=(7, 5))
sns.barplot(x='is_personalised', y='click_rate', data=df, ci=None)
mean_click_rate_personalised = df.groupby('is_personalised')['click_rate'].mean()
for i, mean_rate in enumerate(mean_click_rate_personalised):
    plt.text(i, mean_rate, f'Mean: {mean_rate:.3f}', ha='center', va='bottom')
plt.title('Is Personalised vs Click Rate (Train)')
plt.xlabel('Is Personalised')
plt.ylabel('Mean Click Rate')
plt.show()
```

```
<ipython-input-40-d950cc7af975>:3: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='is_personalised', y='click_rate', data=df, ci=None)
```



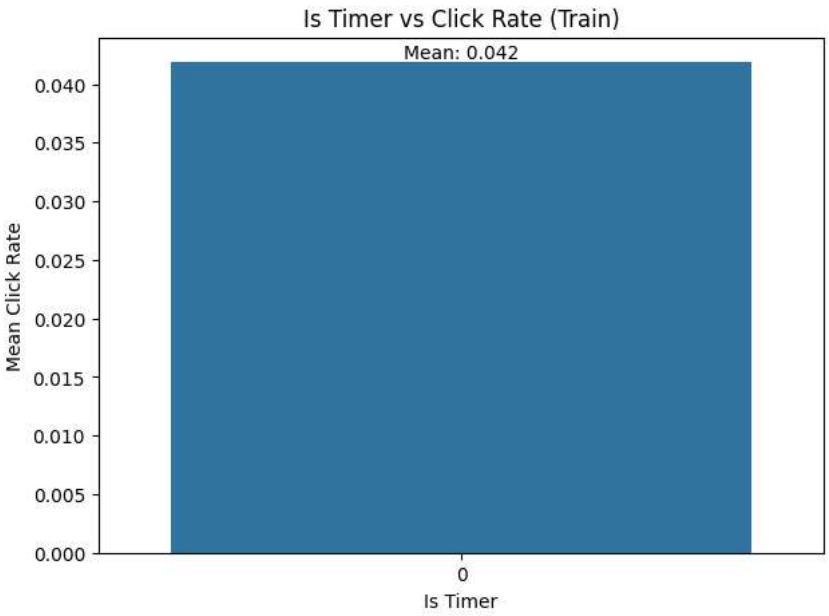

```
# Bar plot for 'is_quote' vs 'click_rate'
plt.figure(figsize=(7, 5))
sns.barplot(x='is_quote', y='click_rate', data=df, ci=None)
mean_click_rate_quote = df.groupby('is_quote')['click_rate'].mean()
for i, mean_rate in enumerate(mean_click_rate_quote):
    plt.text(i, mean_rate, f'Mean: {mean_rate:.3f}', ha='center', va='bottom')
plt.title('Is Quote vs Click Rate (Train)')
plt.xlabel('Is Quote')
plt.ylabel('Mean Click Rate')
plt.show()
```

```
<ipython-input-41-898557361ed0>:3: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='is_quote', y='click_rate', data=df, ci=None)
```



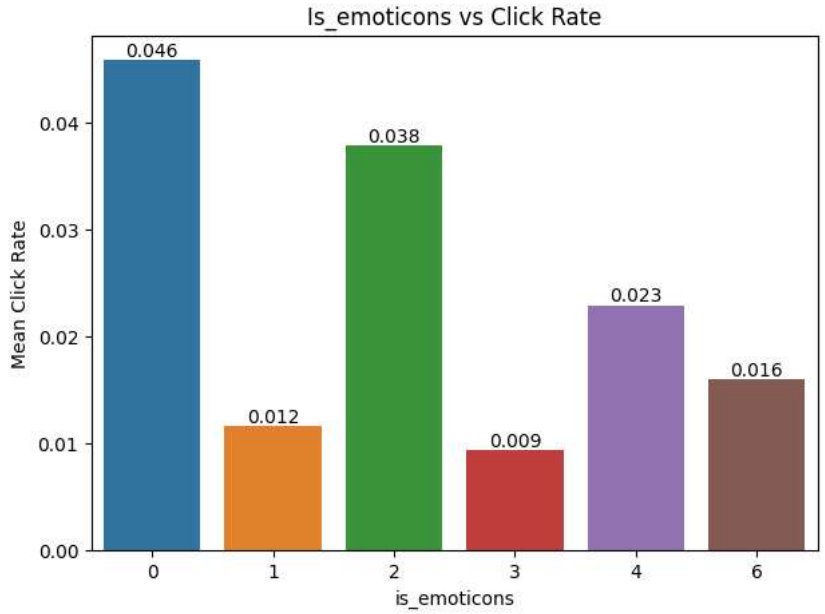

```
# Bar plot for 'is_timer' vs 'click_rate'
plt.figure(figsize=(7, 5))
sns.barplot(x='is_timer', y='click_rate', data=df, ci=None)
mean_click_rate_timer = df.groupby('is_timer')['click_rate'].mean()
for i, mean_rate in enumerate(mean_click_rate_timer):
    plt.text(i, mean_rate, f'Mean: {mean_rate:.3f}', ha='center', va='bottom')
plt.title('Is Timer vs Click Rate (Train)')
plt.xlabel('Is Timer')
plt.ylabel('Mean Click Rate')
plt.show()
```

```
<ipython-input-42-a94cc6a628da>:3: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='is_timer', y='click_rate', data=df, ci=None)
```



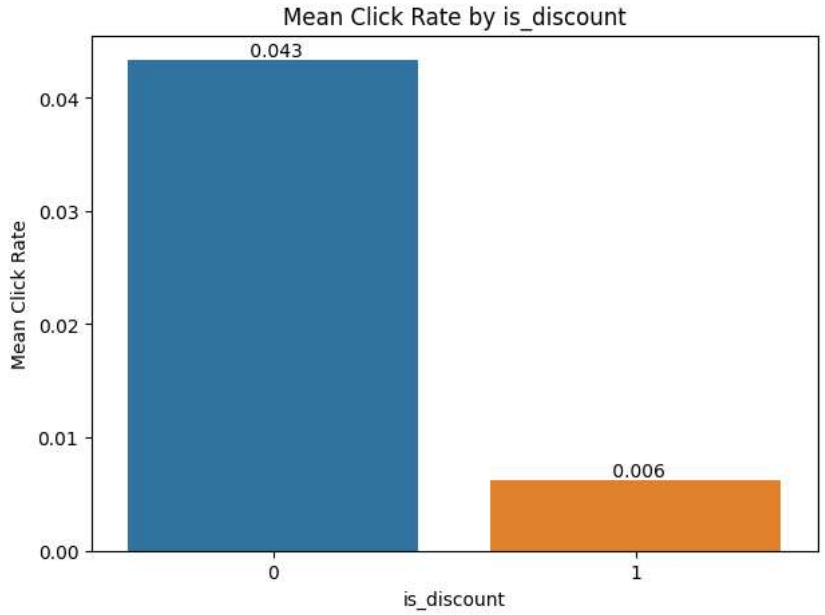
```
#Is_emoticons with respect to click_rate
# Observation
emoticons_distribution = df['is_emoticons'].value_counts(normalize=True)
print("Distribution of is_emoticons:\n", emoticons_distribution)

