🌟 Team Stars 🌟

Member	ID	Badge
محمد مسعد نعيم محمد	23011492	🏆 **Linear Regression**
عبدالرحمن احمد عبدالمنعم	23011311	🚀 **Polynomial Regression**
محمد علاء محمد احمد البرعي	23011481	💡 **Logistic Regression**

🔴 For The Source Code - Click Here for Github Repo

Import libraries

In [4]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.preprocessing import StandardScaler, LabelEncoder, PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score, confusion_matrix, Confusi
```

Exploratory Data Analysis (EDA)

In [5]:
```
SM_data = pd.read_csv("https://raw.githubusercontent.com/mohamed8905/Regression-Mod
SM_data
```

Out[5]:

	Podcast_Name	Episode_Title	Episode_Length_minutes	Genre	Host_Popularity_per
0	Healthy Living	Episode 77	99.25	Health	
1	Mystery Matters	Episode 6	19.43	True Crime	
2	Current Affairs	Episode 1	117.03	News	
3	Mystery Matters	Episode 38	16.97	True Crime	
4	Humor Hub	Episode 73	83.48	Comedy	
...	...	...	...	...	
52495	Home & Living	Episode 17	24.81	Lifestyle	
52496	Melody Mix	Episode 9	92.15	Music	
52497	Comedy Corner	Episode 24	112.27	Comedy	
52498	Business Briefs	Episode 85	NaN	Business	
52499	Healthy Living	Episode 54	94.35	Health	

52500 rows × 11 columns

In [6]:
```
SM_data['Episode_Sentiment'] = SM_data['Episode_Sentiment'].map({'Negative':-1, 'Ne
SM_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52500 entries, 0 to 52499
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Podcast_Name                 52500 non-null  object 
 1   Episode_Title                52500 non-null  object 
 2   Episode_Length_minutes       47254 non-null  float64
 3   Genre                        52500 non-null  object 
 4   Host_Popularity_percentage   52500 non-null  float64
 5   Publication_Day              52500 non-null  object 
 6   Publication_Time             52500 non-null  object 
 7   Guest_Popularity_percentage  47242 non-null  float64
 8   Number_of_Ads                52500 non-null  int64  
 9   Episode_Sentiment            52500 non-null  int64  
 10  Listening_Time_minutes       47105 non-null  float64
dtypes: float64(4), int64(2), object(5)
memory usage: 4.4+ MB
```

In [7]:
```
# Dropping null values
null_rows = SM_data.isnull().any(axis=1).sum()
print(f"Rows with at least one null: {null_rows}")
SM_data[SM_data.isnull().any(axis=1)]
```

