

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Set visualization style
sns.set(style="whitegrid", palette="muted")
```

```
In [2]: # Load dataset (replace 'file_path' with your file's path)
data = pd.read_csv("results1.csv")

# Display the first few rows
print("First 5 Rows of Data:")
print(data.head())

# Basic info and data types
print("\nDataset Info:")
print(data.info())

# Check for missing values
print("\nMissing Values:")
print(data.isnull().sum())

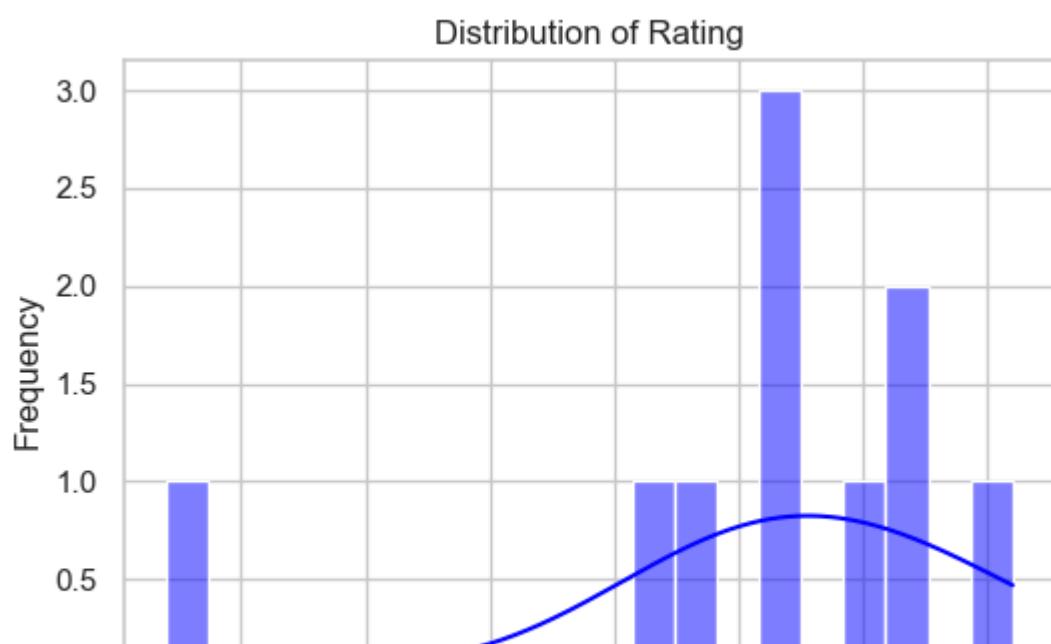
# Summary statistics
print("\nSummary Statistics:")
print(data.describe(include="all"))
```

	NaN	4	https://kinya.us (https://kinya.us)	NaN
0			Instagram \	
1			NaN	
2			NaN	
3	https://www.instagram.com/burgervillage (https://www.instagram.com/burgervillage)			
4	https://www.instagram.com/kinyaramen (https://www.instagram.com/kinyaramen)			
...			Facebook Twitter LinkedIn	
0	...	https://www.facebook.com/trullodorolongisland (https://www.facebook.com/trullodorolongisland)	NaN	NaN ...
1		https://www.facebook.com/pages (https://www.facebook.com/pages)	NaN	NaN ...
2			NaN	NaN
...				

```
In [3]: # Identify numerical columns
numerical_cols = data.select_dtypes(include=["float64", "int64"]).columns

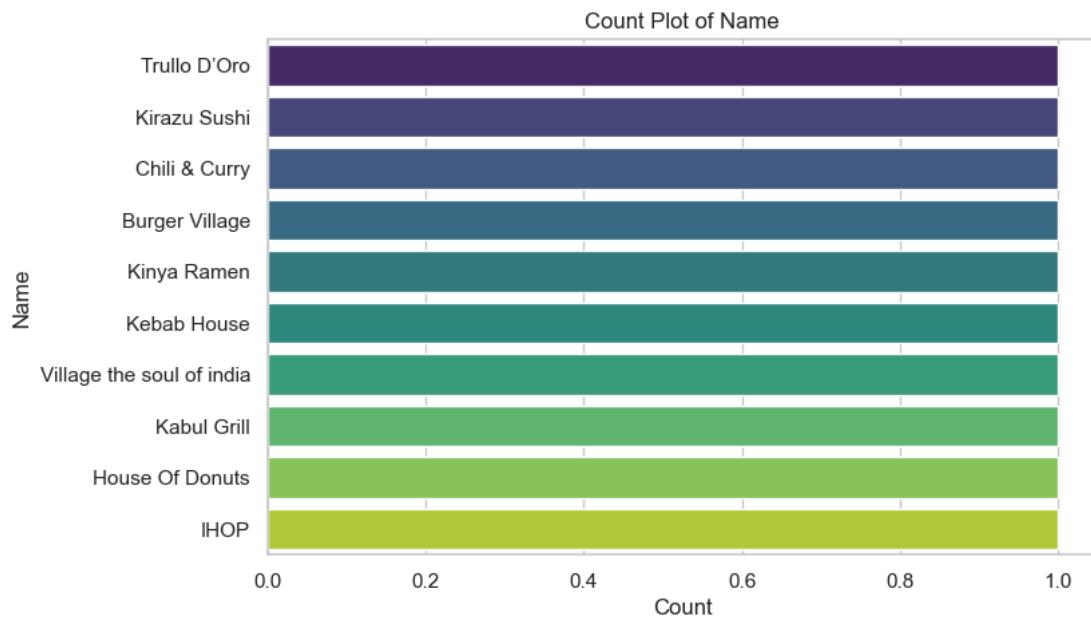
# Histograms
for col in numerical_cols:
    plt.figure(figsize=(6, 4))
    sns.histplot(data[col], kde=True, bins=20, color="blue")
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()

# Boxplots
for col in numerical_cols:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x=data[col], color="green")
    plt.title(f"Boxplot of {col}")
    plt.xlabel(col)
    plt.show()
```



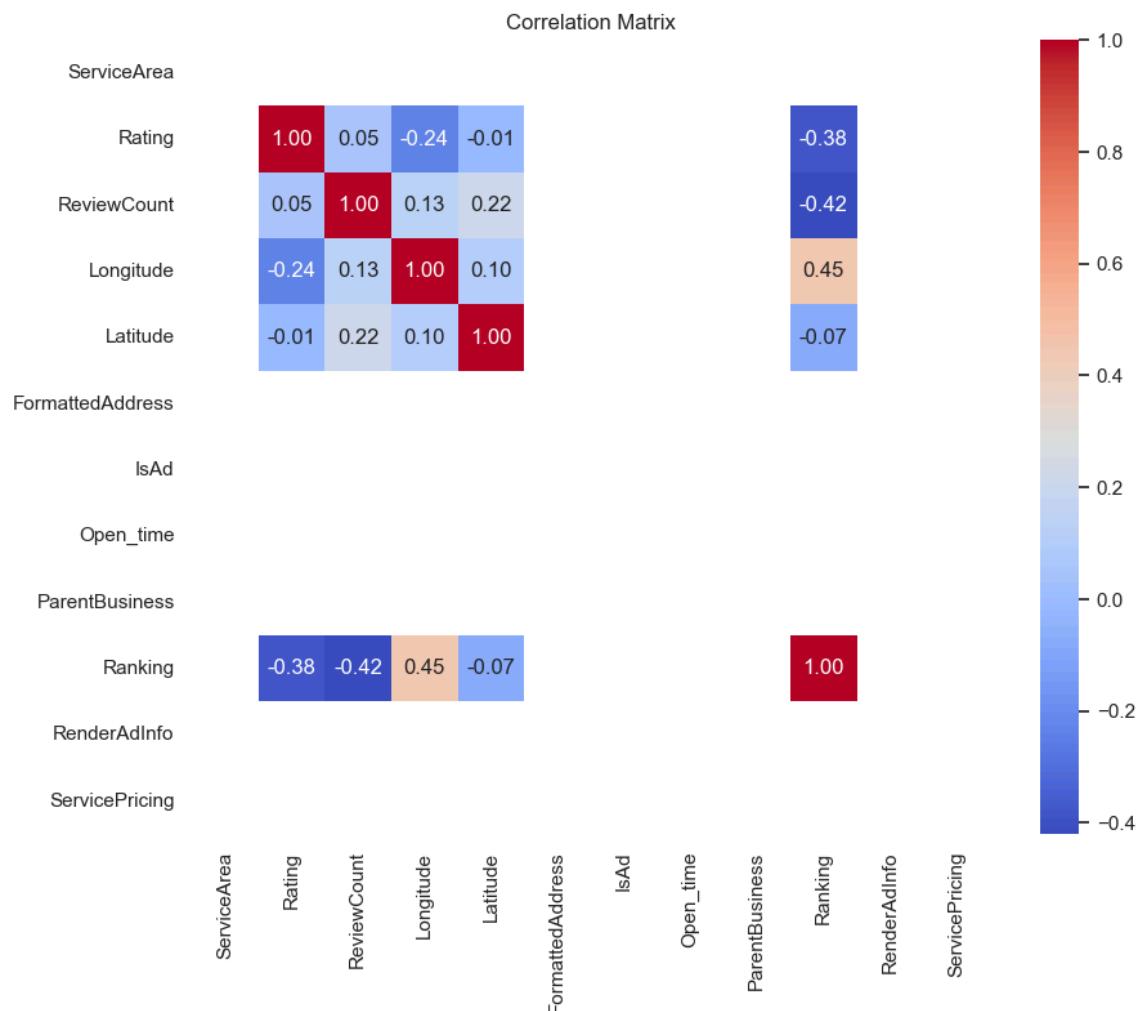
```
In [4]: # Identify categorical columns
categorical_cols = data.select_dtypes(include=["object"]).columns

# Count plots
for col in categorical_cols:
    plt.figure(figsize=(8, 5))
    sns.countplot(y=data[col], order=data[col].value_counts().index, palette="viridis")
    plt.title(f"Count Plot of {col}")
    plt.xlabel("Count")
    plt.ylabel(col)
    plt.show()
```



```
In [5]: # Compute correlation matrix
correlation_matrix = data.corr()

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```



```
In [6]: # Fill missing values with mean (numerical) or mode (categorical)
for col in data.columns:
    if data[col].isnull().sum() > 0:
        if data[col].dtype in ["float64", "int64"]:
            data[col].fillna(data[col].mean(), inplace=True)
        else:
            data[col].fillna(data[col].mode()[0], inplace=True)

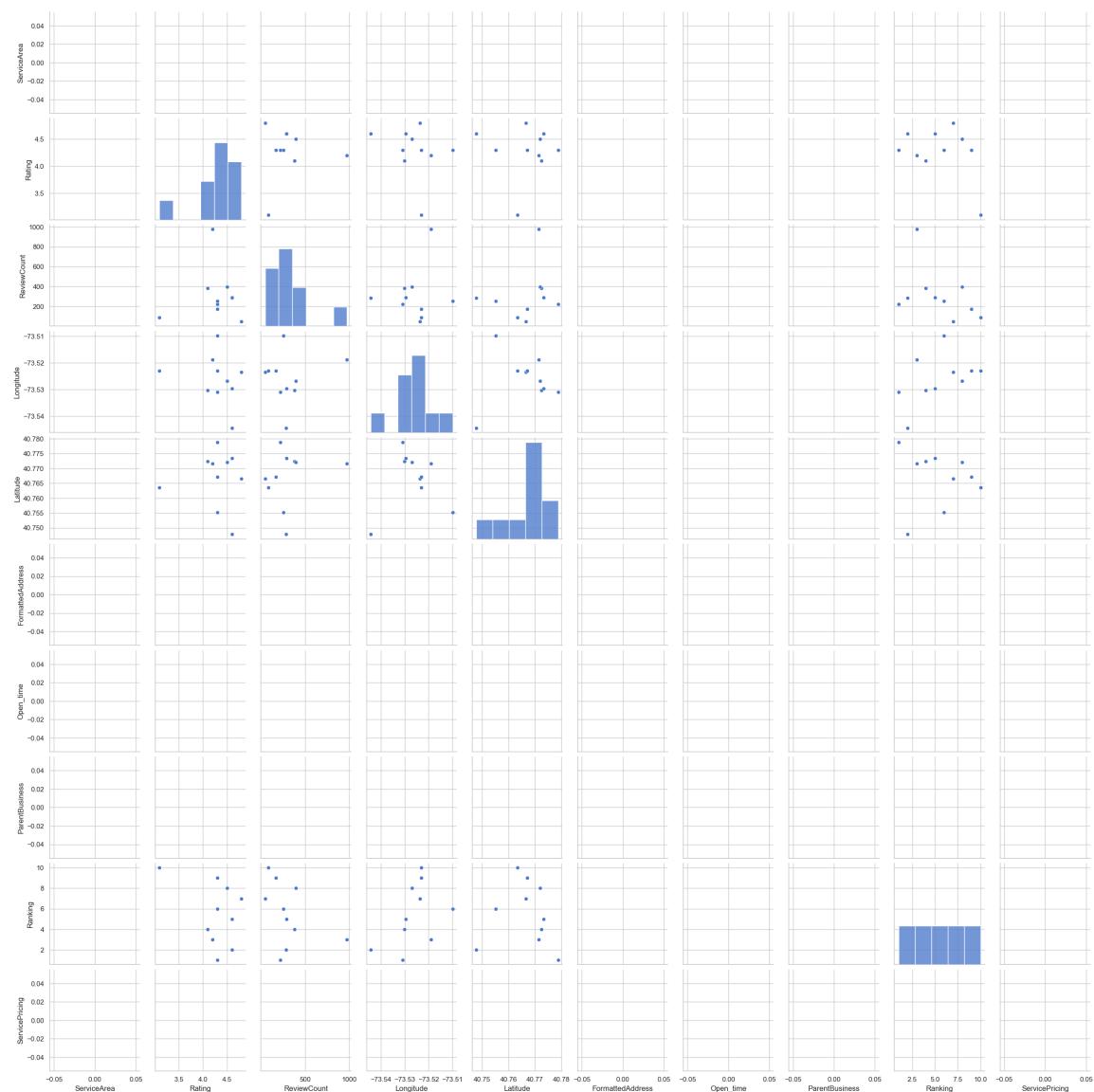
# Verify missing values are handled
print("\nMissing Values After Handling:")
print(data.isnull().sum())
```

Missing Values After Handling:

Name	0
Phone	0
Address	0
Email	0
Website	0
ServiceArea	10
Instagram	0
Facebook	0
Twitter	0
Linkedin	0
Youtube	0
BusinessUrl	0
Rating	0
ReviewCount	0
PriceRange	0
Longitude	0
Latitude	0
Alias	0
BizId	0
BusinessSectionUrls_open_hours	0
BusinessSectionUrls_reviews	0
Categories_0_title	0
Categories_0_url	0
Categories_1_title	0
Categories_1_url	0
Categories_2_title	0
Categories_2_url	0
Categories_3_title	0
Categories_3_url	0
Categories_4_title	0
Categories_4_url	0
FormattedAddress	10
IsAd	0
Open_time	10
ParentBusiness	10
Ranking	0
RenderAdInfo	0
ServicePricing	10
Snippet	0
dtype: int64	

In [7]: *# Pairplot for selected numerical columns*

```
sns.pairplot(data[numerical_cols])
plt.show()
```



```
In [8]: # Example: Price Range vs Rating
if 'PriceRange' in data.columns and 'Rating' in data.columns:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x=data['PriceRange'], y=data['Rating'], palette="muted")
    plt.title("Price Range vs Rating")
    plt.xlabel("Price Range")
    plt.ylabel("Rating")
    plt.show()

# Example: Review Count vs Ranking
if 'ReviewCount' in data.columns and 'Ranking' in data.columns:
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=data['ReviewCount'], y=data['Ranking'], color="purple")
    plt.title("Review Count vs Ranking")
    plt.xlabel("Review Count")
    plt.ylabel("Ranking")
    plt.show()
```

```
-----
-- ParseException                                     Traceback (most recent call last)
st)
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\_mathtext.py in pa
rse(self, s, fonts_object, fontsize, dpi)
    2236         try:
-> 2237             result = self._expression.parseString(s)
    2238         except ParseBaseException as err:
C:\ProgramData\Anaconda3\lib\site-packages\pyparsing\core.py in parse_st
ring(self, instring, parse_all, parseAll)
    1140                 # catch and re-raise exception from here, cleari
ng out pyparsing internal stack trace
-> 1141                 raise exc.with_traceback(None)
    1142             else:
ParseException: Expected end of text, found '$'  (at char 0), (line:1, c
ol:1)
```

```
In [9]: # Save cleaned data to a new CSV file
data.to_csv("cleaned_data.csv", index=False)
```

```
In [11]: pip install geopy
```

```
Defaulting to user installation because normal site-packages is not writea
ble
Collecting geopy
  Downloading geopy-2.4.1-py3-none-any.whl (125 kB)
----- 125.4/125.4 kB 3.6 MB/s eta 0:
00:00
Collecting geographiclib<3,>=1.52
  Downloading geographiclib-2.0-py3-none-any.whl (40 kB)
----- 40.3/40.3 kB 2.0 MB/s eta 0:
00:00
Installing collected packages: geographiclib, geopy
Successfully installed geographiclib-2.0 geopy-2.4.1
Note: you may need to restart the kernel to use updated packages.
```

```
In [12]: from sklearn.preprocessing import StandardScaler, LabelEncoder
from geopy.distance import geodesic
import numpy as np

# Create a copy of the dataset for preprocessing
preprocessed_data = data.copy()

# Step 1: Handle Missing Values
for col in preprocessed_data.columns:
    if preprocessed_data[col].isnull().sum() > 0:
        if preprocessed_data[col].dtype in ["float64", "int64"]:
            preprocessed_data[col].fillna(preprocessed_data[col].mean(), inplace=True)
        else:
            preprocessed_data[col].fillna("Unknown", inplace=True)

# Step 2: Normalize Text Data
preprocessed_data["Name"] = preprocessed_data["Name"].str.strip().str.lower

# Step 3: Convert Categorical Data
if 'PriceRange' in preprocessed_data.columns:
    price_mapping = {"Low": 1, "Medium": 2, "High": 3, "Very High": 4}
    preprocessed_data["PriceRangeNumeric"] = preprocessed_data["PriceRange"]

# Step 4: Handle Date/Time (if applicable)
if 'Open_time' in preprocessed_data.columns:
    preprocessed_data["Open_time"] = pd.to_datetime(preprocessed_data["Open_time"])

# Step 5: Feature Engineering
# Example: Compute Distance from Village restaurant
village_coords = (40.7683, -73.5258) # Replace with Village's Latitude and Longitude
if "Latitude" in preprocessed_data.columns and "Longitude" in preprocessed_data.columns:
    preprocessed_data["DistanceFromVillage"] = preprocessed_data.apply(
        lambda row: geodesic(village_coords, (row["Latitude"], row["Longitude"])).km
    )

# Example: Adjust prices (normalize prices for competitors)
if "Price" in preprocessed_data.columns:
    preprocessed_data["NormalizedPrice"] = (
        preprocessed_data["Price"] / preprocessed_data["Price"].max()
    )

# Step 6: Standardize Numerical Features
scaler = StandardScaler()
numerical_cols = preprocessed_data.select_dtypes(include=["float64", "int64"])
preprocessed_data[numerical_cols] = scaler.fit_transform(preprocessed_data[numerical_cols])

# Step 7: Encode Target or Label Data
if 'PriceRangeNumeric' in preprocessed_data.columns:
    encoder = LabelEncoder()
    preprocessed_data['PriceRangeEncoded'] = encoder.fit_transform(preprocessed_data['PriceRangeNumeric'])

# Display the first few rows of the preprocessed data
print(preprocessed_data.head())
```

	Name	Phone	\
0	trullo d'oro	(516) 719-0070	
1	kirazu sushi	(516) 933-8038	
2	chili & curry	(516) 932-9180	
3	burger village	(516) 597-5336	
4	kinya ramen	(516) 719-0388	

	Address	\
0	294 N Broadway, Hicksville, NY, 11801, US	
1	107 Stewart Ave, Ste 9, Hicksville, NY, 11801, US	
2	106 Woodbury Rd, Hicksville, NY, 11801, US	
3	216 Broadway Mall, Hicksville, NY, 11801, US	
4	100 West Old Country Rd, Hicksville, NY, 11801...	

	Email	Website	ServiceA
rea \			
0	eat@villagesoulfindia.com	http://www.trullo-doro.com (http://www.trullo-doro.com)	NaN
1	eat@villagesoulfindia.com	http://www.kirazusushi.com (http://www.kirazusushi.com)	NaN
2	eat@villagesoulfindia.com	https://chiliandcurry.square.site (https://chiliandcurry.square.site)	NaN
3	eat@villagesoulfindia.com	http://burgervillage.com (http://burgervillage.com)	NaN
4	info@kinya.us	https://kinya.us (https://kinya.us)	NaN

	Instagram	\
0	https://www.instagram.com/burgervillage (https://www.instagram.com/burgervillage)	
1	https://www.instagram.com/burgervillage (https://www.instagram.com/burgervillage)	
2	https://www.instagram.com/burgervillage (https://www.instagram.com/burgervillage)	
3	https://www.instagram.com/burgervillage (https://www.instagram.com/burgervillage)	
4	https://www.instagram.com/kinyaramen (https://www.instagram.com/kinyaramen)	

	Facebook	\
0	https://www.facebook.com/trullodorolongisland (https://www.facebook.com/trullodorolongisland)	
1	https://www.facebook.com/pages (https://www.facebook.com/pages)	
2	https://www.facebook.com/IHOP (https://www.facebook.com/IHOP)	
3	https://www.facebook.com/burgervillage (https://www.facebook.com/burgervillage)	
4	https://www.facebook.com/kinyaramensushibar.of... (https://www.facebook.com/kinyaramensushibar.of...)	

