In [1]:
```
import numpy as np
```

READING THE DATA

In [7]:
```
import pandas as pd
df = pd.read_csv("restaurant_data.csv")
df
```

Out[7]:

	Name	Location	Cuisine	Rating	Seating Capacity	Average Meal Price	Marketing Budget	Social Media Followers	Chef Experience Years	Number of Reviews	Avg Review Length	Ambience Score	Service Quality Score	Parking Availability	Weekend Reservations	Weekday Reservations	Revenue
0	Restaurant 0	Rural	Japanese	4.0	38	73.98	2224	23406	13	185	161.924906	1.3	7.0	Yes	13	4	638945.52
1	Restaurant 1	Downtown	Mexican	3.2	76	28.11	4416	42741	8	533	148.759717	2.6	3.4	Yes	48	6	490207.83
2	Restaurant 2	Rural	Italian	4.7	48	48.29	2796	37285	18	853	56.849189	5.3	6.7	No	27	14	541368.62
3	Restaurant 3	Rural	Italian	4.4	34	51.55	1167	15214	13	82	205.433265	4.6	2.8	Yes	9	17	404556.80
4	Restaurant 4	Downtown	Japanese	4.9	88	75.98	3639	40171	9	78	241.681584	8.6	2.1	No	37	26	1491046.35
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8363	Restaurant 8363	Suburban	Indian	3.4	54	34.85	1102	11298	11	380	253.919515	9.5	5.0	Yes	37	0	434653.45
8364	Restaurant 8364	Rural	Indian	3.7	49	36.88	1988	20432	9	713	175.590194	2.7	2.6	No	37	21	414977.92
8365	Restaurant 8365	Downtown	Italian	4.7	88	46.87	5949	63945	6	436	222.953647	4.8	1.7	Yes	83	21	930395.87
8366	Restaurant 8366	Rural	American	3.1	31	44.53	707	7170	1	729	178.482851	6.1	2.1	No	6	21	311493.48
8367	Restaurant 8367	Rural	Japanese	4.0	33	71.07	2003	24268	8	197	151.838065	5.9	7.5	Yes	5	12	534142.98