```
Rows with at least one null: 14337
```

Out[7]:

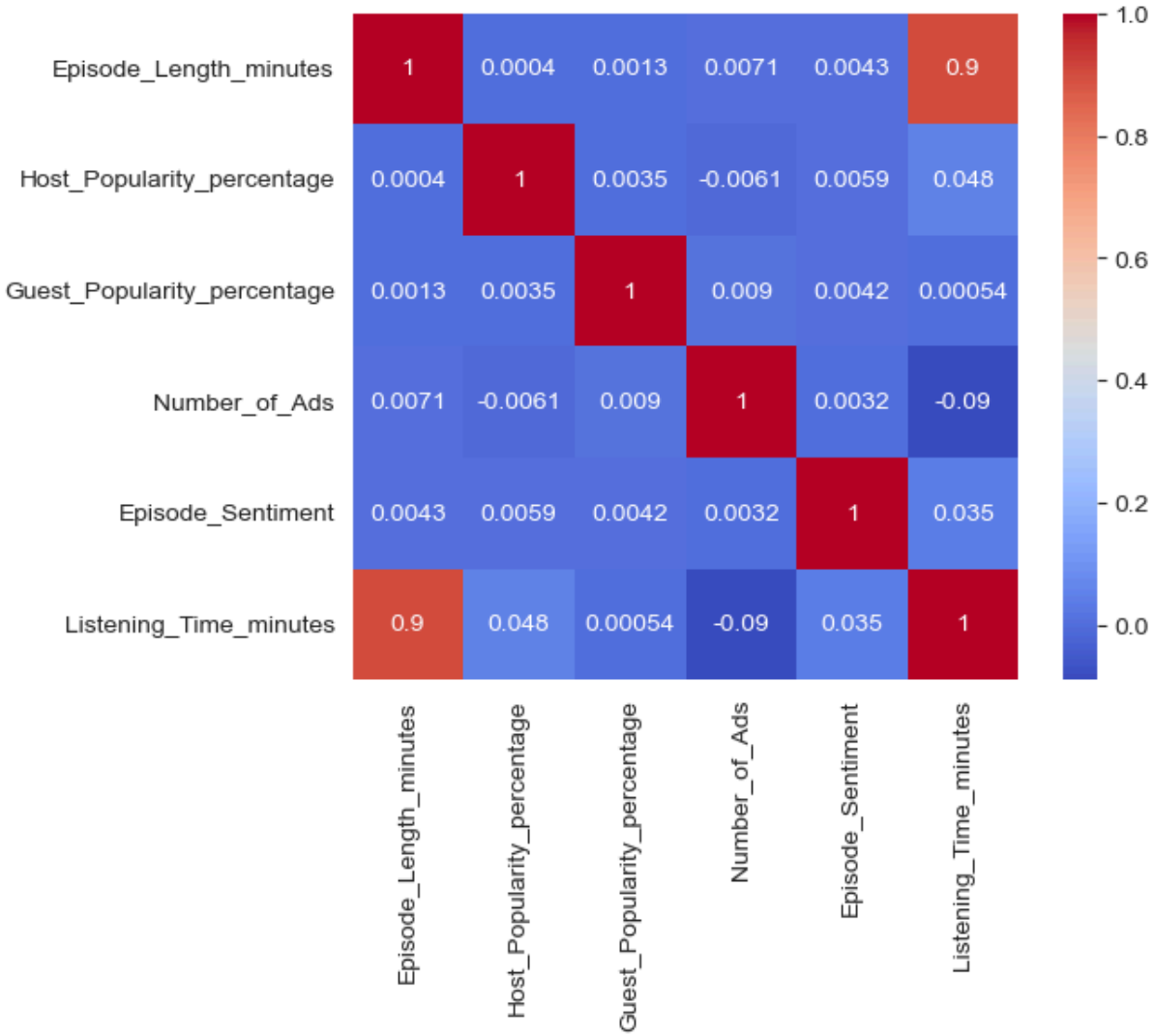
	Podcast_Name	Episode_Title	Episode_Length_minutes	Genre	Host_Popularity_
4	Humor Hub	Episode 73	83.48	Comedy	
5	Money Matters	Episode 87	28.06	Business	
10	Joke Junction	Episode 68	34.97	Comedy	
22	Digital Digest	Episode 48	85.64	Technology	
23	Learning Lab	Episode 18	NaN	Education	
...	...	...	...	...	
52474	News Roundup	Episode 66	44.52	News	
52476	Health Hour	Episode 16	99.19	Health	
52491	Sport Spot	Episode 86	29.47	Sports	
52493	Brain Boost	Episode 71	72.87	Education	
52498	Business Briefs	Episode 85	NaN	Business	

14337 rows × 11 columns

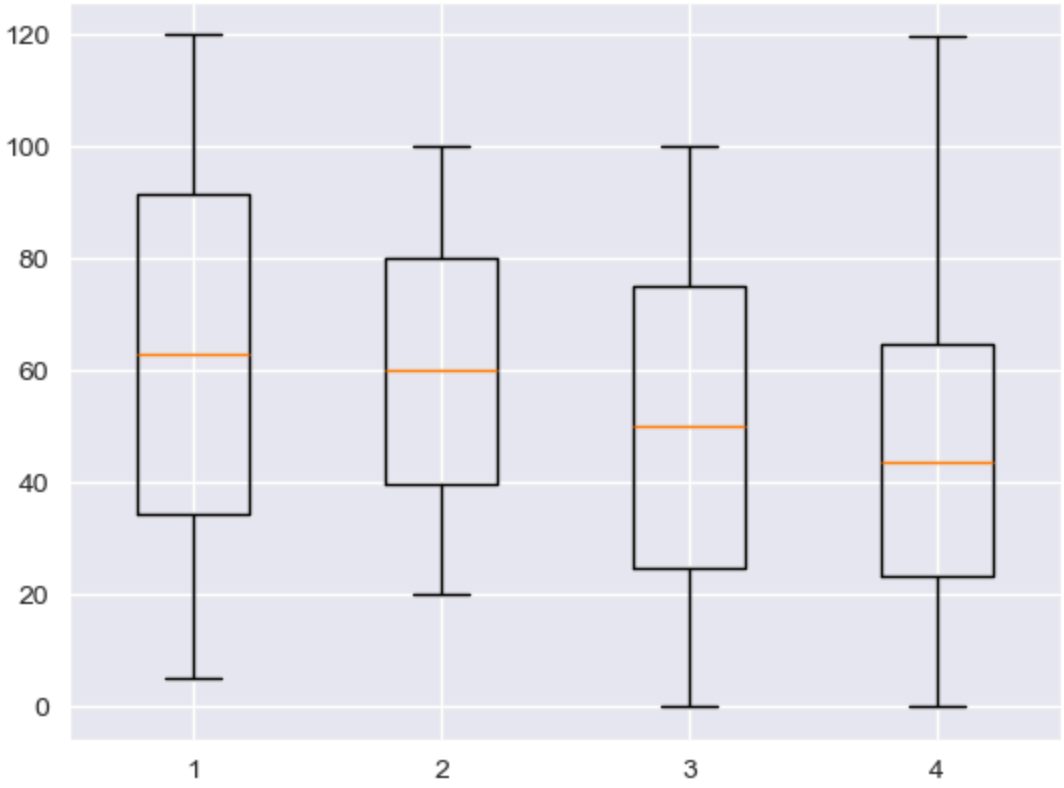
In [8]:
```
SM_data.dropna(inplace=True, axis=0)
SM_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 38163 entries, 0 to 52499
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Podcast_Name                 38163 non-null  object 
 1   Episode_Title                38163 non-null  object 
 2   Episode_Length_minutes       38163 non-null  float64
 3   Genre                        38163 non-null  object 
 4   Host_Popularity_percentage   38163 non-null  float64
 5   Publication_Day              38163 non-null  object 
 6   Publication_Time             38163 non-null  object 
 7   Guest_Popularity_percentage  38163 non-null  float64
 8   Number_of_Ads                38163 non-null  int64  
 9   Episode_Sentiment            38163 non-null  int64  