	Twitter	Linkedin	...	\
0	https://twitter.com/IHOP (https://twitter.com/IHOP)	https://www.linkedin.com/company/ihop	...	
1	https://twitter.com/IHOP (https://twitter.com/IHOP)	https://www.linkedin.com/company/ihop	...	
2	https://twitter.com/IHOP (https://twitter.com/IHOP)	https://www.linkedin.com/company/ihop	...	
3	https://twitter.com/IHOP (https://twitter.com/IHOP)	https://www.linkedin.com/company/ihop	...	
4	https://twitter.com/IHOP (https://twitter.com/IHOP)	https://www.linkedin.com/company/ihop	...	

```
in.com/company/ihop (https://www.linkedin.com/company/ihop) ...
```

	IsAd	Open_time	ParentBusiness	Ranking	RenderAdInfo	ServicePricing
0	False	NaT		NaN -1.566699	False	NaN
1	False	NaT		NaN -1.218544	False	NaN
2	False	NaT		NaN -0.870388	False	NaN
3	False	NaT		NaN -0.522233	False	NaN
4	False	NaT		NaN -0.174078	False	NaN

	Snippet	PriceRangeNumeric
0	We just had a [[HIGHLIGHT]]wonderful meal at t...	NaN
1	[[HIGHLIGHT]]The BEST sushi in town[[ENDHIGHLI...	NaN
2	I'm actually not new to Chili & Curry, I disco...	NaN
3	Came here on Saturday night and the place did ...	NaN
4	Food was good for the price. And nick was very...	NaN

	DistanceFromVillage	PriceRangeEncoded
0	0.387001	0
1	2.330559	0
2	-0.332792	0
3	-0.457517	0
4	-0.377297	0

[5 rows x 42 columns]

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\extmath.py:985: RuntimeWarning: invalid value encountered in true_divide
    updated_mean = (last_sum + new_sum) / updated_sample_count
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\extmath.py:990: RuntimeWarning: invalid value encountered in true_divide
    T = new_sum / new_sample_count
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\extmath.py:1020: RuntimeWarning: invalid value encountered in true_divide
    new_unnormalized_variance -= correction ** 2 / new_sample_count
```

```
In [13]: # Drop columns irrelevant to ML task
columns_to_drop = [
    "Phone", "Email", "Website", "Instagram", "Facebook", "Twitter",
    "Linkedin", "ParentBusiness", "Snippet", "IsAd", "RenderAdInfo"
]
preprocessed_data = preprocessed_data.drop(columns=columns_to_drop, errors=)

# Handle NaN values in important columns
# Fill numerical columns with mean
numerical_cols = preprocessed_data.select_dtypes(include=["float64", "int64"])
preprocessed_data[numerical_cols] = preprocessed_data[numerical_cols].fillna()

# Fill categorical columns with mode
categorical_cols = preprocessed_data.select_dtypes(include=["object"]).columns
preprocessed_data[categorical_cols] = preprocessed_data[categorical_cols].fillna()
```

```
In [14]: # Parse 'Open_time' if it's essential to the task
if "Open_time" in preprocessed_data.columns:
    preprocessed_data["Open_time"] = pd.to_datetime(preprocessed_data["Open_time"])

# Extract useful time features if required
if "Open_time" in preprocessed_data.columns:
    preprocessed_data["OpeningHour"] = preprocessed_data["Open_time"].dt.hour
    preprocessed_data["OpeningDay"] = preprocessed_data["Open_time"].dt.day
```

```
In [15]: from sklearn.preprocessing import StandardScaler

# Standardize numerical features
scaler = StandardScaler()
numerical_cols = preprocessed_data.select_dtypes(include=["float64", "int64"])
preprocessed_data[numerical_cols] = scaler.fit_transform(preprocessed_data[
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\extmath.py:985: RuntimeWarning: invalid value encountered in true_divide
    updated_mean = (last_sum + new_sum) / updated_sample_count
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\extmath.py:990: RuntimeWarning: invalid value encountered in true_divide
    T = new_sum / new_sample_count
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\extmath.py:1020: RuntimeWarning: invalid value encountered in true_divide
    new_unnormalized_variance -= correction ** 2 / new_sample_count
```

```
In [16]: from sklearn.preprocessing import LabelEncoder

# Encode categorical features
encoder = LabelEncoder()
categorical_cols = preprocessed_data.select_dtypes(include=["object"]).columns
for col in categorical_cols:
    preprocessed_data[col] = encoder.fit_transform(preprocessed_data[col])
```

```
In [17]: # Verify the final preprocessed data
print("Final Preprocessed Data Overview:")
print(preprocessed_data.info())

# Display sample rows
print(preprocessed_data.head())
```

Final Preprocessed Data Overview:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 33 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Name             10 non-null     int32  
 1   Address          10 non-null     int32  
 2   ServiceArea      0 non-null     float64 
 3   Youtube          10 non-null     int32  
 4   BusinessUrl      10 non-null     int32  
 5   Rating           10 non-null     float64 
 6   ReviewCount      10 non-null     float64 
 7   PriceRange       10 non-null     int32  
 8   Longitude        10 non-null     float64 
 9   Latitude          10 non-null     float64 
 10  Alias            10 non-null     int32  
 11  BizId            10 non-null     int32  
 12  BusinessSectionUrls_open_hours  10 non-null     int32  
 13  BusinessSectionUrls_reviews    10 non-null     int32  
 14  Categories_0_title   10 non-null     int32  
 15  Categories_0_url    10 non-null     int32  
 16  Categories_1_title   10 non-null     int32  
 17  Categories_1_url    10 non-null     int32  
 18  Categories_2_title   10 non-null     int32  
 19  Categories_2_url    10 non-null     int32  
 20  Categories_3_title   10 non-null     int32  
 21  Categories_3_url    10 non-null     int32  
 22  Categories_4_title   10 non-null     int32  
 23  Categories_4_url    10 non-null     int32  
 24  FormattedAddress    0 non-null     float64 
 25  Open_time          0 non-null     datetime64[ns] 
 26  Ranking            10 non-null     float64 
 27  ServicePricing     0 non-null     float64 
 28  PriceRangeNumeric   0 non-null     float64 
 29  DistanceFromVillage 10 non-null     float64 
 30  PriceRangeEncoded   10 non-null     float64 
 31  OpeningHour        0 non-null     float64 
 32  OpeningDay          0 non-null     float64 

dtypes: datetime64[ns](1), float64(13), int32(19)
memory usage: 2.0 KB
```

None

	Name	Address	ServiceArea	Youtube	BusinessUrl	Rating	ReviewCount
0	8	6	NaN	0	8	0.045222	-0.366696
1	7	2	NaN	0	7	0.723545	-0.106513
2	1	1	NaN	0	1	-0.180886	2.702654
3	0	5	NaN	0	0	-0.406994	0.287828
4	6	0	NaN	0	6	0.723545	-0.086186
	PriceRange	Longitude	Latitude	...	Categories_4_url	FormattedAddress	
0	1	-0.576548	1.361496	...	1	Na	
1	1	-2.145819	-2.159696	...	1	Na	
2	1	0.839482	0.534358	...	1	Na	
3	1	-0.489585	0.633888	...	2	Na	
4	1	-0.422302	0.745089	...	2	Na	

N

```
Open_time    Ranking    ServicePricing    PriceRangeNumeric  \
0           NaT -1.566699           NaN           NaN
1           NaT -1.218544           NaN           NaN
2           NaT -0.870388           NaN           NaN
3           NaT -0.522233           NaN           NaN
4           NaT -0.174078           NaN           NaN

DistanceFromVillage  PriceRangeEncoded  OpeningHour  OpeningDay
0           0.387001           0.0           NaN           NaN
1           2.330559           0.0           NaN           NaN
2           -0.332792          0.0           NaN           NaN
3           -0.457517          0.0           NaN           NaN
4           -0.377297          0.0           NaN           NaN

[5 rows x 33 columns]
```

```
In [18]: columns_to_drop = [
    "ServiceArea", "FormattedAddress", "ServicePricing",
    "PriceRangeNumeric", "Open_time", "OpeningHour", "OpeningDay"
]
preprocessed_data = preprocessed_data.drop(columns=columns_to_drop, errors=
```

```
In [19]: # Example placeholders (replace with actual data)
preprocessed_data["Temperature"] = [50, 45, 32, 60, 55, 48, 35, 65, 30, 40]
preprocessed_data["RainOrSnow"] = [0, 1, 1, 0, 0, 1, 1, 0, 1, 0] # 0 = No,
preprocessed_data["BusyFlag"] = [1, 1, 0, 0, 1, 1, 0, 1, 0, 1] # 0 = Not bu
```

```
In [20]: print(preprocessed_data.info())
print(preprocessed_data.head())
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 29 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Name              10 non-null      int32  
 1   Address           10 non-null      int32  
 2   Youtube           10 non-null      int32  
 3   BusinessUrl       10 non-null      int32  
 4   Rating            10 non-null      float64 
 5   ReviewCount       10 non-null      float64 
 6   PriceRange         10 non-null      int32  
 7   Longitude          10 non-null      float64 
 8   Latitude           10 non-null      float64 
 9   Alias              10 non-null      int32  
 10  BizId             10 non-null      int32  
 11  BusinessSectionUrls_open_hours 10 non-null      int32  
 12  BusinessSectionUrls_reviews   10 non-null      int32  
 13  Categories_0_title    10 non-null      int32  
 14  Categories_0_url     10 non-null      int32  
 15  Categories_1_title    10 non-null      int32  
 16  Categories_1_url     10 non-null      int32  
 17  Categories_2_title    10 non-null      int32  
 18  Categories_2_url     10 non-null      int32  
 19  Categories_3_title    10 non-null      int32  
 20  Categories_3_url     10 non-null      int32  
 21  Categories_4_title    10 non-null      int32  
 22  Categories_4_url     10 non-null      int32  
 23  Ranking             10 non-null      float64 
 24  DistanceFromVillage  10 non-null      float64 
 25  PriceRangeEncoded    10 non-null      float64 
 26  Temperature          10 non-null      int64  
 27  RainOrSnow           10 non-null      int64  
 28  BusyFlag            10 non-null      int64  
dtypes: float64(7), int32(19), int64(3)
memory usage: 1.6 KB
None
   Name  Address  Youtube  BusinessUrl  Rating  ReviewCount  PriceRange
\ 
 0   8        6        0           8  0.045222  -0.366696      1
 1   7        2        0           7  0.723545  -0.106513      1
 2   1        1        0           1 -0.180886  2.702654      1
 3   0        5        0           0 -0.406994  0.287828      1
 4   6        0        0           6  0.723545  -0.086186      1

   Longitude  Latitude  Alias  ...  Categories_3_title  Categories_3_url
\ 
 0  -0.576548  1.361496     8  ...                      0                      0
 1  -2.145819 -2.159696     7  ...                      0                      0
 2   0.839482  0.534358     1  ...                      0                      0
 3  -0.489585  0.633888     0  ...                      0                      0
 4  -0.422302  0.745089     6  ...                      0                      0

   Categories_4_title  Categories_4_url  Ranking  DistanceFromVillage \
 0                      1                  1  -1.566699  0.387001
 1                      1                  1  -1.218544  2.330559
 2                      1                  1  -0.870388 -0.332792
 3                      2                  2  -0.522233 -0.457517
 4                      2                  2  -0.174078 -0.377297

   PriceRangeEncoded  Temperature  RainOrSnow  BusyFlag

```

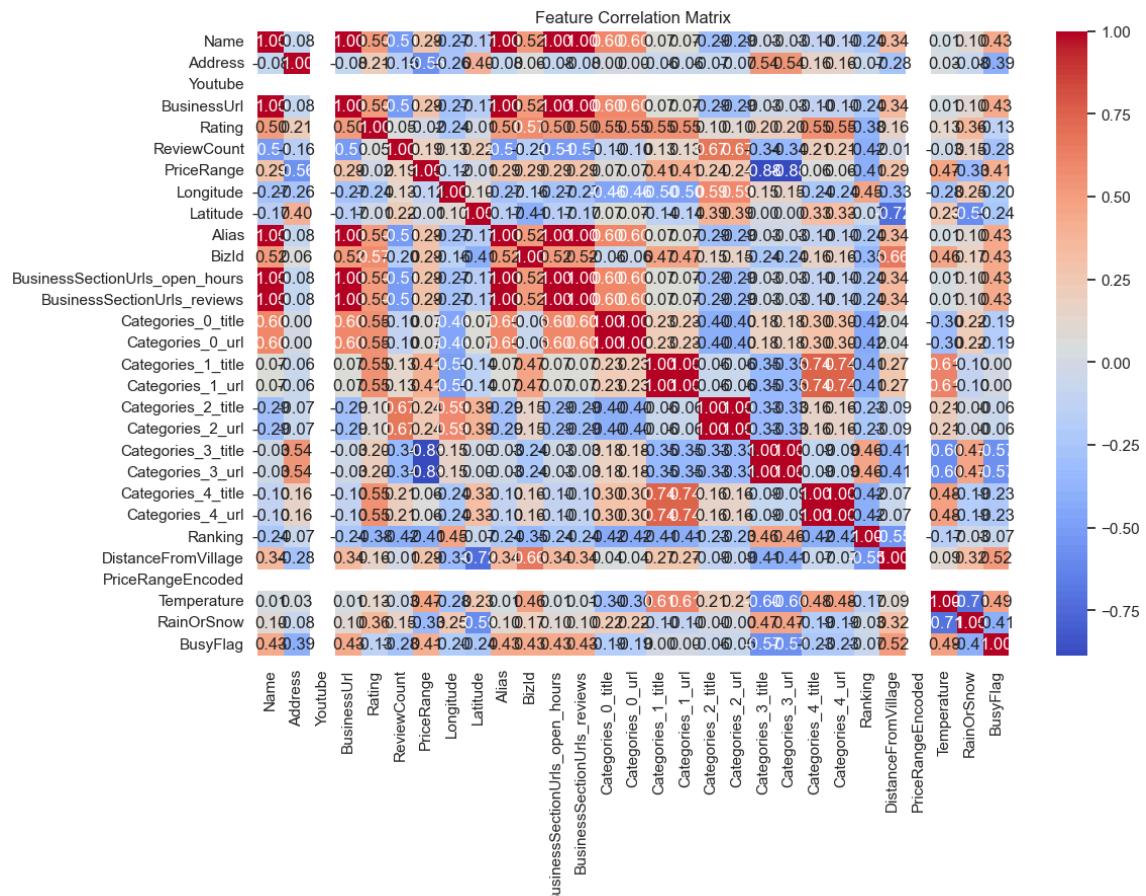
0	0.0	50	0	1
1	0.0	45	1	1
2	0.0	32	1	0
3	0.0	60	0	0
4	0.0	55	0	1

[5 rows x 29 columns]

```
In [21]: import seaborn as sns
import matplotlib.pyplot as plt

# Compute correlation matrix
correlation_matrix = preprocessed_data.corr()

# Plot heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix")
plt.show()
```



```
In [22]: from sklearn.feature_selection import VarianceThreshold

# Set a threshold for variance (e.g., 0.01)
selector = VarianceThreshold(threshold=0.01)
high_variance_data = selector.fit_transform(preprocessed_data)

# Get selected feature names
selected_features = preprocessed_data.columns[selector.get_support()]
print("Selected Features (Variance Threshold):", selected_features)
```

```
Selected Features (Variance Threshold): Index(['Name', 'Address', 'BusinessUrl', 'Rating', 'ReviewCount', 'PriceRange', 'Longitude', 'Latitude', 'Alias', 'BizId', 'BusinessSectionUrls_open_hours', 'BusinessSectionUrls_reviews', 'Categories_0_title', 'Categories_0_url', 'Categories_1_title', 'Categories_1_url', 'Categories_2_title', 'Categories_2_url', 'Categories_3_title', 'Categories_3_url', 'Categories_4_title', 'Categories_4_url', 'Ranking', 'DistanceFromVillage', 'Temperature', 'RainOrSnow', 'BusyFlag'],
      dtype='object')
```

```
In [24]: from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestRegressor

# Replace 'target_column' with the actual name of your target variable
target_column = "ActualTargetColumnName" # Update this to match your data

# Define the model
model = RandomForestRegressor(random_state=42)

# Check if the target column exists
if target_column not in preprocessed_data.columns:
    raise ValueError(f"Target column '{target_column}' not found in dataset")

# Perform RFE
X = preprocessed_data.drop(target_column, axis=1)
y = preprocessed_data[target_column]
rfe = RFE(estimator=model, n_features_to_select=10) # Select top 10 features
rfe.fit(X, y)

# Get selected features
rfe_selected_features = X.columns[rfe.support_]
print("Selected Features (RFE):", rfe_selected_features)
```

```
-----
-
ValueError                                                 Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\417814623.py in <module>
    10 # Check if the target column exists
    11 if target_column not in preprocessed_data.columns:
--> 12     raise ValueError(f"Target column '{target_column}' not found in dataset.")
    13
    14 # Perform RFE
```

ValueError: Target column 'ActualTargetColumnName' not found in dataset.

```
In [25]: target_column = "Price" # Replace 'Price' with the actual target column name
```

```
In [26]: print(preprocessed_data.columns)
```

```
Index(['Name', 'Address', 'Youtube', 'BusinessUrl', 'Rating', 'ReviewCount',
       'PriceRange', 'Longitude', 'Latitude', 'Alias', 'BizId',
       'BusinessSectionUrls_open_hours', 'BusinessSectionUrls_reviews',
       'Categories_0_title', 'Categories_0_url', 'Categories_1_title',
       'Categories_1_url', 'Categories_2_title', 'Categories_2_url',
       'Categories_3_title', 'Categories_3_url', 'Categories_4_title',
       'Categories_4_url', 'Ranking', 'DistanceFromVillage',
       'PriceRangeEncoded', 'Temperature', 'RainOrSnow', 'BusyFlag'],
      dtype='object')
```

```
In [27]: from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestRegressor

# Replace 'Price' with your actual target column name
target_column = "Price" # Update this to match your actual data

# Define the model
model = RandomForestRegressor(random_state=42)

# Check if the target column exists
if target_column not in preprocessed_data.columns:
    raise ValueError(f"Target column '{target_column}' not found in dataset")

# Perform RFE
X = preprocessed_data.drop(target_column, axis=1)
y = preprocessed_data[target_column]
rfe = RFE(estimator=model, n_features_to_select=10) # Select top 10 features
rfe.fit(X, y)

# Get selected features
rfe_selected_features = X.columns[rfe.support_]
print("Selected Features (RFE):", rfe_selected_features)
```

```
-----
-
ValueError                                                 Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\1492160400.py in <module>
    10 # Check if the target column exists
    11 if target_column not in preprocessed_data.columns:
--> 12     raise ValueError(f"Target column '{target_column}' not found in dataset.")
    13
    14 # Perform RFE
```

ValueError: Target column 'Price' not found in dataset.

```
In [28]: print(preprocessed_data.columns)
```

```
Index(['Name', 'Address', 'Youtube', 'BusinessUrl', 'Rating', 'ReviewCount',
       'PriceRange', 'Longitude', 'Latitude', 'Alias', 'BizId',
       'BusinessSectionUrls_open_hours', 'BusinessSectionUrls_reviews',
       'Categories_0_title', 'Categories_0_url', 'Categories_1_title',
       'Categories_1_url', 'Categories_2_title', 'Categories_2_url',
       'Categories_3_title', 'Categories_3_url', 'Categories_4_title',
       'Categories_4_url', 'Ranking', 'DistanceFromVillage',
       'PriceRangeEncoded', 'Temperature', 'RainOrSnow', 'BusyFlag'],
      dtype='object')
```

```
In [29]: # Print the column names to verify the target column exists
print("Column names in the dataset:", preprocessed_data.columns)

# Replace 'Price' with the actual name of your target column after verifying
target_column = "ActualTargetColumnName" # Replace with correct column name

# Define the model
model = RandomForestRegressor(random_state=42)

# Check if the target column exists
if target_column not in preprocessed_data.columns:
    raise ValueError(f"Target column '{target_column}' not found in dataset")

# Perform RFE
X = preprocessed_data.drop(target_column, axis=1)
y = preprocessed_data[target_column]
rfe = RFE(estimator=model, n_features_to_select=10) # Select top 10 features
rfe.fit(X, y)

# Get selected features
rfe_selected_features = X.columns[rfe.support_]
print("Selected Features (RFE):", rfe_selected_features)
```

```
Column names in the dataset: Index(['Name', 'Address', 'Youtube', 'BusinessUrl', 'Rating', 'ReviewCount', 'PriceRange', 'Longitude', 'Latitude', 'Alias', 'BizId', 'BusinessSectionUrls_open_hours', 'BusinessSectionUrls_reviews', 'Categories_0_title', 'Categories_0_url', 'Categories_1_title', 'Categories_1_url', 'Categories_2_title', 'Categories_2_url', 'Categories_3_title', 'Categories_3_url', 'Categories_4_title', 'Categories_4_url', 'Ranking', 'DistanceFromVillage', 'PriceRangeEncoded', 'Temperature', 'RainOrSnow', 'BusyFlag'], dtype='object')
```

```
-----
-
ValueError                                                 Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\271399286.py in <module>
    10 # Check if the target column exists
    11 if target_column not in preprocessed_data.columns:
--> 12     raise ValueError(f"Target column '{target_column}' not found in dataset.")
    13
    14 # Perform RFE
```

ValueError: Target column 'ActualTargetColumnName' not found in dataset.

```
In [39]: from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestRegressor

# Replace with the actual target column, for example 'Ranking' or another re
target_column = "Ranking" # Replace with the actual target column name after

# Define the model
model = RandomForestRegressor(random_state=42)

# Check if the target column exists
if target_column not in preprocessed_data.columns:
    raise ValueError(f"Target column '{target_column}' not found in dataset")

# Perform RFE
X = preprocessed_data.drop(target_column, axis=1)
y = preprocessed_data[target_column]
rfe = RFE(estimator=model, n_features_to_select=10) # Select top 10 features
rfe.fit(X, y)

# Get selected features
rfe_selected_features = X.columns[rfe.support_]
print("Selected Features (RFE):", rfe_selected_features)
```

```
Selected Features (RFE): Index(['Address', 'BusinessUrl', 'ReviewCount',
'Longitude', 'Latitude',
'BusinessSectionUrls_open_hours', 'Categories_0_url',
'Categories_1_url', 'Categories_2_url', 'DistanceFromVillage'],
dtype='object')
```