8368 rows × 17 columns

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8368 entries, 0 to 8367
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Name                    8368 non-null   object 
 1   Location                8368 non-null   object 
 2   Cuisine                 8368 non-null   object 
 3   Rating                  8368 non-null   float64
 4   Seating Capacity        8368 non-null   int64  
 5   Average Meal Price      8368 non-null   float64
 6   Marketing Budget        8368 non-null   int64  
 7   Social Media Followers  8368 non-null   int64  
 8   Chef Experience Years   8368 non-null   int64  
 9   Number of Reviews       8368 non-null   int64  
 10  Avg Review Length       8368 non-null   float64
 11  Ambience Score          8368 non-null   float64
 12  Service Quality Score   8368 non-null   float64
 13  Parking Availability    8368 non-null   object 
 14  Weekend Reservations    8368 non-null   int64  
 15  Weekday Reservations    8368 non-null   int64  
 16  Revenue                 8368 non-null   float64
dtypes: float64(6), int64(7), object(4)
memory usage: 1.1+ MB
```

In [10]:
```
df.shape
```

Out[10]:
```
(8368, 17)
```

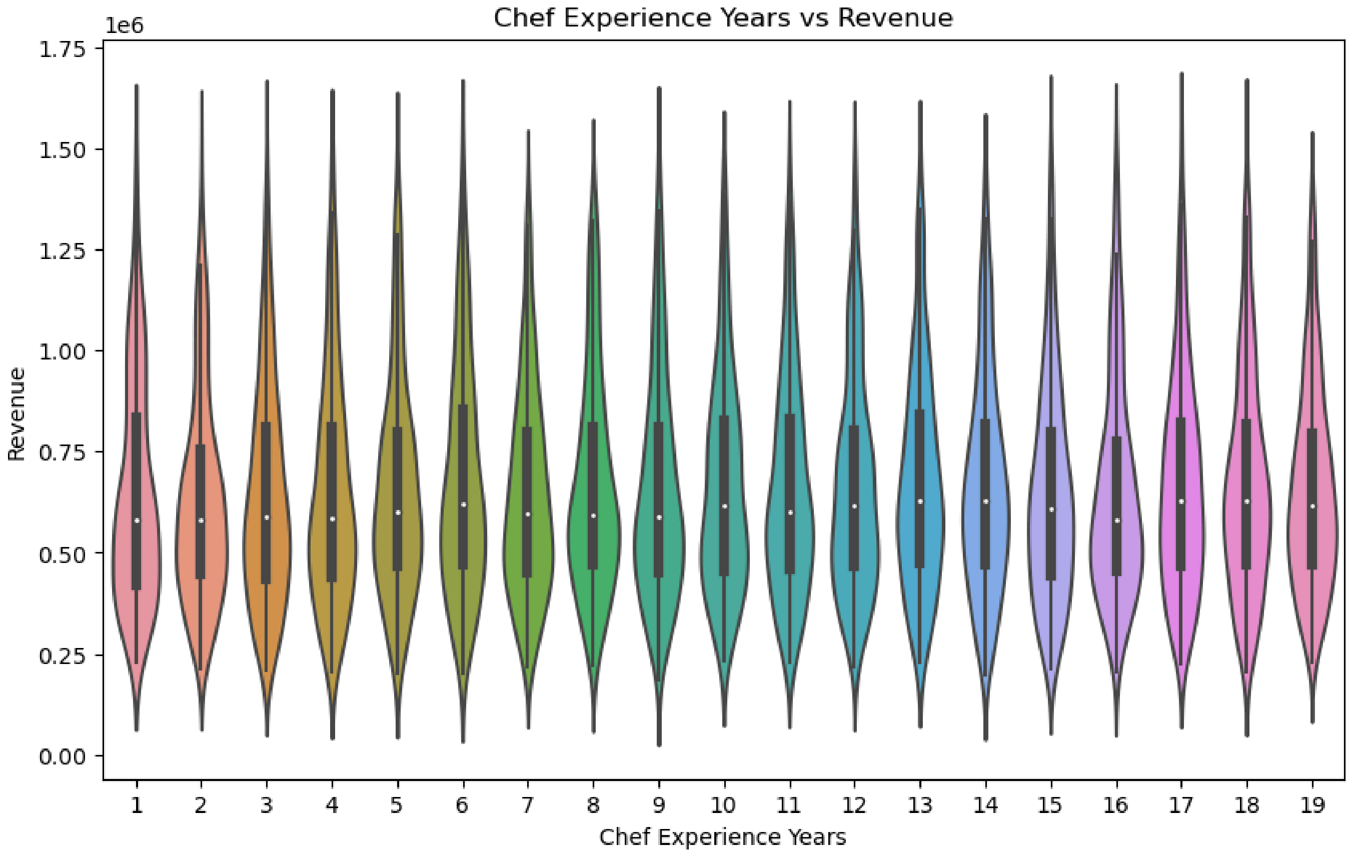
Checking if any row or column has null value or not

In [11]:
```
df.isna().sum()
```

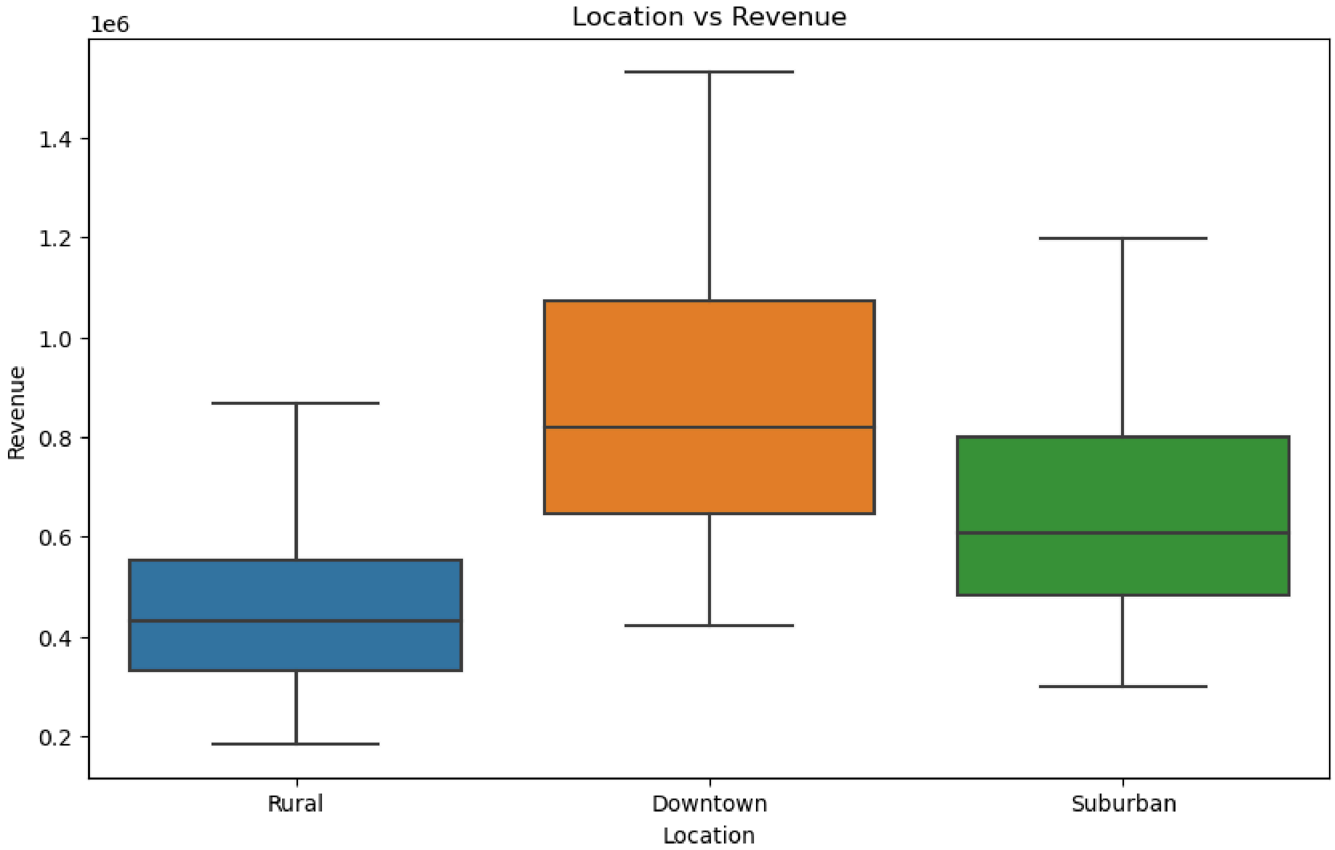
Out[11]:
```
Name                      0
Location                  0
Cuisine                   0
Rating                    0
Seating Capacity          0
Average Meal Price        0
Marketing Budget          0
Social Media Followers    0
Chef Experience Years     0
Number of Reviews         0
Avg Review Length         0
Ambience Score            0
Service Quality Score     0
Parking Availability      0
Weekend Reservations      0
Weekday Reservations      0
Revenue                   0
dtype: int64
```

Making charts and plots using matplotlib and seaborn libs

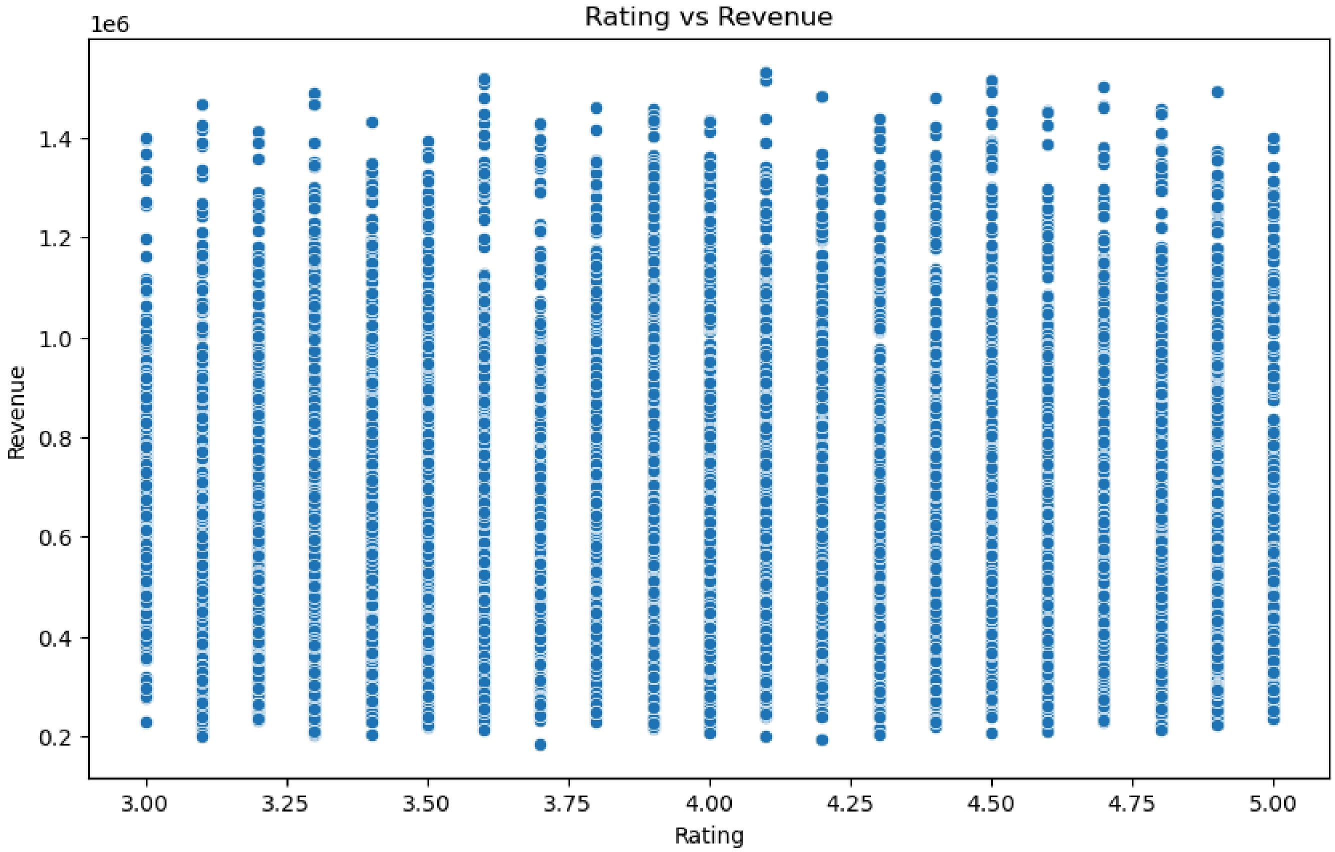
In [22]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10,6))
sns.violinplot(data=df, x = 'Chef Experience Years', y = 'Revenue')
plt.title('Chef Experience Years vs Revenue')
plt.show()
```