 10  Listening_Time_minutes       38163 non-null  float64
dtypes: float64(4), int64(2), object(5)
memory usage: 3.5+ MB
```

In [9]:
```
# Heatmap to find correlation
num_DM_data = SM_data.select_dtypes(include=["number"])
sns.heatmap(num_DM_data.corr(), annot=True, square=True, cmap='coolwarm')
plt.show()
```



In [10]:
```
# Checking outliers
plt.boxplot(num_DM_data.drop(['Number_of_Ads', 'Episode_Sentiment'], axis=1))
plt.show()
```



Preprocessing

In [11]:
```
# Splitting data to input / output
x = num_DM_data.drop('Listening_Time_minutes', axis=1)
y = num_DM_data['Listening_Time_minutes']
x
```

Out[11]:

	Episode_Length_minutes	Host_Popularity_percentage	Guest_Popularity_percentage	N
0	99.25	21.37	70.22	
1	19.43	47.19	75.15	
2	117.03	96.33	57.95	
3	16.97	25.73	24.19	
6	63.77	58.84	82.20	
...	...	...	...	
52494	33.35	63.23	90.81	
52495	24.81	66.15	98.63	
52496	92.15	89.61	25.82	
52497	112.27	26.33	55.29	
52499	94.35	91.08	92.54	

38163 rows × 5 columns

In [12]:
```
y
```

Out[12]:
```
0         55.158695
1          7.686559
2        110.064645
3         12.000380
6         57.827346
             ...
52494     32.133509
52495     20.573795
52496     76.198459
52497    107.602135
52499     52.102024
Name: Listening_Time_minutes, Length: 38163, dtype: float64
```

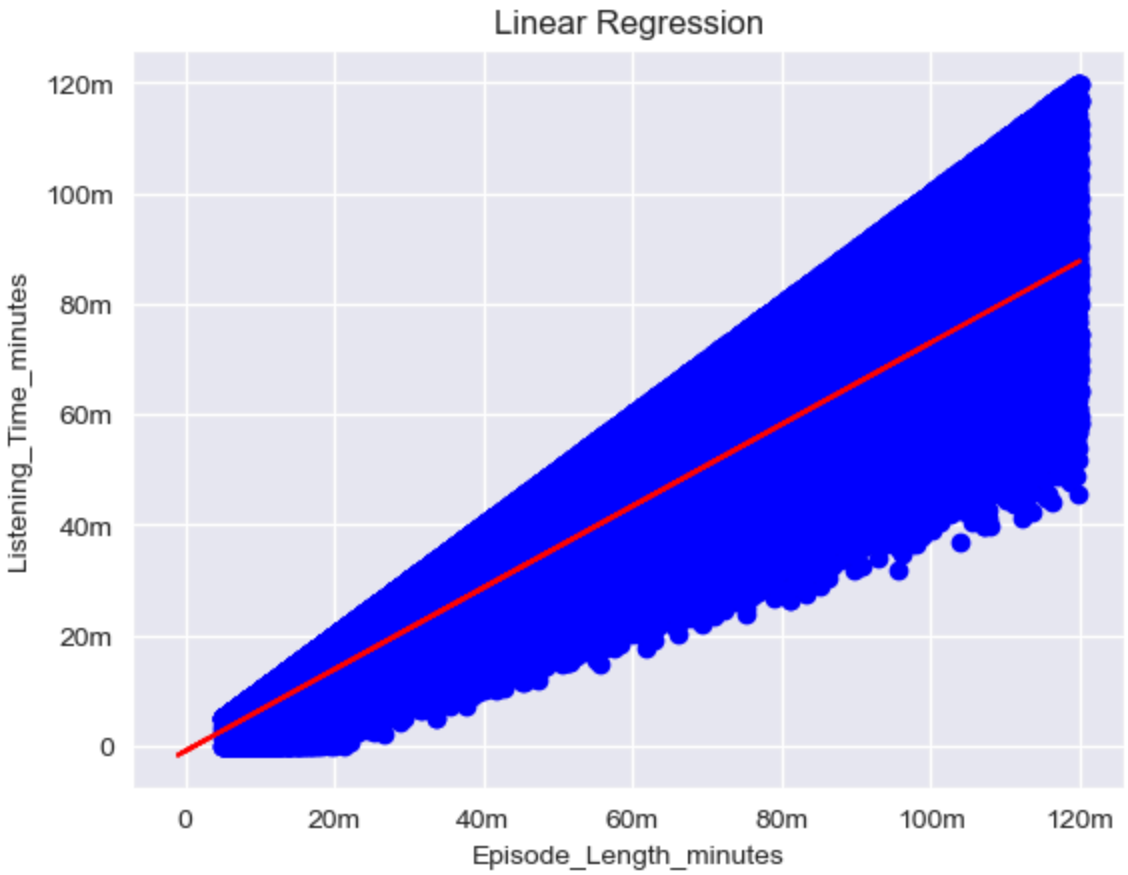
In [13]:
```
# Splitting data to train / test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