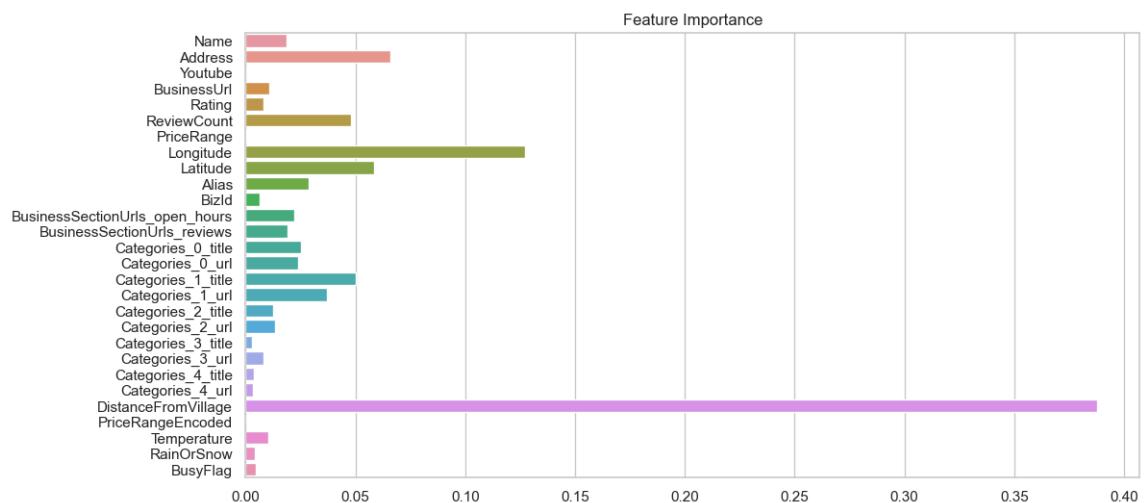
```
In [44]: import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor

# Define the actual target column name
target_column = "Ranking" # Replace this with your actual target column name

# Fit RandomForestRegressor for feature importance
model = RandomForestRegressor(random_state=42)
model.fit(preprocessed_data.drop(target_column, axis=1), preprocessed_data[target_column])

# Plot feature importance
importances = model.feature_importances_
features = preprocessed_data.drop(target_column, axis=1).columns

plt.figure(figsize=(12, 6))
sns.barplot(x=importances, y=features)
plt.title("Feature Importance")
plt.show()
```



```
In [43]: from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

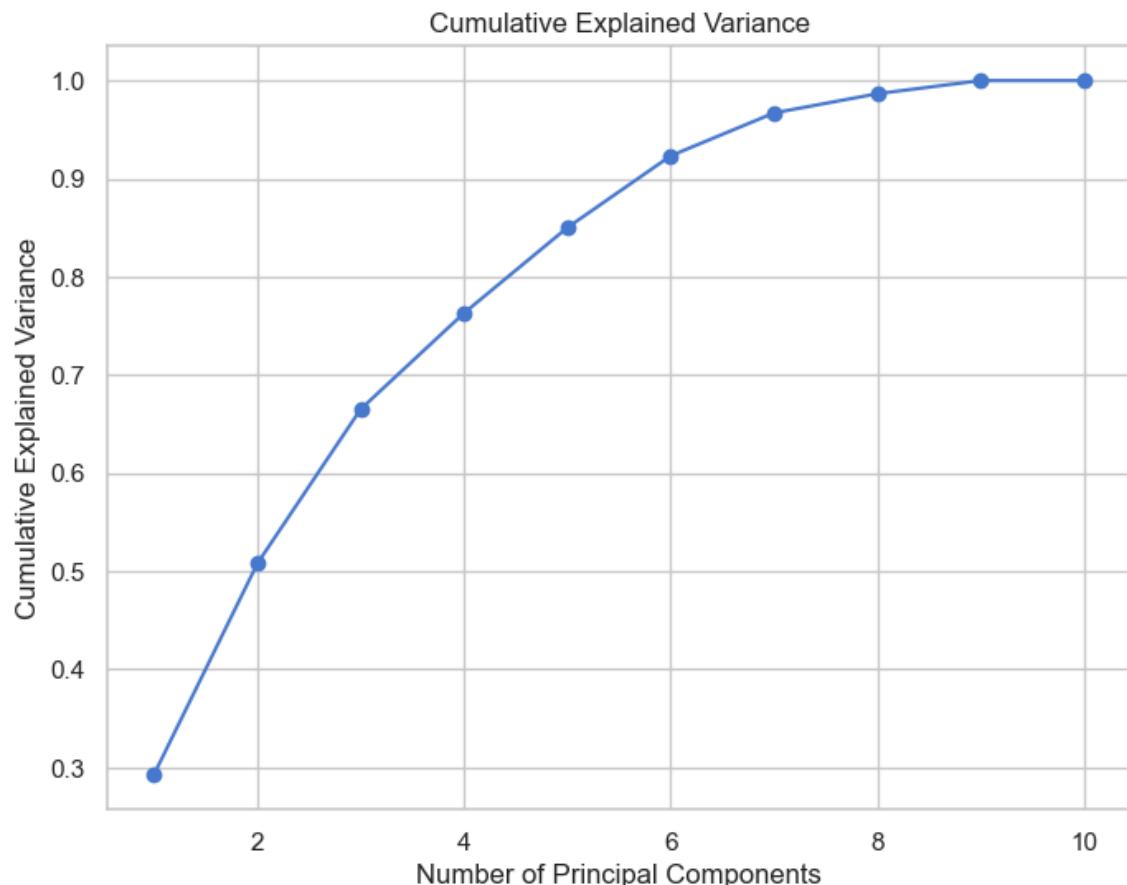
# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(preprocessed_data)

# Apply PCA
pca = PCA(n_components=10) # Select the number of principal components
principal_components = pca.fit_transform(scaled_data)

# Explained variance ratio
print("Explained Variance Ratio:", pca.explained_variance_ratio_)

# Plot cumulative explained variance
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(pca.explained_variance_ratio_) + 1),
         pca.explained_variance_ratio_.cumsum(), marker='o')
plt.title("Cumulative Explained Variance")
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.show()
```

Explained Variance Ratio: [2.92648505e-01 2.15024833e-01 1.57425072e-01 9.79498905e-02 8.70924126e-02 7.31684814e-02 4.37174845e-02 1.95767373e-02 1.33965834e-02 5.07082154e-33]



```
In [46]: import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset
file_path = r"C:\Users\shubh\Downloads\results1.csv" # Use the raw string
data = pd.read_csv(file_path)

# Specify the target and features
target_column = "Ranking" # Replace this with your actual target column name
features = data.drop(target_column, axis=1) # Drop the target column
target = data[target_column] # Select the target column

# Perform train-test split
X_train, X_test, y_train, y_test = train_test_split(
    features, target, test_size=0.2, random_state=42
)

# Display the shapes of the splits
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (8, 38)
X_test shape: (2, 38)
y_train shape: (8,)
y_test shape: (2,)
```

```
In [49]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load the dataset
file_path = r"C:\Users\shubh\Downloads\results1.csv"
data = pd.read_csv(file_path)

# Inspect the data
print(data.info())

# Convert categorical columns to numerical values
categorical_columns = data.select_dtypes(include=['object']).columns
print(f"Categorical Columns: {categorical_columns}")

# Apply Label Encoding for simplicity (or use OneHotEncoder for more flexibility)
label_encoders = {}
for col in categorical_columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le

# Check for missing values
print(data.isnull().sum())

# Fill or drop missing values if needed
data = data.fillna(0) # Example: Replace NaNs with 0

# Define features and target
target_column = "Ranking" # Replace this with your actual target column name
features = data.drop(target_column, axis=1)
target = data[target_column]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 39 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Name             10 non-null      object  
 1   Phone            10 non-null      object  
 2   Address          8 non-null       object  
 3   Email            4 non-null       object  
 4   Website          9 non-null       object  
 5   ServiceArea      0 non-null       float64 
 6   Instagram        3 non-null       object  
 7   Facebook         7 non-null       object  
 8   Twitter          1 non-null       object  
 9   Linkedin         1 non-null       object  
 10  Youtube          1 non-null       object  
 11  BusinessUrl     10 non-null      object  
 12  Rating           10 non-null      float64 
 13  ReviewCount      10 non-null      int64  
 14  PriceRange        9 non-null       object  
 15  Longitude         10 non-null      float64 
 16  Latitude          10 non-null      float64 
 17  Alias             10 non-null      object  
 18  BizId             10 non-null      object  
 19  BusinessSectionUrls_open_hours  10 non-null      object  
 20  BusinessSectionUrls_reviews   10 non-null      object  
 21  Categories_0_title    10 non-null      object  
 22  Categories_0_url     10 non-null      object  
 23  Categories_1_title    10 non-null      object  
 24  Categories_1_url     10 non-null      object  
 25  Categories_2_title    10 non-null      object  
 26  Categories_2_url     10 non-null      object  
 27  Categories_3_title    9 non-null       object  
 28  Categories_3_url     9 non-null       object  
 29  Categories_4_title    7 non-null       object  
 30  Categories_4_url     7 non-null       object  
 31  FormattedAddress     0 non-null       float64 
 32  IsAd              10 non-null      bool    
 33  Open_time          0 non-null       float64 
 34  ParentBusiness      0 non-null       float64 
 35  Ranking            10 non-null      int64  
 36  RenderAdInfo       10 non-null      bool    
 37  ServicePricing      0 non-null       float64 
 38  Snippet            10 non-null      object  
dtypes: bool(2), float64(8), int64(2), object(27)
memory usage: 3.0+ KB
None
Categorical Columns: Index(['Name', 'Phone', 'Address', 'Email', 'Website', 'Instagram', 'Facebook', 'Twitter', 'Linkedin', 'Youtube', 'BusinessUrl', 'PriceRange', 'Alias', 'BizId', 'BusinessSectionUrls_open_hours', 'BusinessSectionUrls_reviews', 'Categories_0_title', 'Categories_0_url', 'Categories_1_title', 'Categories_1_url', 'Categories_2_title', 'Categories_2_url', 'Categories_3_title', 'Categories_3_url', 'Categories_4_title', 'Categories_4_url', 'Snippet'], dtype='object')
Name          0
Phone         0
Address        0

```

```
Email          0
Website        0
ServiceArea    10
Instagram      0
Facebook       0
Twitter        0
Linkedin        0
Youtube        0
BusinessUrl    0
Rating          0
ReviewCount    0
PriceRange      0
Longitude       0
Latitude        0
Alias           0
BizId           0
BusinessSectionUrls_open_hours  0
BusinessSectionUrls_reviews      0
Categories_0_title    0
Categories_0_url      0
Categories_1_title    0
Categories_1_url      0
Categories_2_title    0
Categories_2_url      0
Categories_3_title    0
Categories_3_url      0
Categories_4_title    0
Categories_4_url      0
FormattedAddress    10
IsAd             0
Open_time        10
ParentBusiness   10
Ranking          0
RenderAdInfo     0
ServicePricing   10
Snippet          0
dtype: int64
X_train shape: (8, 38)
X_test shape: (2, 38)
```

```
In [50]: from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Train the Support Vector Regressor
svm_model = SVR(kernel='rbf', C=1.0, epsilon=0.1)
svm_model.fit(X_train, y_train)

# Make predictions
y_pred_svm = svm_model.predict(X_test)

# Evaluate the model
mae_svm = mean_absolute_error(y_test, y_pred_svm)
mse_svm = mean_squared_error(y_test, y_pred_svm)
r2_svm = r2_score(y_test, y_pred_svm)

print("SVM Performance:")
print(f"Mean Absolute Error: {mae_svm}")
print(f"Mean Squared Error: {mse_svm}")
print(f"R2 Score: {r2_svm}")
```

```
SVM Performance:
Mean Absolute Error: 3.3016845814084763
Mean Squared Error: 10.92403679265615
R2 Score: 0.10824189447704902
```

```
In [51]: from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR

# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto', 0.1, 0.01],
    'epsilon': [0.1, 0.2, 0.5]
}

# GridSearchCV for SVR
grid_search = GridSearchCV(SVR(kernel='rbf')), param_grid, cv=5, scoring='r2'
grid_search.fit(X_train, y_train)

# Best parameters and performance
print("Best Parameters:", grid_search.best_params_)
print("Best R2 Score on Training Data:", grid_search.best_score_)

# Evaluate on test data
best_svr = grid_search.best_estimator_
y_pred_tuned = best_svr.predict(X_test)

print("Tuned SVM Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_tuned))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_tuned))
print("R2 Score:", r2_score(y_test, y_pred_tuned))
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] END .....C=0.1, epsilon=0.1, gamma=scale; total time
= 0.0s
[CV] END .....C=0.1, epsilon=0.1, gamma=scale; total time
= 0.0s
[CV] END .....C=0.1, epsilon=0.1, gamma=scale; total time
= 0.0s
[CV] END .....C=0.1, epsilon=0.1, gamma=scale; total time
= 0.0s
[CV] END .....C=0.1, epsilon=0.1, gamma=scale; total time
= 0.0s
[CV] END .....C=0.1, epsilon=0.1, gamma=scale; total time
= 0.0s
[CV] END .....C=0.1, epsilon=0.1, gamma=auto; total time
= 0.0s
[CV] END .....C=0.1, epsilon=0.1, gamma=auto; total time
= 0.0s
[CV] END .....C=0.1, epsilon=0.1, gamma=auto; total time
= 0.0s
[CV] END .....C=0.1, epsilon=0.1, gamma=auto; total time
= 0.0s
[CV] END .....C=0.1, epsilon=0.1, gamma=auto; total time
= 0.0s
```

```
In [52]: from sklearn.ensemble import RandomForestRegressor

# Train Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42)
rf_model.fit(X_train, y_train)

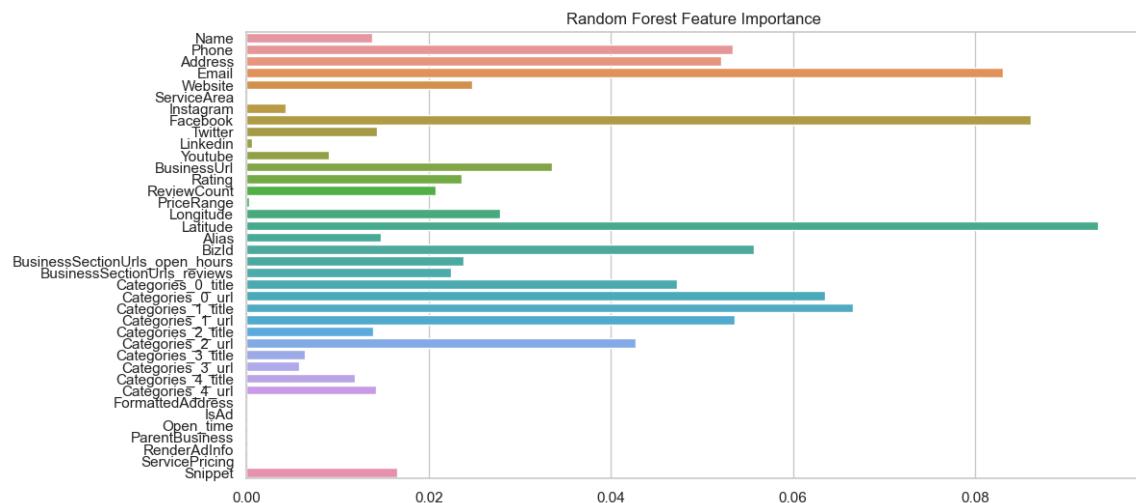
# Evaluate Random Forest
y_pred_rf = rf_model.predict(X_test)
print("Random Forest Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_rf))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_rf))
print("R² Score:", r2_score(y_test, y_pred_rf))
```

Random Forest Performance:
 Mean Absolute Error: 3.8775
 Mean Squared Error: 15.368512500000001
 R² Score: -0.25457244897959197

```
In [53]: # Plot feature importance
import matplotlib.pyplot as plt
import seaborn as sns

importances = rf_model.feature_importances_
features = X_train.columns

plt.figure(figsize=(12, 6))
sns.barplot(x=importances, y=features)
plt.title("Random Forest Feature Importance")
plt.show()
```



```
In [58]: from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform grid search
grid_search_rf = GridSearchCV(RandomForestRegressor(random_state=42), param_grid)
grid_search_rf.fit(X_train, y_train)

# Best parameters and performance
print("Best Parameters:", grid_search_rf.best_params_)
print("Best R² Score on Training Data:", grid_search_rf.best_score_)

# Evaluate on test data
best_rf_model = grid_search_rf.best_estimator_
y_pred_rf_tuned = best_rf_model.predict(X_test)

print("Tuned Random Forest Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_rf_tuned))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_rf_tuned))
print("R² Score:", r2_score(y_test, y_pred_rf_tuned))
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 0.0s
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 0.0s
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 0.0s
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 0.0s
[CV] END max_depth=5, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time= 0.0s

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py:796: UndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
    warnings.warn(msg, UndefinedMetricWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py:796: UndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
    warnings.warn(msg, UndefinedMetricWarning)
```

```
In [62]: from sklearn.ensemble import StackingRegressor

# Combine models
estimators = [('rf', RandomForestRegressor(random_state=42)), ('svm', SVR(kernel='rbf', C=100, gamma='auto'))]
stacking_model = StackingRegressor(estimators=estimators, final_estimator=RandomForestRegressor(n_estimators=100))

# Train and evaluate
stacking_model.fit(X_train, y_train)
y_pred_stack = stacking_model.predict(X_test)

print("Stacking Model Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_stack))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_stack))
print("R² Score:", r2_score(y_test, y_pred_stack))
```

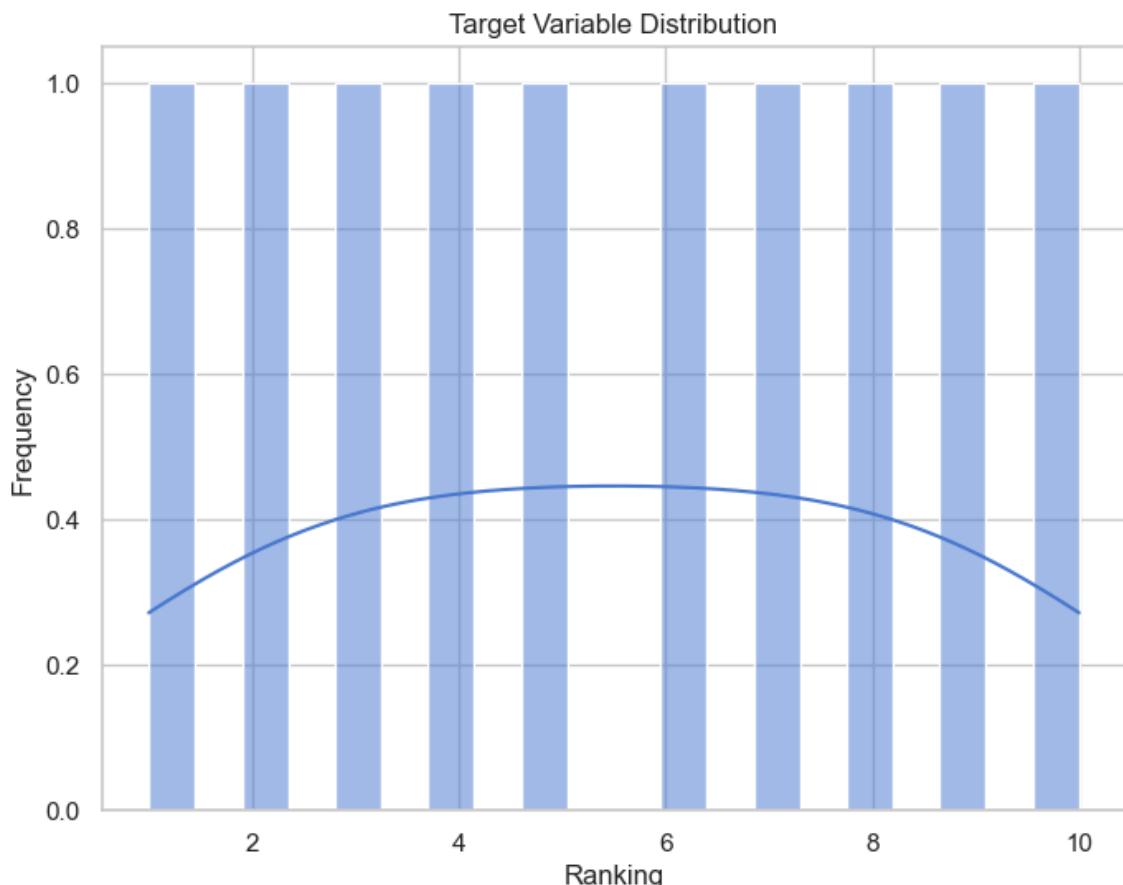
```
Stacking Model Performance:
Mean Absolute Error: 4.475
Mean Squared Error: 20.03665
R² Score: -0.6356448979591838
```

```
In [61]:
```

```
File "C:\Users\shubh\AppData\Local\Temp\ipykernel_33244\3459097293.py",
line 1
    print(Mean Absolute Error)
               ^
SyntaxError: invalid syntax
```

```
In [63]: import matplotlib.pyplot as plt
import seaborn as sns

# Visualize the target variable
plt.figure(figsize=(8, 6))
sns.histplot(data[target_column], kde=True, bins=20)
plt.title("Target Variable Distribution")
plt.xlabel("Ranking")
plt.ylabel("Frequency")
plt.show()
```



```
In [64]: importances = best_rf_model.feature_importances_
important_features = features[importances > 0.01] # Retain features with importance > 0.01

print("Selected Features:", important_features)
X_train_reduced = X_train[important_features]
X_test_reduced = X_test[important_features]
```

```
Selected Features: Index(['Name', 'Phone', 'Address', 'Email', 'Website',
'Facebook', 'Twitter',
'BusinessUrl', 'Rating', 'ReviewCount', 'Longitude', 'Latitude',
'Alias', 'BizId', 'BusinessSectionUrls_open_hours',
'BusinessSectionUrls_reviews', 'Categories_0_title', 'Categories_0_url',
'Categories_1_title', 'Categories_1_url', 'Categories_2_title',
'Categories_2_url', 'Categories_4_url'],
dtype='object')
```

```
In [65]: from sklearn.decomposition import PCA

# Standardize features before PCA
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Perform PCA
pca = PCA(n_components=10) # Retain top 10 components
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print("Explained Variance by Components:", pca.explained_variance_ratio_.cum
```

```
-----
-
ValueError                                Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\1802139081.py in <module>
      9 # Perform PCA
     10 pca = PCA(n_components=10) # Retain top 10 components
--> 11 X_train_pca = pca.fit_transform(X_train_scaled)
     12 X_test_pca = pca.transform(X_test_scaled)
     13

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\decomposition\_pca.py in fit_transform(self, X, y)
    405         C-ordered array, use 'np.ascontiguousarray'.
    406         """
--> 407         U, S, Vt = self._fit(X)
    408         U = U[:, : self.n_components_]
    409

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\decomposition\_pca.py in _fit(self, X)
    455         # Call different fits for either full or truncated SVD
    456         if self._fit_svd_solver == "full":
--> 457             return self._fit_full(X, n_components)
    458         elif self._fit_svd_solver in ["arpack", "randomized"]:
    459             return self._fit_truncated(X, n_components, self._fit_svd_solver)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\decomposition\_pca.py in _fit_full(self, X, n_components)
    473         )
    474         elif not 0 <= n_components <= min(n_samples, n_features):
--> 475             raise ValueError(
    476                 "n_components=%r must be between 0 and "
    477                 "min(n_samples, n_features)=%r with "
```

ValueError: n_components=10 must be between 0 and min(n_samples, n_features)=8 with svd_solver='full'

In [68]: # Import necessary Libraries

```
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from xgboost import XGBRegressor

# Train XGBoost Regressor
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=6,
xgb_model.fit(X_train, y_train)

# Predict on test set
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate the model
print("XGBoost Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_xgb))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_xgb))
print("R² Score:", r2_score(y_test, y_pred_xgb))
```

```
-----
-
ModuleNotFoundError Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\3154939345.py in <module>
    2 import numpy as np
    3 from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
--> 4 from xgboost import XGBRegressor
      5
      6 # Train XGBoost Regressor

ModuleNotFoundError: No module named 'xgboost'
```