# Relation to click_rate
mean_click_rate_emoticons = df.groupby('is_emoticons')['click_rate'].mean()
print("Mean Click Rate by is_emoticons:\n", mean_click_rate_emoticons)

# Visualization
plt.figure(figsize=(7, 5))
sns.barplot(x='is_emoticons', y='click_rate', data=df, errorbar=None)
mean_click_rate_emoticons = df.groupby('is_emoticons')['click_rate'].mean()
for i, mean_rate in enumerate(mean_click_rate_emoticons):
    plt.text(i, mean_rate, f'{mean_rate:.3f}', ha='center', va='bottom')
plt.title('Is_emoticons vs Click Rate')
plt.xlabel('is_emoticons')
plt.ylabel('Mean Click Rate')
plt.show()
```

```
Distribution of is_emoticons:
 0    0.861229
1    0.091102
2    0.028602
3    0.015890
4    0.002119
6    0.001059
Name: is_emoticons, dtype: float64
Mean Click Rate by is_emoticons:
 is_emoticons
0    0.045905
1    0.011607
2    0.037820
3    0.009313
4    0.022907
6    0.015958
Name: click_rate, dtype: float64
```



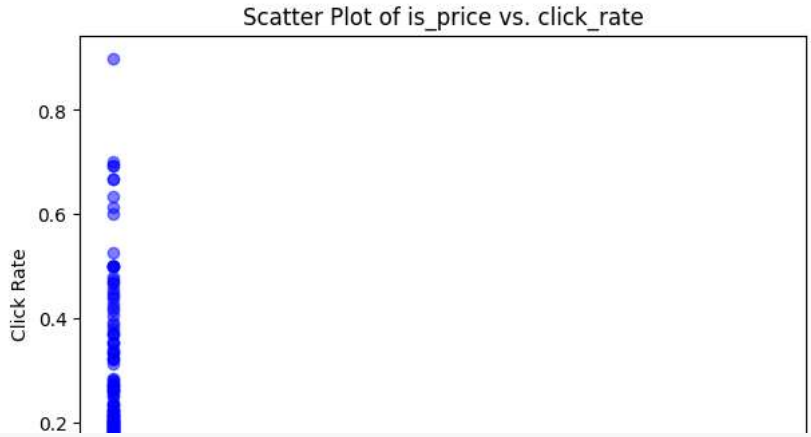
```
#Is_discount with respect to click_rate
# Observation
discount_distribution = df['is_discount'].value_counts(normalize=True)
print("Distribution of is_discount:\n", discount_distribution)

# Relation to click_rate
mean_click_rate_discount = df.groupby('is_discount')['click_rate'].mean()
print("Mean Click Rate by is_discount:\n", mean_click_rate_discount)

# Visualization
plt.figure(figsize=(7, 5))
sns.barplot(x='is_discount', y='click_rate', data=df, errorbar=None)
mean_click_rate_discount = df.groupby('is_discount')['click_rate'].mean()
for i, mean_rate in enumerate(mean_click_rate_discount):
    plt.text(i, mean_rate, f' {mean_rate:.3f}', ha='center', va='bottom')
plt.title('is_discount vs Click Rate')
plt.title('Mean Click Rate by is_discount')
plt.xlabel('is_discount')
plt.ylabel('Mean Click Rate')

plt.show()
```

```
Distribution of is_discount:
 0    0.960275
1    0.039725
Name: is_discount, dtype: float64
Mean Click Rate by is_discount:
 is_discount
0    0.043362
1    0.006242
Name: click_rate, dtype: float64
```




```
#Is_price with respect to click_rate
# Observation
price_distribution = df['is_price'].value_counts(normalize=True)
print("Distribution of is_price:\n", price_distribution)

# Relation to click_rate
mean_click_rate_price = df.groupby('is_price')['click_rate'].mean()
print("Mean Click Rate by is_price:\n", mean_click_rate_price)


# Scatter Plot
plt.figure(figsize=(7, 5))
plt.scatter(df['is_price'], df['click_rate'], color='blue', alpha=0.5)
plt.title('Scatter Plot of is_price vs. click_rate')
plt.xlabel('is_price')
plt.ylabel('Click Rate')
plt.show()
```

```
Distribution of is_price:
 0        0.993114
5499     0.002648
8000     0.001059
5000     0.001059
4999     0.000530
1399     0.000530
1000     0.000530
14999    0.000530
Name: is_price, dtype: float64
Mean Click Rate by is_price:
 is_price
0        0.042109
1000     0.054237
1399     0.002767
4999     0.012607
5000     0.009735
5499     0.005973
8000     0.002936
14999    0.004036
Name: click_rate, dtype: float64
```



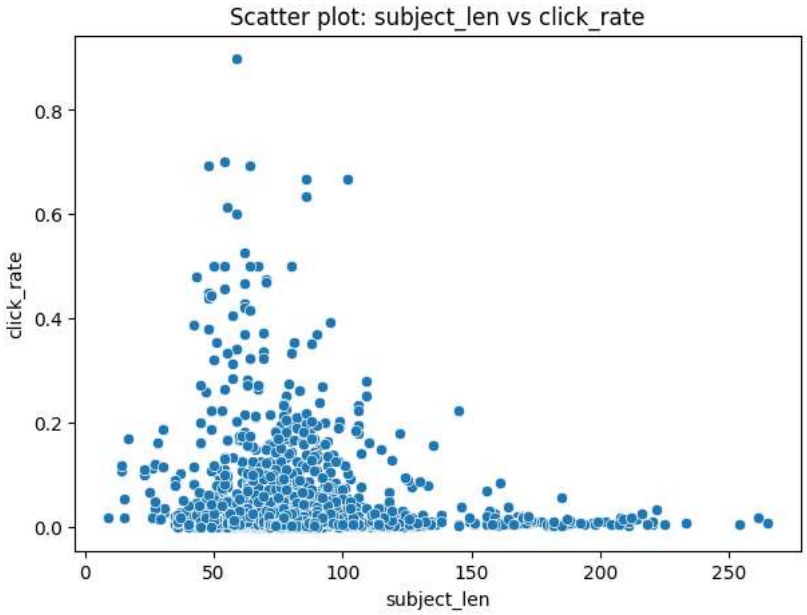

