```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load datasets
user_details = pd.read_excel('Data Analyst Intern Assignment - Excel.xlsx', sheet_
cooking_sessions = pd.read_excel('Data Analyst Intern Assignment - Excel.xlsx', sh
order_details = pd.read_excel('Data Analyst Intern Assignment - Excel.xlsx', sheet


# Standardize column names
user_details.columns = user_details.columns.str.strip().str.lower().str.replace('
cooking_sessions.columns = cooking_sessions.columns.str.strip().str.lower().str.re
order_details.columns = order_details.columns.str.strip().str.lower().str.replace(



# Convert date fields to datetime
user_details['registration_date'] = pd.to_datetime(user_details['registration_date
cooking_sessions['session_start'] = pd.to_datetime(cooking_sessions['session_start
cooking_sessions['session_end'] = pd.to_datetime(cooking_sessions['session_end'])
order_details['order_date'] = pd.to_datetime(order_details['order_date'])


# Handle missing values
user_details.fillna({'favorite_meal': 'Unknown', 'total_orders': 0}, inplace=True)
cooking_sessions['session_rating'] = cooking_sessions['session_rating'].fillna(coo
order_details.dropna(subset=['amount_(usd)'], inplace=True)




# Remove duplicates
user_details.drop_duplicates(inplace=True)
cooking_sessions.drop_duplicates(inplace=True)
order_details.drop_duplicates(inplace=True)