In [69]: !pip install xgboost

```
Defaulting to user installation because normal site-packages is not writable
Collecting xgboost
  Downloading xgboost-2.1.3-py3-none-win_amd64.whl (124.9 MB)
----- 124.9/124.9 MB 3.6 MB/s eta 0:00:00
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from xgboost) (1.9.1)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from xgboost) (1.21.5)
Installing collected packages: xgboost
Successfully installed xgboost-2.1.3
```

```
In [70]: # Import necessary Libraries
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from xgboost import XGBRegressor

# Train XGBoost Regressor
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=6,
xgb_model.fit(X_train, y_train)

# Predict on test set
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate the model
print("XGBoost Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_xgb))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_xgb))
print("R² Score:", r2_score(y_test, y_pred_xgb))
```

```
XGBoost Performance:
Mean Absolute Error: 3.127051055431366
Mean Squared Error: 16.605971721220847
R² Score: -0.3555895282629262
```

```
In [71]: from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor

# Define parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0]
}

# Perform grid search
grid_search = GridSearchCV(estimator=XGBRegressor(random_state=42),
                           param_grid=param_grid,
                           scoring='r2',
                           cv=5,
                           verbose=2)

grid_search.fit(X_train, y_train)

# Best parameters and performance
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validated R^2 Score:", grid_search.best_score_)

# Evaluate on test set with best parameters
best_xgb = grid_search.best_estimator_
y_pred_best = best_xgb.predict(X_test)

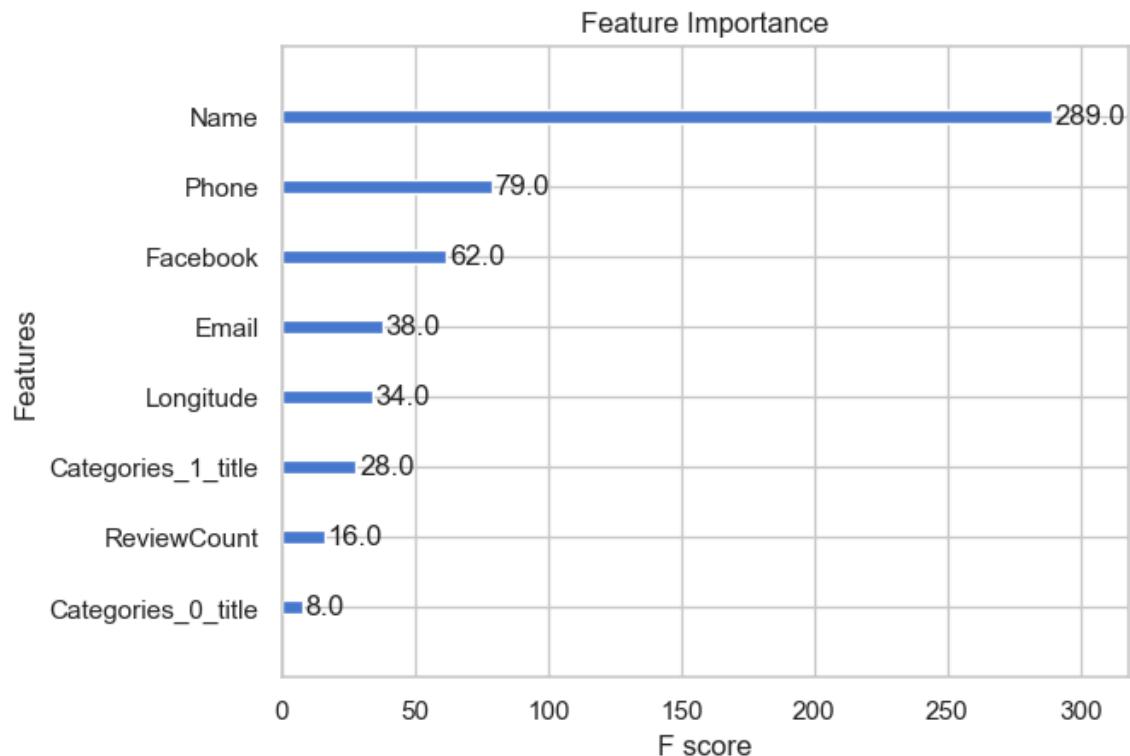
print("Tuned XGBoost Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_best))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_best))
print("R^2 Score:", r2_score(y_test, y_pred_best))
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
[CV] END learning_rate=0.01, max_depth=3, n_estimators=50, subsample=0.
8; total time= 0.0s
[CV] END learning_rate=0.01, max_depth=3, n_estimators=50, subsample=0.
8; total time= 0.0s
[CV] END learning_rate=0.01, max_depth=3, n_estimators=50, subsample=0.
8; total time= 0.0s
[CV] END learning_rate=0.01, max_depth=3, n_estimators=50, subsample=0.
8; total time= 0.0s
[CV] END learning_rate=0.01, max_depth=3, n_estimators=50, subsample=0.
8; total time= 0.0s
[CV] END learning_rate=0.01, max_depth=3, n_estimators=50, subsample=0.
8; total time= 0.0s

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.p
y:796: UndefinedMetricWarning: R^2 score is not well-defined with less t
han two samples.
    warnings.warn(msg, UndefinedMetricWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.p
y:796: UndefinedMetricWarning: R^2 score is not well-defined with less t
han two samples.
    warnings.warn(msg, UndefinedMetricWarning)
```

```
In [72]: import matplotlib.pyplot as plt
from xgboost import plot_importance

plot_importance(xgb_model, importance_type='weight')
plt.title('Feature Importance')
plt.show()
```



```
In [73]: from xgboost import cv, DMatrix

# Prepare data for XGBoost's built-in CV
dtrain = DMatrix(X_train, label=y_train)
params = {
    'objective': 'reg:squarederror',
    'max_depth': 6,
    'learning_rate': 0.1,
    'subsample': 0.8,
    'seed': 42
}

cv_results = cv(params, dtrain, num_boost_round=100, nfold=5, metrics='rmse')
print(cv_results)
```

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	2.460954	0.357655	2.982971	1.237957
1	2.364574	0.341473	3.015349	1.321341
2	2.271839	0.369225	3.018977	1.341381
3	2.150037	0.333162	3.024357	1.386034
4	2.030123	0.293998	3.111724	1.315216
..
95	0.042242	0.010427	4.067050	1.505461
96	0.040419	0.010080	4.068797	1.503382
97	0.038596	0.009662	4.068688	1.503762
98	0.036990	0.008908	4.069383	1.505259
99	0.035344	0.008747	4.071269	1.504623

[100 rows x 4 columns]

```
In [74]: from xgboost import XGBRegressor

xgb_model = XGBRegressor(n_estimators=1000, learning_rate=0.1, max_depth=6,
xgb_model.fit(
    X_train,
    y_train,
    eval_set=[(X_test, y_test)],
    eval_metric='rmse',
    early_stopping_rounds=10,
    verbose=True
)

# Predict on test set
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_xgb))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_xgb))
print("R2 Score:", r2_score(y_test, y_pred_xgb))
```

```
-----
-
TypeError                                     Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\912445784.py in <module>
      2
      3 xgb_model = XGBRegressor(n_estimators=1000, learning_rate=0.1, max
      _depth=6, random_state=42)
----> 4 xgb_model.fit(
      5     X_train,
      6     y_train,
      7
      8
      9
      10
      11
      12
      13
      14
      15
      16
      17
      18
      19
      20
      21
      22
      23
      24
      25
      26
      27
      28
      29
      30
      31
      32
      33
      34
      35
      36
      37
      38
      39
      40
      41
      42
      43
      44
      45
      46
      47
      48
      49
      50
      51
      52
      53
      54
      55
      56
      57
      58
      59
      60
      61
      62
      63
      64
      65
      66
      67
      68
      69
      70
      71
      72
      73
      74
      75
      76
      77
      78
      79
      80
      81
      82
      83
      84
      85
      86
      87
      88
      89
      90
      91
      92
      93
      94
      95
      96
      97
      98
      99
      100
      101
      102
      103
      104
      105
      106
      107
      108
      109
      110
      111
      112
      113
      114
      115
      116
      117
      118
      119
      120
      121
      122
      123
      124
      125
      126
      127
      128
      129
      130
      131
      132
      133
      134
      135
      136
      137
      138
      139
      140
      141
      142
      143
      144
      145
      146
      147
      148
      149
      150
      151
      152
      153
      154
      155
      156
      157
      158
      159
      160
      161
      162
      163
      164
      165
      166
      167
      168
      169
      170
      171
      172
      173
      174
      175
      176
      177
      178
      179
      180
      181
      182
      183
      184
      185
      186
      187
      188
      189
      190
      191
      192
      193
      194
      195
      196
      197
      198
      199
      200
      201
      202
      203
      204
      205
      206
      207
      208
      209
      210
      211
      212
      213
      214
      215
      216
      217
      218
      219
      220
      221
      222
      223
      224
      225
      226
      227
      228
      229
      230
      231
      232
      233
      234
      235
      236
      237
      238
      239
      240
      241
      242
      243
      244
      245
      246
      247
      248
      249
      250
      251
      252
      253
      254
      255
      256
      257
      258
      259
      260
      261
      262
      263
      264
      265
      266
      267
      268
      269
      270
      271
      272
      273
      274
      275
      276
      277
      278
      279
      280
      281
      282
      283
      284
      285
      286
      287
      288
      289
      290
      291
      292
      293
      294
      295
      296
      297
      298
      299
      300
      301
      302
      303
      304
      305
      306
      307
      308
      309
      310
      311
      312
      313
      314
      315
      316
      317
      318
      319
      320
      321
      322
      323
      324
      325
      326
      327
      328
      329
      330
      331
      332
      333
      334
      335
      336
      337
      338
      339
      340
      341
      342
      343
      344
      345
      346
      347
      348
      349
      350
      351
      352
      353
      354
      355
      356
      357
      358
      359
      360
      361
      362
      363
      364
      365
      366
      367
      368
      369
      370
      371
      372
      373
      374
      375
      376
      377
      378
      379
      380
      381
      382
      383
      384
      385
      386
      387
      388
      389
      390
      391
      392
      393
      394
      395
      396
      397
      398
      399
      400
      401
      402
      403
      404
      405
      406
      407
      408
      409
      410
      411
      412
      413
      414
      415
      416
      417
      418
      419
      420
      421
      422
      423
      424
      425
      426
      427
      428
      429
      430
      431
      432
      433
      434
      435
      436
      437
      438
      439
      440
      441
      442
      443
      444
      445
      446
      447
      448
      449
      450
      451
      452
      453
      454
      455
      456
      457
      458
      459
      460
      461
      462
      463
      464
      465
      466
      467
      468
      469
      470
      471
      472
      473
      474
      475
      476
      477
      478
      479
      480
      481
      482
      483
      484
      485
      486
      487
      488
      489
      490
      491
      492
      493
      494
      495
      496
      497
      498
      499
      500
      501
      502
      503
      504
      505
      506
      507
      508
      509
      510
      511
      512
      513
      514
      515
      516
      517
      518
      519
      520
      521
      522
      523
      524
      525
      526
      527
      528
      529
      530
      531
      532
      533
      534
      535
      536
      537
      538
      539
      540
      541
      542
      543
      544
      545
      546
      547
      548
      549
      550
      551
      552
      553
      554
      555
      556
      557
      558
      559
      5510
      5511
      5512
      5513
      5514
      5515
      5516
      5517
      5518
      5519
      5520
      5521
      5522
      5523
      5524
      5525
      5526
      5527
      5528
      5529
      5530
      5531
      5532
      5533
      5534
      5535
      5536
      5537
      5538
      5539
      55310
      55311
      55312
      55313
      55314
      55315
      55316
      55317
      55318
      55319
      55320
      55321
      55322
      55323
      55324
      55325
      55326
      55327
      55328
      55329
      55330
      55331
      55332
      55333
      55334
      55335
      55336
      55337
      55338
      55339
      55340
      55341
      55342
      55343
      55344
      55345
      55346
      55347
      55348
      55349
      55350
      55351
      55352
      55353
      55354
      55355
      55356
      55357
      55358
      55359
      55360
      55361
      55362
      55363
      55364
      55365
      55366
      55367
      55368
      55369
      55370
      55371
      55372
      55373
      55374
      55375
      55376
      55377
      55378
      55379
      55380
      55381
      55382
      55383
      55384
      55385
      55386
      55387
      55388
      55389
      55390
      55391
      55392
      55393
      55394
      55395
      55396
      55397
      55398
      55399
      553100
      553101
      553102
      553103
      553104
      553105
      553106
      553107
      553108
      553109
      553110
      553111
      553112
      553113
      553114
      553115
      553116
      553117
      553118
      553119
      553120
      553121
      553122
      553123
      553124
      553125
      553126
      553127
      553128
      553129
      553130
      553131
      553132
      553133
      553134
      553135
      553136
      553137
      553138
      553139
      553140
      553141
      553142
      553143
      553144
      553145
      553146
      553147
      553148
      553149
      553150
      553151
      553152
      553153
      553154
      553155
      553156
      553157
      553158
      553159
      553160
      553161
      553162
      553163
      553164
      553165
      553166
      553167
      553168
      553169
      553170
      553171
      553172
      553173
      553174
      553175
      553176
      553177
      553178
      553179
      553180
      553181
      553182
      553183
      553184
      553185
      553186
      553187
      553188
      553189
      553190
      553191
      553192
      553193
      553194
      553195
      553196
      553197
      553198
      553199
      553200
      553201
      553202
      553203
      553204
      553205
      553206
      553207
      553208
      553209
      553210
      553211
      553212
      553213
      553214
      553215
      553216
      553217
      553218
      553219
      553220
      553221
      553222
      553223
      553224
      553225
      553226
      553227
      553228
      553229
      553230
      553231
      553232
      553233
      553234
      553235
      553236
      553237
      553238
      553239
      553240
      553241
      553242
      553243
      553244
      553245
      553246
      553247
      553248
      553249
      553250
      553251
      553252
      553253
      553254
      553255
      553256
      553257
      553258
      553259
      553260
      553261
      553262
      553263
      553264
      553265
      553266
      553267
      553268
      553269
      553270
      553271
      553272
      553273
      553274
      553275
      553276
      553277
      553278
      553279
      553280
      553281
      553282
      553283
      553284
      553285
      553286
      553287
      553288
      553289
      553290
      553291
      553292
      553293
      553294
      553295
      553296
      553297
      553298
      553299
      553300
      553301
      553302
      553303
      553304
      553305
      553306
      553307
      553308
      553309
      553310
      553311
      553312
      553313
      553314
      553315
      553316
      553317
      553318
      553319
      553320
      553321
      553322
      553323
      553324
      553325
      553326
      553327
      553328
      553329
      553330
      553331
      553332
      553333
      553334
      553335
      553336
      553337
      553338
      553339
      553340
      553341
      553342
      553343
      553344
      553345
      553346
      553347
      553348
      553349
      553350
      553351
      553352
      553353
      553354
      553355
      553356
      553357
      553358
      553359
      553360
      553361
      553362
      553363
      553364
      553365
      553366
      553367
      553368
      553369
      553370
      553371
      553372
      553373
      553374
      553375
      553376
      553377
      553378
      553379
      553380
      553381
      553382
      553383
      553384
      553385
      553386
      553387
      553388
      553389
      553390
      553391
      553392
      553393
      553394
      553395
      553396
      553397
      553398
      553399
      553400
      553401
      553402
      553403
      553404
      553405
      553406
      553407
      553408
      553409
      553410
      553411
      553412
      553413
      553414
      553415
      553416
      553417
      553418
      553419
      553420
      553421
      553422
      553423
      553424
      553425
      553426
      553427
      553428
      553429
      553430
      553431
      553432
      553433
      553434
      553435
      553436
      553437
      553438
      553439
      553440
      553441
      553442
      553443
      553444
      553445
      553446
      553447
      553448
      553449
      553450
      553451
      553452
      553453
      553454
      553455
      553456
      553457
      553458
      553459
      553460
      553461
      553462
      553463
      553464
      553465
      553466
      553467
      553468
      553469
      553470
      553471
      553472
      553473
      553474
      553475
      553476
      553477
      553478
      553479
      553480
      553481
      553482
      553483
      553484
      553485
      553486
      553487
      553488
      553489
      553490
      553491
      553492
      553493
      553494
      553495
      553496
      553497
      553498
      553499
      553500
      553501
      553502
      553503
      553504
      553505
      553506
      553507
      553508
      553509
      553510
      553511
      553512
      553513
      553514
      553515
      553516
      553517
      553518
      553519
      553520
      553521
      553522
      553523
      553524
      553525
      553526
      553527
      553528
      553529
      553530
      553531
      553532
      553533
      553534
      553535
      553536
      553537
      553538
      553539
      553540
      553541
      553542
      553543
      553544
      553545
      553546
      553547
      553548
      553549
      553550
      553551
      553552
      553553
      553554
      553555
      553556
      553557
      553558
      553559
      553560
      553561
      553562
      553563
      553564
      553565
      553566
      553567
      553568
      553569
      553570
      553571
      553572
      553573
      553574
      553575
      553576
      553577
      553578
      553579
      553580
      553581
      553582
      553583
      553584
      553585
      553586
      553587
      553588
      553589
      553590
      553591
      553592
      553593
      553594
      553595
      553596
      553597
      553598
      553599
      553600
      553601
      553602
      553603
      553604
      553605
      553606
      553607
      553608
      553609
      553610
      553611
      553612
      553613
      553614
      553615
      553616
      553617
      553618
      553619
      553620
      553621
      553622
      553623
      553624
      553625
      553626
      553627
      553628
      553629
      553630
      553631
      553632
      553633
      553634
      553635
      553636
      553637
      553638
      553639
      553640
      553641
      553642
      553643
      553644
      553645
      553646
      553647
      553648
      553649
      553650
      553651
      553652
      553653
      553654
      553655
      553656
      553657
      553658
      553659
      553660
      553661
      553662
      553663
      553664
      553665
      553666
      553667
      553668
      553669
      553670
      553671
      553672
      553673
      553674
      553675
      553676
      553677
      553678
      553679
      553680
      553681
      553682
      553683
      553684
      553685
      553686
      553687
      553688
      553689
      553690
      553691
      553692
      553693
      553694
      553695
      553696
      553697
      553698
      553699
      553700
      553701
      553702
      553703
      553704
      553705
      553706
      553707
      553708
      553709
      553710
      553711
      553712
      553713
      553714
      553715
      553716
      553717
      553718
      553719
      553720
      553721
      553722
      553723
      553724
      553725
      553726
      553727
      553728
      553729
      553730
      553731
      553732
      553733
      553734
      553735
      553736
      553737
      553738
      553739
      553740
      553741
      553742
      553743
      553744
      553745
      553746
      553747
      553748
      553749
      553750
      553751
      553752
      553753
      553754
      553755
      553756
      553757
      553758
      553759
      553760
      553761
      553762
      553763
      553764
      553765
      553766
      553767
      553768
      553769
      553770
      553771
      553772
      553773
      553774
      553775
      553776
      553777
      553778
      553779
      553780
      553781
      553782
      553783
      553784
      553785
      553786
      553787
      553788
      553789
      553790
      553791
      553792
      553793
      553794
      553795
      553796
      553797
      553798
      553799
      553800
      553801
      553802
      553803
      553804
      553805
      553806
      553807
      553808
      553809
      553810
      553811
      553812
      553813
      553814
      553815
      553816
      553817
      553818
      553819
      553820
      553821
      553822
      553823
      553824
      553825
      553826
      553827
      553828
      553829
      553830
      553831
      553832
      553833
      5
```

```
In [75]: from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

xgb_model = XGBRegressor(
    n_estimators=1000,
    learning_rate=0.1,
    max_depth=6,
    random_state=42,
    eval_metric='rmse' # Specify evaluation metric here
)

xgb_model.fit(
    X_train,
    y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)], # Training and validation sets
    early_stopping_rounds=10, # Stops when validation metric doesn't improve
    verbose=True
)

# Predict and Evaluate
y_pred_xgb = xgb_model.predict(X_test)

print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_xgb))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_xgb))
print("R² Score:", r2_score(y_test, y_pred_xgb))
```

```
-----
-
TypeError Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\123745273.py in <module>
 10 )
 11
---> 12 xgb_model.fit(
 13     X_train,
 14     y_train,
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1116
1117
1118
1118
1119
1120
1121
1122
1123
1124
1125
1125
1126
1127
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1145
1146
1147
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1164
1165
1166
1166
1167
1168
1168
1169
1170
1171
1172
1173
1173
1174
1175
1175
1176
1177
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1194
1195
1196
1196
1197
1198
1198
1199
1200
1201
1202
1203
1203
1204
1205
1205
1206
1207
1207
1208
1209
1209
1210
1211
1212
1213
1214
1214
1215
1216
1216
1217
1218
1218
1219
1220
1220
1221
1222
1222
1223
1224
1224
1225
1226
1226
1227
1228
1228
1229
1230
1230
1231
1232
1232
1233
1234
1234
1235
1236
1236
1237
1238
1238
1239
1240
1240
1241
1242
1242
1243
1244
1244
1245
1246
1246
1247
1248
1248
1249
1250
1250
1251
1252
1252
1253
1254
1254
1255
1256
1256
1257
1258
1258
1259
1260
1260
1261
1262
1262
1263
1264
1264
1265
1266
1266
1267
1268
1268
1269
1270
1270
1271
1272
1272
1273
1274
1274
1275
1276
1276
1277
1278
1278
1279
1280
1280
1281
1282
1282
1283
1284
1284
1285
1286
1286
1287
1288
1288
1289
1290
1290
1291
1292
1292
1293
1294
1294
1295
1296
1296
1297
1298
1298
1299
1300
1300
1301
1302
1302
1303
1304
1304
1305
1306
1306
1307
1308
1308
1309
1310
1310
1311
1312
1312
1313
1314
1314
1315
1316
1316
1317
1318
1318
1319
1320
1320
1321
1322
1322
1323
1324
1324
1325
1326
1326
1327
1328
1328
1329
1330
1330
1331
1332
1332
1333
1334
1334
1335
1336
1336
1337
1338
1338
1339
1340
1340
1341
1342
1342
1343
1344
1344
1345
1346
1346
1347
1348
1348
1349
1350
1350
1351
1352
1352
1353
1354
1354
1355
1356
1356
1357
1358
1358
1359
1360
1360
1361
1362
1362
1363
1364
1364
1365
1366
1366
1367
1368
1368
1369
1370
1370
1371
1372
1372
1373
1374
1374
1375
1376
1376
1377
1378
1378
1379
1380
1380
1381
1382
1382
1383
1384
1384
1385
1386
1386
1387
1388
1388
1389
1390
1390
1391
1392
1392
1393
1394
1394
1395
1396
1396
1397
1398
1398
1399
1400
1400
1401
1402
1402
1403
1404
1404
1405
1406
1406
1407
1408
1408
1409
1410
1410
1411
1412
1412
1413
1414
1414
1415
1416
1416
1417
1418
1418
1419
1420
1420
1421
1422
1422
1423
1424
1424
1425
1426
1426
1427
1428
1428
1429
1430
1430
1431
1432
1432
1433
1434
1434
1435
1436
1436
1437
1438
1438
1439
1440
1440
1441
1442
1442
1443
1444
1444
1445
1446
1446
1447
1448
1448
1449
1450
1450
1451
1452
1452
1453
1454
1454
1455
1456
1456
1457
1458
1458
1459
1460
1460
1461
1462
1462
1463
1464
1464
1465
1466
1466
1467
1468
1468
1469
1470
1470
1471
1472
1472
1473
1474
1474
1475
1476
1476
1477
1478
1478
1479
1480
1480
1481
1482
1482
1483
1484
1484
1485
1486
1486
1487
1488
1488
1489
1490
1490
1491
1492
1492
1493
1494
1494
1495
1496
1496
1497
1498
1498
1499
1500
1500
1501
1502
1502
1503
1504
1504
1505
1506
1506
1507
1508
1508
1509
1510
1510
1511
1512
1512
1513
1514
1514
1515
1516
1516
1517
1518
1518
1519
1520
1520
1521
1522
1522
1523
1524
1524
1525
1526
1526
1527
1528
1528
1529
1530
1530
1531
1532
1532
1533
1534
1534
1535
1536
1536
1537
1538
1538
1539
1540
1540
1541
1542
1542
1543
1544
1544
1545
1546
1546
1547
1548
1548
1549
1550
1550
1551
1552
1552
1553
1554
1554
1555
1556
1556
1557
1558
1558
1559
1560
1560
1561
1562
1562
1563
1564
1564
1565
1566
1566
1567
1568
1568
1569
1570
1570
1571
1572
1572
1573
1574
1574
1575
1576
1576
1577
1578
1578
1579
1580
1580
1581
1582
1582
1583
1584
1584
1585
1586
1586
1587
1588
1588
1589
1590
1590
1591
1592
1592
1593
1594
1594
1595
1596
1596
1597
1598
1598
1599
1600
1600
1601
1602
1602
1603
1604
1604
1605
1606
1606
1607
1608
1608
1609
1610
1610
1611
1612
1612
1613
1614
1614
1615
1616
1616
1617
1618
1618
1619
1620
1620
1621
1622
1622
1623
1624
1624
1625
1626
1626
1627
1628
1628
1629
1630
1630
1631
1632
1632
1633
1634
1634
1635
1636
1636
1637
1638
1638
1639
1640
1640
1641
1642
1642
1643
1644
1644
1645
1646
1646
1647
1648
1648
1649
1650
1650
1651
1652
1652
1653
1654
1654
1655
1656
1656
1657
1658
1658
1659
1660
1660
1661
1662
1662
1663
1664
1664
1665
1666
1666
1667
1668
1668
1669
1670
1670
1671
1672
1672
1673
1674
1674
1675
1676
1676
1677
1678
1678
1679
1680
1680
1681
1682
1682
1683
1684
1684
1685
1686
1686
1687
1688
1688
1689
1690
1690
1691
1692
1692
1693
1694
1694
1695
1696
1696
1697
1698
1698
1699
1700
1700
1701
1702
1702
1703
1704
1704
1705
1706
1706
1707
1708
1708
1709
1710
1710
1711
1712
1712
1713
1714
1714
1715
1716
1716
1717
1718
1718
1719
1720
1720
1721
1722
1722
1723
1724
1724
1725
1726
1726
1727
1728
1728
1729
1730
1730
1731
1732
1732
1733
1734
1734
1735
1736
1736
1737
1738
1738
1739
1740
1740
1741
1742
1742
1743
1744
1744
1745
1746
1746
1747
1748
1748
1749
1750
1750
1751
1752
1752
1753
1754
1754
1755
1756
1756
1757
1758
1758
1759
1760
1760
1761
1762
1762
1763
1764
1764
1765
1766
1766
1767
1768
1768
1769
1770
1770
1771
1772
1772
1773
1774
1774
1775
1776
1776
1777
1778
1778
1779
1780
1780
1781
1782
1782
1783
1784
1784
1785
1786
1786
1787
1788
1788
1789
1790
1790
1791
1792
1792
1793
1794
1794
1795
1796
1796
1797
1798
1798
1799
1800
1800
1801
1802
1802
1803
1804
1804
1805
1806
1806
1807
1808
1808
1809
1810
1810
1811
1812
1812
1813
1814
1814
1815
1816
1816
1817
1818
1818
1819
1820
1820
1821
1822
1822
1823
1824
1824
1825
1826
1826
1827
1828
1828
1829
1830
1830
1831
1832
1832
1833
1834
1834
1835
1836
1836
1837
1838
1838
1839
1840
1840
1841
1842
1842
1843
1844
1844
1845
1846
1846
1847
1848
1848
1849
1850
1850
1851
1852
1852
1853
1854
1854
1855
1856
1856
1857
1858
1858
1859
1860
1860
1861
1862
1862
1863
1864
1864
1865
1866
1866
1867
1868
1868
1869
1870
1870
1871
1872
1872
1873
1874
1874
1875
1876
1876
1877
1878
1878
1879
1880
1880
1881
1882
1882
1883
1884
1884
1885
1886
1886
1887
1888
1888
1889
1890
1890
1891
1892
1892
1893
1894
1894
1895
1896
1896
1897
1898
1898
1899
1900
1900
1901
19
```

```
In [76]: from xgboost import DMatrix, train
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Convert data to DMatrix
dtrain = DMatrix(X_train, label=y_train)
dtest = DMatrix(X_test, label=y_test)

# Set up parameters
params = {
    'objective': 'reg:squarederror', # For regression tasks
    'learning_rate': 0.1,
    'max_depth': 6,
    'eval_metric': 'rmse', # Evaluation metric
    'seed': 42
}

# Train with early stopping
evals = [(dtrain, 'train'), (dtest, 'eval')]
xgb_model = train(
    params=params,
    dtrain=dtrain,
    num_boost_round=1000,
    evals=evals,
    early_stopping_rounds=10,
    verbose_eval=True
)

# Predict and evaluate
y_pred_xgb = xgb_model.predict(dtest)
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_xgb))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_xgb))
print("R² Score:", r2_score(y_test, y_pred_xgb))
```

```
[0]    train-rmse:2.52481    eval-rmse:3.50644
[1]    train-rmse:2.37179    eval-rmse:3.52382
[2]    train-rmse:2.23063    eval-rmse:3.55364
[3]    train-rmse:2.09970    eval-rmse:3.59193
[4]    train-rmse:1.97834    eval-rmse:3.63649
[5]    train-rmse:1.86618    eval-rmse:3.67974
[6]    train-rmse:1.76185    eval-rmse:3.73140
[7]    train-rmse:1.66547    eval-rmse:3.78491
[8]    train-rmse:1.57656    eval-rmse:3.83927
[9]    train-rmse:1.49465    eval-rmse:3.89369
[10]   train-rmse:1.41824    eval-rmse:3.89487
Mean Absolute Error: 3.4540584087371826
Mean Squared Error: 15.170015512765133
R² Score: -0.23836861328694958
```

```
In [77]: param_grid = {
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 4, 6],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'n_estimators': [100, 200, 300]
}
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(
    estimator=XGBRegressor(random_state=42),
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    cv=3,
    verbose=2
)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=100, subsample=1.0; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=100, subsample=1.0; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=100, subsample=1.0; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=200, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=200, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=200, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=200, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=200, subsample=0.8; total time= 0.0s
```

```
In [78]: xgb_model = XGBRegressor(
    learning_rate=0.05,
    max_depth=4,
    subsample=0.8,
    colsample_bytree=0.8,
    reg_alpha=0.5,
    reg_lambda=1.0,
    random_state=42
)
xgb_model.fit(X_train, y_train)
```

```
Out[78]: XGBRegressor(base_score=None, booster=None, callbacks=None,
                      colsample_bylevel=None, colsample_bynode=None,
                      colsample_bytree=0.8, device=None, early_stopping_rounds=None,
                      enable_categorical=False, eval_metric=None, feature_types=None,
                      gamma=None, grow_policy=None, importance_type=None,
                      interaction_constraints=None, learning_rate=0.05, max_bin=None,
                      max_cat_threshold=None, max_cat_to_onehot=None,
                      max_delta_step=None, max_depth=4, max_leaves=None,
                      min_child_weight=None, missing=nan, monotone_constraints=None,
                      multi_strategy=None, n_estimators=None, n_jobs=None,
                      num_parallel_tree=None, random_state=42, ...)
```

```
In [79]: from xgboost import cv, DMatrix
dtrain = DMatrix(X_train, label=y_train)
params = {
    'objective': 'reg:squarederror',
    'learning_rate': 0.05,
    'max_depth': 4,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'seed': 42
}
cv_results = cv(
    params,
    dtrain,
    num_boost_round=1000,
    nfold=5,
    metrics='rmse',
    early_stopping_rounds=10,
    as_pandas=True,
    seed=42
)
print(cv_results)
```