```
#Is_urgency with respect to click_rate
# Observation
urgency_distribution = df['is_urgency'].value_counts(normalize=True)
print("Distribution of is_urgency:\n", urgency_distribution)

# Relation to click_rate
mean_click_rate_urgency = df.groupby('is_urgency')['click_rate'].mean()
print("Mean Click Rate by is_urgency:\n", mean_click_rate_urgency)

# Visualization

plt.figure(figsize=(7, 5))
sns.barplot(x='is_urgency', y='click_rate', data=df, errorbar=None)
mean_click_rate_urgency = df.groupby('is_urgency')['click_rate'].mean()
for i, mean_rate in enumerate(mean_click_rate_urgency):
    plt.text(i, mean_rate, f' {mean_rate:.3f}', ha='center', va='bottom')
plt.title('is_urgency vs Click Rate')
plt.title('Mean Click Rate by is_urgency')
plt.xlabel('is_urgency')
plt.ylabel('Mean Click Rate')

plt.show()
```

```
Distribution of is_urgency:
 0    0.887712
1    0.112288
Name: is_urgency, dtype: float64
Mean Click Rate by is_urgency:
 is_urgency
0    0.045310
1    0.014831
Name: click_rate, dtype: float64
```

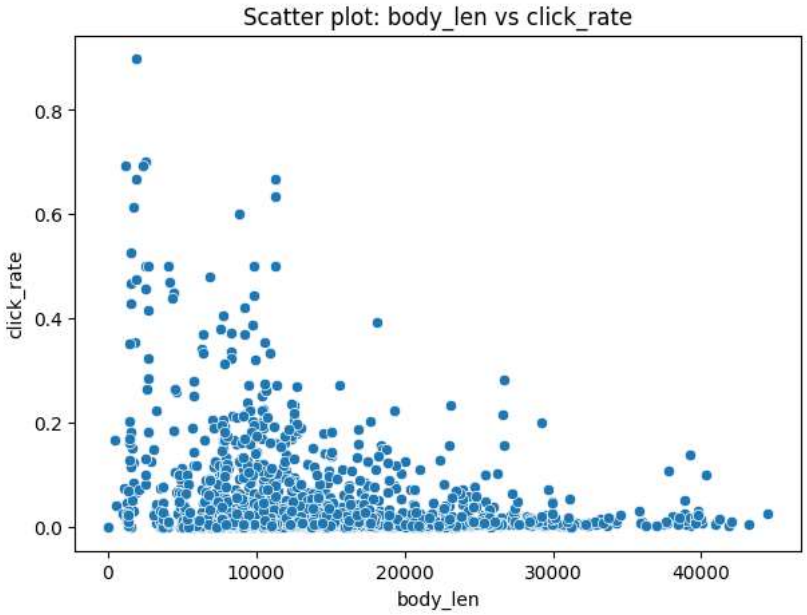

```
#target_audience with respect to click_rate
# Observation
audience_distribution = df['target_audience'].value_counts(normalize=True)
print("Distribution of target_audience:\n", audience_distribution)

# Relation to click_rate
mean_click_rate_audience = df.groupby('target_audience')['click_rate'].mean()
print("Mean Click Rate by target_audience:\n", mean_click_rate_audience)

# Visualization
plt.figure(figsize=(7, 5))
sns.barplot(x='target_audience', y='click_rate', data=df, errorbar=None)
mean_click_rate_audience = df.groupby('target_audience')['click_rate'].mean()
for i, mean_rate in enumerate(mean_click_rate_audience):
    plt.text(i, mean_rate, f'{mean_rate:.2f}', ha='center', va='bottom')
plt.title('target_audience vs Click Rate')
plt.title('Mean Click Rate by target_audience')
plt.xlabel('target_audience')
plt.ylabel('Mean Click Rate')

plt.show()
```

```
Distribution of target_audience:
 12    0.619174
14    0.092691
10    0.072034
16    0.069915
15    0.032309
7     0.029661
1     0.026483
2     0.010593
5     0.008475
6     0.007415
11    0.006356
4     0.005297
13    0.005297
```

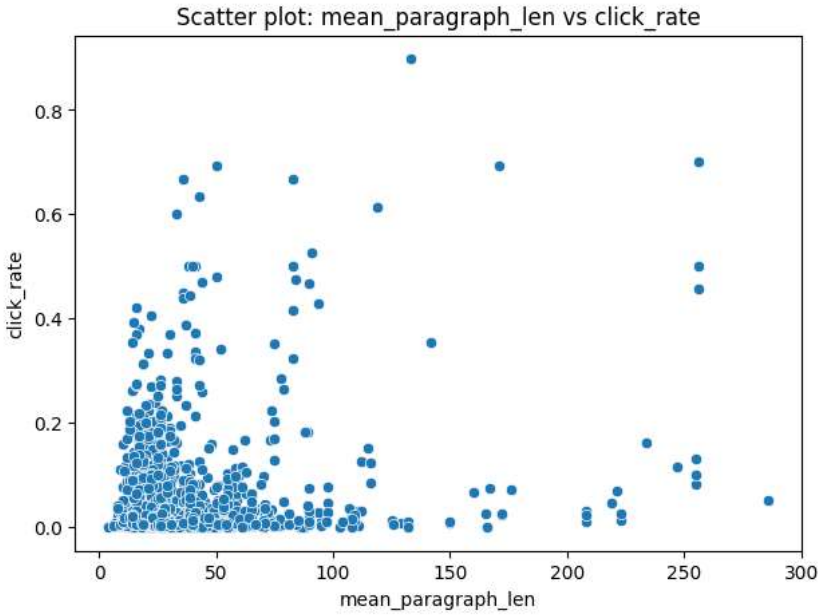
```
# A scatter plot to understand the relation between subject_len feature and target variable click_rate
plt.figure(figsize=(7, 5))
sns.scatterplot(x='subject_len', y='click_rate', data=df)
plt.title('Scatter plot: subject_len vs click_rate')
plt.xlabel('subject_len')
plt.ylabel('click_rate')
plt.show()
```