In [21]:
```
plt.figure(figsize=(10,6))
plt.title('Location vs Revenue')
sns.boxplot(data=df, x = 'Location', y = 'Revenue')
plt.show()
```



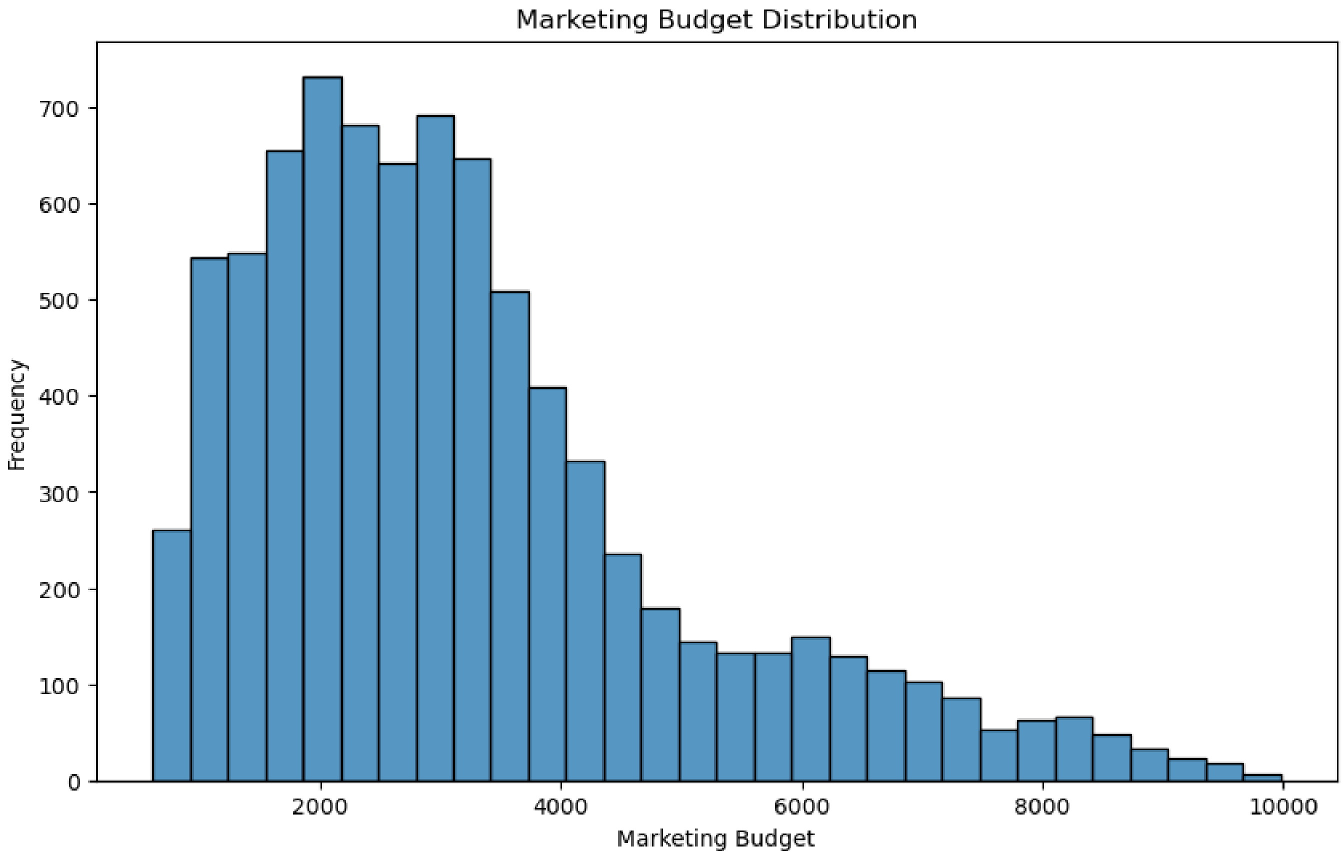
In [23]:
```
plt.figure(figsize=(10,6))
plt.title('Rating vs Revenue')
sns.scatterplot(data=df, x = 'Rating', y = 'Revenue')
plt.show()
```