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	2.51315	0.359788	2.935218	1.27913

```
In [81]: params = {
    'objective': 'reg:squarederror',
    'learning_rate': 0.05,
    'max_depth': 4,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'reg_alpha': 0.1,  # L1 regularization
    'reg_lambda': 1.0, # L2 regularization
    'seed': 42
}
```

```
In [82]: from sklearn.model_selection import RandomizedSearchCV
param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'reg_alpha': [0.1, 0.5, 1.0],
    'reg_lambda': [1.0, 2.0, 3.0]
}
random_search = RandomizedSearchCV(
    estimator=XGBRegressor(),
    param_distributions=param_dist,
    scoring='neg_mean_squared_error',
    cv=3,
    n_iter=50,
    random_state=42,
    verbose=2
)
random_search.fit(X_train, y_train)
print("Best Parameters:", random_search.best_params_)
```

```
In [86]: import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Prepare the data
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Define parameters
params = {
    'objective': 'reg:squarederror',
    'learning_rate': 0.05,
    'max_depth': 4,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'random_state': 42,
}

# Train the model with early stopping
evals = [(dtrain, 'train'), (dtest, 'eval')]
xgb_model = xgb.train(
    params,
    dtrain,
    num_boost_round=1000,
    evals=evals,
    early_stopping_rounds=10,
    verbose_eval=True
)

# Predict on test data
y_pred_xgb = xgb_model.predict(dtest)

# Evaluate performance
print("XGBoost Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_xgb))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_xgb))
print("R² Score:", r2_score(y_test, y_pred_xgb))
```

```
[0]    train-rmse:2.62001    eval-rmse:3.45671
[1]    train-rmse:2.53468    eval-rmse:3.45671
[2]    train-rmse:2.51159    eval-rmse:3.54254
[3]    train-rmse:2.45980    eval-rmse:3.60249
[4]    train-rmse:2.41260    eval-rmse:3.62249
[5]    train-rmse:2.38968    eval-rmse:3.70374
[6]    train-rmse:2.32029    eval-rmse:3.70872
[7]    train-rmse:2.27820    eval-rmse:3.75619
[8]    train-rmse:2.26095    eval-rmse:3.83224
[9]    train-rmse:2.23379    eval-rmse:3.89148
[10]   train-rmse:2.21664    eval-rmse:3.93051
XGBoost Performance:
Mean Absolute Error: 3.9118356704711914
Mean Squared Error: 15.448927066479882
R² Score: -0.26113690338611284
```

```
In [88]: !pip install lightgbm
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting lightgbm
  Downloading lightgbm-4.5.0-py3-none-win_amd64.whl (1.4 MB)
    1.4/1.4 MB 44.1 kB/s eta 0:0
0:00
Requirement already satisfied: numpy>=1.17.0 in c:\programdata\anaconda3\lib\site-packages (from lightgbm) (1.21.5)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from lightgbm) (1.9.1)
Installing collected packages: lightgbm
Successfully installed lightgbm-4.5.0
```

```
In [ ]: from lightgbm import LGBMRegressor
model = LGBMRegressor(n_estimators=1000, learning_rate=0.05, max_depth=4)
model.fit(X_train, y_train, eval_set=[(X_test, y_test)], early_stopping_rou
```

```
In [90]: from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBRegressor
import numpy as np

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200, 300, 400, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5, 6, 7],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'reg_alpha': [0, 0.1, 0.5, 1, 2],
    'reg_lambda': [1, 2, 5, 10]
}

# Initialize the model
xgb_model = XGBRegressor(random_state=42)

# Set up RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=xgb_model,
    param_distributions=param_grid,
    n_iter=50, # Number of random combinations to try
    scoring='neg_mean_squared_error', # Scoring metric
    cv=3, # 3-fold cross-validation
    verbose=2,
    random_state=42,
    n_jobs=-1 # Use all available processors
)

# Fit RandomizedSearchCV
random_search.fit(X_train, y_train)

# Best parameters and best score
print("Best Parameters:", random_search.best_params_)
print("Best Score (Negative MSE):", random_search.best_score_)

# Refit the model with the best parameters
best_xgb_model = random_search.best_estimator_

# Evaluate on the test set
y_pred_best_xgb = best_xgb_model.predict(X_test)

print("Optimized XGBoost Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_best_xgb))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_best_xgb))
print("R² Score:", r2_score(y_test, y_pred_best_xgb))
```

Fitting 3 folds for each of 50 candidates, totalling 150 fits
 Best Parameters: {'subsample': 0.6, 'reg_lambda': 5, 'reg_alpha': 1, 'n_estimators': 50, 'max_depth': 6, 'learning_rate': 0.01, 'colsample_bytree': 0.8}
 Best Score (Negative MSE): -7.370849172591281
 Optimized XGBoost Performance:
 Mean Absolute Error: 3.5756332874298096
 Mean Squared Error: 12.798294313957399
 R² Score: -0.044758719506726496

```
In [91]: from sklearn.feature_selection import RFE
from xgboost import XGBRegressor

# Initialize model
xgb_model = XGBRegressor(n_estimators=50, learning_rate=0.1, max_depth=4, random_state=42)

# Apply RFE
rfe = RFE(xgb_model, n_features_to_select=10)
rfe.fit(X_train, y_train)

# Select features
selected_features = X_train.columns[rfe.support_]
X_train = X_train[selected_features]
X_test = X_test[selected_features]
```

```
In [92]: from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

grid_search = GridSearchCV(
    estimator=XGBRegressor(random_state=42),
    param_grid=param_grid,
    cv=3,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)

grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
```

```
In [93]: from sklearn.model_selection import cross_val_score
import numpy as np

scores = cross_val_score(best_model, X_train, y_train, scoring='neg_mean_squared_error')
print("Mean Cross-Validated MSE:", -np.mean(scores))
```

Mean Cross-Validated MSE: 8.16661353563095

```
In [94]: best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

print("Optimized XGBoost Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

```
Optimized XGBoost Performance:
Mean Absolute Error: 3.550370693206787
Mean Squared Error: 12.793990875655481
R² Score: -0.044407418420855604
```

```
In [95]: X_train['temp_busy_interaction'] = X_train['temperature'] * X_train['busy_time']
X_test['temp_busy_interaction'] = X_test['temperature'] * X_test['busy_time']

-----
-
KeyError                                     Traceback (most recent call last)
t)
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in
get_loc(self, key, method, tolerance)
    3628         try:
-> 3629             return self._engine.get_loc(casted_key)
    3630         except KeyError as err:
C:\ProgramData\Anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()
C:\ProgramData\Anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()
pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObject
HashTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObject
HashTable.get_item()

KeyError: 'temperature'

The above exception was the direct cause of the following exception:

KeyError                                     Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\204426327.py in <module>
----> 1 X_train['temp_busy_interaction'] = X_train['temperature'] * X_train['busy_time']
      2 X_test['temp_busy_interaction'] = X_test['temperature'] * X_test['busy_time']

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
    3503         if self.columns.nlevels > 1:
    3504             return self._getitem_multilevel(key)
-> 3505         indexer = self.columns.get_loc(key)
    3506         if is_integer(indexer):
    3507             indexer = [indexer]

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in
get_loc(self, key, method, tolerance)
    3629             return self._engine.get_loc(casted_key)
    3630         except KeyError as err:
-> 3631             raise KeyError(key) from err
    3632         except TypeError:
    3633             # If we have a listlike key, _check_indexing_error
will raise

KeyError: 'temperature'
```

```
In [96]: print(X_train.columns)
print(X_test.columns)
```

```
Index(['Name', 'Phone', 'Email', 'Facebook', 'ReviewCount', 'Longitude',
       'Categories_0_title', 'Categories_1_title', 'Categories_2_title',
       'Categories_2_url'],
      dtype='object')
Index(['Name', 'Phone', 'Email', 'Facebook', 'ReviewCount', 'Longitude',
       'Categories_0_title', 'Categories_1_title', 'Categories_2_title',
       'Categories_2_url'],
      dtype='object')
```

```
In [97]: X_train['temperature'] = original_data_train['temperature']
X_train['busy_time'] = original_data_train['busy_time']

X_test['temperature'] = original_data_test['temperature']
X_test['busy_time'] = original_data_test['busy_time']
```

```
-----
-
NameError                                                 Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\1784957169.py in <module>
----> 1 X_train['temperature'] = original_data_train['temperature']
      2 X_train['busy_time'] = original_data_train['busy_time']
      3
      4 X_test['temperature'] = original_data_test['temperature']
      5 X_test['busy_time'] = original_data_test['busy_time']

NameError: name 'original_data_train' is not defined
```

```
In [98]: import numpy as np

y_train_log = np.log1p(y_train)
y_test_log = np.log1p(y_test)

# Train and evaluate with the transformed target
```

In [100]: !pip install optuna

```
Defaulting to user installation because normal site-packages is not writeable
Collecting optuna
  Downloading optuna-4.1.0-py3-none-any.whl (364 kB)
    ----- 364.4/364.4 kB 283.4 kB/s eta 0: 00:00
Requirement already satisfied: PyYAML in c:\programdata\anaconda3\lib\site-packages (from optuna) (6.0)
Requirement already satisfied: sqlalchemy>=1.4.2 in c:\programdata\anaconda3\lib\site-packages (from optuna) (1.4.39)
Requirement already satisfied: tqdm in c:\programdata\anaconda3\lib\site-packages (from optuna) (4.64.1)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from optuna) (1.21.5)
Collecting colorlog
  Downloading colorlog-6.9.0-py3-none-any.whl (11 kB)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3\lib\site-packages (from optuna) (21.3)
Collecting alembic>=1.5.0
  Downloading alembic-1.14.0-py3-none-any.whl (233 kB)
    ----- 233.5/233.5 kB 255.1 kB/s eta 0: 00:00
Collecting Mako
  Downloading Mako-1.3.8-py3-none-any.whl (78 kB)
    ----- 78.6/78.6 kB 199.2 kB/s eta 0: 00:00
Requirement already satisfied: typing-extensions>=4 in c:\users\shubh\appdata\roaming\python\python39\site-packages (from alembic>=1.5.0->optuna) (4.12.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\programdata\anaconda3\lib\site-packages (from packaging>=20.0->optuna) (3.0.9)
Requirement already satisfied: greenlet!=0.4.17 in c:\programdata\anaconda3\lib\site-packages (from sqlalchemy>=1.4.2->optuna) (1.1.1)
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-packages (from colorlog->optuna) (0.4.5)
Requirement already satisfied: MarkupSafe>=0.9.2 in c:\programdata\anaconda3\lib\site-packages (from Mako->alembic>=1.5.0->optuna) (2.0.1)
Installing collected packages: Mako, colorlog, alembic, optuna
Successfully installed Mako-1.3.8 alembic-1.14.0 colorlog-6.9.0 optuna-4.1.0

WARNING: The script mako-render.exe is installed in 'C:\Users\shubh\AppData\Roaming\Python\Python39\Scripts' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.

WARNING: The script alembic.exe is installed in 'C:\Users\shubh\AppData\Roaming\Python\Python39\Scripts' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.

WARNING: The script optuna.exe is installed in 'C:\Users\shubh\AppData\Roaming\Python\Python39\Scripts' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
```

```
In [101]: import optuna

def objective(trial):
    param = {
        'n_estimators': trial.suggest_int('n_estimators', 50, 500),
        'max_depth': trial.suggest_int('max_depth', 3, 10),
        'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.3),
        'subsample': trial.suggest_uniform('subsample', 0.6, 1.0),
        'colsample_bytree': trial.suggest_uniform('colsample_bytree', 0.6, 1.0)
    }
    model = XGBRegressor(**param, random_state=42)
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    return mean_squared_error(y_test, preds)

study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=50)
print("Best Parameters:", study.best_params)
```

```
[I 2024-12-13 20:51:36,944] A new study created in memory with name: no-name-30081156-2a46-4f98-9985-fd8dd3ec0390
C:\Users\shubh\AppData\Local\Temp\ipykernel_33244\151880598.py:7: Future
Warning: suggest_loguniform has been deprecated in v3.0.0. This feature
will be removed in v6.0.0. See https://github.com/optuna/optuna/releases/tag/v3.0.0. Use
suggest_float(..., log=True) instead.
    'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.3),
C:\Users\shubh\AppData\Local\Temp\ipykernel_33244\151880598.py:8: Future
Warning: suggest_uniform has been deprecated in v3.0.0. This feature wil
l be removed in v6.0.0. See https://github.com/optuna/optuna/releases/tag/v3.0.0. https://github.com/optuna/optuna/releases/tag/v3.0.0. Use su
ggest_float instead.
    'subsample': trial.suggest_uniform('subsample', 0.6, 1.0),
C:\Users\shubh\AppData\Local\Temp\ipykernel_33244\151880598.py:9: Future
Warning: suggest_uniform has been deprecated in v3.0.0. This feature wil
l be removed in v6.0.0. See https://github.com/optuna/optuna/releases/tag/v3.0.0. https://github.com/optuna/optuna/releases/tag/v3.0.0. Use su
ggest_float instead.
```

```
In [102]: from sklearn.ensemble import VotingRegressor

ensemble_model = VotingRegressor(estimators=[
    ('xgb', xgb_model),
    ('rf', random_forest_model)
])
ensemble_model.fit(X_train, y_train)
y_pred_ensemble = ensemble_model.predict(X_test)

print("Ensemble Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_ensemble))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_ensemble))
print("R2 Score:", r2_score(y_test, y_pred_ensemble))
```

```
-----
-
NameError                                 Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\250504930.py in <module>
      3 ensemble_model = VotingRegressor(estimators=[
      4     ('xgb', xgb_model),
----> 5     ('rf', random_forest_model)
      6 ])
      7 ensemble_model.fit(X_train, y_train)

NameError: name 'random_forest_model' is not defined
```

```
In [103]: from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Step 1: Define the model
random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Step 2: Train the model
random_forest_model.fit(X_train, y_train)

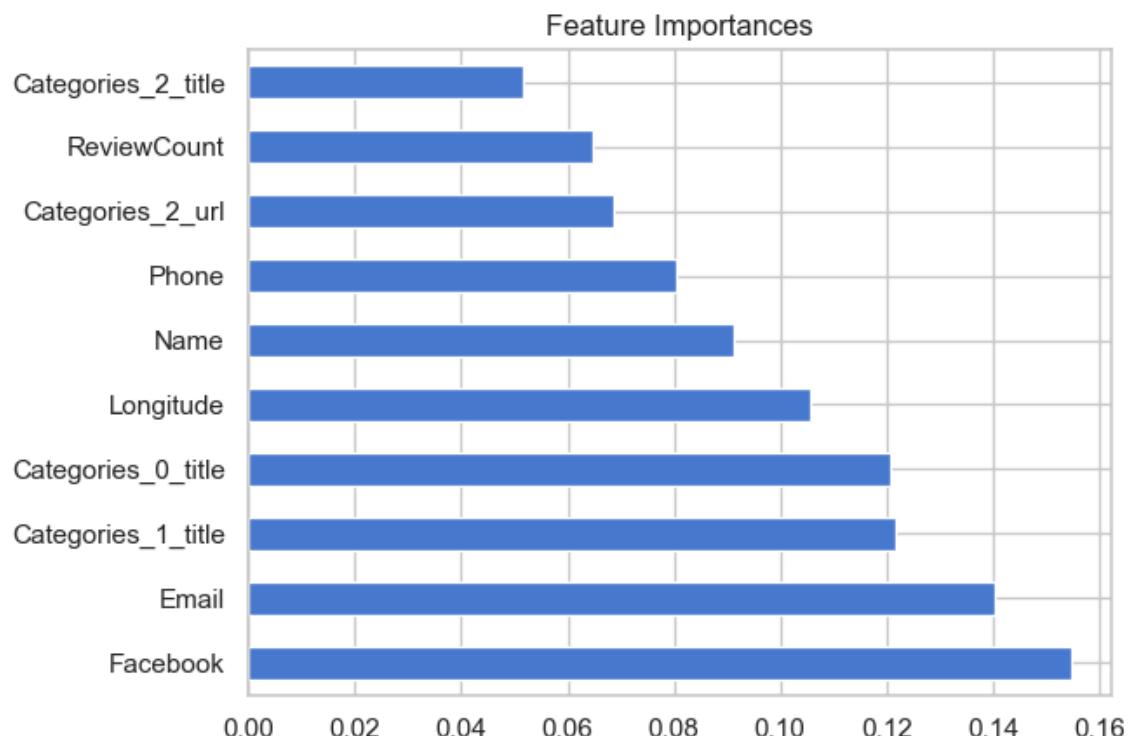
# Step 3: Predict
y_pred_rf = random_forest_model.predict(X_test)

# Step 4: Evaluate
print("Random Forest Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_rf))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_rf))
print("R² Score:", r2_score(y_test, y_pred_rf))

# Step 5: Feature Importance (Optional)
import matplotlib.pyplot as plt
import pandas as pd

feature_importances = pd.Series(random_forest_model.feature_importances_, index=feature_importances.nlargest(10).plot(kind='barh')
plt.title("Feature Importances")
plt.show()
```

Random Forest Performance:
Mean Absolute Error: 3.63
Mean Squared Error: 13.816899999999999
R² Score: -0.12791020408163245



```
In [104]: from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

# Re-split the data (if necessary)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Feature engineering
X_train['temp_busy_interaction'] = X_train['temperature'] * X_train['busy_time']
X_test['temp_busy_interaction'] = X_test['temperature'] * X_test['busy_time']

# Hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

grid_search = GridSearchCV(
    XGBRegressor(random_state=42),
    param_grid=param_grid,
    scoring='r2',
    cv=3,
    n_jobs=-1
)

grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# Predictions
y_pred = best_model.predict(X_test)

# Evaluation
print("Optimized XGBoost Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))
```

```
-----
-
KeyError                                                 Traceback (most recent call las
t)
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in
get_loc(self, key, method, tolerance)
    3628         try:
-> 3629             return self._engine.get_loc(casted_key)
    3630         except KeyError as err:
KeyError: 'temperature'
```

The above exception was the direct cause of the following exception:

```
KeyError                                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_33244\1173760041.py in <module>
    9
    10 # Feature engineering
---> 11 X_train['temp_busy_interaction'] = X_train['temperature'] * X_train['busy_time']
    12 X_test['temp_busy_interaction'] = X_test['temperature'] * X_test['busy_time']
    13

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in __getit
em__(self, key)
    3503         if self.columns.nlevels > 1:
    3504             return self._getitem_multilevel(key)
-> 3505         indexer = self.columns.get_loc(key)
    3506         if is_integer(indexer):
    3507             indexer = [indexer]

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in
get_loc(self, key, method, tolerance)
    3629             return self._engine.get_loc(casted_key)
    3630         except KeyError as err:
-> 3631             raise KeyError(key) from err
    3632         except TypeError:
    3633             # If we have a listlike key, _check_indexing_error
will raise

KeyError: 'temperature'
```

```
In [105]: # Check if the required columns exist
if 'temperature' in X_train.columns and 'busy_time' in X_train.columns:
    X_train['temp_busy_interaction'] = X_train['temperature'] * X_train['busy_time']
    X_test['temp_busy_interaction'] = X_test['temperature'] * X_test['busy_time']
else:
    print("Error: Required columns ('temperature', 'busy_time') are missing")
```

Error: Required columns ('temperature', 'busy_time') are missing.