```
# A scatter plot to understand the relation between body_len feature and target variable click_rate

plt.figure(figsize=(7, 5))
sns.scatterplot(x='body_len', y='click_rate', data=df)
plt.title('Scatter plot: body_len vs click_rate')
plt.xlabel('body_len')
plt.ylabel('click_rate')
plt.show()
```



```
# A scatter plot to understand the relation between mean_paragraph_len feature and target variable click_rate

plt.figure(figsize=(7, 5))
sns.scatterplot(x='mean_paragraph_len', y='click_rate', data=df)
plt.title('Scatter plot: mean_paragraph_len vs click_rate')
plt.xlabel('mean_paragraph_len')
plt.ylabel('click_rate')
plt.show()
```




```
from scipy.stats import ttest_ind

target_variable = 'click_rate'
feature = 'sender'

# Finding the correlation coeff and p-value for the correlation between sender feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for sender and click_rate: 57.6930966328559
Correlation between sender and click_rate: -0.03139754532663776
P-value: 0.0
```

```
target_variable = 'click_rate'
feature = 'product'

# Finding the correlation coeff and p-value for the correlation between product feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for product and click_rate: 61.41396447050734
Correlation between product and click_rate: 0.12160227524547376
P-value: 0.0
```

```
target_variable = 'click_rate'
feature = 'category'

# Finding the correlation coeff and p-value for the correlation between category feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for category and click_rate: 81.20604616902446
Correlation between category and click_rate: -0.1677562090120046
P-value: 0.0
```

```
target_variable = 'click_rate'
feature = 'no_of_CTA'

# Finding the correlation coeff and p-value for the correlation between no_of_CTA feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for no_of_CTA and click_rate: 39.24086830319124
Correlation between no_of_CTA and click_rate: -0.17263684027135914
P-value: 9.134744523310117e-283
```

```
target_variable = 'click_rate'
feature = 'mean_CTA_len'

# Finding the correlation coeff and p-value for the correlation between mean_CTA_len feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for mean_CTA_len and click_rate: 110.71348748658511
Correlation between mean_CTA_len and click_rate: -0.031161971861755042
P-value: 0.0
```

```
target_variable = 'click_rate'
feature = 'is_image'

# Finding the correlation coeff and p-value for the correlation between is_image feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for is_image and click_rate: 43.327320574569356
Correlation between is_image and click_rate: -0.021527005413653107
P-value: 0.0
```

```
target_variable = 'click_rate'
feature = 'is_personalised'

# Finding the correlation coeff and p-value for the correlation between is_personalised feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for is_personalised and click_rate: 2.610230594587902
Correlation between is_personalised and click_rate: 0.032921776197264614
P-value: 0.009083886126673523
```

```
target_variable = 'click_rate'
feature = 'is_quote'

# Finding the correlation coeff and p-value for the correlation between is_quote feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for is_quote and click_rate: 33.18877917296929
Correlation between is_quote and click_rate: -0.09840186004996657
P-value: 3.715233804811014e-212
```

```
target_variable = 'click_rate'
feature = 'is_timer'

# Finding the correlation coeff and p-value for the correlation between is_timer feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for is_timer and click_rate: -21.61008854449971
Correlation between is_timer and click_rate: nan
P-value: 9.660453833609957e-98

/usr/local/lib/python3.10/dist-packages/scipy/stats/_stats_py.py:4781: ConstantInputWarning: An input array is constant; the correla
  warnings.warn(stats.ConstantInputWarning(msg))
```

```
target_variable = 'click_rate'
feature = 'is_emoticons'

# Finding the correlation coeff and p-value for the correlation between is_emoticons feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for is_emoticons and click_rate: 11.85350767297251
Correlation between is_emoticons and click_rate: -0.09430522829404966
P-value: 7.529884282975184e-32
```

```
target_variable = 'click_rate'
feature = 'is_discount'

# Finding the correlation coeff and p-value for the correlation between is_discount feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for is_discount and click_rate: -0.4417707076025 73
Correlation between is_discount and click_rate: -0.08610469841088633
P-value: 0.6586804482802313
```

```
target_variable = 'click_rate'
feature = 'is_price'

# Finding the correlation coeff and p-value for the correlation between is_price feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for is_price and click_rate: 3.1496752618933246
Correlation between is_price and click_rate: -0.030539707503137233
P-value: 0.0016472948869968689
```

```
target_variable = 'click_rate'
feature = 'is_urgency'

# Finding the correlation coeff and p-value for the correlation between is_urgency feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for is_urgency and click_rate: 9.35920746026809
Correlation between is_urgency and click_rate: -0.11428382324961081
P-value: 1.3404585624352308e-20
```

```
target_variable = 'click_rate'
feature = 'target_audience'

# Finding the correlation coeff and p-value for the correlation between target_audience feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for target_audience and click_rate: 170.73164168905603
Correlation between target_audience and click_rate: 0.03871266572890302
P-value: 0.0
```

```
target_variable = 'click_rate'
feature = 'subject_len'

# Finding the correlation coeff and p-value for the correlation between subject_len feature and target click_rate
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for subject_len and click_rate: 124.26227968845419
Correlation between subject_len and click_rate: -0.18014337944849376
P-value: 0.0
```

```
# Finding the correlation coeff and p-value for the correlation between body_len feature and target click_rate
feature = 'body_len'
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for body_len and click_rate: 84.11825939542686
Correlation between body_len and click_rate: -0.247865558903977
P-value: 0.0
```

```
# Finding the correlation coeff and p-value for the correlation between mean_paragraph_len feature and target click_rate

feature = 'mean_paragraph_len'
t_statistic, p_value = ttest_ind(df[feature], df[target_variable])
correlation, _ = pearsonr(df[feature], df[target_variable])

print(f"T Statistic for {feature} and {target_variable}: {t_statistic}")
print(f"Correlation between {feature} and {target_variable}: {correlation}")
print(f"P-value: {p_value}\n")
```