Linear Regression (Simple)

In [14]:
```
# Building Simple Linear model
reg = LinearRegression()
reg.fit(x_train[["Episode_Length_minutes"]], y_train)
print(f"The Equation is: Y = {reg.coef_[0]:0.3f} X + {reg.intercept_:0.3f}")
```

```
The Equation is: Y = 0.739 X + -0.905
```

In [15]:
```
plt.scatter(x_train[["Episode_Length_minutes"]], y_train , color='blue')
plt.plot(x_train, reg.coef_[0]*x_train + reg.intercept_, '-r')
plt.xlabel('Episode_Length_minutes')
plt.ylabel('Listening_Time_minutes')
plt.title("Linear Regression")
plt.xticks([0,20,40,60,80,100,120], ['0','20m','40m','60m','80m','100m','120m'])
plt.yticks([0,20,40,60,80,100,120], ['0','20m','40m','60m','80m','100m','120m'])
plt.show()
```



In [16]:
```
# Making prediction
y_hat = reg.predict(x_test[["Episode_Length_minutes"]])
```

In [17]:
```
# testing score and error
r2 = r2_score(y_test, y_hat)
print(f"Model Accuracy (R² Score): {r2*100:.2f}%")
```

```
Model Accuracy (R² Score): 81.20%
```

In [18]:
```
mse = mean_squared_error(y_test, y_hat)
print(f"Mean Squared Error: {mse:.2f}")
```

```
Mean Squared Error: 137.99
```

Multiple Linear Regression

In [19]:
```
# Building Multiple Linear model
reg2 = LinearRegression()
reg2.fit(x_train, y_train)

eq = ""
for i in range(len(reg2.coef_)):
    if i==0:
        eq = eq + f"Y = {reg2.intercept_:0.3f} + {reg2.coef_[i]:0.3f} X{i+1}"
    else :
        eq = eq + f" + {reg2.coef_[i]:0.3f} X{i+1}"

print(f"The Equation is: {eq}")
```

```
The Equation is: Y = -0.790 + 0.739 X1 + 0.056 X2 + 0.000 X3 + -2.343 X4 + 1.017 X5
```

In [20]:
```
# Prediction
y_hat2 = reg2.predict(x_test)
```

In [21]:
```
# Score and error
r2_2 = r2_score(y_test, y_hat2)
print(f"Model Accuracy (R² Score): {r2_2*100:.2f}%")
```

```
Model Accuracy (R² Score): 82.41%
```

In [22]:
```
mse2 = mean_squared_error(y_test, y_hat)
print(f"Mean Squared Error: {mse2:.2f}")
```

```
Mean Squared Error: 137.99
```

The Multiple Linear Regression Model has higher r2 score.

Polynomial Regression

In [23]:
```
Poly_reg = pd.read_csv("https://raw.githubusercontent.com/mohamed8905/Regression-Mo

# Clean the data: Drop missing or duplicate rows
Poly_reg = Poly_reg.dropna().drop_duplicates()

# Convert columns to numeric
Poly_reg["price"] = pd.to_numeric(Poly_reg["price"], errors='coerce')
Poly_reg["log_price"] = pd.to_numeric(Poly_reg["log_price"], errors='coerce')
Poly_reg = Poly_reg.dropna(subset=["price", "log_price"])

Poly_reg
```

Out[23]:

	carat	cut	color	clarity	depth	table	price	x	y	z	log_price	v
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43	5.786897	38.
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31	5.786897	34.
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31	5.789960	38.
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63	5.811141	46.
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75	5.814131	51.
...	...	...	...	...	...	...	...	...	...	...	...	
53935	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50	7.921898	115.
53936	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61	7.921898	118.
53937	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56	7.921898	114.
53938	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74	7.921898	140.
53939	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64	7.921898	124.

53794 rows × 12 columns

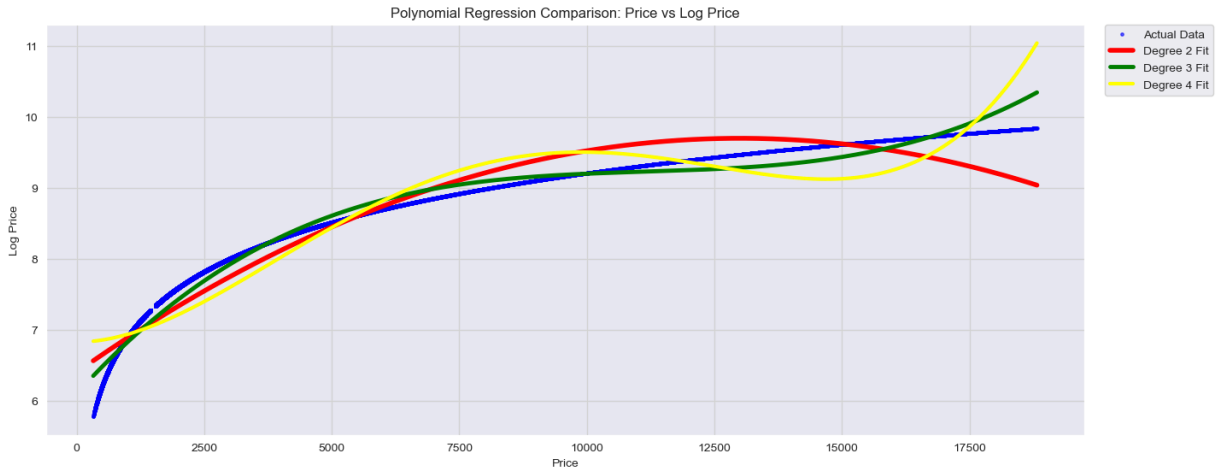
In [24]:

```
# Comparing 2nd, 3rd & 4th degree
X = Poly_reg["price"].values.reshape(-1, 1)
y = Poly_reg["log_price"].values

degrees = [2, 3, 4]
models = {}
predictions = {}
metrics = {}

for degree in degrees:
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(X)

    model = LinearRegression()
    model.fit(X_poly, y)
    y_pred = model.predict(X_poly)

    models[degree] = model
    predictions[degree] = y_pred

    mse = mean_squared_error(y, y_pred)
    r2 = r2_score(y, y_pred)
    metrics[degree] = {
        'MSE': mse,
        'R2': r2,
        'Accuracy%': r2 * 100
    }
```

In [25]:
```
for degree in degrees:
    print(f"\nDegree {degree} Results:")
    print(f"Mean Squared Error: {metrics[degree]['MSE']:.4f}")
    print(f"R² Score: {metrics[degree]['R2']:.4f}")
    print(f"Accuracy: {metrics[degree]['Accuracy%']:.2f}%")
```

```
Degree 2 Results:
Mean Squared Error: 0.0535
R² Score: 0.9480
Accuracy: 94.80%

Degree 3 Results:
Mean Squared Error: 0.0181
R² Score: 0.9824
Accuracy: 98.24%

Degree 4 Results:
Mean Squared Error: 0.1106
R² Score: 0.8925
Accuracy: 89.25%
```