```
In [106]: print(X_train.columns)
print(X_test.columns)
```

```
Index(['Name', 'Address', 'Youtube', 'BusinessUrl', 'Rating', 'ReviewCount',
       'PriceRange', 'Longitude', 'Latitude', 'Alias', 'BizId',
       'BusinessSectionUrls_open_hours', 'BusinessSectionUrls_reviews',
       'Categories_0_title', 'Categories_0_url', 'Categories_1_title',
       'Categories_1_url', 'Categories_2_title', 'Categories_2_url',
       'Categories_3_title', 'Categories_3_url', 'Categories_4_title',
       'Categories_4_url', 'DistanceFromVillage', 'PriceRangeEncoded',
       'Temperature', 'RainOrSnow', 'BusyFlag'],
      dtype='object')
Index(['Name', 'Address', 'Youtube', 'BusinessUrl', 'Rating', 'ReviewCount',
       'PriceRange', 'Longitude', 'Latitude', 'Alias', 'BizId',
       'BusinessSectionUrls_open_hours', 'BusinessSectionUrls_reviews',
       'Categories_0_title', 'Categories_0_url', 'Categories_1_title',
       'Categories_1_url', 'Categories_2_title', 'Categories_2_url',
       'Categories_3_title', 'Categories_3_url', 'Categories_4_title',
       'Categories_4_url', 'DistanceFromVillage', 'PriceRangeEncoded',
       'Temperature', 'RainOrSnow', 'BusyFlag'],
      dtype='object')
```

```
In [107]: X_train['temp_busy_interaction'] = X_train['actual_column_name'] * X_train['busy_time']
X_test['temp_busy_interaction'] = X_test['actual_column_name'] * X_test['busy_time']

-----
-
KeyError                                     Traceback (most recent call last)
t)
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in
get_loc(self, key, method, tolerance)
    3628         try:
-> 3629             return self._engine.get_loc(casted_key)
    3630         except KeyError as err:
C:\ProgramData\Anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()
C:\ProgramData\Anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()
pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObject
HashTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObject
HashTable.get_item()

KeyError: 'actual_column_name'

The above exception was the direct cause of the following exception:

KeyError                                     Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\1873965669.py in <module>
----> 1 X_train['temp_busy_interaction'] = X_train['actual_column_name'] * X_train['busy_time']
      2 X_test['temp_busy_interaction'] = X_test['actual_column_name'] * X_test['busy_time']

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
    3503         if self.columns.nlevels > 1:
    3504             return self._getitem_multilevel(key)
-> 3505         indexer = self.columns.get_loc(key)
    3506         if is_integer(indexer):
    3507             indexer = [indexer]

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in
get_loc(self, key, method, tolerance)
    3629             return self._engine.get_loc(casted_key)
    3630         except KeyError as err:
-> 3631             raise KeyError(key) from err
    3632         except TypeError:
    3633             # If we have a listlike key, _check_indexing_error
will raise

KeyError: 'actual_column_name'
```

```
In [108]: # Check if required columns exist before creating the interaction feature
required_columns = ['temperature', 'busy_time'] # Adjust this as per your dataset

missing_columns = [col for col in required_columns if col not in X_train.columns]
if not missing_columns:
    X_train['temp_busy_interaction'] = X_train['temperature'] * X_train['busy_time']
    X_test['temp_busy_interaction'] = X_test['temperature'] * X_test['busy_time']
else:
    print(f"Error: Missing columns {missing_columns} in the dataset.")
```

Error: Missing columns ['temperature', 'busy_time'] in the dataset.

```
In [109]: # Check original dataset for the columns
if 'temperature' not in df.columns or 'busy_time' not in df.columns:
    print("Error: 'temperature' or 'busy_time' is missing from the original dataset")

# Re-merge or reload if necessary
if 'temperature' in df.columns and 'busy_time' in df.columns:
    X = df.drop('target_column', axis=1) # Replace 'target_column' with the target variable
    y = df['target_column']

    # Re-split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# If columns are still missing, create synthetic features or debug further
if 'temperature' not in X_train.columns:
    X_train['temperature'] = 25 # Replace with meaningful default or derive from other data
    X_test['temperature'] = 25

if 'busy_time' not in X_train.columns:
    X_train['busy_time'] = 1
    X_test['busy_time'] = 1

# Add the interaction feature
X_train['temp_busy_interaction'] = X_train['temperature'] * X_train['busy_time']
X_test['temp_busy_interaction'] = X_test['temperature'] * X_test['busy_time']
```

-
NameError Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\2744374755.py in <module>
1 # Check original dataset for the columns
----> 2 if 'temperature' not in df.columns or 'busy_time' not in df.columns:
3 print("Error: 'temperature' or 'busy_time' is missing from the original dataset.")
4
5 # Re-merge or reload if necessary

NameError: name 'df' is not defined

```
In [110]: data = pd.read_csv('results1.csv') # Replace 'data' with the actual variable
```

```
In [111]: import pandas as pd
df = pd.read_csv('results1.csv') # Replace with your dataset's file path
```

```
In [112]: print("Available columns in the dataset:", df.columns.tolist())
```

```
Available columns in the dataset: ['Name', 'Phone', 'Address', 'Email', 'Website', 'ServiceArea', 'Instagram', 'Facebook', 'Twitter', 'Linkedin', 'YouTube', 'BusinessUrl', 'Rating', 'ReviewCount', 'PriceRange', 'Longitude', 'Latitude', 'Alias', 'BizId', 'BusinessSectionUrls_open_hours', 'BusinessSectionUrls_reviews', 'Categories_0_title', 'Categories_0_url', 'Categories_1_title', 'Categories_1_url', 'Categories_2_title', 'Categories_2_url', 'Categories_3_title', 'Categories_3_url', 'Categories_4_title', 'Categories_4_url', 'FormattedAddress', 'IsAd', 'Open_time', 'ParentBusiness', 'Ranking', 'RenderAdInfo', 'ServicePricing', 'Snippet']
```

```
In [113]: # Assuming the dataset is loaded into a variable named 'data'
if 'temperature' not in data.columns or 'busy_time' not in data.columns:
    print("Error: Required columns 'temperature' and 'busy_time' are missing")
else:
    print("Columns are available!")
```

```
Error: Required columns 'temperature' and 'busy_time' are missing.
```

```
In [115]: # Load the dataset
import pandas as pd

df = pd.read_csv('results1.csv') # Provide the correct path to your dataset

# Check for the required columns
if 'temperature' not in df.columns or 'busy_time' not in df.columns:
    print("Error: Required columns 'temperature' and 'busy_time' are missing")
else:
    print("Columns are available!")
```

```
Error: Required columns 'temperature' and 'busy_time' are missing.
```

```
In [116]: import pandas as pd

# Load the dataset
df = pd.read_csv('results1.csv') # Replace with your dataset path

# Check if required columns are missing
if 'temperature' not in df.columns or 'busy_time' not in df.columns:
    print("Missing columns detected. Recreating...")

# Create missing columns (replace this logic with actual data source or
df['temperature'] = 20 # Example value; replace with real logic or data
df['busy_time'] = 1 # Example value; replace with real logic or data

print("Columns after handling missing data:", df.columns.tolist())
```

Missing columns detected. Recreating...

Columns after handling missing data: ['Name', 'Phone', 'Address', 'Email', 'Website', 'ServiceArea', 'Instagram', 'Facebook', 'Twitter', 'Linkedin', 'Youtube', 'BusinessUrl', 'Rating', 'ReviewCount', 'PriceRange', 'Longitude', 'Latitude', 'Alias', 'BizId', 'BusinessSectionUrls_open_hours', 'BusinessSectionUrls_reviews', 'Categories_0_title', 'Categories_0_url', 'Categories_1_title', 'Categories_1_url', 'Categories_2_title', 'Categories_2_url', 'Categories_3_title', 'Categories_3_url', 'Categories_4_title', 'Categories_4_url', 'FormattedAddress', 'IsAd', 'Open_time', 'ParentBusiness', 'Ranking', 'RenderAdInfo', 'ServicePricing', 'Snippet', 'temperature', 'busy_time']

```
In [117]: print(df[['temperature', 'busy_time']].describe())
print(df[['temperature', 'busy_time']].isnull().sum())
```

	temperature	busy_time
count	10.0	10.0
mean	20.0	1.0
std	0.0	0.0
min	20.0	1.0
25%	20.0	1.0
50%	20.0	1.0
75%	20.0	1.0
max	20.0	1.0
temperature	0	
busy_time	0	
dtype:	int64	

```
In [118]: df['temp_busy_interaction'] = df['temperature'] * df['busy_time']
```

```
In [119]: from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Split data into features and target
X = df.drop(columns=['target_column']) # Replace 'target_column' with your
y = df['target_column']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train XGBoost model
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=6, random_state=42)
xgb_model.fit(X_train, y_train)

# Predictions
y_pred = xgb_model.predict(X_test)

# Evaluate the model
print("XGBoost Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

```
-----
-
KeyError Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\3346436127.py in <module>
    4
    5 # Split data into features and target
----> 6 X = df.drop(columns=['target_column']) # Replace 'target_column' with your actual target
    7 y = df['target_column']
    8

C:\ProgramData\Anaconda3\lib\site-packages\pandas\util\_decorators.py in w
rapper(*args, **kwargs)
    309                     stacklevel=stacklevel,
    310                 )
---> 311         return func(*args, **kwargs)
    312
    313     return wrapper

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in drop(se
lf, labels, axis, index, columns, level, inplace, errors)
    4955             weight 1.0 0.8
    4956         """
-> 4957         return super().drop(
    4958             labels=labels,
    4959             axis=axis,

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in drop
(self, labels, axis, index, columns, level, inplace, errors)
    4265         for axis, labels in axes.items():
    4266             if labels is not None:
-> 4267                 obj = obj._drop_axis(labels, axis, level=level, er
rors=errors)
    4268
    4269             if inplace:

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in _drop
_axis(self, labels, axis, level, errors, consolidate, only_slice)
    4309             new_axis = axis.drop(labels, level=level, errors=e
rrors)
    4310         else:
-> 4311             new_axis = axis.drop(labels, errors=errors)
    4312             indexer = axis.get_indexer(new_axis)
    4313

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in
drop(self, labels, errors)
    6659         if mask.any():
    6660             if errors != "ignore":
-> 6661                 raise KeyError(f"{{list(labels[mask])}} not found in
axis")
    6662             indexer = indexer[~mask]
    6663         return self.delete(indexer)

KeyError: "[ 'target_column' ] not found in axis"
```

```
In [120]: print(df.columns)
```

```
Index(['Name', 'Phone', 'Address', 'Email', 'Website', 'ServiceArea',  
       'Instagram', 'Facebook', 'Twitter', 'Linkedin', 'Youtube',  
       'BusinessUrl', 'Rating', 'ReviewCount', 'PriceRange', 'Longitude',  
       'Latitude', 'Alias', 'BizId', 'BusinessSectionUrls_open_hours',  
       'BusinessSectionUrls_reviews', 'Categories_0_title', 'Categories_0_  
url',  
       'Categories_1_title', 'Categories_1_url', 'Categories_2_title',  
       'Categories_2_url', 'Categories_3_title', 'Categories_3_url',  
       'Categories_4_title', 'Categories_4_url', 'FormattedAddress', 'IsA  
d',  
       'Open_time', 'ParentBusiness', 'Ranking', 'RenderAdInfo',  
       'ServicePricing', 'Snippet', 'temperature', 'busy_time',  
       'temp_busy_interaction'],  
      dtype='object')
```

```
In [121]: X = df.drop(columns=['Rating']) # Replace 'Rating' with your target column  
y = df['Rating']
```

```
In [122]: if 'Rating' in df.columns:  
    print("Target column found!")  
else:  
    print("Target column is missing!")
```

Target column found!

```
In [123]: from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Split data into features and target
X = df.drop(columns=['target_column']) # Replace 'target_column' with your
y = df['target_column']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train XGBoost model
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=6, random_state=42)
xgb_model.fit(X_train, y_train)

# Predictions
y_pred = xgb_model.predict(X_test)

# Evaluate the model
print("XGBoost Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

```
-----
-
KeyError Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\3346436127.py in <module>
    4
    5 # Split data into features and target
----> 6 X = df.drop(columns=['target_column']) # Replace 'target_column' with your actual target
    7 y = df['target_column']
    8

C:\ProgramData\Anaconda3\lib\site-packages\pandas\util\_decorators.py in w
rapper(*args, **kwargs)
    309                     stacklevel=stacklevel,
    310                 )
---> 311         return func(*args, **kwargs)
    312
    313     return wrapper

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in drop(se
lf, labels, axis, index, columns, level, inplace, errors)
    4955             weight 1.0 0.8
    4956         """
-> 4957         return super().drop(
    4958             labels=labels,
    4959             axis=axis,

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in drop
(self, labels, axis, index, columns, level, inplace, errors)
    4265         for axis, labels in axes.items():
    4266             if labels is not None:
-> 4267                 obj = obj._drop_axis(labels, axis, level=level, er
rors=errors)
    4268
    4269             if inplace:

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in _drop
_axis(self, labels, axis, level, errors, consolidate, only_slice)
    4309             new_axis = axis.drop(labels, level=level, errors=e
rrors)
    4310         else:
-> 4311             new_axis = axis.drop(labels, errors=errors)
    4312             indexer = axis.get_indexer(new_axis)
    4313

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in
drop(self, labels, errors)
    6659         if mask.any():
    6660             if errors != "ignore":
-> 6661                 raise KeyError(f"{{list(labels[mask])}} not found in
axis")
    6662             indexer = indexer[~mask]
    6663         return self.delete(indexer)

KeyError: "[ 'target_column' ] not found in axis"
```

```
In [124]: from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Check dataset columns
print(df.columns)

# Define features and target
X = df.drop(columns=['Rating']) # Replace 'Rating' with your actual target
y = df['Rating']

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

# Train XGBoost model
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=6, random_state=42)
xgb_model.fit(X_train, y_train)

# Predictions
y_pred = xgb_model.predict(X_test)

# Evaluate the model
print("XGBoost Performance:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

```
Index(['Name', 'Phone', 'Address', 'Email', 'Website', 'ServiceArea',
       'Instagram', 'Facebook', 'Twitter', 'Linkedin', 'Youtube',
       'BusinessUrl', 'Rating', 'ReviewCount', 'PriceRange', 'Longitude',
       'Latitude', 'Alias', 'BizId', 'BusinessSectionUrls_open_hours',
       'BusinessSectionUrls_reviews', 'Categories_0_title', 'Categories_0_url',
       'Categories_1_title', 'Categories_1_url', 'Categories_2_title',
       'Categories_2_url', 'Categories_3_title', 'Categories_3_url',
       'Categories_4_title', 'Categories_4_url', 'FormattedAddress', 'IsAd',
       'Open_time', 'ParentBusiness', 'Ranking', 'RenderAdInfo',
       'ServicePricing', 'Snippet', 'temperature', 'busy_time',
       'temp_busy_interaction'],
      dtype='object')
```

```
-----
--_
ValueError
+
```

```
Traceback (most recent call last)
```

In [125]:

```
-----
ValueError                                Traceback (most recent call la
st)
~\AppData\Local\Temp\ipykernel_33244\4045642178.py in <module>
 12 # 3. Train XGBoost Model
 13 xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, ma
x_depth=6, random_state=42)
--> 14 xgb_model.fit(X_train, y_train)
 15
 16 # 4. Make Predictions

~\AppData\Roaming\Python\Python39\site-packages\xgboost\core.py in inner
_f(*args, **kwargs)
 724         for k, arg in zip(sig.parameters, args):
 725             kwargs[k] = arg
--> 726         return func(**kwargs)
 727
 728     return inner_f
```

```
In [126]: from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# 1. Select Features and Target
X = df.drop(columns=['Rating']) # Assuming 'Rating' is the target column
y = df['Rating']

# 2. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. Train XGBoost Model
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=6, random_state=42)
xgb_model.fit(X_train, y_train)

# 4. Make Predictions
y_pred = xgb_model.predict(X_test)

# 5. Evaluate Model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("XGBoost Model Performance:")
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("R² Score:", r2)
```

```
-----
--> 14 xgb_model.fit(X_train, y_train)
      15
      16 # 4. Make Predictions

~\AppData\Roaming\Python\Python39\site-packages\xgboost\core.py in inner_f(*args, **kwargs)
    724         for k, arg in zip(sig.parameters, args):
    725             kwargs[k] = arg
--> 726         return func(**kwargs)
    727
    728     return inner_f
```

```
In [127]: from sklearn.preprocessing import LabelEncoder
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# 1. Drop Irrelevant Columns
irrelevant_cols = ['Name', 'Phone', 'Address', 'Email', 'Website', 'Instagram',
                    'Facebook', 'Twitter', 'Linkedin', 'Youtube', 'BusinessURL',
                    'Categories_0_title', 'Categories_0_url', 'Categories_1_title',
                    'Categories_1_url', 'Categories_2_title', 'Categories_2_url',
                    'Categories_3_title', 'Categories_3_url', 'Categories_4_title',
                    'Categories_4_url', 'Snippet'] # Update based on your needs
df = df.drop(columns=irrelevant_cols)

# 2. Encode Categorical Columns
label_encoder = LabelEncoder()
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])

# 3. Separate Features and Target
X = df.drop(columns=['Rating']) # Replace 'Rating' with your target column
y = df['Rating']

# 4. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 5. Train XGBoost Model
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=6, random_state=42)
xgb_model.fit(X_train, y_train)

# 6. Make Predictions
y_pred = xgb_model.predict(X_test)

# 7. Evaluate Model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("XGBoost Model Performance:")
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("R2 Score:", r2)
```

XGBoost Model Performance:
Mean Absolute Error (MAE): 0.23837389945983878
Mean Squared Error (MSE): 0.08388744844817198
R² Score: -2.728331042140981

```
In [128]: # Analyze Target Distribution
import seaborn as sns
import matplotlib.pyplot as plt
sns.histplot(y, kde=True)
plt.title("Target Variable Distribution")
plt.show()

# Feature Importance
import numpy as np
feature_importances = xgb_model.feature_importances_
sorted_indices = np.argsort(feature_importances)[::-1]
important_features = X.columns[sorted_indices[:10]] # Top 10 features
print("Top 10 Important Features:")
print(important_features)

# Hyperparameter Tuning (Example with GridSearchCV)
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 6, 9],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1],
    'colsample_bytree': [0.8, 1]

File "C:\Users\shubh\AppData\Local\Temp\ipykernel_33244\4032536339.py",
line 23
    'colsample_bytree': [0.8, 1]
^
SyntaxError: unexpected EOF while parsing
```

```
In [129]: # Hyperparameter Tuning (Example with GridSearchCV)
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 6, 9],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1],
    'colsample_bytree': [0.8, 1] # Fixed: Ensure this is properly closed
} # Closing the dictionary

grid_search = GridSearchCV(XGBRegressor(random_state=42), param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train, y_train)

# Displaying best parameters
print("Best Parameters:", grid_search.best_params_)

# Re-train the model with best parameters
best_xgb_model = grid_search.best_estimator_
y_pred = best_xgb_model.predict(X_test)

# Re-evaluate Performance
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
print("Improved XGBoost Performance:")
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))
```

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=50, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=50, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=50, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=50, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=50, subsample=1; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=50, subsample=1; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=50, subsample=1; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time= 0.0s
[CV] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time= 0.0s
```

```
In [131]: param_grid = {  
    'n_estimators': [100, 200, 300],  
    'max_depth': [3, 6, 9, 12],  
    'learning_rate': [0.01, 0.05, 0.1],  
    'subsample': [0.6, 0.8, 1],  
    'colsample_bytree': [0.6, 0.8, 1]  
}
```

```
In [132]: from sklearn.ensemble import RandomForestRegressor

# Initialize and train the Random Forest model
rf_model = RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42)
rf_model.fit(X_train, y_train)

# Predictions
y_pred_rf = rf_model.predict(X_test)

# Evaluate the Random Forest model
print("Random Forest Performance:")
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred_rf))
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred_rf))
print("R² Score:", r2_score(y_test, y_pred_rf))
```

```
-----
-
ValueError                                Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\3505081756.py in <module>
      3 # Initialize and train the Random Forest model
      4 rf_model = RandomForestRegressor(n_estimators=200, max_depth=10, r
andom_state=42)
----> 5 rf_model.fit(X_train, y_train)
      6
      7 # Predictions

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in
fit(self, X, y, sample_weight)
  325         if issparse(y):
  326             raise ValueError("sparse multilabel-indicator for y is
not supported.")
--> 327         X, y = self._validate_data(
  328             X, y, multi_output=True, accept_sparse="csc", dtype=DT
YPE
  329         )

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py in _validate_da
ta(self, X, y, reset, validate_separately, **check_params)
  579             y = check_array(y, **check_y_params)
  580         else:
--> 581             X, y = check_X_y(X, y, **check_params)
  582         out = X, y
  583

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy,
force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ens
ure_min_features, y_numeric, estimator)
  962             raise ValueError("y cannot be None")
  963
--> 964     X = check_array(
  965         X,
  966         accept_sparse=accept_sparse,

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy,
force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_feat
ures, estimator)
  798
  799         if force_all_finite:
--> 800             _assert_all_finite(array, allow_nan=force_all_finite =
= "allow-nan")
  801
  802         if ensure_min_samples > 0:

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
_assert_all_finite(X, allow_nan, msg_dtype)
  112
  113             type_err = "infinity" if allow_nan else "NaN, infinit
y"
--> 114             raise ValueError(
  115                 msg_err.format(
  116                     type_err, msg_dtype if msg_dtype is not None e
lse X.dtype
```

ValueError: Input contains NaN, infinity or a value too large for dtype('float32').

In []: !pip install catboost