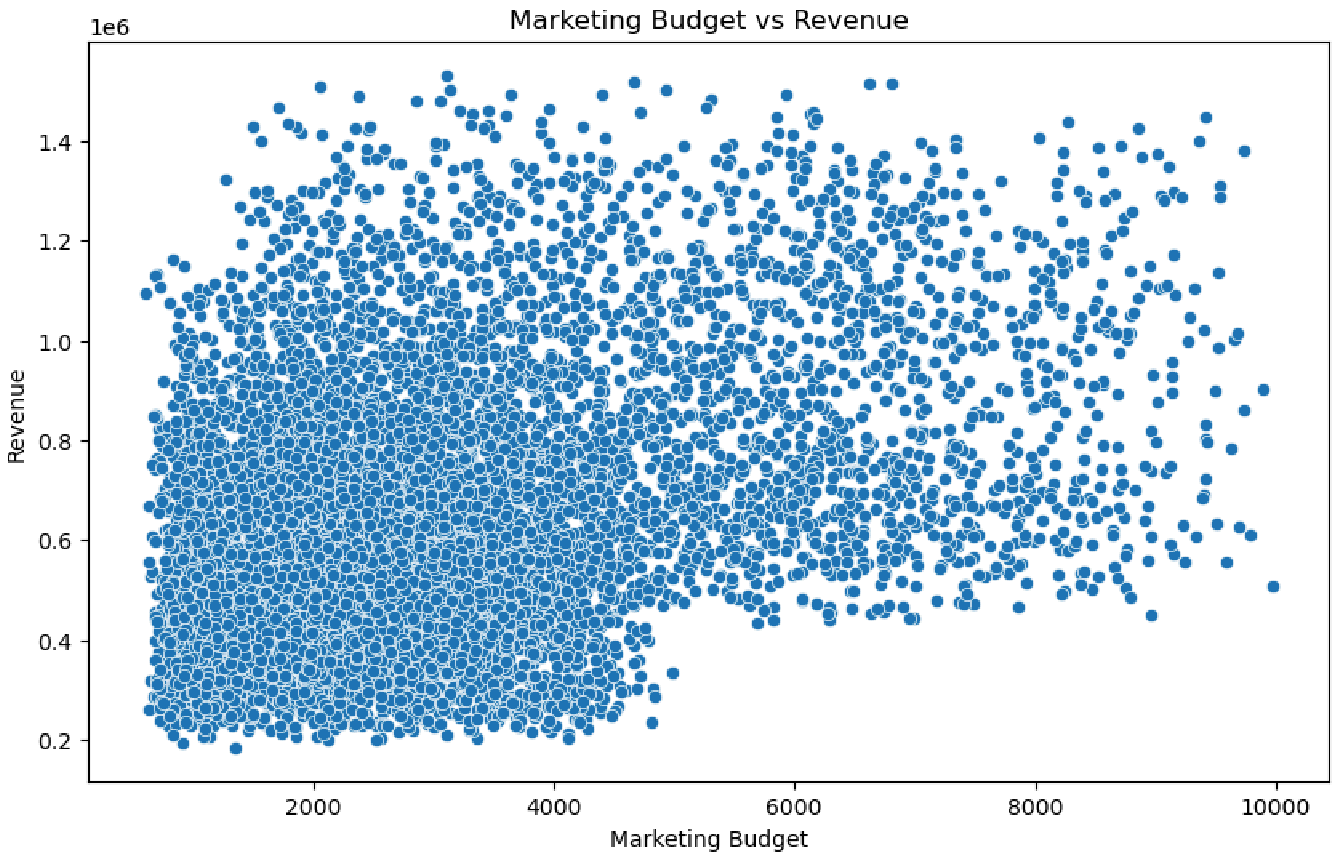
In [26]:
```
plt.figure(figsize=(10,6))
sns.lmplot(x='Average Meal Price', y='Revenue', data=df)
plt.title('Average Meal Price vs Revenue')
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```

In [27]:
```
plt.figure(figsize=(10,6))
sns.histplot(df['Marketing Budget'], bins=30)
plt.title('Marketing Budget Distribution')
plt.xlabel('Marketing Budget')
plt.ylabel('Frequency')
plt.show()
```

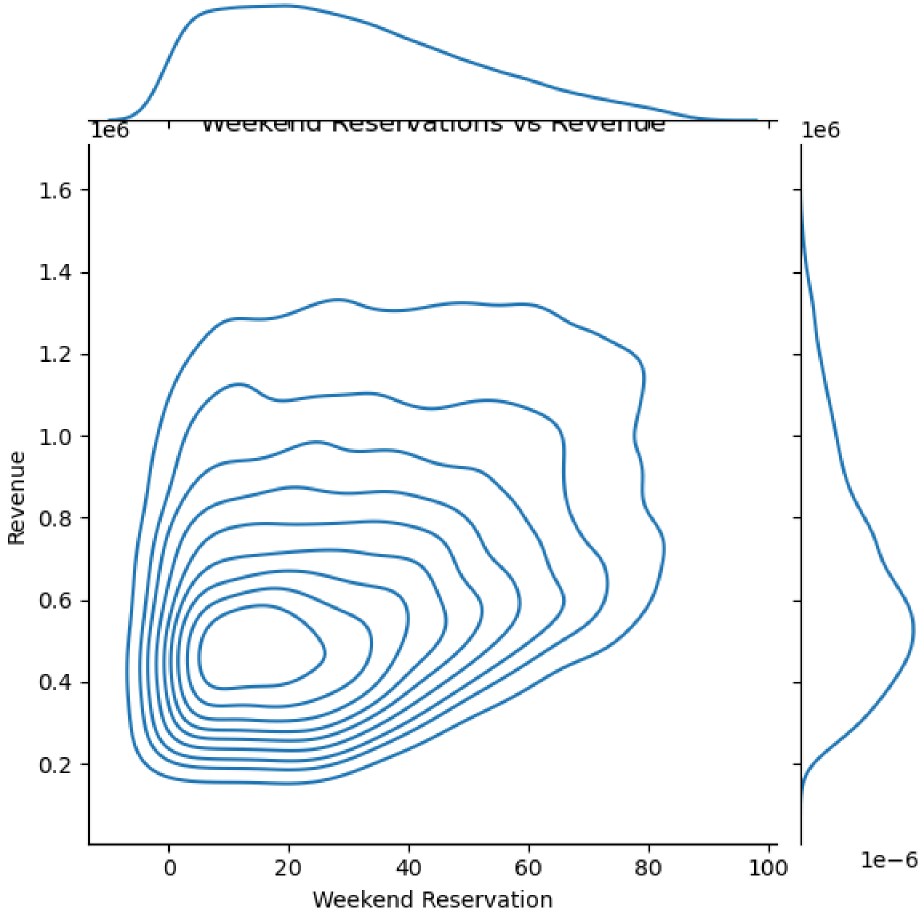


In [32]:
```
plt.figure(figsize=(10,6))
sns.scatterplot(x='Marketing Budget', y='Revenue', data=df)
plt.title('Marketing Budget vs Revenue')
plt.xlabel('Marketing Budget')
plt.ylabel('Revenue')
plt.show()
```



In [42]:
```
plt.figure(figsize=(10,6))
sns.jointplot(x='Weekend Reservations', y='Revenue', data=df, kind='kde')
plt.title('Weekend Reservations vs Revenue')
plt.xlabel('Weekend Reservation')
plt.ylabel('Revenue')
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