```
T Statistic for mean_paragraph_len and click_rate: 54.34939998091778
Correlation between mean_paragraph_len and click_rate: 0.17804218919263282
P-value: 0.0
```

```

```

```
from scipy import stats
# evaluate p-value and t statistic by t-test
# t statistic and p value
weekend_t, weekend_p = stats.ttest_ind(weekend, click_rate)
daysOfWeek_t, daysOfWeek_p = stats.ttest_ind(daysOfWeek, click_rate)
times_t, times_p = stats.ttest_ind(times_encoded, click_rate)

print(f"is_weekend:  t-statistic - {weekend_t}, p-value - {weekend_p}")
print(f"days of week:  t-statistic - {daysOfWeek_t}, p-value - {daysOfWeek_p}")
print(f"times of day:  t-statistic - {times_t}, p-value - {times_p}")
```

```
is_weekend:  t-statistic - 16.778433177564494, p-value - 5.413031606085736e-61
days of week:  t-statistic - 68.5907688094779, p-value - 0.0
times of day:  t-statistic - 184.70050705243636, p-value - 0.0
```

```
#Finding the relation between feature and target click_rate using OLS linear regression model
columns_to_analyze = ['sender', 'category', 'product', 'day_of_week', 'is_weekend', 'times_of_day',
                      'no_of_CTA', 'mean_CTA_len', 'is_image', 'is_personalised', 'is_quote',
                      'is_timer', 'is_emoticons', 'is_discount', 'is_price', 'is_urgency',
                      'target_audience', 'subject_len', 'body_len', 'mean_paragraph_len']


# Perform linear regression for each column
for column in columns_to_analyze:
    if df[column].dtype != 'object':

        X = sm.add_constant(df[column])
        y = df['click_rate']  # Replace 'target_column'
        model = sm.OLS(y, X).fit()
        print(f"Linear Regression for {column}:\n{model.summary()}\n")
```

```
Linear Regression for sender:
                            OLS Regression Results
==============================================================================
Dep. Variable:             click_rate   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                  0.000
Method:                 Least Squares   F-statistic:                     1.861
Date:                Mon, 04 Dec 2023   Prob (F-statistic):              0.173
Time:                        11:26:07   Log-Likelihood:                 1993.9
No. Observations:                1888   AIC:                            -3984.
Df Residuals:                    1886   BIC:                            -3973.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0454      0.003     14.013      0.000       0.039       0.052
sender        -0.0008      0.001     -1.364      0.173      -0.002       0.000
==============================================================================
Omnibus:                     1704.635   Durbin-Watson:                   1.709
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            51363.198
Skew:                           4.314   Prob(JB):                         0.00
Kurtosis:                      27.052   Cond. No.                         9.37
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Linear Regression for category:
                            OLS Regression Results
==============================================================================
Dep. Variable:             click_rate   R-squared:                       0.028
Model:                            OLS   Adj. R-squared:                  0.028
Method:                 Least Squares   F-statistic:                     54.61
Date:                Mon, 04 Dec 2023   Prob (F-statistic):           2.19e-13
Time:                        11:26:07   Log-Likelihood:                 2020.0
No. Observations:                1888   AIC:                            -4036.
Df Residuals:                    1886   BIC:                            -4025.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0684      0.004     16.825      0.000       0.060       0.076
category      -0.0027      0.000     -7.390      0.000      -0.003      -0.002
==============================================================================
Omnibus:                     1676.897   Durbin-Watson:                   1.712
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            49184.752
Skew:                           4.212   Prob(JB):                         0.00
Kurtosis:                      26.543   Cond. No.                         24.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Linear Regression for product:
                            OLS Regression Results
```

```
==========================================================================
```

```
num_bootstrap_samples = 500

# Creating array to save bootstrap sample means
bootstrap_means = np.zeros(num_bootstrap_samples)

#Performing Bootstrap Sampling to find the range of mean for click_rate
for i in range(num_bootstrap_samples):
    bootstrap_sample = np.random.choice(df['click_rate'], size=len(df), replace=True)

    # Calculate the mean of the bootstrap sample
    bootstrap_means[i] = np.mean(bootstrap_sample)

# Calculating mean and confidence intervals of bootstrap sample means
mean_bootstrap = np.mean(bootstrap_means)
conf_interval_bootstrap = np.percentile(bootstrap_means, [2.5, 97.5])

print(f"Bootstrap Sample Mean: {mean_bootstrap:.4f}")
print(f"95% Confidence Interval: {conf_interval_bootstrap}")
```

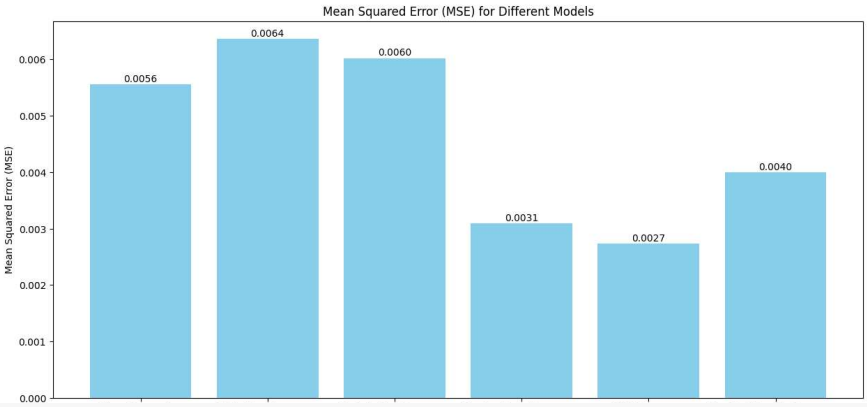
```
Bootstrap Sample Mean: 0.0420
95% Confidence Interval: [0.03840429 0.0456543 ]
```

```
numerical_features = ['no_of_CTA', 'mean_CTA_len', 'subject_len', 'body_len', 'mean_paragraph_len']

# Defining a function to remove outliers based on IQR
def remove_outliers_iqr(data, feature, threshold=1.5):
    Q1 = data[feature].quantile(0.25)
    Q3 = data[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - threshold * IQR
    upper_bound = Q3 + threshold * IQR
    return data[(data[feature] >= lower_bound) & (data[feature] <= upper_bound)]

# Removing the outliers for each numerical feature
for feature in numerical_features:
    df = remove_outliers_iqr(df, feature)
```