# Rename 'user_id' columns to avoid suffixes before merging
user_details.rename(columns={'user_id': 'user_id_details'}, inplace=True)
cooking_sessions.rename(columns={'user_id': 'user_id_sessions'}, inplace=True)


# Merge datasets
user_sessions = pd.merge(user_details, cooking_sessions, left_on='user_id_details', right
final_data = pd.merge(user_sessions, order_details, on='session_id', how='inner')


# Inspect the final dataset to see available columns
print(final_data.columns)
```

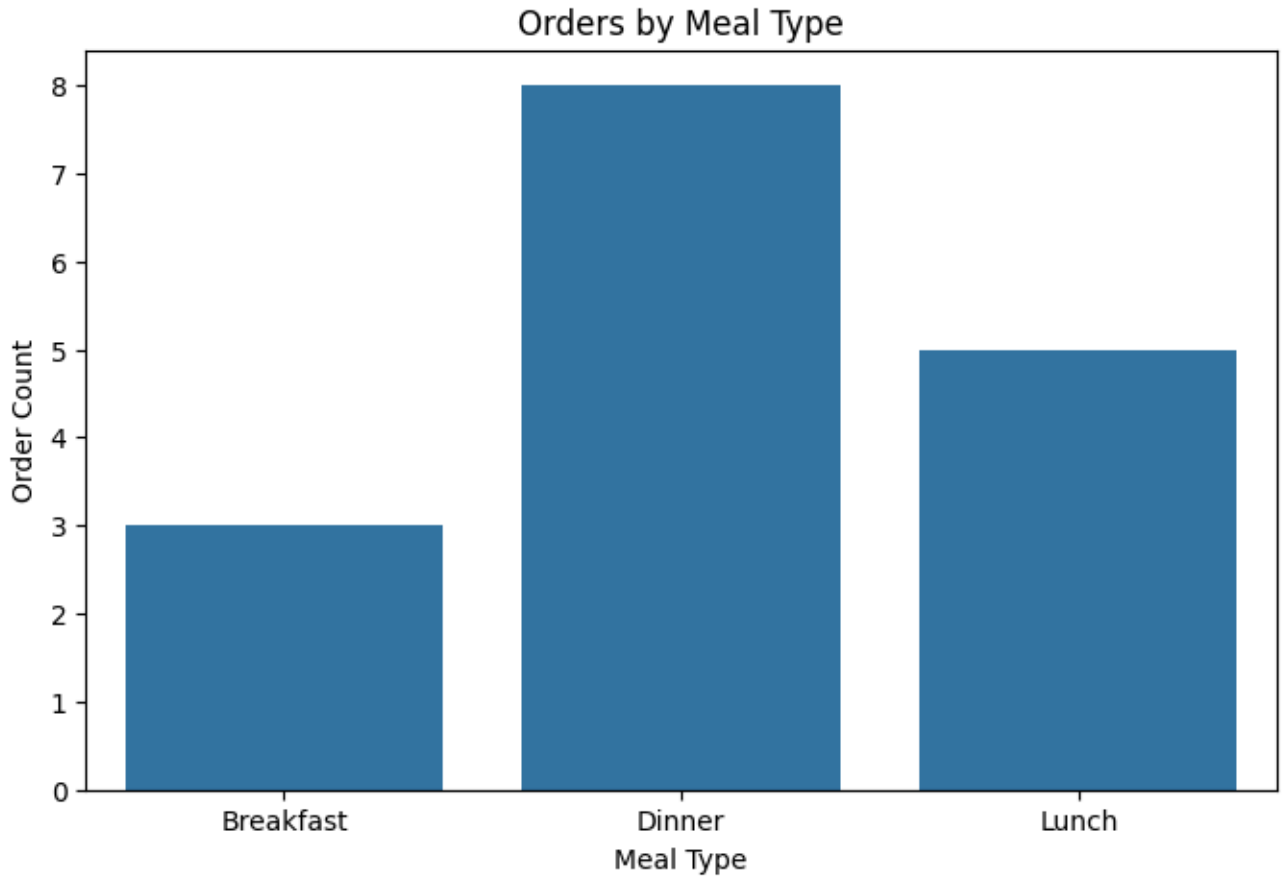
```
Index(['user_id_details', 'user_name', 'age', 'location', 'registration_date',
       'phone', 'email', 'favorite_meal', 'total_orders', 'session_id',
       'user_id_sessions', 'dish_name_x', 'meal_type_x', 'session_start',
```

```
       'session_end', 'duration_(mins)', 'session_rating', 'order_id',
       'user_id', 'order_date', 'meal_type_y', 'dish_name_y', 'order_status',
       'amount_(usd)', 'time_of_day', 'rating'],
      dtype='object')
```

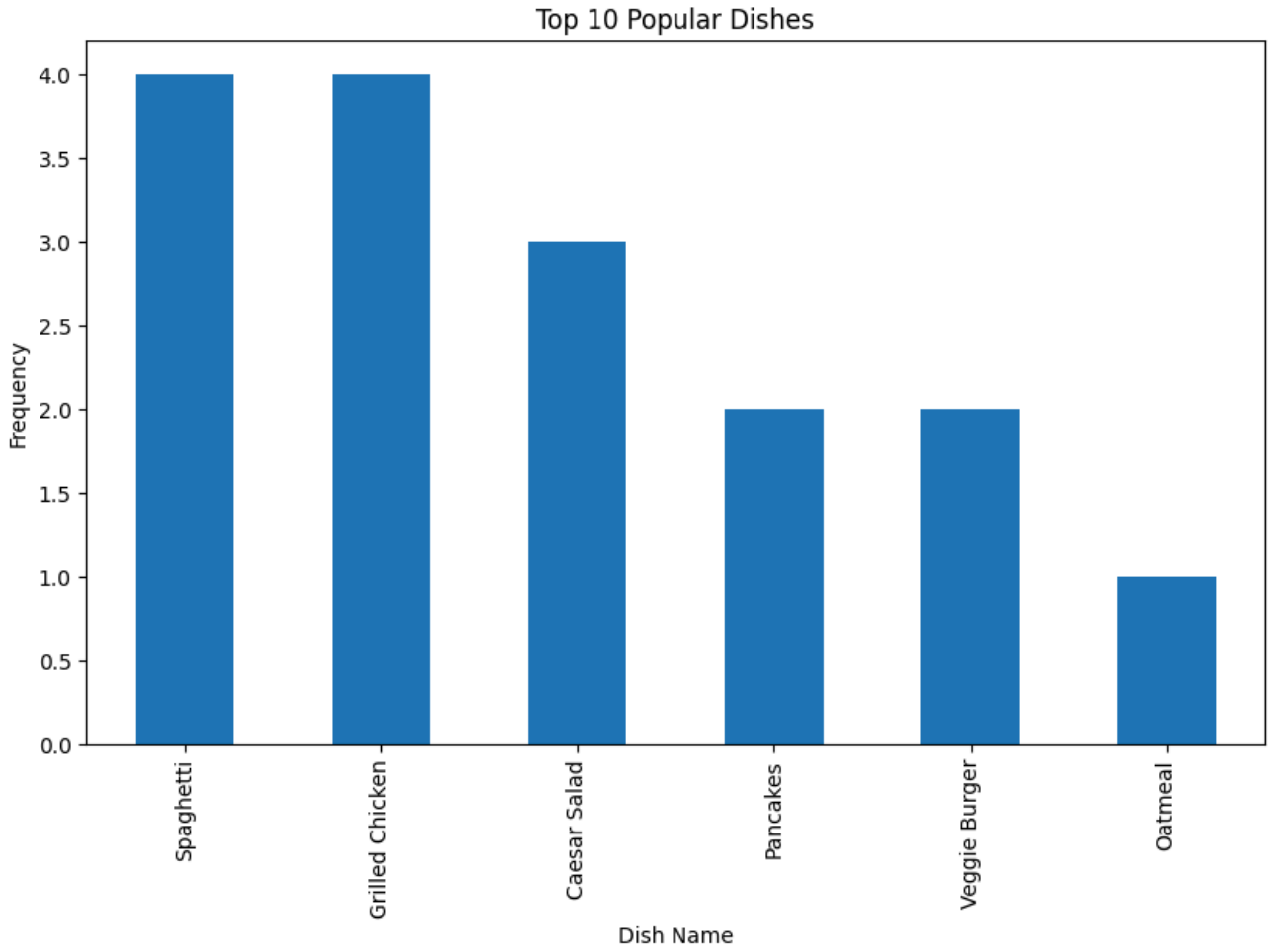
```
# Check if 'user_id_x' exists before dropping it
if 'user_id_x' in final_data.columns:
    final_data['user_id'] = final_data['user_id_x']
    final_data.drop(columns=['user_id_x'], inplace=True)

# Check if 'user_id_y' exists before dropping it
if 'user_id_y' in final_data.columns:
    final_data.drop(columns=['user_id_y'], inplace=True)