In [47]:
```
string_columns = df.select_dtypes(include=['object'])
string_columns.head()
```

Out[47]:

	Name	Location	Cuisine	Parking Availability
0	Restaurant 0	Rural	Japanese	Yes
1	Restaurant 1	Downtown	Mexican	Yes
2	Restaurant 2	Rural	Italian	No
3	Restaurant 3	Rural	Italian	Yes
4	Restaurant 4	Downtown	Japanese	No

READING THE DATA/ TAKING INSIGHTS

In [55]:
```
df['Parking Availability'].unique()
```

Out[55]:
```
array(['Yes', 'No'], dtype=object)
```

In [56]:
```
df['Parking Availability'] = df['Parking Availability'].replace({'Yes': 1, 'No': 0})
df['Parking Availability'].head()
```

Out[56]:
```
0    1
1    1
2    0
3    1
4    0
Name: Parking Availability, dtype: int64
```

In [57]:
```
df['Location'].unique()
```

Out[57]:
```
array(['Rural', 'Downtown', 'Suburban'], dtype=object)
```

In [58]:
```
df['Location'].value_counts()
```

Out[58]:
```
Downtown    2821
Suburban    2785
Rural       2762
Name: Location, dtype: int64
```

In [59]:
```
location_variables = pd.get_dummies(df['Location'], dtype=int)
location_variables.head()
```

Out[59]:

	Downtown	Rural	Suburban
0	0	1	0
1	1	0	0
2	0	1	0
3	0	1	0
4	1	0	0

In [60]:
```
updated_df = pd.concat([df, location_variables], axis=1)
updated_df.head()
```

Out[60]:

	Name	Location	Cuisine	Rating	Seating Capacity	Average Meal Price	Marketing Budget	Social Media Followers	Chef Experience Years	Number of Reviews	Avg Review Length	Ambience Score	Service Quality Score	Parking Availability	Weekend Reservations	Weekday Reservations	Revenue	Dov
0	Restaurant 0	Rural	Japanese	4.0	38	73.98	2224	23406	13	185	161.924906	1.3	7.0	1	13	4	638945.52	
1	Restaurant 1	Downtown	Mexican	3.2	76	28.11	4416	42741	8	533	148.759717	2.6	3.4	1	48	6	490207.83	
2	Restaurant 2	Rural	Italian	4.7	48	48.29	2796	37285	18	853	56.849189	5.3	6.7	0	27	14	541368.62	
3	Restaurant 3	Rural	Italian	4.4	34	51.55	1167	15214	13	82	205.433265	4.6	2.8	1	9	17	404556.80	
4	Restaurant 4	Downtown	Japanese	4.9	88	75.98	3639	40171	9	78	241.681584	8.6	2.1	0	37	26	1491046.35	

In [61]:

```
updated_df.shape
```

Out[61]:

```
(8368, 20)
```

In [62]:

```
print(updated_df['Downtown'].sum())
print(updated_df['Rural'].sum())
print(updated_df['Suburban'].sum())
```

```
2821
2762
2785
```

In [63]:

```
updated_df = updated_df.drop('Location', axis=1)
updated_df.head()
```

Out[63]:

	Name	Cuisine	Rating	Seating Capacity	Average Meal Price	Marketing Budget	Social Media Followers	Chef Experience Years	Number of Reviews	Avg Review Length	Ambience Score	Service Quality Score	Parking Availability	Weekend Reservations	Weekday Reservations	Revenue	Downtown	Ru
0	Restaurant 0	Japanese	4.0	38	73.98	2224	23406	13	185	161.924906	1.3	7.0	1	13	4	638945.52	0	
1	Restaurant 1	Mexican	3.2	76	28.11	4416	42741	8	533	148.759717	2.6	3.4	1	48	6	490207.83	1	
2	Restaurant 2	Italian	4.7	48	48.29	2796	37285	18	853	56.849189	5.3	6.7	0	27	14	541368.62	0	
3	Restaurant 3	Italian	4.4	34	51.55	1167	15214	13	82	205.433265	4.6	2.8	1	9	17	404556.80	0	
4	Restaurant 4	Japanese	4.9	88	75.98	3639	40171	9	78	241.681584	8.6	2.1	0	37	26	1491046.35	1	

In [64]:
```
updated_df.shape
```
Out[64]:
```
(8368, 19)
```

In [65]:
```
string_columns = updated_df.select_dtypes(include=['object'])
string_columns.head()
```
Out[65]:

	Name	Cuisine
0	Restaurant 0	Japanese
1	Restaurant 1	Mexican
2	Restaurant 2	Italian
3	Restaurant 3	Italian
4	Restaurant 4	Japanese

In [85]:
```
df['Cuisine'].unique()
```
Out[85]:
```
array(['Japanese', 'Mexican', 'Italian', 'Indian', 'French', 'American'],
      dtype=object)
```

In [86]:
```
df['Cuisine'].value_counts()
```
Out[86]:
```
French      1433
American    1416
Italian     1413
Mexican     1393
Indian      1369
Japanese    1344
Name: Cuisine, dtype: int64
```

In [66]:
```
numeric_cuisine_values = pd.get_dummies(df['Cuisine'], dtype=int)
numeric_cuisine_values.head()
```
Out[66]:

	American	French	Indian	Italian	Japanese	Mexican
0	0	0	0	0	1	0
1	0	0	0	0	0	1
2	0	0	0	1	0	0
3	0	0	0	1	0	0
4	0	0	0	0	1	0

In [67]:
```
print("counts for French: ", numeric_cuisine_values['French'].sum())
print("counts for American: ", numeric_cuisine_values['American'].sum())
print("counts for Italian: ", numeric_cuisine_values['Italian'].sum())
print("counts for Mexican: ", numeric_cuisine_values['Mexican'].sum())
print("counts for Indian: ", numeric_cuisine_values['Indian'].sum())
print("counts for Japanese: ", numeric_cuisine_values['Japanese'].sum())
```

```
counts for French:  1433
counts for American:  1416
counts for Italian:  1413
counts for Mexican:  1393
counts for Indian:  1369
counts for Japanese:  1344
```