```
# Selecting features decided after statistically proving dependancy
selected_features = ['sender', 'category', 'product', 'day_of_week', 'is_weekend', 'times_of_day',
                     'no_of_CTA', 'mean_CTA_len', 'is_image', 'is_personalised', 'is_quote',
                     'is_emoticons', 'is_price', 'is_urgency', 'target_audience', 'subject_len',
                     'body_len', 'mean_paragraph_len', 'click_rate']

df_selected = df[selected_features]

# Changing categorical variables to numberical variables
df_selected = pd.get_dummies(df_selected, columns=['sender', 'category', 'product', 'day_of_week', 'times_of_day', 'target_audience'])
```

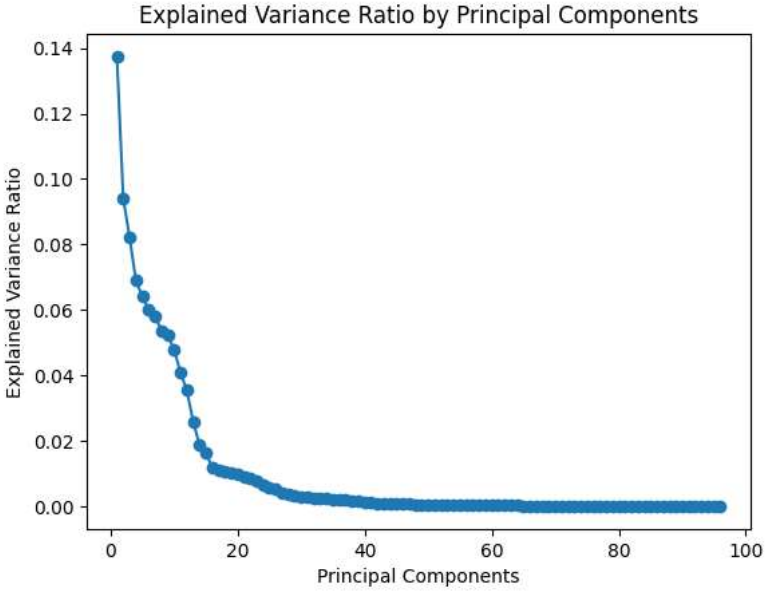
```
import time

X = df_selected.drop('click_rate', axis=1)
y = df_selected['click_rate']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# List of models to train
models = [
    LinearRegression(),
    KNeighborsRegressor(n_neighbors=5),
    DecisionTreeRegressor(),
    RandomForestRegressor(n_estimators=100, random_state=42),
    XGBRegressor(n_estimators=100),
    GradientBoostingRegressor(n_estimators=100, random_state=42),
]


model_names = []
mse_values = []
value_counts = []
training_times = []

# Training and evaluating each model:
for model in models:
    model_name = type(model).__name__

    # Training the model on train data
    start_time = time.time()

    model.fit(X_train, y_train)

    elapsed_time = time.time() - start_time
    training_times.append(elapsed_time)

    # Using the model, predict target values of test data
    y_pred = model.predict(X_test)

    # Model Evaluation
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    explained_var = explained_variance_score(y_test, y_pred)

    model_names.append(model_name)
    mse_values.append(mse)
    value_counts.append(len(y_pred))

    # Printing the metrics of each model
    print(f"Metrics for {model_name}:")
    print(f"Mean Squared Error: {mse}")
    print(f"R-squared: {r2}")
    print(f"Explained Variance Score: {explained_var}")
    print(f"Training Time: {elapsed_time:.4f} seconds")
    print("-" * 50)


plt.figure(figsize=(15, 7))
plt.bar(model_names, mse_values, color='skyblue')
plt.title('Mean Squared Error (MSE) for Different Models')
plt.xlabel('Model')
plt.ylabel('Mean Squared Error (MSE)')
for i, (count, mse) in enumerate(zip(value_counts, mse_values)):
    plt.text(i, mse + 0.00001, f'{mse:.4f}', ha='center', va='bottom', fontsize=10, color='black')

plt.show()
```

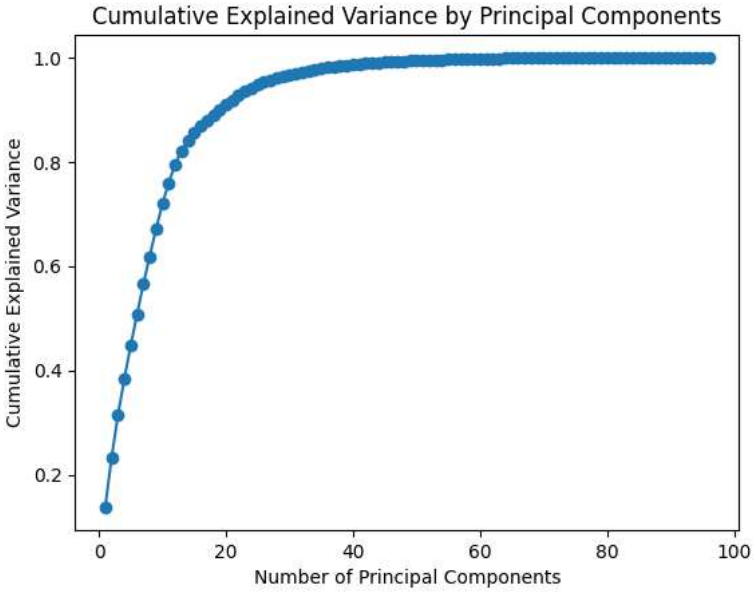
```
Metrics for LinearRegression:
Mean Squared Error: 0.005553286522615929
R-squared: 0.11331284536531394
Explained Variance Score: 0.1140909815440112
Training Time: 0.0268 seconds
--------------------------------------------------
Metrics for KNeighborsRegressor:
Mean Squared Error: 0.0063589119051526075
R-squared: -0.015320473883344565
Explained Variance Score: -0.01226507013804179
Training Time: 0.0081 seconds
--------------------------------------------------
Metrics for DecisionTreeRegressor:
Mean Squared Error: 0.00602065659392684
R-squared: 0.038688379834068476
Explained Variance Score: 0.03909252281191655
Training Time: 0.0578 seconds
--------------------------------------------------
Metrics for RandomForestRegressor:
Mean Squared Error: 0.0030907639368515302
R-squared: 0.5065011197147045
Explained Variance Score: 0.5068963177609815
Training Time: 2.6327 seconds
--------------------------------------------------
Metrics for XGBRegressor:
Mean Squared Error: 0.0027290424990348864
R-squared: 0.5642567840698227
Explained Variance Score: 0.5653417302413777
Training Time: 1.2496 seconds
--------------------------------------------------
Metrics for GradientBoostingRegressor:
Mean Squared Error: 0.003994873824072106
R-squared: 0.36214288786192605
Explained Variance Score: 0.36216623338434006
Training Time: 0.4349 seconds
--------------------------------------------------
```



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.decomposition import PCA

# Assuming df_selected contains your dataset with 'selected_features'
# If you've dropped the 'click_rate' column, make sure to include it in the dataset
X_pca = df[selected_features]

# Separate numerical and categorical features
categorical_features = ['sender', 'category', 'product', 'day_of_week', 'times_of_day', 'target_audience']
numerical_features = list(set(selected_features) - set(categorical_features))

# Standardize the numerical features
scaler = StandardScaler()
X_numerical_scaled = scaler.fit_transform(X_pca[numerical_features])

# One-hot encode the categorical features
encoder = OneHotEncoder(drop='first', sparse=False)
X_categorical_encoded = encoder.fit_transform(X_pca[categorical_features])

# Combine the scaled numerical and encoded categorical features
X_pca_scaled = np.concatenate((X_numerical_scaled, X_categorical_encoded), axis=1)

# Apply PCA
pca = PCA()
X_pca_transformed = pca.fit_transform(X_pca_scaled)

# Explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_explained_variance = np.cumsum(explained_variance_ratio)

# Plot explained variance ratio
plt.plot(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio, marker='o')
plt.title('Explained Variance Ratio by Principal Components')
plt.xlabel('Principal Components')
plt.ylabel('Explained Variance Ratio')
plt.show()

# Plot cumulative explained variance
plt.plot(range(1, len(cumulative_explained_variance) + 1), cumulative_explained_variance, marker='o')
plt.title('Cumulative Explained Variance by Principal Components')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()

# Print cumulative explained variance
print("Cumulative Explained Variance:")
for i, var in enumerate(cumulative_explained_variance):
    print(f"Principal Component {i + 1}: {var:.4f}")
```

```
usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureW
 warnings.warn(
```