In [26]:
```
plt.figure(figsize=(13.5, 6))

plt.scatter(X.flatten(), y, color='blue', s=6, alpha=0.6, label='Actual Data')

# Plot polynomial fits
colors = ['red', 'green', 'yellow']
sorted_idx = np.argsort(X.flatten())

for i, degree in enumerate(degrees):
    plt.plot(X[sorted_idx].flatten(), predictions[degree][sorted_idx],
             color=colors[i], linewidth=4-0.5*i,
             label=f'Degree {degree} Fit')

plt.title('Polynomial Regression Comparison: Price vs Log Price')
plt.xlabel('Price')
plt.ylabel('Log Price')

# Adjust legend position
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left', borderaxespad=0.)

# Grid settings
plt.grid(True, color='LightGray', linewidth=1)

# Adjust margins
plt.subplots_adjust(left=0.05, right=0.95, bottom=0.1, top=0.9)

plt.show()
```



In [27]:
```
# The models
for degree in degrees:
    print(f"\nDegree {degree} Polynomial Equation:")
    equation = f"Y = {models[degree].intercept_}"
    for power, coef in enumerate(models[degree].coef_[1:], 1):
        equation += f" + {coef} X^{power}"
    print(equation)
```

```
Degree 2 Polynomial Equation:
Y = 6.404746158769985 + 0.0005069698301360856 X^1 + -1.9492525316013317e-08 X^2

Degree 3 Polynomial Equation:
Y = 6.103030973684653 + 0.0007968307881509356 X^1 + -6.94722237262789e-08 X^2 + 2.07
8127513751212e-12 X^3

Degree 4 Polynomial Equation:
Y = 6.830825765370128 + 3.155340445445842e-11 X^1 + 1.2340557074382692e-07 X^2 + -1.
3904774232973876e-11 X^3 + 4.239562356902578e-16 X^4
```

Logistic Regression with Classification

In [28]:
```
Log_reg=pd.read_csv("https://raw.githubusercontent.com/mohamed8905/Regression-Model
Log_reg
```

Out[28]:

	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights
0	INN00001	2	0	1	2
1	INN00002	2	0	2	3
2	INN00003	1	0	2	1
3	INN00004	2	0	0	2
4	INN00005	2	0	1	1
...	...	...	...	...	...
36270	INN36271	3	0	2	6
36271	INN36272	2	0	1	3
36272	INN36273	2	0	2	6
36273	INN36274	2	0	0	3
36274	INN36275	2	0	1	2

36275 rows × 19 columns

In [29]:
```
Log_reg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36275 entries, 0 to 36274
Data columns (total 19 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   Booking_ID                            36275 non-null  object 
 1   no_of_adults                          36275 non-null  int64  
 2   no_of_children                        36275 non-null  int64  
 3   no_of_weekend_nights                  36275 non-null  int64  
 4   no_of_week_nights                     36275 non-null  int64  
 5   type_of_meal_plan                     36275 non-null  object 
 6   required_car_parking_space            36275 non-null  int64  
 7   room_type_reserved                    36275 non-null  object 
 8   lead_time                             36275 non-null  int64  
 9   arrival_year                          36275 non-null  int64  
 10  arrival_month                         36275 non-null  int64  
 11  arrival_date                          36275 non-null  int64  
 12  market_segment_type                   36275 non-null  object 
 13  repeated_guest                        36275 non-null  int64  
 14  no_of_previous_cancellations          36275 non-null  int64  
 15  no_of_previous_bookings_not_canceled  36275 non-null  int64  
 16  avg_price_per_room                    36275 non-null  float64
 17  no_of_special_requests                36275 non-null  int64  
 18  booking_status                        36275 non-null  object 
dtypes: float64(1), int64(13), object(5)
memory usage: 5.3+ MB
```

In [30]:
```
Log_reg.describe()
```

Out[30]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	required_ca
count	36275.000000	36275.000000	36275.000000	36275.000000	
mean	1.844962	0.105279	0.810724	2.204300	
std	0.518715	0.402648	0.870644	1.410905	
min	0.000000	0.000000	0.000000	0.000000	
25%	2.000000	0.000000	0.000000	1.000000	
50%	2.000000	0.000000	1.000000	2.000000	
75%	2.000000	0.000000	2.000000	3.000000	
max	4.000000	10.000000	7.000000	17.000000	

In [31]:
```
Log_reg.isna().sum()
```

Out[31]:
```
Booking_ID                            0
no_of_adults                          0
no_of_children                        0
no_of_weekend_nights                  0
no_of_week_nights                     0
type_of_meal_plan                     0
required_car_parking_space            0
room_type_reserved                    0
lead_time                             0
arrival_year                          0
arrival_month                         0
arrival_date                          0
market_segment_type                   0
repeated_guest                        0
no_of_previous_cancellations          0
no_of_previous_bookings_not_canceled  0
avg_price_per_room                    0
no_of_special_requests                0
booking_status                        0
dtype: int64
```