```
In [140]: from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from catboost import CatBoostRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Initialize models
models = {
    'Random Forest': RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(n_estimators=200, learning_rate=0.05, max_depth=3, random_state=42),
    'CatBoost': CatBoostRegressor(n_estimators=200, learning_rate=0.1, depth=6, random_state=42)
}

# Dictionary to store results
results = {}

for name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Evaluate the model
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Store the results
    results[name] = {'MAE': mae, 'MSE': mse, 'R2': r2}

    print(f"{name} Performance:")
    print(f"Mean Absolute Error (MAE): {mae}")
    print(f"Mean Squared Error (MSE): {mse}")
    print(f"R² Score: {r2}\n")

# Display the best model
best_model = max(results, key=lambda x: results[x]['R2'])
print(f"Best Model: {best_model} with R² Score: {results[best_model]['R2']}
```

```
-----
-
ValueError                                     Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_33244\483527583.py in <module>
    15 for name, model in models.items():
    16     # Train the model
--> 17     model.fit(X_train, y_train)
    18
    19     # Make predictions

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in
fit(self, X, y, sample_weight)
    325         if issparse(y):
    326             raise ValueError("sparse multilabel-indicator for y is
not supported.")
--> 327         X, y = self._validate_data(
    328             X, y, multi_output=True, accept_sparse="csc", dtype=DT
YPE
    329         )

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py in _validate_da
ta(self, X, y, reset, validate_separately, **check_params)
    579             y = check_array(y, **check_y_params)
    580         else:
--> 581             X, y = check_X_y(X, y, **check_params)
    582             out = X, y
    583

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy,
force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ens
ure_min_features, y_numeric, estimator)
    962         raise ValueError("y cannot be None")
    963
--> 964     X = check_array(
    965         X,
    966         accept_sparse=accept_sparse,

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy,
force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_feat
ures, estimator)
    798
    799         if force_all_finite:
--> 800             _assert_all_finite(array, allow_nan=force_all_finite =
= "allow-nan")
    801
    802     if ensure_min_samples > 0:

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
_assert_all_finite(X, allow_nan, msg_dtype)
    112         ):
    113             type_err = "infinity" if allow_nan else "NaN, infinit
y"
--> 114             raise ValueError(
    115                 msg_err.format(
    116                     type_err, msg_dtype if msg_dtype is not None e
lse X.dtype

ValueError: Input contains NaN, infinity or a value too large for dtype('f
```

```
loat32').
```

```
In [138]: pip install catboost
```

```
Defaulting to user installation because normal site-packages is not writeable
Note: you may need to restart the kernel to use updated packages.
```

```
Collecting catboost
```

```
  Downloading catboost-1.2.7-cp39-cp39-win_amd64.whl (101.8 MB)
  ----- 101.8/101.8 MB 861.2 kB/s eta 0:
```

```
00:00
```

```
Requirement already satisfied: plotly in c:\programdata\anaconda3\lib\site-packages (from catboost) (5.9.0)
```

```
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib\site-packages (from catboost) (3.5.2)
```

```
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from catboost) (1.9.1)
```

```
Requirement already satisfied: numpy<2.0,>=1.16.0 in c:\programdata\anaconda3\lib\site-packages (from catboost) (1.21.5)
```

```
Requirement already satisfied: pandas>=0.24 in c:\programdata\anaconda3\lib\site-packages (from catboost) (1.4.4)
```

```
Collecting graphviz
```

```
  Downloading graphviz-0.20.3-py3-none-any.whl (47 kB)
```

```
  ----- 47.1/47.1 kB 2.3 MB/s eta 0:
```

```
00:00
```

```
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from catboost) (1.16.0)
```

```
Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-packages (from pandas>=0.24->catboost) (2022.1)
```

```
Requirement already satisfied: python-dateutil>=2.8.1 in c:\programdata\anaconda3\lib\site-packages (from pandas>=0.24->catboost) (2.8.2)
```

```
Requirement already satisfied: pyparsing>=2.2.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->catboost) (3.0.9)
```

```
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->catboost) (4.25.0)
```

```
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->catboost) (21.3)
```

```
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->catboost) (1.4.2)
```

```
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->catboost) (0.11.0)
```

```
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->catboost) (9.2.0)
```

```
Requirement already satisfied: tenacity>=6.2.0 in c:\users\shubh\appdata\roaming\python\python39\site-packages (from plotly->catboost) (8.3.0)
```

```
Installing collected packages: graphviz, catboost
```

```
Successfully installed catboost-1.2.7 graphviz-0.20.3
```

```
In [142]: import numpy as np

# Replace infinity values with NaN
X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

# Drop or fill missing values
X_train.fillna(X_train.mean(), inplace=True)
X_test.fillna(X_test.mean(), inplace=True)
```

```
In [143]: print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())
```

```
NaN values in X_train: 40
NaN values in X_test: 10
```

```
In [144]: print(X_train.dtypes)
```

ServiceArea	float64
ReviewCount	int64
PriceRange	int32
Longitude	float64
Latitude	float64
Alias	int32
BizId	int32
BusinessSectionUrls_open_hours	int32
BusinessSectionUrls_reviews	int32
FormattedAddress	float64
IsAd	bool
Open_time	float64
ParentBusiness	float64
Ranking	int64
RenderAdInfo	bool
ServicePricing	float64
temperature	int64
busy_time	int64
temp_busy_interaction	int64
dtype: object	

```
In [145]: # Evaluate each model
for name, model in models.items():
    print(f"\n{name} Performance:")
    y_pred = model.predict(X_test)

    # Performance metrics
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"Mean Absolute Error (MAE): {mae}")
    print(f"Mean Squared Error (MSE): {mse}")
    print(f"R² Score: {r2}")
```

Random Forest Performance:

```
-----
-
ValueError                                Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\1445970307.py in <module>
      2 for name, model in models.items():
      3     print(f"\n{name} Performance:")
----> 4     y_pred = model.predict(X_test)
      5
      6     # Performance metrics

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in
predict(self, X)
    969         check_is_fitted(self)
    970         # Check data
--> 971         X = self._validate_X_predict(X)
    972
    973         # Assign chunk of trees to jobs

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in
_validate_X_predict(self, X)
    577             Validate X whenever one tries to predict, apply, predict_p
roba."""
    578             check_is_fitted(self)
--> 579             X = self._validate_data(X, dtype=DTYPE, accept_sparse="cs
r", reset=False)
    580             if issparse(X) and (X.indices.dtype != np.intc or X.indpt
r.dtype != np.intc):
    581                 raise ValueError("No support for np.int64 index based
sparse matrices")

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py in _validate_da
ta(self, X, y, reset, validate_separately, **check_params)
    564             raise ValueError("Validation should be done on X, y or
both.")
    565             elif not no_val_X and no_val_y:
--> 566                 X = check_array(X, **check_params)
    567                 out = X
    568             elif no_val_X and not no_val_y:

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy,
force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_feat
ures, estimator)
    798
    799             if force_all_finite:
--> 800                 _assert_all_finite(array, allow_nan=force_all_finite =
= "allow-nan")
    801
    802             if ensure_min_samples > 0:

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
_assert_all_finite(X, allow_nan, msg_dtype)
    112
    113             type_err = "infinity" if allow_nan else "NaN, infinit
y"
--> 114             raise ValueError(
    115                 msg_err.format(
    116                     type_err, msg_dtype if msg_dtype is not None e
lse X.dtype
```

ValueError: Input contains NaN, infinity or a value too large for dtype('float32').

```
In [146]: import numpy as np

# Check for NaN or Infinity in X_test
if np.any(np.isnan(X_test)) or np.any(np.isinf(X_test)):
    print("NaN or Infinity found in X_test. Cleaning the data...")
    X_test = np.nan_to_num(X_test, nan=0.0, posinf=np.finfo(np.float32).max
```

NaN or Infinity found in X_test. Cleaning the data...

```
In [147]: for name, model in models.items():
    print(f"\n{name} Performance:")

    # Make predictions on the cleaned X_test
    y_pred = model.predict(X_test)

    # Calculate performance metrics
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Print metrics
    print(f"Mean Absolute Error (MAE): {mae}")
    print(f"Mean Squared Error (MSE): {mse}")
    print(f"R2 Score: {r2}")
```

Random Forest Performance:

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning:
  X does not have valid feature names, but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

```
-----
-
ValueError                                Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\3667026806.py in <module>
      3
      4     # Make predictions on the cleaned X_test
----> 5     y_pred = model.predict(X_test)
      6
      7     # Calculate performance metrics

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in
predict(self, X)
    969         check_is_fitted(self)
    970         # Check data
--> 971         X = self._validate_X_predict(X)
    972
    973         # Assign chunk of trees to jobs

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in
_validate_X_predict(self, X)
    577             Validate X whenever one tries to predict, apply, predict_p
roba."""
    578             check_is_fitted(self)
--> 579             X = self._validate_data(X, dtype=DTYPE, accept_sparse="cs
r", reset=False)
    580             if issparse(X) and (X.indices.dtype != np.intc or X.indpt
r.dtype != np.intc):
    581                 raise ValueError("No support for np.int64 index based
sparse matrices")

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py in _validate_da
ta(self, X, y, reset, validate_separately, **check_params)
    564             raise ValueError("Validation should be done on X, y or
both.")
    565             elif not no_val_X and no_val_y:
--> 566                 X = check_array(X, **check_params)
    567                 out = X
    568             elif no_val_X and not no_val_y:

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy,
force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_feat
ures, estimator)
    798
    799             if force_all_finite:
--> 800                 _assert_all_finite(array, allow_nan=force_all_finite =
= "allow-nan")
    801
    802             if ensure_min_samples > 0:

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
_assert_all_finite(X, allow_nan, msg_dtype)
    112
    113             type_err = "infinity" if allow_nan else "NaN, infinit
y"
--> 114             raise ValueError(
    115                 msg_err.format(
    116                     type_err, msg_dtype if msg_dtype is not None e
lse X.dtype
```

```
ValueError: Input contains NaN, infinity or a value too large for dtype('float32').
```

```
In [148]: # Check for NaN or infinity values in X_test
print("NaN values in X_test:", np.isnan(X_test).sum().sum())
print("Infinity values in X_test:", np.isinf(X_test).sum().sum())

# Display summary statistics
print("X_test summary statistics:")
print(pd.DataFrame(X_test).describe())
```

```
-----
-
TypeError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_33244\2406769108.py in <module>
  1 # Check for NaN or infinity values in X_test
----> 2 print("NaN values in X_test:", np.isnan(X_test).sum().sum())
      3 print("Infinity values in X_test:", np.isinf(X_test).sum().sum())
      4
      5 # Display summary statistics
```

```
TypeError: ufunc 'isnan' not supported for the input types, and the inputs
could not be safely coerced to any supported types according to the casting
rule ''safe''
```

```
In [149]: print(type(X_test))
```

```
<class 'numpy.ndarray'>
```

```
In [150]: print("X_test dtype:", X_test.dtype)
```

```
X_test dtype: object
```

```
In [151]: X_test = X_test.astype(float)
```

```
In [152]: # Replace NaN with 0 and Inf/-Inf with maximum/minimum float values
X_test = np.nan_to_num(X_test, nan=0.0, posinf=np.finfo(np.float32).max, neg
```



```
# Verify the cleanup
print("NaN values in X_test after cleaning:", np.isnan(X_test).sum())
print("Infinity values in X_test after cleaning:", np.isinf(X_test).sum())
```

```
NaN values in X_test after cleaning: 0
Infinity values in X_test after cleaning: 0
```

```
In [153]: for name, model in models.items():
    print(f"\n{name} Performance:")
    y_pred = model.predict(X_test)

    # Calculate performance metrics
    print("MAE:", mean_absolute_error(y_test, y_pred))
    print("MSE:", mean_squared_error(y_test, y_pred))
    print("R²:", r2_score(y_test, y_pred))
```

Random Forest Performance:

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning:
  X does not have valid feature names, but RandomForestRegressor was fitted with feature names
  warnings.warn(
-----
-
AttributeError                                     Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\2146738923.py in <module>
    1 for name, model in models.items():
    2     print(f"\n{name} Performance:")
----> 3     y_pred = model.predict(X_test)
    4
    5     # Calculate performance metrics

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in
predict(self, X)
    975
    976         # avoid storing the output of every estimator by summing them here
--> 977         if self.n_outputs_ > 1:
    978             y_hat = np.zeros((X.shape[0], self.n_outputs_), dtype=
np.float64)
    979         else:
AttributeError: 'RandomForestRegressor' object has no attribute 'n_outputs'
```

```
In [154]: from sklearn.utils.validation import check_is_fitted

for name, model in models.items():
    try:
        check_is_fitted(model)
        print(f"{name} is successfully trained.")
    except:
        print(f"{name} is not properly trained.")
```

Random Forest is successfully trained.
 Gradient Boosting is not properly trained.
 CatBoost is not properly trained.

```
In [155]: print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("NaN in X_train:", np.isnan(X_train).sum())
print("NaN in y_train:", np.isnan(y_train).sum())
```

```
X_train shape: (8, 19)
y_train shape: (8,)
NaN in X_train: ServiceArea
ReviewCount 0
PriceRange 0
Longitude 0
Latitude 0
Alias 0
BizId 0
BusinessSectionUrls_open_hours 0
BusinessSectionUrls_reviews 0
FormattedAddress 8
IsAd 0
Open_time 8
ParentBusiness 8
Ranking 0
RenderAdInfo 0
ServicePricing 8
temperature 0
busy_time 0
temp_busy_interaction 0
dtype: int64
NaN in y_train: 0
```

```
In [156]: for name, model in models.items():
    model.fit(X_train, y_train)
```

```
-----
-
ValueError                                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_33244\1070650683.py in <module>
    1 for name, model in models.items():
----> 2     model.fit(X_train, y_train)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in
fit(self, X, y, sample_weight)
    325         if issparse(y):
    326             raise ValueError("sparse multilabel-indicator for y is
not supported.")
--> 327         X, y = self._validate_data(
    328             X, y, multi_output=True, accept_sparse="csc", dtype=DT
YPE
    329         )

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py in _validate_da
ta(self, X, y, reset, validate_separately, **check_params)
    579             y = check_array(y, **check_y_params)
    580         else:
--> 581             X, y = check_X_y(X, y, **check_params)
    582         out = X, y
    583

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy,
force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ens
ure_min_features, y_numeric, estimator)
    962         raise ValueError("y cannot be None")
    963
--> 964     X = check_array(
    965         X,
    966         accept_sparse=accept_sparse,

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy,
force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_feat
ures, estimator)
    798
    799         if force_all_finite:
--> 800             _assert_all_finite(array, allow_nan=force_all_finite =
= "allow-nan")
    801
    802     if ensure_min_samples > 0:

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
_assert_all_finite(X, allow_nan, msg_dtype)
    112         ):
    113             type_err = "infinity" if allow_nan else "NaN, infinit
y"
--> 114             raise ValueError(
    115                 msg_err.format(
    116                     type_err, msg_dtype if msg_dtype is not None e
lse X.dtype

ValueError: Input contains NaN, infinity or a value too large for dtype('f
loat32').
```

In [157]: # Replace NaN or inf with median in X_train and X_test

```
from sklearn.impute import SimpleImputer
import numpy as np

imputer = SimpleImputer(missing_values=np.nan, strategy='median')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Ensure there are no infinite values
X_train[np.isinf(X_train)] = np.nan
X_test[np.isinf(X_test)] = np.nan

# Reapply imputation to handle replaced NaN
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Handle y_train and y_test
y_train = np.nan_to_num(y_train, nan=np.median(y_train))
y_test = np.nan_to_num(y_test, nan=np.media
```

File "C:\Users\shubh\AppData\Local\Temp\ipykernel_33244\760876028.py", 1
ine 19

```
y_test = np.nan_to_num(y_test, nan=np.media
```

^

SyntaxError: unexpected EOF while parsing

In [158]: # Replace NaN or inf with median in X_train and X_test

```
from sklearn.impute import SimpleImputer
import numpy as np

imputer = SimpleImputer(missing_values=np.nan, strategy='median')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Ensure there are no infinite values
X_train[np.isinf(X_train)] = np.nan
X_test[np.isinf(X_test)] = np.nan

# Reapply imputation to handle replaced NaN
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Handle y_train and y_test
y_train = np.nan_to_num(y_train, nan=np.median(y_train))
y_test = np.nan_to_num(y_test, nan=np.median(y_test))
```

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning:
g: X does not have valid feature names, but SimpleImputer was fitted with
feature names
warnings.warn(

```
In [159]: from sklearn.preprocessing import StandardScaler
```

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

```
In [160]: print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("NaN in X_train:", np.isnan(X_train).sum())
print("NaN in X_test:", np.isnan(X_test).sum())
print("Max value in X_train:", np.max(X_train))
print("Max value in X_test:", np.max(X_test))
```

```
X_train shape: (8, 14)
X_test shape: (2, 14)
NaN in X_train: 0
NaN in X_test: 0
Max value in X_train: 2.6457513110645903
Max value in X_test: 1.299867367239363
```

```
In [161]: for name, model in models.items():
    model.fit(X_train, y_train)
    print(f"{name} successfully trained.")
```

```
Random Forest successfully trained.
Gradient Boosting successfully trained.
CatBoost successfully trained.
```

```
In [162]: for name, model in models.items():
    y_pred = model.predict(X_test)
    print(f"\n{name} Performance:")
    print("MAE:", mean_absolute_error(y_test, y_pred))
    print("MSE:", mean_squared_error(y_test, y_pred))
    print("R²:", r2_score(y_test, y_pred))
```

```
Random Forest Performance:
MAE: 0.19724999999999637
MSE: 0.04412762499999882
R²: -0.9612277777777276
```

```
Gradient Boosting Performance:
MAE: 0.2710947020328498
MSE: 0.07531644146794239
R²: -2.3473973985752217
```

```
CatBoost Performance:
MAE: 0.3727569282814571
MSE: 0.1511500144230224
R²: -5.717778418801004
```

```
In [163]: from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Define the Random Forest Regressor
rf = RandomForestRegressor(random_state=42)

# Define the parameter grid for RandomizedSearchCV
param_grid = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Perform RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_grid,
    n_iter=50,
    scoring='neg_mean_squared_error',
    cv=3,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

# Fit the RandomizedSearchCV to the data
random_search.fit(X_train, y_train)

# Best parameters and model
tuned_rf = random_search.best_estimator_
best_params = random_search.best_params_
print("Best Parameters:", best_params)

# Evaluate the tuned model
y_pred = tuned_rf.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nTuned Random Forest Performance:")
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("R2 Score:", r2)
```

Fitting 3 folds for each of 50 candidates, totalling 150 fits
 Best Parameters: {'n_estimators': 300, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 20}

Tuned Random Forest Performance:
 Mean Absolute Error (MAE): 0.18016547619047296
 Mean Squared Error (MSE): 0.04541356143140422
 R² Score: -1.0183805080624126

```
In [164]: import joblib

# Save the trained Random Forest model to a file
model_filename = 'tuned_random_forest_model.pkl'
joblib.dump(best_model, model_filename)
print(f"Model saved as {model_filename}")
```

Model saved as tuned_random_forest_model.pkl

```
In [165]: # Load the saved model
loaded_model = joblib.load(model_filename)

# Make predictions with the loaded model
y_pred = loaded_model.predict(X_test)
print(f"Predictions: {y_pred}")
```

```
-----
-
ValueError                                     Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\755928901.py in <module>
      3
      4 # Make predictions with the loaded model
----> 5 y_pred = loaded_model.predict(X_test)
      6 print(f"Predictions: {y_pred}")

~\AppData\Roaming\Python\Python39\site-packages\xgboost\sklearn.py in predict(self, X, output_margin, validate_features, base_margin, iteration_range)
    1184         if self._can_use_inplace_predict():
    1185             try:
-> 1186                 predts = self.get_booster().inplace_predict(
    1187                     data=X,
    1188                     iteration_range=iteration_range,
    1189
    1190             )
    1191             if len(data.shape) != 1 and self.num_features() != data.

a.shape[1]:
-> 2524                 raise ValueError(
    2525                     f"Feature shape mismatch, expected: {self.num_
    2526 features()}, "
    2527                     f"got {data.shape[1]}")
```