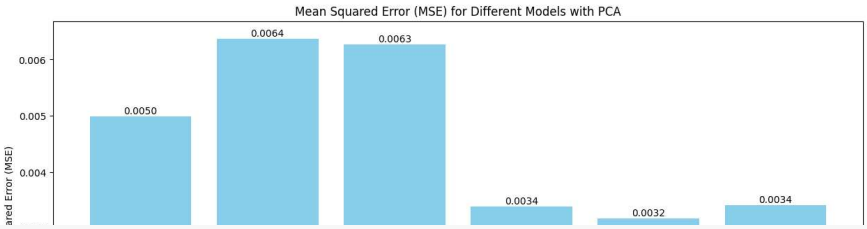



```
umulative Explained Variance:
rincipal Component 1: 0.1374
rincipal Component 2: 0.2316
rincipal Component 3: 0.3137
rincipal Component 4: 0.3826
rincipal Component 5: 0.4468
rincipal Component 6: 0.5067
rincipal Component 7: 0.5649
rincipal Component 8: 0.6185
rincipal Component 9: 0.6708
rincipal Component 10: 0.7187
rincipal Component 11: 0.7597
rincipal Component 12: 0.7951
rincipal Component 13: 0.8210
rincipal Component 14: 0.8398
rincipal Component 15: 0.8560
rincipal Component 16: 0.8679
rincipal Component 17: 0.8790
rincipal Component 18: 0.8897
rincipal Component 19: 0.8999
rincipal Component 20: 0.9097
rincipal Component 21: 0.9188
rincipal Component 22: 0.9274
rincipal Component 23: 0.9353
rincipal Component 24: 0.9418
rincipal Component 25: 0.9473
rincipal Component 26: 0.9527
rincipal Component 27: 0.9567
rincipal Component 28: 0.9604
rincipal Component 29: 0.9635
rincipal Component 30: 0.9664
rincipal Component 31: 0.9693
rincipal Component 32: 0.9719
rincipal Component 33: 0.9744
rincipal Component 34: 0.9766
rincipal Component 35: 0.9788
```

```
rincipal Component 36: 0.9807
rincipal Component 37: 0.9826
rincipal Component 38: 0.9842
rincipal Component 39: 0.9856
rincipal Component 40: 0.9870
rincipal Component 41: 0.9881
rincipal Component 42: 0.9890
rincipal Component 43: 0.9897
rincipal Component 44: 0.9905
rincipal Component 45: 0.9911
rincipal Component 46: 0.9918
rincipal Component 47: 0.9924
rincipal Component 48: 0.9930
rincipal Component 49: 0.9935
rincipal Component 50: 0.9940
rincipal Component 51: 0.9945
rincipal Component 52: 0.9949
rincipal Component 53: 0.9953
rincipal Component 54: 0.9957
rincipal Component 55: 0.9961
rincipal Component 56: 0.9964
rincipal Component 57: 0.9967
rincipal Component 58: 0.9970
rincipal Component 59: 0.9973
rincipal Component 60: 0.9976
rincipal Component 61: 0.9978
rincipal Component 62: 0.9981
rincipal Component 63: 0.9983
rincipal Component 64: 0.9986
rincipal Component 65: 0.9987
```

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

# Apply PCA with 40 components
pca = PCA(n_components=40)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

model_names = []
mse_values = []
value_counts = []
training_times = []

# Training and evaluating each model:
for model in models:
    model_name = type(model).__name__

    start_time = time.time()

    # Training the model on PCA-transformed train data
    model.fit(X_train_pca, y_train)

    elapsed_time = time.time() - start_time
    training_times.append(elapsed_time)


    # Using the model, predict target values of PCA-transformed test data
    y_pred = model.predict(X_test_pca)

    # Model Evaluation
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    explained_var = explained_variance_score(y_test, y_pred)

    model_names.append(model_name)
    mse_values.append(mse)
    value_counts.append(len(y_pred))

    # Printing the metrics of each model
    print(f"Metrics for {model_name} with PCA:")
    print(f"Mean Squared Error: {mse}")
    print(f"R-squared: {r2}")
    print(f"Explained Variance Score: {explained_var}")
    print(f"Training Time: {elapsed_time:.4f} seconds")
    print("-" * 50)