# Relationship between cooking sessions and orders
session_order_relationship = final_data.groupby('meal_type_y').agg({
    'order_id': 'count',
    'session_rating': 'mean'
}).rename(columns={'order_id': 'order_count'})

# Most popular dishes
popular_dishes = final_data['dish_name_y'].value_counts()

# Demographic factors
age_group_behavior = final_data.groupby(pd.cut(final_data['age'], bins=[18, 25, 35, 50, 1
```

```
<ipython-input-11-8a4983a0a23c>:11: FutureWarning: The default of observed=False is d
  age_group_behavior = final_data.groupby(pd.cut(final_data['age'], bins=[18, 25, 35,
```

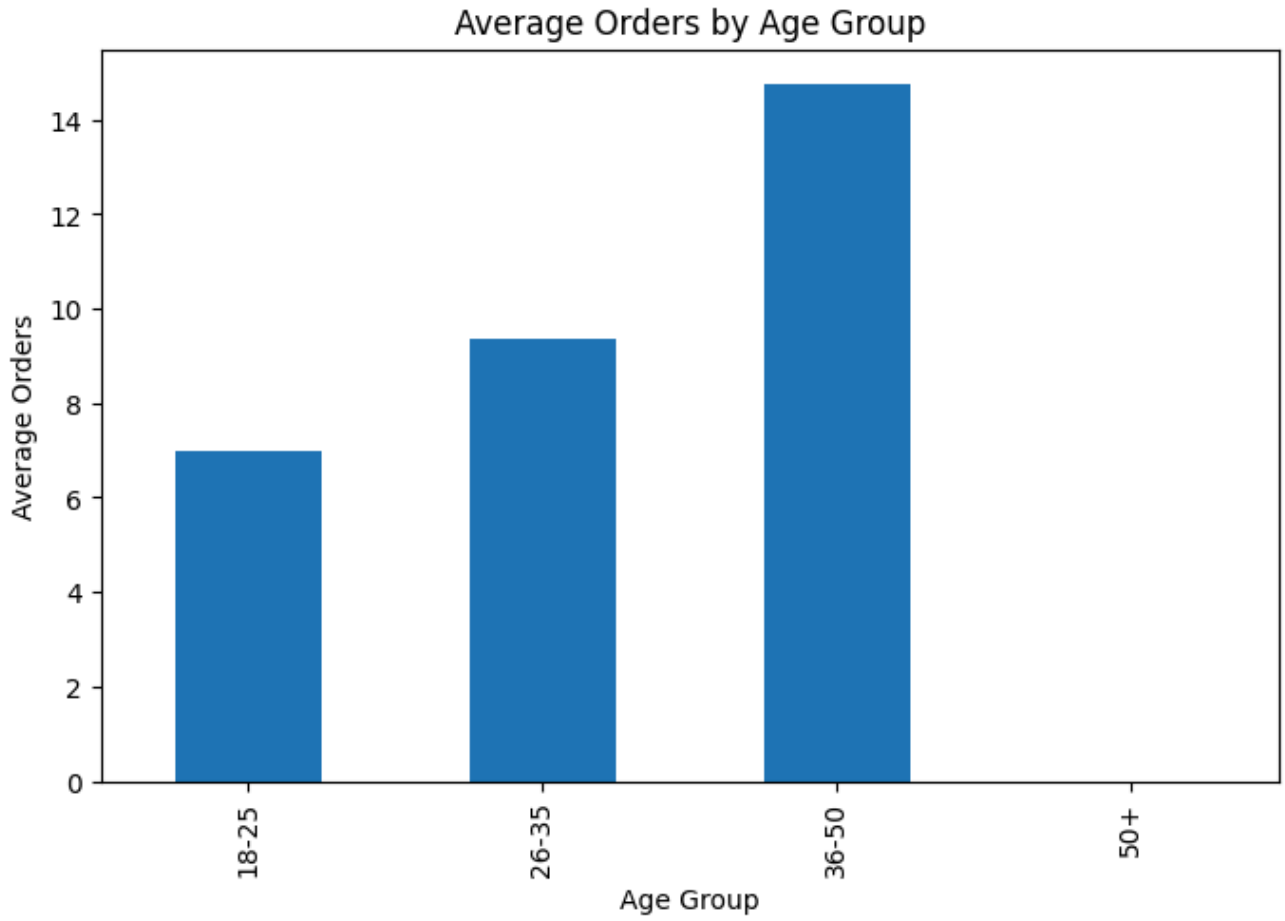
```
# Visualization 1: Orders by Meal Type
plt.figure(figsize=(8, 5))
sns.barplot(x=session_order_relationship.index, y=session_order_relationship['order_count
plt.title('Orders by Meal Type')
plt.xlabel('Meal Type')
plt.ylabel('Order Count')
plt.show()
```




```
# Visualization 2: Popular Dishes
popular_dishes[:10].plot(kind='bar', figsize=(10, 6), title='Top 10 Popular Dishes')
plt.xlabel('Dish Name')
plt.ylabel('Frequency')
plt.show()
```

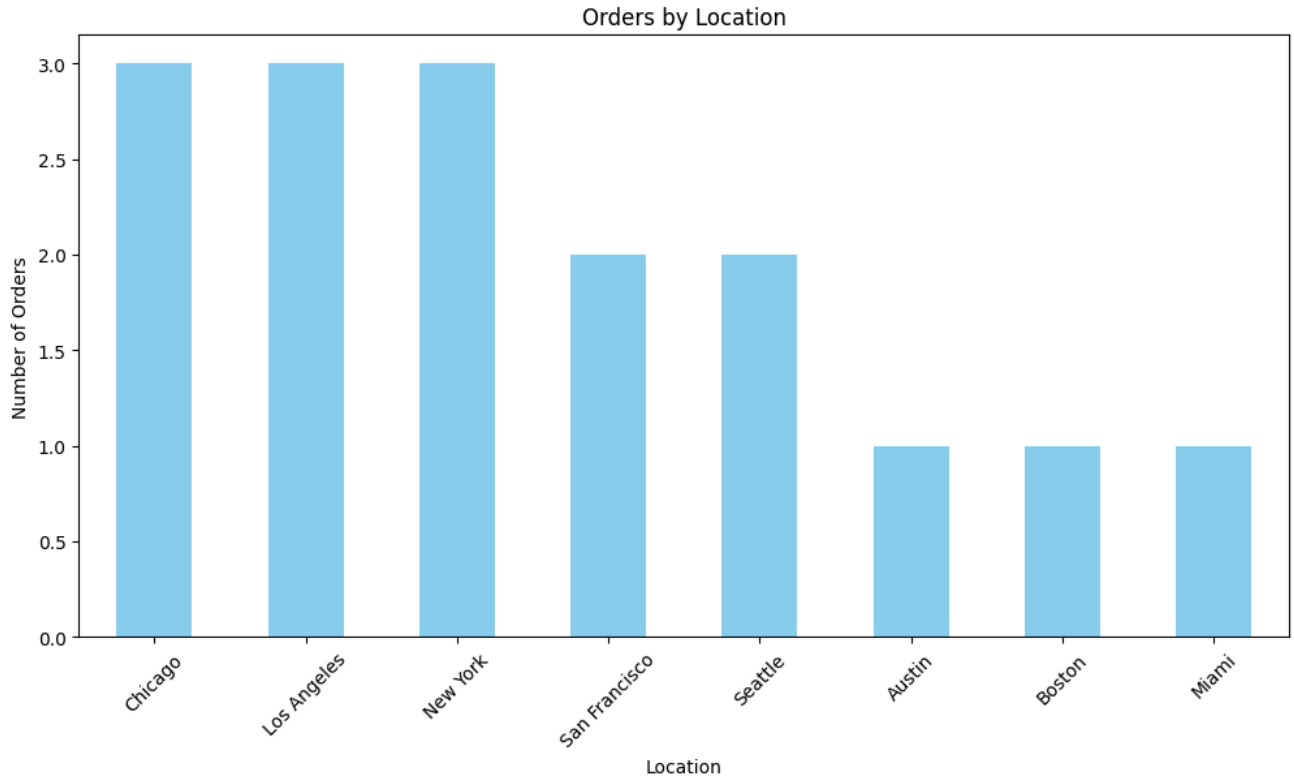



```
# Visualization 3: Age Group Behavior
age_group_behavior.plot(kind='bar', figsize=(8, 5), title='Average Orders by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Average Orders')
plt.show()
```




```