In [32]:
```
Log_reg.duplicated().sum()
```

Out[32]:
```
np.int64(0)
```

In [33]:
```
# Encoding
#label encoding to type_of_meal_plan
Log_reg['type_of_meal_plan']=Log_reg['type_of_meal_plan'].str.replace("Meal Plan","
Log_reg['type_of_meal_plan']=Log_reg['type_of_meal_plan'].str.replace("Not Selected
Log_reg['type_of_meal_plan']=Log_reg['type_of_meal_plan'].astype(int)
#label encoding to room_type_reserved
Log_reg['room_type_reserved']=Log_reg['room_type_reserved'].str.replace("Room_Type"
Log_reg['room_type_reserved']=Log_reg['room_type_reserved'].astype(int)
#label encoding to booking_status
encoder = LabelEncoder()
```

```
Log_reg['booking_status'] = encoder.fit_transform(Log_reg['booking_status'])
Log_reg['booking_status']=Log_reg['booking_status'].astype(int)
Log_reg['booking_status']=Log_reg['booking_status']^1
#label encoding to market_segment_type
encoder = LabelEncoder()
Log_reg['market_segment_type'] = encoder.fit_transform(Log_reg['market_segment_type
Log_reg['booking_status']=Log_reg['booking_status'].astype(int)
Log_reg
```

Out[33]:

	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights
0	INN00001	2	0	1	2
1	INN00002	2	0	2	3
2	INN00003	1	0	2	1
3	INN00004	2	0	0	2
4	INN00005	2	0	1	1
...	...	...	...	...	...
36270	INN36271	3	0	2	6
36271	INN36272	2	0	1	3
36272	INN36273	2	0	2	6
36273	INN36274	2	0	0	3
36274	INN36275	2	0	1	2

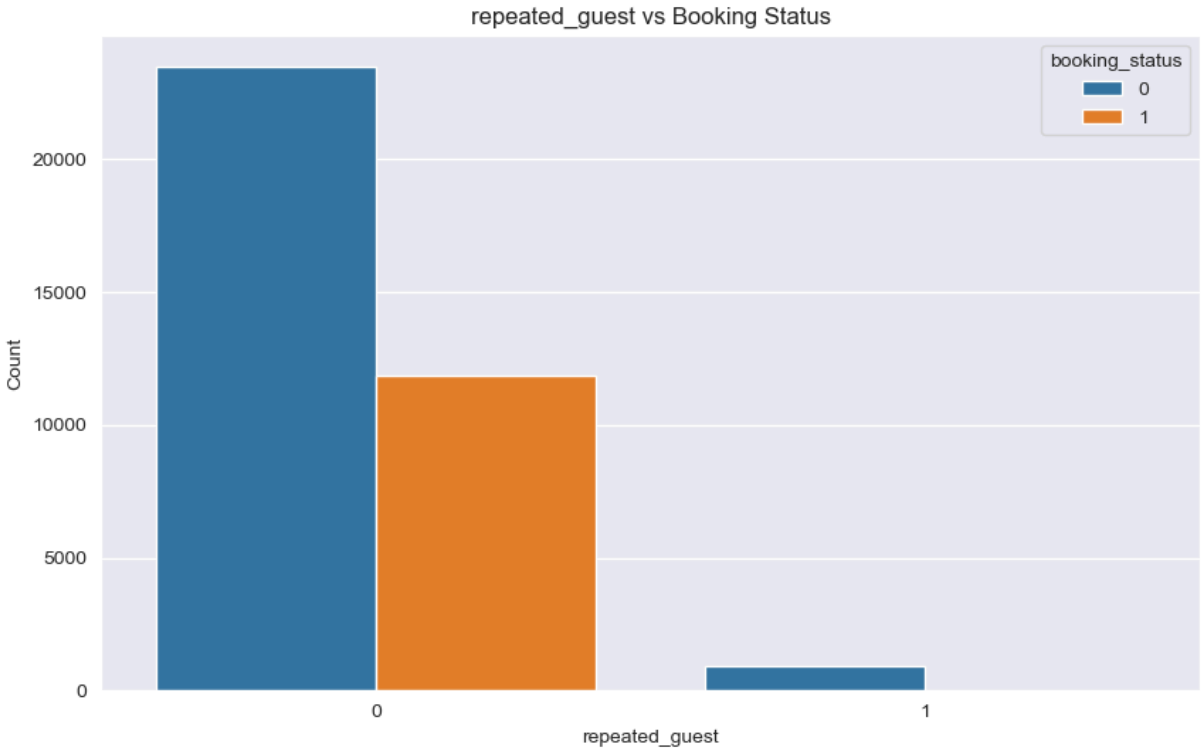
36275 rows × 19 columns

In [34]:

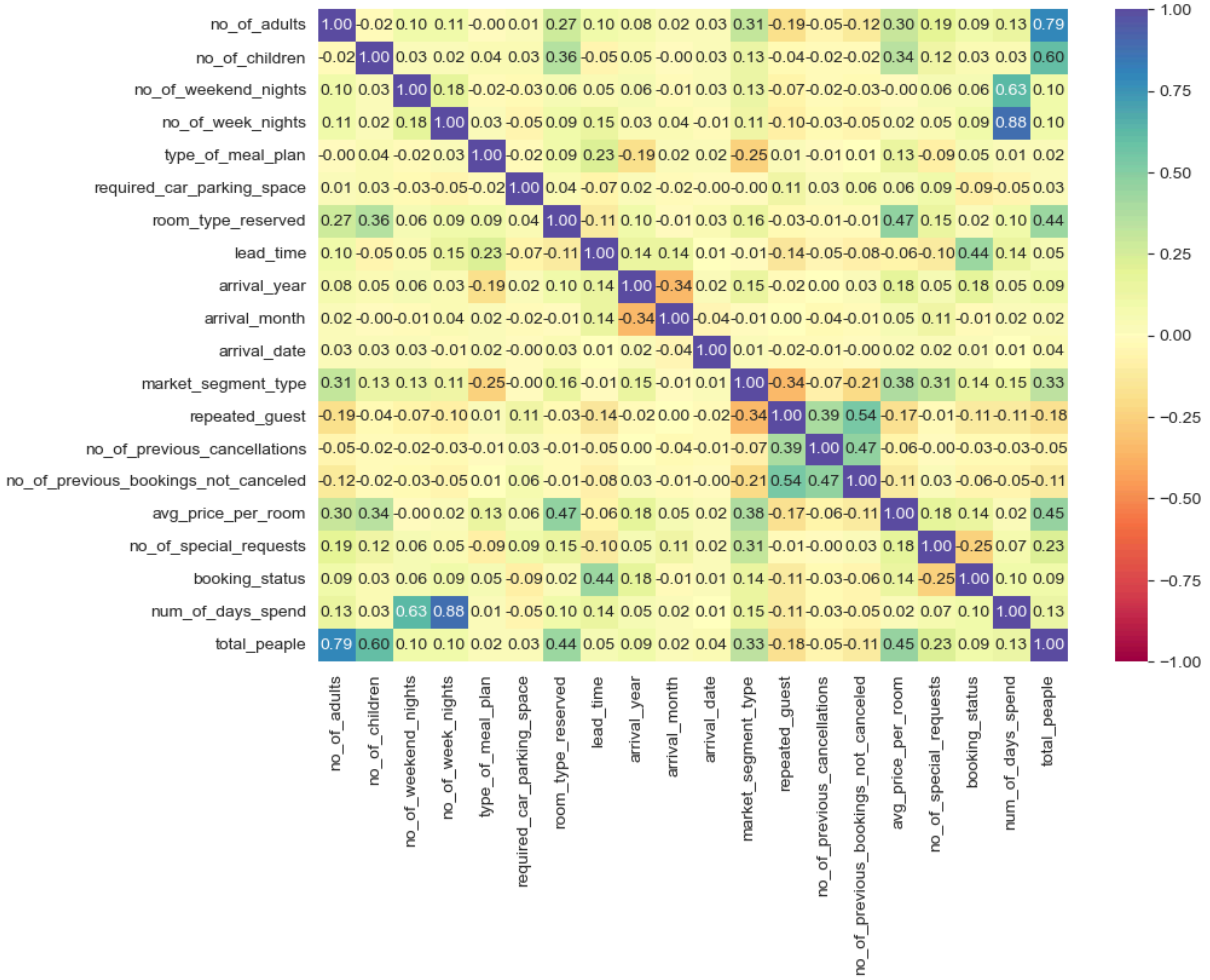
```
# Feature Engineering
Log_reg['num_of_days_spend']=Log_reg['no_of_weekend_nights']+Log_reg['no_of_week_ni
Log_reg['total_peaple']=Log_reg['no_of_adults']+Log_reg['no_of_children']
```

In [35]:

```
plt.figure(figsize=(10, 6))
sns.countplot(data=Log_reg, x='repeated_guest', hue='booking_status')
plt.title('repeated_guest vs Booking Status')
plt.xlabel('repeated_guest')
plt.ylabel('Count')

plt.show()
```