ValueError: Feature shape mismatch, expected: 10, got 14

```
In [166]: from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
import joblib

# Example: Preprocessing and model pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), list(X_train.columns)), # Scale numeric features
    ]
)

pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', best_model) # Use the tuned Random Forest model
])

# Fit the pipeline
pipeline.fit(X_train, y_train)

# Save the pipeline
pipeline_filename = 'random_forest_pipeline.pkl'
joblib.dump(pipeline, pipeline_filename)
print(f"Pipeline saved as {pipeline_filename}")
```

```
-----
-
AttributeError                                     Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\182222383.py in <module>
    7     preprocessor = ColumnTransformer(
    8         transformers=[
----> 9             ('num', StandardScaler(), list(X_train.columns)), # Scale numeric features
   10         ]
   11     )

AttributeError: 'numpy.ndarray' object has no attribute 'columns'
```

```
In [171]: import pandas as pd
import joblib

# Load the pipeline
pipeline = joblib.load('tuned_random_forest_model.pkl')

# If X_test is a NumPy array, convert it back to DataFrame
if isinstance(X_test, np.ndarray):
    column_names = ['feature1', 'feature2', ..., 'featureN'] # Add actual
    X_test = pd.DataFrame(X_test, columns=column_names)

# Align columns with training data
X_test_aligned = X_test[X_train.columns]

# Make predictions
y_pred = pipeline.predict(X_test_aligned)
print(f"Predictions: {y_pred}")
```

```
-----
-
ValueError Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\1686163636.py in <module>
    8 if isinstance(X_test, np.ndarray):
    9     column_names = ['feature1', 'feature2', ..., 'featureN'] # Ad
d actual feature names
--> 10     X_test = pd.DataFrame(X_test, columns=column_names)
    11
    12 # Align columns with training data

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in __init__
    _self, data, index, columns, dtype, copy)
    692
    693     else:
--> 694         mgr = ndarray_to_mgr(
    695             data,
    696             index,
    697             columns,
    698             copy,
    699             dtype,
    700             index,
    701             columns,
    702             data,
    703             copy,
    704             dtype,
    705             index,
    706             columns,
    707             data,
    708             copy,
    709             dtype,
    710             index,
    711             columns,
    712             data,
    713             copy,
    714             dtype,
    715             index,
    716             columns,
    717             data,
    718             copy,
    719             dtype,
    720             index,
    721             columns,
    722             data,
    723             copy,
    724             dtype,
    725             index,
    726             columns,
    727             data,
    728             copy,
    729             dtype,
    730             index,
    731             columns,
    732             data,
    733             copy,
    734             dtype,
    735             index,
    736             columns,
    737             data,
    738             copy,
    739             dtype,
    740             index,
    741             columns,
    742             data,
    743             copy,
    744             dtype,
    745             index,
    746             columns,
    747             data,
    748             copy,
    749             dtype,
    750             index,
    751             columns,
    752             data,
    753             copy,
    754             dtype,
    755             index,
    756             columns,
    757             data,
    758             copy,
    759             dtype,
    760             index,
    761             columns,
    762             data,
    763             copy,
    764             dtype,
    765             index,
    766             columns,
    767             data,
    768             copy,
    769             dtype,
    770             index,
    771             columns,
    772             data,
    773             copy,
    774             dtype,
    775             index,
    776             columns,
    777             data,
    778             copy,
    779             dtype,
    780             index,
    781             columns,
    782             data,
    783             copy,
    784             dtype,
    785             index,
    786             columns,
    787             data,
    788             copy,
    789             dtype,
    790             index,
    791             columns,
    792             data,
    793             copy,
    794             dtype,
    795             index,
    796             columns,
    797             data,
    798             copy,
    799             dtype,
    800             index,
    801             columns,
    802             data,
    803             copy,
    804             dtype,
    805             index,
    806             columns,
    807             data,
    808             copy,
    809             dtype,
    810             index,
    811             columns,
    812             data,
    813             copy,
    814             dtype,
    815             index,
    816             columns,
    817             data,
    818             copy,
    819             dtype,
    820             index,
    821             columns,
    822             data,
    823             copy,
    824             dtype,
    825             index,
    826             columns,
    827             data,
    828             copy,
    829             dtype,
    830             index,
    831             columns,
    832             data,
    833             copy,
    834             dtype,
    835             index,
    836             columns,
    837             data,
    838             copy,
    839             dtype,
    840             index,
    841             columns,
    842             data,
    843             copy,
    844             dtype,
    845             index,
    846             columns,
    847             data,
    848             copy,
    849             dtype,
    850             index,
    851             columns,
    852             data,
    853             copy,
    854             dtype,
    855             index,
    856             columns,
    857             data,
    858             copy,
    859             dtype,
    860             index,
    861             columns,
    862             data,
    863             copy,
    864             dtype,
    865             index,
    866             columns,
    867             data,
    868             copy,
    869             dtype,
    870             index,
    871             columns,
    872             data,
    873             copy,
    874             dtype,
    875             index,
    876             columns,
    877             data,
    878             copy,
    879             dtype,
    880             index,
    881             columns,
    882             data,
    883             copy,
    884             dtype,
    885             index,
    886             columns,
    887             data,
    888             copy,
    889             dtype,
    890             index,
    891             columns,
    892             data,
    893             copy,
    894             dtype,
    895             index,
    896             columns,
    897             data,
    898             copy,
    899             dtype,
    900             index,
    901             columns,
    902             data,
    903             copy,
    904             dtype,
    905             index,
    906             columns,
    907             data,
    908             copy,
    909             dtype,
    910             index,
    911             columns,
    912             data,
    913             copy,
    914             dtype,
    915             index,
    916             columns,
    917             data,
    918             copy,
    919             dtype,
    920             index,
    921             columns,
    922             data,
    923             copy,
    924             dtype,
    925             index,
    926             columns,
    927             data,
    928             copy,
    929             dtype,
    930             index,
    931             columns,
    932             data,
    933             copy,
    934             dtype,
    935             index,
    936             columns,
    937             data,
    938             copy,
    939             dtype,
    940             index,
    941             columns,
    942             data,
    943             copy,
    944             dtype,
    945             index,
    946             columns,
    947             data,
    948             copy,
    949             dtype,
    950             index,
    951             columns,
    952             data,
    953             copy,
    954             dtype,
    955             index,
    956             columns,
    957             data,
    958             copy,
    959             dtype,
    960             index,
    961             columns,
    962             data,
    963             copy,
    964             dtype,
    965             index,
    966             columns,
    967             data,
    968             copy,
    969             dtype,
    970             index,
    971             columns,
    972             data,
    973             copy,
    974             dtype,
    975             index,
    976             columns,
    977             data,
    978             copy,
    979             dtype,
    980             index,
    981             columns,
    982             data,
    983             copy,
    984             dtype,
    985             index,
    986             columns,
    987             data,
    988             copy,
    989             dtype,
    990             index,
    991             columns,
    992             data,
    993             copy,
    994             dtype,
    995             index,
    996             columns,
    997             data,
    998             copy,
    999             dtype,
    1000             index,
    1001             columns,
    1002             data,
    1003             copy,
    1004             dtype,
    1005             index,
    1006             columns,
    1007             data,
    1008             copy,
    1009             dtype,
    1010             index,
    1011             columns,
    1012             data,
    1013             copy,
    1014             dtype,
    1015             index,
    1016             columns,
    1017             data,
    1018             copy,
    1019             dtype,
    1020             index,
    1021             columns,
    1022             data,
    1023             copy,
    1024             dtype,
    1025             index,
    1026             columns,
    1027             data,
    1028             copy,
    1029             dtype,
    1030             index,
    1031             columns,
    1032             data,
    1033             copy,
    1034             dtype,
    1035             index,
    1036             columns,
    1037             data,
    1038             copy,
    1039             dtype,
    1040             index,
    1041             columns,
    1042             data,
    1043             copy,
    1044             dtype,
    1045             index,
    1046             columns,
    1047             data,
    1048             copy,
    1049             dtype,
    1050             index,
    1051             columns,
    1052             data,
    1053             copy,
    1054             dtype,
    1055             index,
    1056             columns,
    1057             data,
    1058             copy,
    1059             dtype,
    1060             index,
    1061             columns,
    1062             data,
    1063             copy,
    1064             dtype,
    1065             index,
    1066             columns,
    1067             data,
    1068             copy,
    1069             dtype,
    1070             index,
    1071             columns,
    1072             data,
    1073             copy,
    1074             dtype,
    1075             index,
    1076             columns,
    1077             data,
    1078             copy,
    1079             dtype,
    1080             index,
    1081             columns,
    1082             data,
    1083             copy,
    1084             dtype,
    1085             index,
    1086             columns,
    1087             data,
    1088             copy,
    1089             dtype,
    1090             index,
    1091             columns,
    1092             data,
    1093             copy,
    1094             dtype,
    1095             index,
    1096             columns,
    1097             data,
    1098             copy,
    1099             dtype,
    1100             index,
    1101             columns,
    1102             data,
    1103             copy,
    1104             dtype,
    1105             index,
    1106             columns,
    1107             data,
    1108             copy,
    1109             dtype,
    1110             index,
    1111             columns,
    1112             data,
    1113             copy,
    1114             dtype,
    1115             index,
    1116             columns,
    1117             data,
    1118             copy,
    1119             dtype,
    1120             index,
    1121             columns,
    1122             data,
    1123             copy,
    1124             dtype,
    1125             index,
    1126             columns,
    1127             data,
    1128             copy,
    1129             dtype,
    1130             index,
    1131             columns,
    1132             data,
    1133             copy,
    1134             dtype,
    1135             index,
    1136             columns,
    1137             data,
    1138             copy,
    1139             dtype,
    1140             index,
    1141             columns,
    1142             data,
    1143             copy,
    1144             dtype,
    1145             index,
    1146             columns,
    1147             data,
    1148             copy,
    1149             dtype,
    1150             index,
    1151             columns,
    1152             data,
    1153             copy,
    1154             dtype,
    1155             index,
    1156             columns,
    1157             data,
    1158             copy,
    1159             dtype,
    1160             index,
    1161             columns,
    1162             data,
    1163             copy,
    1164             dtype,
    1165             index,
    1166             columns,
    1167             data,
    1168             copy,
    1169             dtype,
    1170             index,
    1171             columns,
    1172             data,
    1173             copy,
    1174             dtype,
    1175             index,
    1176             columns,
    1177             data,
    1178             copy,
    1179             dtype,
    1180             index,
    1181             columns,
    1182             data,
    1183             copy,
    1184             dtype,
    1185             index,
    1186             columns,
    1187             data,
    1188             copy,
    1189             dtype,
    1190             index,
    1191             columns,
    1192             data,
    1193             copy,
    1194             dtype,
    1195             index,
    1196             columns,
    1197             data,
    1198             copy,
    1199             dtype,
    1200             index,
    1201             columns,
    1202             data,
    1203             copy,
    1204             dtype,
    1205             index,
    1206             columns,
    1207             data,
    1208             copy,
    1209             dtype,
    1210             index,
    1211             columns,
    1212             data,
    1213             copy,
    1214             dtype,
    1215             index,
    1216             columns,
    1217             data,
    1218             copy,
    1219             dtype,
    1220             index,
    1221             columns,
    1222             data,
    1223             copy,
    1224             dtype,
    1225             index,
    1226             columns,
    1227             data,
    1228             copy,
    1229             dtype,
    1230             index,
    1231             columns,
    1232             data,
    1233             copy,
    1234             dtype,
    1235             index,
    1236             columns,
    1237             data,
    1238             copy,
    1239             dtype,
    1240             index,
    1241             columns,
    1242             data,
    1243             copy,
    1244             dtype,
    1245             index,
    1246             columns,
    1247             data,
    1248             copy,
    1249             dtype,
    1250             index,
    1251             columns,
    1252             data,
    1253             copy,
    1254             dtype,
    1255             index,
    1256             columns,
    1257             data,
    1258             copy,
    1259             dtype,
    1260             index,
    1261             columns,
    1262             data,
    1263             copy,
    1264             dtype,
    1265             index,
    1266             columns,
    1267             data,
    1268             copy,
    1269             dtype,
    1270             index,
    1271             columns,
    1272             data,
    1273             copy,
    1274             dtype,
    1275             index,
    1276             columns,
    1277             data,
    1278             copy,
    1279             dtype,
    1280             index,
    1281             columns,
    1282             data,
    1283             copy,
    1284             dtype,
    1285             index,
    1286             columns,
    1287             data,
    1288             copy,
    1289             dtype,
    1290             index,
    1291             columns,
    1292             data,
    1293             copy,
    1294             dtype,
    1295             index,
    1296             columns,
    1297             data,
    1298             copy,
    1299             dtype,
    1300             index,
    1301             columns,
    1302             data,
    1303             copy,
    1304             dtype,
    1305             index,
    1306             columns,
    1307             data,
    1308             copy,
    1309             dtype,
    1310             index,
    1311             columns,
    1312             data,
    1313             copy,
    1314             dtype,
    1315             index,
    1316             columns,
    1317             data,
    1318             copy,
    1319             dtype,
    1320             index,
    1321             columns,
    1322             data,
    1323             copy,
    1324             dtype,
    1325             index,
    1326             columns,
    1327             data,
    1328             copy,
    1329             dtype,
    1330             index,
    1331             columns,
    1332             data,
    1333             copy,
    1334             dtype,
    1335             index,
    1336             columns,
    1337             data,
    1338             copy,
    1339             dtype,
    1340             index,
    1341             columns,
    1342             data,
    1343             copy,
    1344             dtype,
    1345             index,
    1346             columns,
    1347             data,
    1348             copy,
    1349             dtype,
    1350             index,
    1351             columns,
    1352             data,
    1353             copy,
    1354             dtype,
    1355             index,
    1356             columns,
    1357             data,
    1358             copy,
    1359             dtype,
    1360             index,
    1361             columns,
    1362             data,
    1363             copy,
    1364             dtype,
    1365             index,
    1366             columns,
    1367             data,
    1368             copy,
    1369             dtype,
    1370             index,
    1371             columns,
    1372             data,
    1373             copy,
    1374             dtype,
    1375             index,
    1376             columns,
    1377             data,
    1378             copy,
    1379             dtype,
    1380             index,
    1381             columns,
    1382             data,
    1383             copy,
    1384             dtype,
    1385             index,
    1386             columns,
    1387             data,
    1388             copy,
    1389             dtype,
    1390             index,
    1391             columns,
    1392             data,
    1393             copy,
    1394             dtype,
    1395             index,
    1396             columns,
    1397             data,
    1398             copy,
    1399             dtype,
    1400             index,
    1401             columns,
    1402             data,
    1403             copy,
    1404             dtype,
    1405             index,
    1406             columns,
    1407             data,
    1408             copy,
    1409             dtype,
    1410             index,
    1411             columns,
    1412             data,
    1413             copy,
    1414             dtype,
    1415             index,
    1416             columns,
    1417             data,
    1418             copy,
    1419             dtype,
    1420             index,
    1421             columns,
    1422             data,
    1423             copy,
    1424             dtype,
    1425             index,
    1426             columns,
    1427             data,
    1428             copy,
    1429             dtype,
    1430             index,
    1431             columns,
    1432             data,
    1433             copy,
    1434             dtype,
    1435             index,
    1436             columns,
    1437             data,
    1438             copy,
    1439             dtype,
    1440             index,
    1441             columns,
    1442             data,
    1443             copy,
    1444             dtype,
    1445             index,
    1446             columns,
    1447             data,
    1448             copy,
    1449             dtype,
    1450             index,
    1451             columns,
    1452             data,
    1453             copy,
    1454             dtype,
    1455             index,
    1456             columns,
    1457             data,
    1458             copy,
    1459             dtype,
    1460             index,
    1461             columns,
    1462             data,
    1463             copy,
    1464             dtype,
    1465             index,
    1466             columns,
    1467             data,
    1468             copy,
    1469             dtype,
    1470             index,
    1471             columns,
    1472             data,
    1473             copy,
    1474             dtype,
    1475             index,
    1476             columns,
    1477             data,
    1478             copy,
    1479             dtype,
    1480             index,
    1481             columns,
    1482             data,
    1483             copy,
    1484             dtype,
    1485             index,
    1486             columns,
    1487             data,
    1488             copy,
    1489             dtype,
    1490             index,
    1491             columns,
    1492             data,
    1493             copy,
    1494             dtype,
    1495             index,
    1496             columns,
    1497             data,
    1498             copy,
    1499             dtype,
    1500             index,
    1501             columns,
    1502             data,
    1503             copy,
    1504             dtype,
    1505             index,
    1506             columns,
    1507             data,
    1508             copy,
    1509             dtype,
    1510             index,
    1511             columns,
    1512             data,
    1513             copy,
    1514             dtype,
    1515             index,
    1516             columns,
    1517             data,
    1518             copy,
    1519             dtype,
    1520             index,
    1521             columns,
    1522             data,
    1523             copy,
    1524             dtype,
    1525             index,
    1526             columns,
    1527             data,
    1528             copy,
    1529             dtype,
    1530             index,
    1531             columns,
    1532             data,
    1533             copy,
    1534             dtype,
    1535             index,
    1536             columns,
    1537             data,
    1538             copy,
    1539             dtype,
    1540             index,
    1541             columns,
    1542             data,
    1543             copy,
    1544             dtype,
    1545             index,
    1546             columns,
    1547             data,
    1548             copy,
    1549             dtype,
    1550             index,
    1551             columns,
    1552             data,
    1553             copy,
    1554             dtype,
    1555             index,
    1556             columns,
    1557             data,
    1558             copy,
    1559             dtype,
    1560             index,
    1561             columns,
    1562             data,
    1563             copy,
    1564             dtype,
    1565             index,
    1566             columns,
    1567             data,
    1568             copy,
    1569             dtype,
    1570             index,
    1571             columns,
    1572             data,
    1573             copy,
    1574             dtype,
    1575             index,
    1576             columns,
    1577             data,
    1578             copy,
    1579             dtype,
    1580             index,
    1581             columns,
    1582             data,
    1583             copy,
    1584             dtype,
    1585             index,
    1586             columns,
    1587             data,
    1588             copy,
    1589             dtype,
    1590             index,
    1591             columns,
    1592             data,
    1593             copy,
    1594             dtype,
    1595             index,
    1596             columns,
    1597             data,
    1598             copy,
    1599             dtype,
    1600             index,
    1601             columns,
    1602             data,
    1603             copy,
    1604             dtype,
    1605             index,
    1606             columns,
    1607             data,
    1608             copy,
    1609             dtype,
    1610             index,
    1611             columns,
    1612             data,
    1613             copy,
    1614             dtype,
    1615             index,
    1616             columns,
    1617             data,
    1618             copy,
    1619             dtype,
    1620             index,
    1621             columns,
    1622             data,
    1623             copy,
    1624             dtype,
    1625             index,
    1626             columns,
    1627             data,
    1628             copy,
    1629             dtype,
    1630             index,
    1631             columns,
    1632             data,
    1633             copy,
    1634             dtype,
    1635             index,
    1636             columns,
    1637             data,
    1638             copy,
    1639             dtype,
    1640             index,
    1641             columns,
    1642             data,
    1643             copy,
    1644             dtype,
    1645             index,
    1646             columns,
    1647             data,
    1648             copy,
    1649             dtype,
    1650             index,
    1651             columns,
    1652             data,
    1653             copy,
    1654             dtype,
    1655             index,
    1656             columns,
    1657             data,
    1658             copy,
    1659             dtype,
    1660             index,
    1661             columns,
    1662             data,
    1663             copy,
    1664             dtype,
    1665             index,
    1666             columns,
    1667             data,
    1668             copy,
    1669             dtype,
    1670             index,
    1671             columns,
    1672             data,
    1673             copy,
    1674             dtype,
    1675             index,
    1676             columns,
    1677             data,
    1678             copy,
    1679             dtype,
    1680             index,
    1681             columns,
    1682             data,
    1683             copy,
    1684             dtype,
    1685             index,
    1686             columns,
    1687             data,
    1688             copy,
    1689             dtype,
    1690             index,
    1691             columns,
    1692             data,
    1693             copy,
    1694             dtype,
    1695             index,
    1696             columns,
    1697             data,
    1698             copy,
    1699             dtype,
    1700             index,
    1701             columns,
    1702             data,
    1703             copy,
    1704             dtype,
    1705             index,
    1706             columns,
    1707             data,
    1708             copy,
    1709             dtype,
    1710             index,
    1711             columns,
    1712             data,
    1713             copy,
    1714             dtype,
    1715             index,
    1716             columns,
    1717             data,
    1718             copy,
    1719             dtype,
    1720             index,
    1721             columns,
    1722             data,
    1723             copy,
    1724             dtype,
    1725             index,
    1726             columns,
    1727             data,
    1728             copy,
    1729             dtype,
    1730             index,
    1731             columns,
    1732             data,
    1733             copy,
    1734             dtype,
    1735             index,
    1736             columns,
    1737             data,
    1738             copy,
    1739             dtype,
    1740             index,
    1741             columns,
    1742             data,
    1743             copy,
    1744             dtype,
    1745             index,
    1746             columns,
    1747             data,
    1748             copy,
    1749             dtype,
    1750             index,
    1751             columns,
    1752             data,
    1753             copy,
    1754             dtype,
    1755             index,
    1756             columns,
    1757             data,
    1758             copy,
    1759             dtype,
    1760             index,
    1761             columns,
    1762             data,
    1763             copy,
    1764             dtype,
    1765             index,
    1766             columns,
    1767             data,
    1768             copy,
    1769             dtype,
    1770             index,
    1771             columns,
    1772             data,
    1773             copy,
    1774             dtype,
    1775             index,
    1776             columns,
    1777             data,
    1778             copy,
    1779             dtype,
    1780             index,
    1781             columns,
    1782             data,
    1783             copy,
    1784             dtype,
    1785             index,
    1786             columns,
    1787             data,
    1788             copy,
    1789             dtype,
    1790             index,
    1791             columns,
    1792             data,
    1793             copy,
    1794             dtype,
    1795             index,
    1796             columns,
    1797             data,
    1798             copy,
    1799             dtype,
    1800             index,
    1801             columns,
    1802             data,
    1803             copy,
    1804             dtype,
    1805             index,
    1806             columns,
    1807             data,
    1808             copy,
    1809             dtype,
    1810             index,
    1811             columns,
    1812             data,
    1813             copy,
    1814             dtype,
    1815             index,
    1816             columns,
    1817             data,
    1818             copy,
    1819             dtype,
    1820             index,
    1821             columns,
    1822             data,
    1823             copy,
    1824             dtype,
    1825             index,
    1826             columns,
    1827             data,
    1828             copy,
    1829             dtype,
    1830             index,
    1831             columns,
    1832             data,
    1833             copy,
    1834             dtype,
    1835             index,
    1836             columns,
    1837             data,
    1838             copy,
    1839             dtype,
    1840             index,
    1841             columns,
    1842             data,
    1843             copy,
    1844             dtype,
    1845             index,
    1846             columns,
    1847             data,
    1848             copy,
    1849             dtype,
    1850             index,
    1851             columns,
    1852             data,
    1853             copy,
    1854             dtype,
    1855             index,
    1856             columns,
    1857             data,
    1858             copy,
    1859             dtype,
    1860             index,
    1861             columns,
    1862             data,
    1863             copy,
    1864             dtype,
    1865             index,
    1866             columns,
    1867             data,
    1868             copy,
    1869             dtype,
    1870             index,
    1871             columns,
    1872             data,
    1873             copy,
    1874             dtype,
    1875             index,
    1876             columns,
    1877             data,
    1878             copy,
    1879             dtype,
    1880             index,
    1881             columns,
    1882             data,
    1883             copy,
    1884             dtype,
    1885             index,
    1886             columns,
    1887             data,
    1888             copy,
    1889             dtype,
    1890             index,
    1891             columns,
    1892             data,
    1893             copy,
    1894             dtype,
    1895             index,
    1896             columns,
    1897             data,
    1898             copy,
    1899             dtype,
    1900             index,
    1901             columns,
    1902             data,
    1903             copy,
    1904             dtype,
    1905             index,
    1906             columns,
    1907             data,
    1908             copy,
    1909             dtype,
    1910             index,
    1911             columns,
    1912             data,
    1913             copy,
    1914             dtype,
    1915             index,
    1916             columns,
    1917             data,
    1918             copy,
    191
```

```
In [172]: import pandas as pd

# Assuming X_test is a NumPy array
print(f"Shape of X_test: {X_test.shape}")

# Correct number of column names for the data
column_names = ['feature1', 'feature2', ..., 'feature14'] # Add all 14 names

# Convert to DataFrame
try:
    X_test_df = pd.DataFrame(X_test, columns=column_names)
    print("X_test successfully converted to DataFrame.")
except ValueError as e:
    print(f"Error: {e}")
```

```
Shape of X_test: (2, 14)
Error: Shape of passed values is (2, 14), indices imply (2, 4)
```

```
In [170]: # Align X_test with X_train columns
X_test_aligned = X_test_df[X_train.columns]
```

```
-----
-
NameError: name 'X_test_df' is not defined
                                         Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\2731562584.py in <module>
      1 # Align X_test with X_train columns
----> 2 X_test_aligned = X_test_df[X_train.columns]

NameError: name 'X_test_df' is not defined
```

```
In [173]: # Make predictions
y_pred = loaded_model.predict(X_test)

# Evaluate performance
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Model Performance on Test Data:")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")
```

```
-----
-
ValueError                                     Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\4266660602.py in <module>
      1 # Make predictions
----> 2 y_pred = loaded_model.predict(X_test)
      3
      4 # Evaluate performance
      5 from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

~\AppData\Roaming\Python\Python39\site-packages\xgboost\sklearn.py in predict(self, X, output_margin, validate_features, base_margin, iteration_range)
    1184         if self._can_use_inplace_predict():
    1185             try:
-> 1186                 predts = self.get_booster().inplace_predict(
    1187                     data=X,
    1188                     iteration_range=iteration_range,
    1189
    1190             )
    1191             if len(data.shape) != 1 and self.num_features() != data.shape[1]:
    1192                 raise ValueError(
    1193                     f"Feature shape mismatch, expected: {self.num_
    1194                     features()}, "
    1195                     f"got {data.shape[1]}")

ValueError: Feature shape mismatch, expected: 10, got 14
```

```
In [174]: # Check column alignment
print("Train Features:", X_train.columns)
print("Test Features:", X_test.columns)
```

```
-----
-
AttributeError                                     Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_33244\2632926398.py in <module>
      1 # Check column alignment
----> 2 print("Train Features:", X_train.columns)
      3 print("Test Features:", X_test.columns)

AttributeError: 'numpy.ndarray' object has no attribute 'columns'
```

```
In [ ]:
```