In [68]:
```
updated_df.shape
```

Out[68]:
```
(8368, 19)
```

In [69]:
```
updated_df = pd.concat([updated_df, numeric_cuisine_values], axis=1)
updated_df.head()
```

Out[69]:

	Name	Cuisine	Rating	Seating Capacity	Average Meal Price	Marketing Budget	Social Media Followers	Chef Experience Years	Number of Reviews	Avg Review Length	...	Revenue	Downtown	Rural	Suburban	American	French	Indian	Italian	Japane
0	Restaurant 0	Japanese	4.0	38	73.98	2224	23406	13	185	161.924906	...	638945.52	0	1	0	0	0	0	0	
1	Restaurant 1	Mexican	3.2	76	28.11	4416	42741	8	533	148.759717	...	490207.83	1	0	0	0	0	0	0	
2	Restaurant 2	Italian	4.7	48	48.29	2796	37285	18	853	56.849189	...	541368.62	0	1	0	0	0	0	1	
3	Restaurant 3	Italian	4.4	34	51.55	1167	15214	13	82	205.433265	...	404556.80	0	1	0	0	0	0	1	
4	Restaurant 4	Japanese	4.9	88	75.98	3639	40171	9	78	241.681584	...	1491046.35	1	0	0	0	0	0	0	

5 rows × 25 columns

In [89]:
```
updated_df = updated_df.drop('Cuisine', axis=1)
updated_df.head()
```

Out[89]:

	Rating	Seating Capacity	Average Meal Price	Marketing Budget	Social Media Followers	Chef Experience Years	Number of Reviews	Avg Review Length	Ambience Score	Service Quality Score	...	Revenue	Downtown	Rural	Suburban	American	French	Indian	Italian	Japanes
0	4.0	38	73.98	2224	23406	13	185	161.924906	1.3	7.0	...	638945.52	0	1	0	0	0	0	0	
1	3.2	76	28.11	4416	42741	8	533	148.759717	2.6	3.4	...	490207.83	1	0	0	0	0	0	0	
2	4.7	48	48.29	2796	37285	18	853	56.849189	5.3	6.7	...	541368.62	0	1	0	0	0	0	1	
3	4.4	34	51.55	1167	15214	13	82	205.433265	4.6	2.8	...	404556.80	0	1	0	0	0	0	1	
4	4.9	88	75.98	3639	40171	9	78	241.681584	8.6	2.1	...	1491046.35	1	0	0	0	0	0	0	

5 rows × 23 columns

In [90]:

```
updated_df.head()
```

Out[90]:

	Rating	Seating Capacity	Average Meal Price	Marketing Budget	Social Media Followers	Chef Experience Years	Number of Reviews	Avg Review Length	Ambience Score	Service Quality Score	...	Revenue	Downtown	Rural	Suburban	American	French	Indian	Italian	Japanes
0	4.0	38	73.98	2224	23406	13	185	161.924906	1.3	7.0	...	638945.52	0	1	0	0	0	0	0	
1	3.2	76	28.11	4416	42741	8	533	148.759717	2.6	3.4	...	490207.83	1	0	0	0	0	0	0	
2	4.7	48	48.29	2796	37285	18	853	56.849189	5.3	6.7	...	541368.62	0	1	0	0	0	0	1	
3	4.4	34	51.55	1167	15214	13	82	205.433265	4.6	2.8	...	404556.80	0	1	0	0	0	0	1	
4	4.9	88	75.98	3639	40171	9	78	241.681584	8.6	2.1	...	1491046.35	1	0	0	0	0	0	0	

5 rows × 23 columns

In [91]:

```
data = updated_df
data.shape
```

Out[91]:

```
(8368, 23)
```

In [92]:

```
np.random.seed(42)
```

In [93]:

```
x = data.drop('Revenue', axis=1)
x.head(2)
```

Out[93]:

	Rating	Seating Capacity	Average Meal Price	Marketing Budget	Social Media Followers	Chef Experience Years	Number of Reviews	Avg Review Length	Ambience Score	Service Quality Score	...	Weekday Reservations	Downtown	Rural	Suburban	American	French	Indian	Italian	Japan
0	4.0	38	73.98	2224	23406	13	185	161.924906	1.3	7.0	...	4	0	1	0	0	0	0	0	
1	3.2	76	28.11	4416	42741	8	533	148.759717	2.6	3.4	...	6	1	0	0	0	0	0	0	

2 rows × 22 columns

In [94]:
```
y = data['Revenue']
y.head()
```

Out[94]:
```
0     638945.52
1     490207.83
2     541368.62
3     404556.80
4    1491046.35
Name: Revenue, dtype: float64
```

Let's train the model

Starting with Linear Regression Model

In [101…
```
from sklearn.model_selection import train_test_split

x_train , x_test, y_train, y_test = train_test_split(x , y, test_size=0.4)

print(f"x_train:{x_train.shape} , x_test:{x_test.shape} y_train:{y_train.shape} , y_test:{y_test.shape}")
```

```
x_train:(5020, 22) , x_test:(3348, 22) y_train:(5020,) , y_test:(3348,)
```