In [36]:
```
# Heatmap to show correlation between features
normacalCol=Log_reg.drop(['Booking_ID'],axis=1)
plt.figure(figsize=(10, 7))
sns.heatmap(
     normacalCol.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```



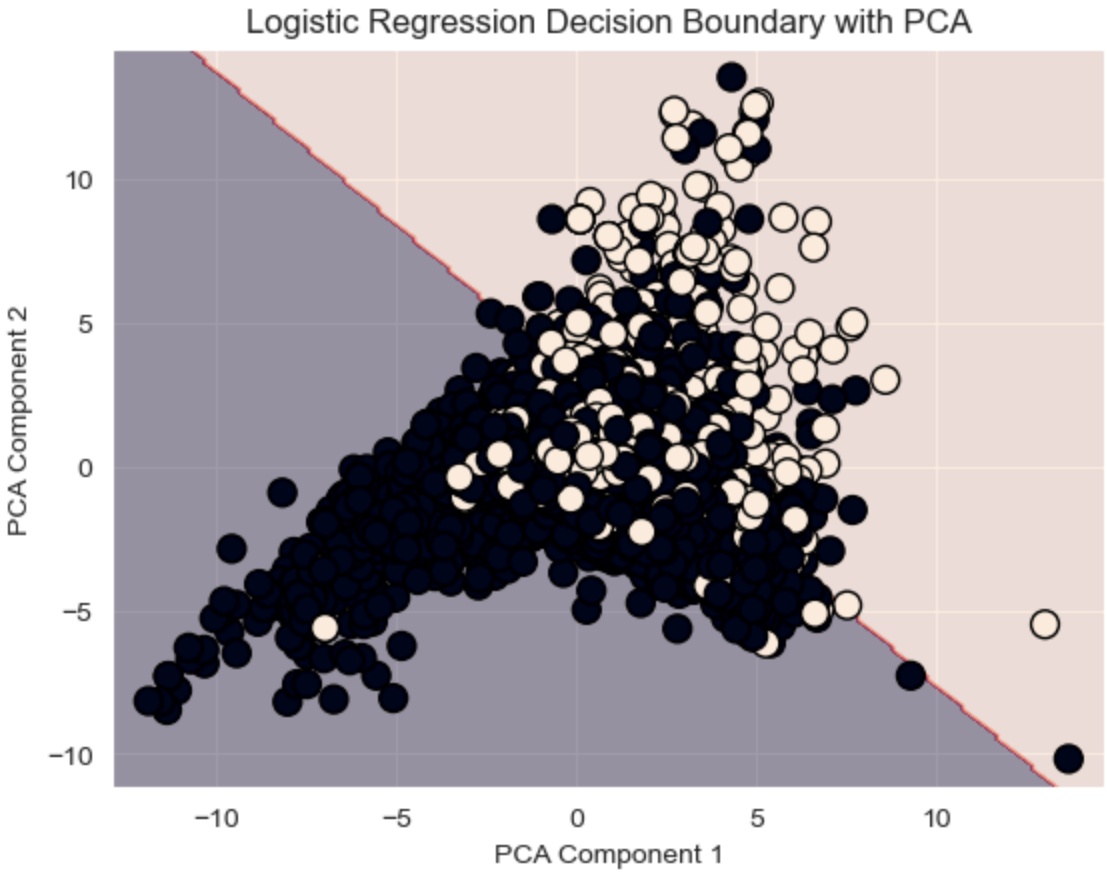
In [37]:
```
# Split data to train and test
x_data=normacalCol.drop(['booking_status'],axis=1)
y_data=Log_reg['booking_status']
x_train, x_test, y_train, y_test = train_test_split(x_data,y_data, test_size=0.2, r
```

In [38]:
```
# Feature Scaling
scaler = StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.transform(x_test)
```

In [39]:
```
# Fit data to make Logistic Regression
log=LogisticRegression()
log.fit(x_train,y_train)
# Prediction
predict=log.predict(x_test)
```

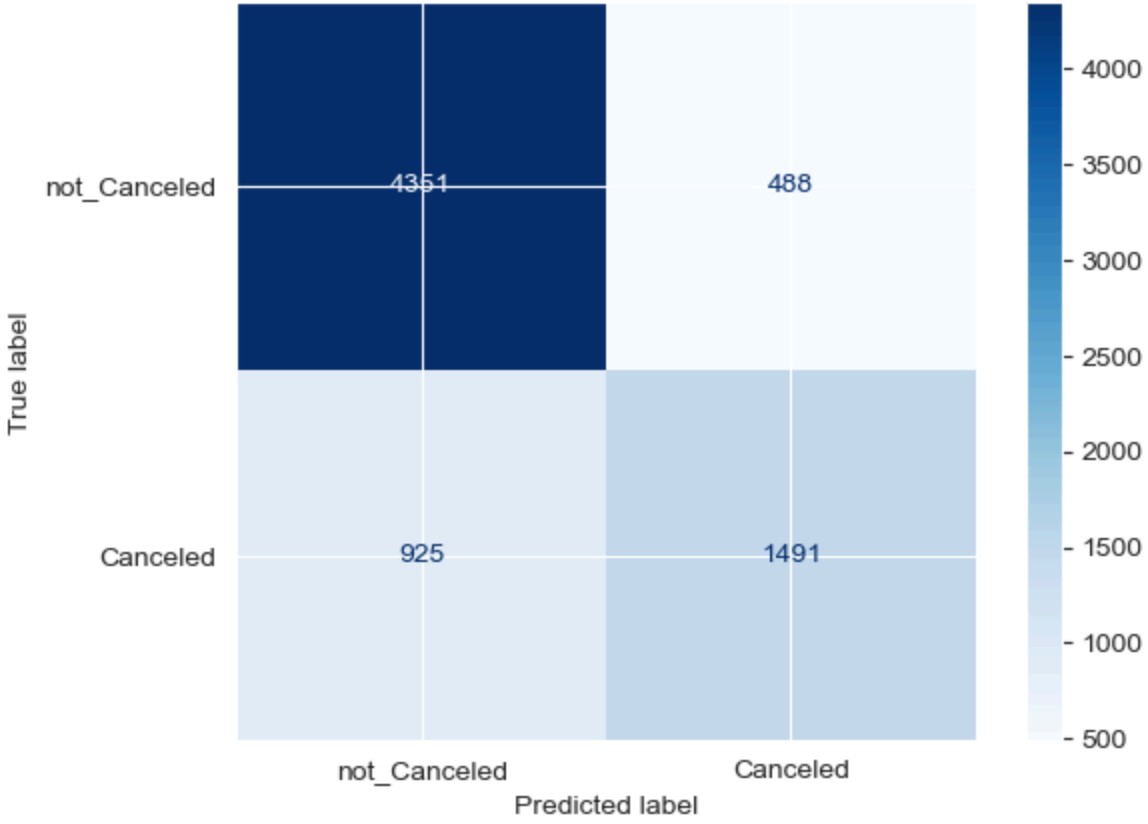
In [40]:
```
# Logistic Regression Decision Boundary with PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(x_train)
xx, yy = np.meshgrid(np.linspace(X_pca[:, 0].min() - 1, X_pca[:, 0].max() + 1, 200)
                     np.linspace(X_pca[:, 1].min() - 1, X_pca[:, 1].max() + 1, 200)
Z = log.predict(pca.inverse_transform(np.c_[xx.ravel(), yy.ravel()]))
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_train, edgecolors='k', marker='o', s=100)
```

```
plt.title("Logistic Regression Decision Boundary with PCA")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")

plt.show()
```



In [41]:
```
# Confusion Matrix
cm = confusion_matrix(y_test, predict)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['not_Canceled',
disp.plot(cmap='Blues')
```

Out[41]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x179f06d7d10>



In [42]:
```
# Accuracy
accuracy = accuracy_score(y_test,predict)
print(f"Model Accuracy: {accuracy*100:.2f}%")
```

```
Model Accuracy: 80.52%
```