# Order Trends Over Time
final_data['month'] = final_data['order_date'].dt.month
monthly_orders = final_data.groupby('month')['order_id'].count()
plt.figure(figsize=(8, 5))
monthly_orders.plot(kind='line', marker='o')
plt.title('Monthly Orders Trend')
plt.xlabel('Month')
plt.ylabel('Number of Orders')
plt.xticks(range(1, 13))
plt.grid(True)
plt.show()
```


```
# Geographic Distribution of Orders (if location data is available)
if 'location' in final_data.columns:
    location_orders = final_data.groupby('location')['order_id'].count().sort_values(asce
    plt.figure(figsize=(12, 6))
    location_orders.plot(kind='bar', color='skyblue')
    plt.title('Orders by Location')
    plt.xlabel('Location')
    plt.ylabel('Number of Orders')
    plt.xticks(rotation=45)
    plt.show()
```




```
# Visualization: Correlation Heatmap between session rating and order amount
correlation = final_data[['session_rating', 'amount_(usd)']].corr()
print("Correlation between Session Rating and Order Amount:")
print(correlation)
plt.figure(figsize=(6, 4))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt='.2f', cbar=True)
plt.title('Correlation between Session Rating and Order Amount')
plt.show()
```

```
Correlation between Session Rating and Order Amount:
                session_rating  amount_(usd)
session_rating        1.000000      0.502733
amount_(usd)          0.502733      1.000000
```


```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


# Prepare the data for modeling
features = final_data[['duration_(mins)', 'age', 'meal_type_y']]  # Changed session_durat
features = pd.get_dummies(features, drop_first=True)  # One-hot encoding for categorical
target = final_data['amount_(usd)']


# Split the data for order amount prediction
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, rand

# Train the Linear Regression model for order amount prediction
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Make predictions and evaluate the model
predictions = model.predict(X_test)
```

```
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error (Order Amount Prediction): {mse}')
```

```
Mean Squared Error (Order Amount Prediction): 1.874902608517318
```

```
# Visualization for Order Amount Prediction
plt.figure(figsize=(10, 6))
plt.scatter(y_test, predictions, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linesty
plt.title('Actual vs Predicted Order Amount')
plt.xlabel('Actual Order Amount')
plt.ylabel('Predicted Order Amount')
plt.show()
```


```
# Predicting Session Rating using Linear Regression
target_rating = final_data['session_rating']

# Split the data for session rating prediction
X_train, X_test, y_train, y_test = train_test_split(features, target_rating, test_size=0.

# Train the Linear Regression model for session rating prediction
model_rating = LinearRegression()
model_rating.fit(X_train, y_train)
```

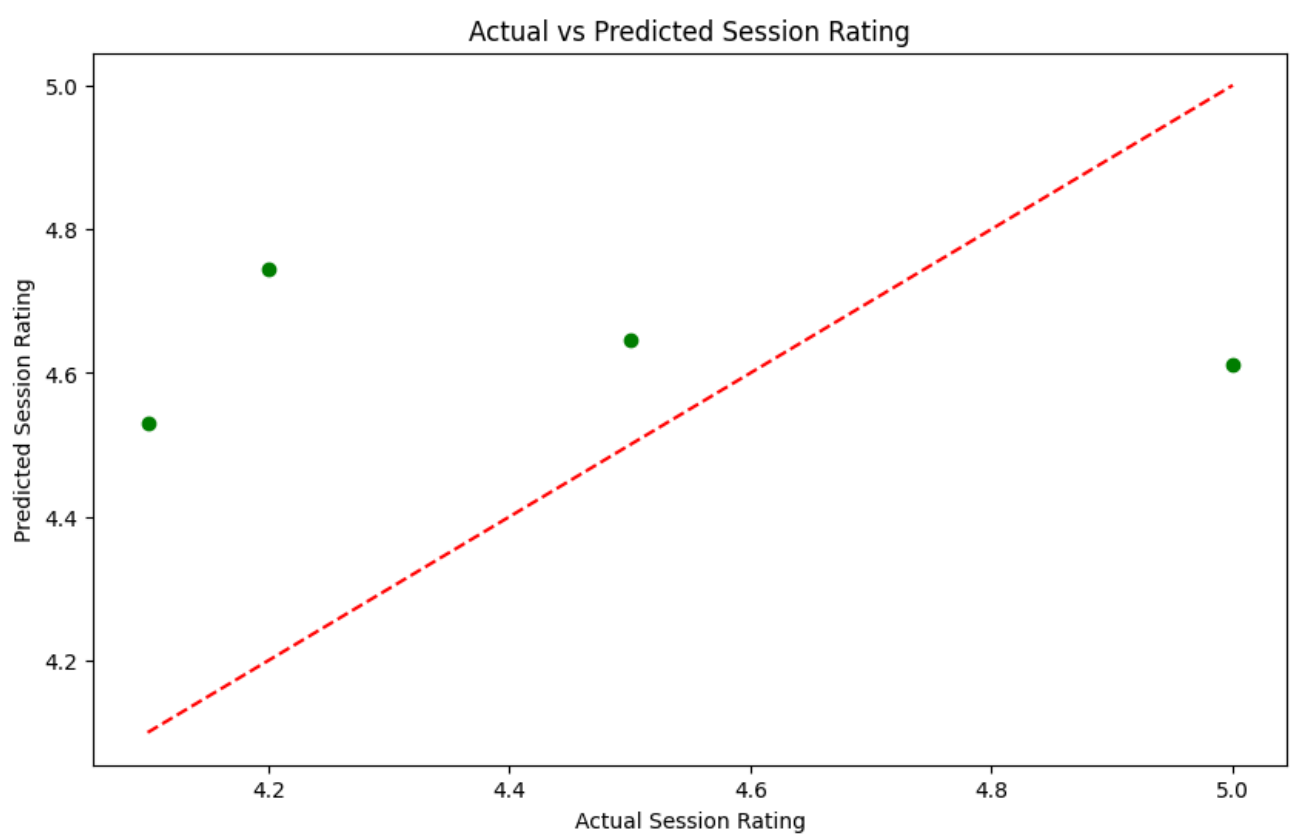
```
LinearRegression
LinearRegression()
```

```
# Make predictions and evaluate the model for session rating
predictions_rating = model_rating.predict(X_test)
```

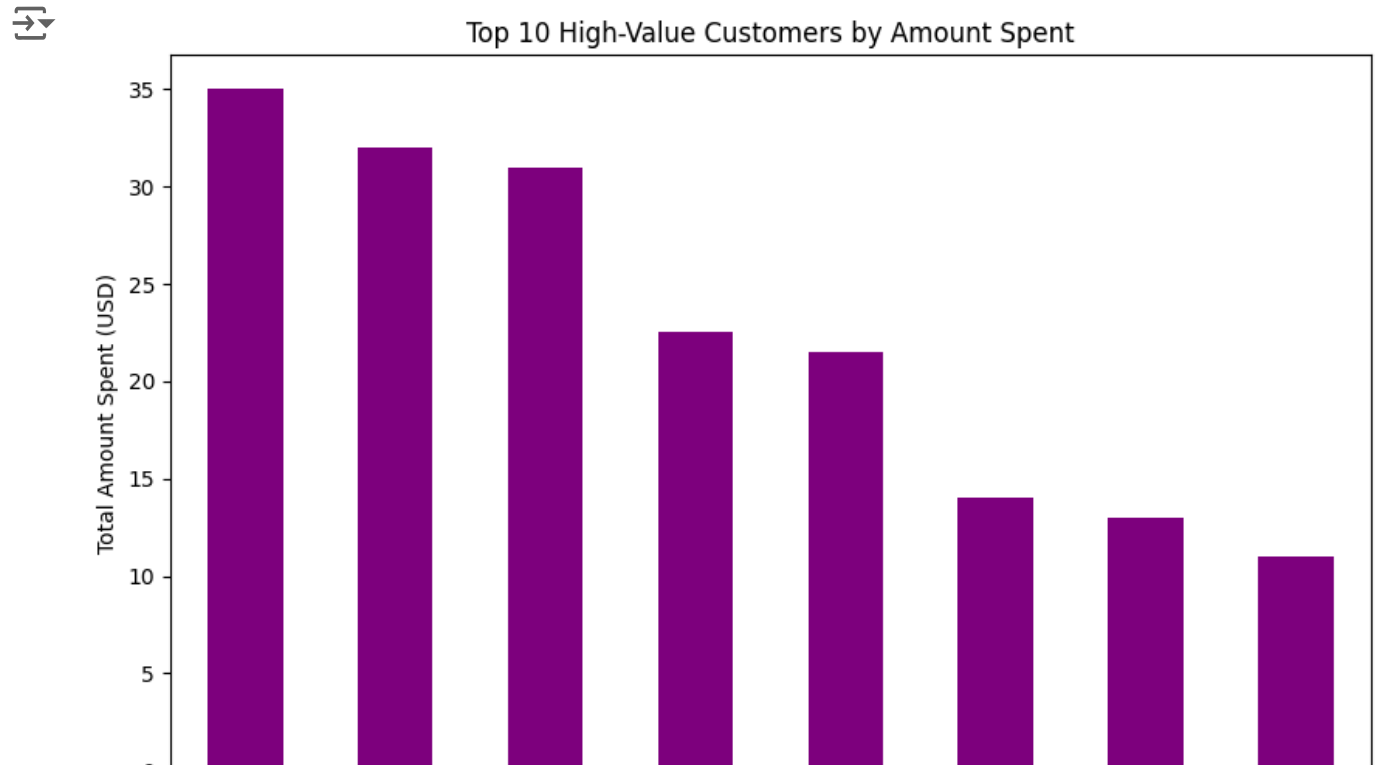
```
mse_rating = mean_squared_error(y_test, predictions_rating)
print(f'Mean Squared Error (Session Rating Prediction): {mse_rating}')
```

```
Mean Squared Error (Session Rating Prediction): 0.16331685991103384
```

```
# Visualization for Session Rating Prediction
plt.figure(figsize=(10, 6))
plt.scatter(y_test, predictions_rating, color='green')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linesty
plt.title('Actual vs Predicted Session Rating')
plt.xlabel('Actual Session Rating')
plt.ylabel('Predicted Session Rating')
plt.show()
```




```
# Business Recommendations
# Visualizing the recommendations with bar charts

# 1. Target high-value customers based on the amount spent.
high_value_customers = final_data.groupby('user_id')['amount_(usd)'].sum().sort_values(as

# Bar chart for high-value customers
plt.figure(figsize=(10, 6))
high_value_customers.plot(kind='bar', color='purple')
plt.title('Top 10 High-Value Customers by Amount Spent')
plt.xlabel('User ID')
plt.ylabel('Total Amount Spent (USD)')
plt.show()
```




```
# 2. Age group behavior - Average orders by age group
age_group_behavior = final_data.groupby(pd.cut(final_data['age'], bins=[18, 25, 35, 50, 1

# Bar chart for Age Group Behavior
plt.figure(figsize=(8, 5))
age_group_behavior.plot(kind='bar', color='orange')
plt.title('Average Orders by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Average Orders')
plt.show()
```

```
<ipython-input-27-d0b17776658f>:2: FutureWarning: The default of observed=False is de
```