In [102…
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(x_train, y_train)
```

Out[102]:
```
▾ LinearRegression
LinearRegression()
```

In [103…
```
y_pred = model.predict(x_test)
```

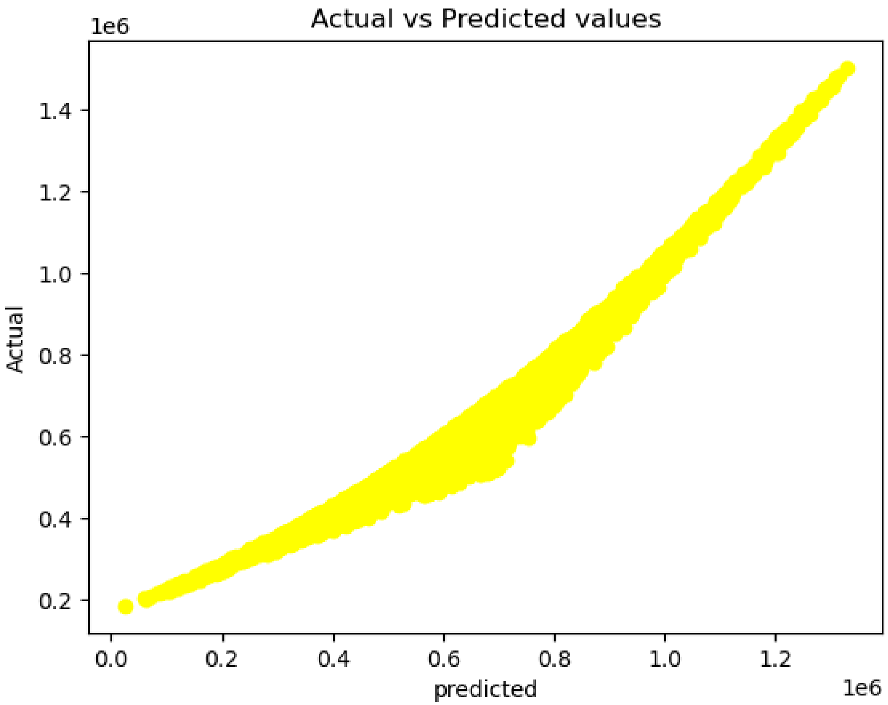
In [104…
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
mse = mean_squared_error(y_test, y_pred)
r2Score = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Squared Error: ", mse)
print("R squared Error: " , r2Score)
print("Mean Absolute Error: ", mae)
```

```
print("R2 score for training: ",  model.score(x_train,y_train))
print(model.score(x_test, y_test))
```

```
Mean Squared Error:  3033168683.267786
R squared Error:  0.9572660348922483
Mean Absolute Error:  41909.96555274624
R2 score for training:  0.9584679045757779
0.9572660348922483
```

In [114…

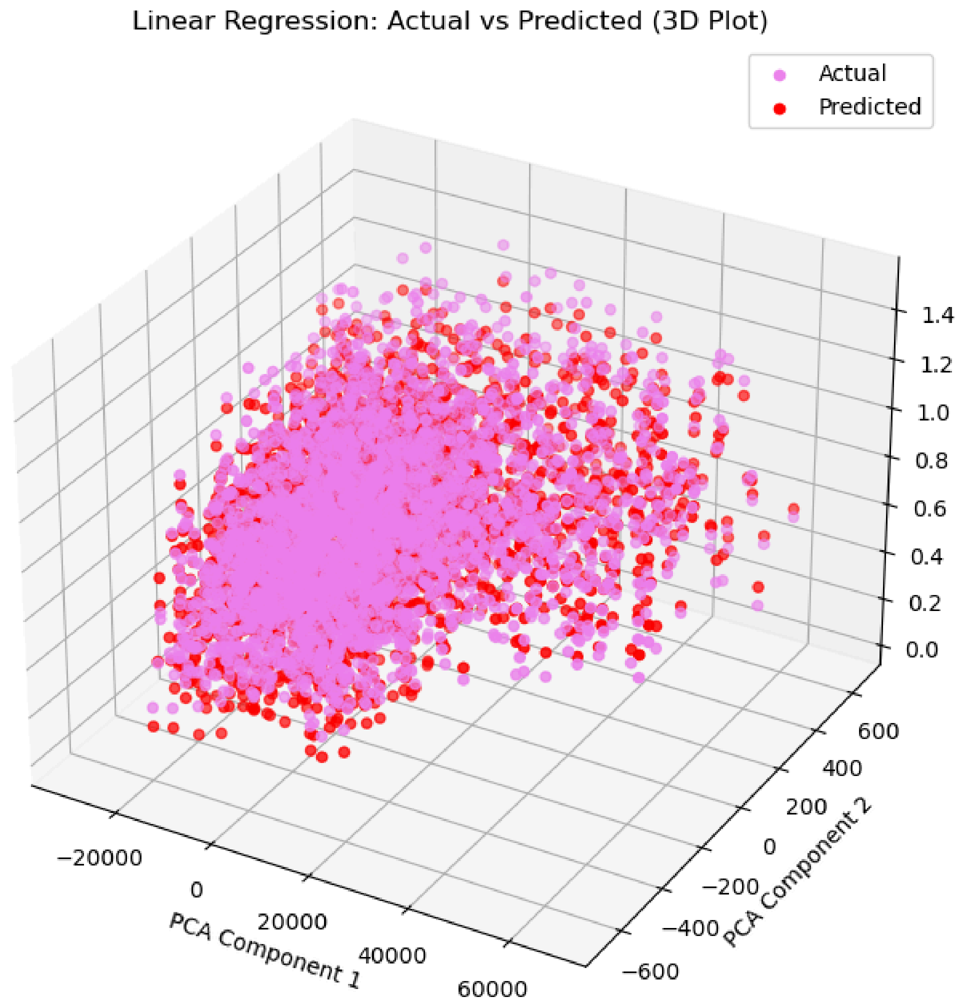
```
plt.scatter(y_pred, y_test, color='yellow')
plt.xlabel("predicted")
plt.ylabel("Actual")
plt.title("Actual vs Predicted values")
plt.show()
```



In [118…

```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
x_train_pca = pca.fit_transform(x_train)
x_test_pca = pca.transform(x_test)

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# actual values
ax.scatter(x_test_pca[:, 0], x_test_pca[:, 1], y_test, color='violet', label='Actual')
```

```python
# predicted values
ax.scatter(x_test_pca[:, 0], x_test_pca[:, 1], y_pred, color='red', label='Predicted')
ax.set_xlabel('PCA Component 1')
ax.set_ylabel('PCA Component 2')
ax.set_zlabel('Y')
ax.set_title('Linear Regression: Actual vs Predicted (3D Plot)')
ax.legend()
plt.show()
```



Linear Regression Accuracy = 95.72%

Do these for Random Forest also !