# Plotting Mean Squared Error for Different Models
plt.figure(figsize=(15, 7))
plt.bar(model_names, mse_values, color='skyblue')
plt.title('Mean Squared Error (MSE) for Different Models with PCA')
plt.xlabel('Model')
plt.ylabel('Mean Squared Error (MSE)')
for i, (count, mse) in enumerate(zip(value_counts, mse_values)):
    plt.text(i, mse + 0.00001, f'{mse:.4f}', ha='center', va='bottom', fontsize=10, color='black')

plt.show()
```

```
Metrics for LinearRegression with PCA:
Mean Squared Error: 0.004985078101287111
R-squared: 0.2040380593977038
Explained Variance Score: 0.20406959684278436
Training Time: 0.0034 seconds
--------------------------------------------------
Metrics for KNeighborsRegressor with PCA:
Mean Squared Error: 0.006359445008500308
R-squared: -0.015405593908859982
Explained Variance Score: -0.012339323766201504
Training Time: 0.0007 seconds
--------------------------------------------------
Metrics for DecisionTreeRegressor with PCA:
Mean Squared Error: 0.0062682451692138465
R-squared: -0.0008438158211050428
Explained Variance Score: 0.008183051412578801
Training Time: 0.1381 seconds
--------------------------------------------------
Metrics for RandomForestRegressor with PCA:
Mean Squared Error: 0.0033898706324683894
R-squared: 0.45874308241758255
Explained Variance Score: 0.4628071159581818
Training Time: 8.4036 seconds
--------------------------------------------------
Metrics for XGBRegressor with PCA:
Mean Squared Error: 0.0031772437771733133
R-squared: 0.49269297867319684
Explained Variance Score: 0.49279873025553556
Training Time: 1.1477 seconds
--------------------------------------------------
Metrics for GradientBoostingRegressor with PCA:
Mean Squared Error: 0.003410124404448405
R-squared: 0.4555091849094236
Explained Variance Score: 0.4555288676420963
Training Time: 1.4785 seconds
--------------------------------------------------
```



```
import time
from sklearn.model_selection import cross_val_score

model_names = []
mse_values = []
r2_values = []
explained_var_values = []
training_times = []

# Training and evaluating each model:
for model in models:
    model_name = type(model).__name__

    # Training the model using cross-validation
    start_time = time.time()
    mse_scores = -cross_val_score(model, X_train, y_train, cv=8, scoring='neg_mean_squared_error')
    elapsed_time = time.time() - start_time

    # Model Evaluation Metrics
    mse = np.mean(mse_scores)
    r2 = np.mean(cross_val_score(model, X_train, y_train, cv=8, scoring='r2'))
    explained_var = np.mean(cross_val_score(model, X_train, y_train, cv=8, scoring='explained_variance'))

    model_names.append(model_name)
    mse_values.append(mse)
    r2_values.append(r2)
    explained_var_values.append(explained_var)
    training_times.append(elapsed_time)

    # Printing the metrics of each model
    print(f"Metrics for {model_name} with Cross-Validation:")
    print(f"Mean Squared Error: {mse}")
    print(f"R-squared: {r2}")
    print(f"Explained Variance Score: {explained_var}")
    print(f"Training Time: {elapsed_time:.4f} seconds")
    print("-" * 50)

# Plotting Mean Squared Error for Different Models
plt.figure(figsize=(15, 7))
plt.bar(model_names, mse_values, color='skyblue')
plt.title('Mean Squared Error (MSE) for Different Models with Cross-Validation')
plt.xlabel('Model')
plt.ylabel('Mean Squared Error (MSE)')
for i, (count, mse) in enumerate(zip(value_counts, mse_values)):
    plt.text(i, mse + 0.00001, f'{mse:.4f}', ha='center', va='bottom', fontsize=10, color='black')

plt.show()
```

```
def evaluate_model(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    evs = explained_variance_score(y_true, y_pred)
    print(f"{model_name} Metrics:")
    print(f"Mean Squared Error: {mse}")
    print(f"R-squared: {r2}")
    print(f"Explained Variance Score: {evs}")
    print("\n")
```

```
 --------------------------------------------------
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, explained_variance_score
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers


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

# Reshape input for CNN
X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], X_train_scaled.shape[1], 1)
X_test_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0], X_test_scaled.shape[1], 1)

# Simple CNN Model
model_cnn = keras.Sequential([
    layers.Conv1D(32, kernel_size=3, activation='relu', input_shape=(X_train_scaled.shape[1], 1)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(1)  # Output layer for regression
])

model_cnn.compile(optimizer='adam', loss='mean_squared_error')
model_cnn.fit(X_train_reshaped, y_train, epochs=50, batch_size=32, validation_split=0.1, verbose=0)
y_pred_cnn = model_cnn.predict(X_test_reshaped).flatten()
evaluate_model(y_test, y_pred_cnn, "Simple CNN")
```

```
10/10 [==============================] - 0s 2ms/step
Simple CNN Metrics:
Mean Squared Error: 0.003881184148800234
R-squared: 0.3802955933395955
Explained Variance Score: 0.4012015333655684
```


```
from tensorflow.keras.layers import Dense, Conv1D, GlobalAveragePooling1D, Input, Concatenate
from tensorflow.keras.models import Model

# Assuming X_train and X_test are already scaled and reshaped

# Define input layer
input_layer = Input(shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2]))

# Convolutional block
conv1 = Conv1D(64, kernel_size=3, activation='relu')(input_layer)
conv2 = Conv1D(128, kernel_size=3, activation='relu')(conv1)
conv3 = Conv1D(256, kernel_size=3, activation='relu')(conv2)

# Global Average Pooling
gap = GlobalAveragePooling1D()(conv3)

# Fully connected layers
dense1 = Dense(128, activation='relu')(gap)
dropout1 = layers.Dropout(0.2)(dense1)
dense2 = Dense(64, activation='relu')(dropout1)
output_layer = Dense(1)(dense2)

# Build the model
model_dcn = Model(inputs=input_layer, outputs=output_layer)

optimizer_dcn = keras.optimizers.Adam(learning_rate=0.001)
model_dcn.compile(optimizer=optimizer_dcn, loss='mean_squared_error')
model_dcn.fit(X_train_reshaped, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=0)
y_pred_dcn = model_dcn.predict(X_test_reshaped).flatten()
evaluate_model(y_test, y_pred_dcn, "DCN Model")
```