In [121…
```
from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor()
clf.fit(x_train, y_train)
```

Out[121]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

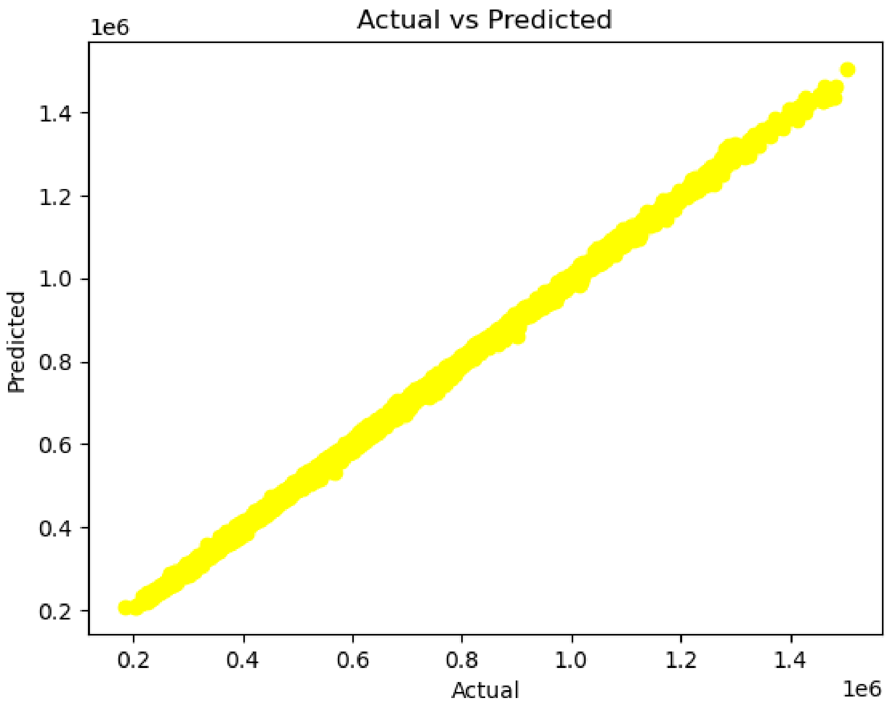
In [109…
```
y2_pred = clf.predict(x_test)
```

In [110…
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
mse = mean_squared_error(y_test, y2_pred)
r2Score = r2_score(y_test, y2_pred)
mae = mean_absolute_error(y_test, y2_pred)
print("Mean Squared Error: ", mse)
print("R squared Error: " , r2Score)
print("Mean Absolute Error: ", mae)
print("R2 score for training: ",  clf.score(x_train,y_train))
print("R2 score for testing: ", clf.score(x_test, y_test))
clf.score(x_train, y_train)
```

```
Mean Squared Error:  66913733.25353832
R squared Error:  0.9990572600996904
Mean Absolute Error:  6361.774014516131
R2 score for training:  0.9998523578720899
R2 score for testing:  0.9990572600996904
```

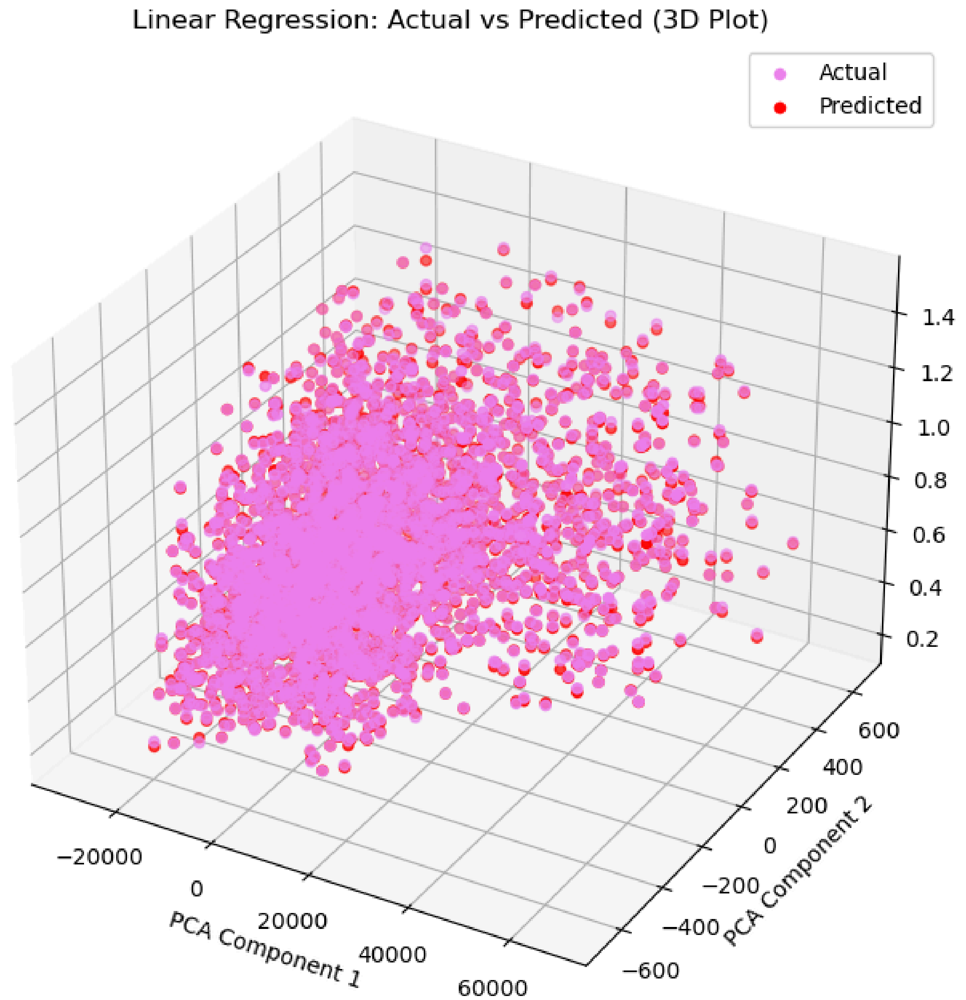
Out[110]:
```
0.9998523578720899
```

In [119…
```
plt.scatter(y_test, y2_pred , color='yellow')
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs Predicted")
plt.show()
```



In [120…

```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
x_train_pca = pca.fit_transform(x_train)
x_test_pca = pca.transform(x_test)

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# actual values
ax.scatter(x_test_pca[:, 0], x_test_pca[:, 1], y_test, color='violet', label='Actual')

# predicted values
ax.scatter(x_test_pca[:, 0], x_test_pca[:, 1], y2_pred, color='red', label='Predicted')

ax.set_xlabel('PCA Component 1')
ax.set_ylabel('PCA Component 2')
ax.set_zlabel('Y')
ax.set_title('Linear Regression: Actual vs Predicted (3D Plot)')
ax.legend()

plt.show()
```



In [113…

```
from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor(n_estimators=120)
clf.fit(x_train, y_train)
clf.score(x_train, y_train)
```

Out[113]:

```
0.9998547400585148
```

Random Forest Accuracy = 99.98%

conclusion

Accuracy of the selected two models are-

1. Linear Regression Model: 95.72%
2. Random Forest Model: 99.98%

So selecting 'Random Forest Model' as a model deployment.

This marks as the end of the project.

- Thank You -

In [ ]: