Importing Libraries

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Load and Preprocess Dataset

In [5]:
```
#importing the dataset
df = pd.read_csv("Rainfall.csv")
```

In [7]:
```
df.head()
```

Out[7]:

	day	pressure	maxtemp	temparature	mintemp	dewpoint	humidity	cloud	rainfa
0	1	1025.9	19.9	18.3	16.8	13.1	72	49	ye
1	2	1022.0	21.7	18.9	17.2	15.6	81	83	ye
2	3	1019.7	20.3	19.3	18.0	18.4	95	91	ye
3	4	1018.9	22.3	20.6	19.1	18.8	90	88	ye
4	5	1015.9	21.3	20.7	20.2	19.9	95	81	ye

In [9]:
```
# Convert 'rainfall' column to binary (1 for 'yes', 0 for 'no')
df['rainfall'] = df['rainfall'].str.strip().str.lower().map({'yes': 1, 'no': 0})
```

In [11]:
```
df.head()
```

Out[11]:

	day	pressure	maxtemp	temparature	mintemp	dewpoint	humidity	cloud	rainfa
0	1	1025.9	19.9	18.3	16.8	13.1	72	49	
1	2	1022.0	21.7	18.9	17.2	15.6	81	83	
2	3	1019.7	20.3	19.3	18.0	18.4	95	91	
3	4	1018.9	22.3	20.6	19.1	18.8	90	88	
4	5	1015.9	21.3	20.7	20.2	19.9	95	81	

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 366 entries, 0 to 365
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   day                     366 non-null    int64  
 1   pressure                366 non-null    float64
 2   maxtemp                 366 non-null    float64
 3   temparature             366 non-null    float64
 4   mintemp                 366 non-null    float64
 5   dewpoint                366 non-null    float64
 6   humidity                366 non-null    int64  
 7   cloud                   366 non-null    int64  
 8   rainfall                366 non-null    int64  
 9   sunshine                366 non-null    float64
 10          winddirection   365 non-null    float64
 11  windspeed               365 non-null    float64
dtypes: float64(8), int64(4)
memory usage: 34.4 KB
```

In [15]:
```
# Drop rows with missing data from the dataframe
df.dropna(inplace=True)
```

In [17]:
```
df.isnull().sum()
```

Out[17]:
```
day                     0
pressure                0
maxtemp                 0
temparature             0
mintemp                 0
dewpoint                0
humidity                0
cloud                   0
rainfall                0
sunshine                0
         winddirection  0
windspeed               0
dtype: int64
```

In [19]:
```
df.describe()
```

Out[19]:

	day	pressure	maxtemp	temparature	mintemp	dewpoint	hur
count	365.000000	365.000000	365.000000	365.000000	365.000000	365.000000	365.0
mean	15.775342	1013.764658	26.176164	23.735068	21.881644	19.973425	80.1
std	8.828584	6.409697	5.979563	5.635701	5.596385	5.997768	10.0
min	1.000000	998.500000	7.100000	4.900000	3.100000	-0.400000	36.0
25%	8.000000	1008.500000	21.200000	18.800000	17.100000	16.100000	75.0
50%	16.000000	1013.000000	27.700000	25.400000	23.700000	21.900000	80.0
75%	23.000000	1018.100000	31.200000	28.600000	26.500000	25.000000	87.0
max	31.000000	1034.600000	36.300000	32.400000	30.000000	26.700000	98.0

In [21]:
```
df.shape
```

Out[21]:
```
(365, 12)
```

In [23]:
```
df.head()
```

Out[23]:

	day	pressure	maxtemp	temparature	mintemp	dewpoint	humidity	cloud	rainfa
0	1	1025.9	19.9	18.3	16.8	13.1	72	49	
1	2	1022.0	21.7	18.9	17.2	15.6	81	83	
2	3	1019.7	20.3	19.3	18.0	18.4	95	91	
3	4	1018.9	22.3	20.6	19.1	18.8	90	88	
4	5	1015.9	21.3	20.7	20.2	19.9	95	81	

In [25]:
```
df.tail()
```

Out[25]:

	day	pressure	maxtemp	temparature	mintemp	dewpoint	humidity	cloud	rain
361	27	1022.7	18.8	17.7	16.9	15.0	84	90	
362	28	1026.6	18.6	17.3	16.3	12.8	75	85	
363	29	1025.9	18.9	17.7	16.4	13.3	75	78	
364	30	1025.3	19.2	17.3	15.2	13.3	78	86	
365	31	1026.4	20.5	17.8	15.5	13.0	74	66	

In [31]:
```
df.columns
```

Out[31]:
```
Index(['day', 'pressure ', 'maxtemp', 'temparature', 'mintemp', 'dewpoint',
       'humidity ', 'cloud ', 'rainfall', 'sunshine', '         winddirection',
       'windspeed'],
      dtype='object')
```

In [33]:
```
df.columns = df.columns.str.strip()
```

In [35]:
```
df.columns
```

Out[35]:
```
Index(['day', 'pressure', 'maxtemp', 'temparature', 'mintemp', 'dewpoint',
       'humidity', 'cloud', 'rainfall', 'sunshine', 'winddirection',
       'windspeed'],
      dtype='object')
```

In [37]:
```
# Separate independent variables (X) and target variable (y)
X = df.drop(columns=['rainfall', 'day'])
y = df['rainfall']
```

In [39]:
```
X
```

Out[39]:

	pressure	maxtemp	temparature	mintemp	dewpoint	humidity	cloud	sunshine
0	1025.9	19.9	18.3	16.8	13.1	72	49	9.3
1	1022.0	21.7	18.9	17.2	15.6	81	83	0.6
2	1019.7	20.3	19.3	18.0	18.4	95	91	0.0
3	1018.9	22.3	20.6	19.1	18.8	90	88	1.0
4	1015.9	21.3	20.7	20.2	19.9	95	81	0.0
...	...	...	...	...	...	...	...	...
361	1022.7	18.8	17.7	16.9	15.0	84	90	0.0
362	1026.6	18.6	17.3	16.3	12.8	75	85	1.0
363	1025.9	18.9	17.7	16.4	13.3	75	78	4.6
364	1025.3	19.2	17.3	15.2	13.3	78	86	1.2
365	1026.4	20.5	17.8	15.5	13.0	74	66	5.7

365 rows × 10 columns

In [41]:
```
y
```

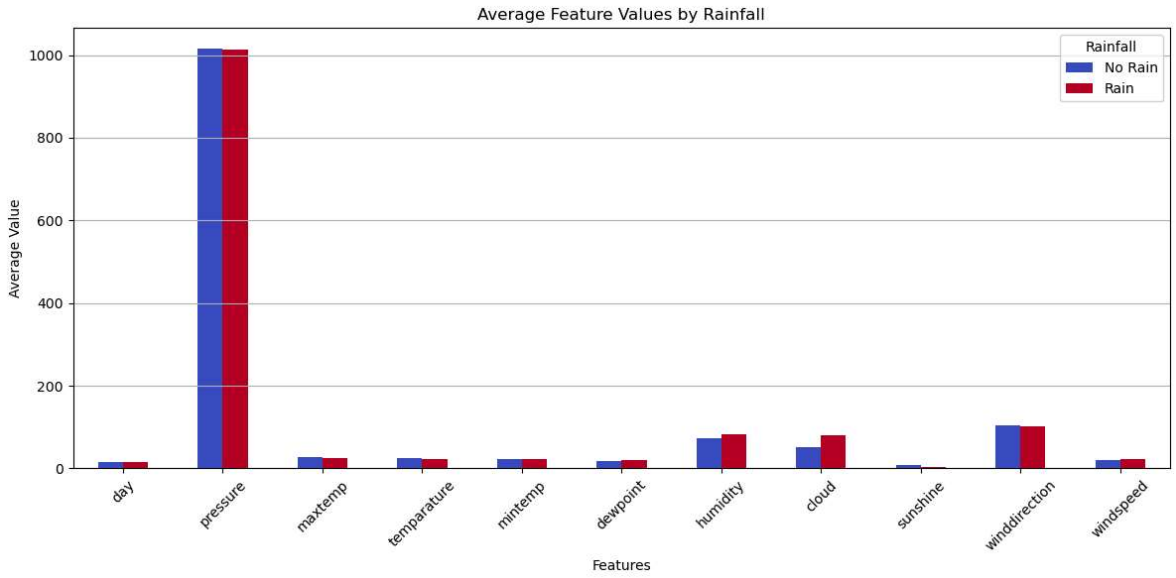
Out[41]:
```
0      1
1      1
2      1
3      1
4      1
      ..
361    1
362    1
363    1
364    1
365    0
Name: rainfall, Length: 365, dtype: int64
```

Visualization of Dataset

In [85]:
```
feature_means = df.groupby('rainfall').mean().T  # Transpose for plotting

# Plot the feature means for both classes
plt.figure(figsize=(12, 6))
feature_means.plot(kind='bar', figsize=(12, 6), colormap='coolwarm')
plt.title("Average Feature Values by Rainfall")
plt.ylabel("Average Value")
plt.xlabel("Features")
plt.xticks(rotation=45)
plt.legend(title='Rainfall', labels=['No Rain', 'Rain'])
plt.tight_layout()
plt.grid(axis='y')
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



Split Dataset into Training and Test Sets

In [48]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [50]:
```
X.shape,X_train.shape,X_test.shape
```

Out[50]:
```
((365, 10), (292, 10), (73, 10))
```

In [52]:
```
y.shape,y_train.shape,y_test.shape
```

Out[52]:
```
((365,), (292,), (73,))
```

Feature Scaling (Standardization)

In [55]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()  # Initialize a StandardScaler object
X_train_scaled = scaler.fit_transform(X_train)  # Fit and transform the training
X_test_scaled = scaler.transform(X_test) # Transform the test data using the alr
```

In [62]:
```
X_train_scaled
```

Out[62]:
```
array([[-1.34714744,  1.23857664,  1.08151854, ...,  1.43643207,
        -0.66716214, -0.70375742],
       [ 1.55224527, -1.25166102, -1.23603891, ..., -0.96165566,
        -1.14758469,  0.09786788],
       [-0.5095451 ,  1.06801242,  1.06341263, ...,  1.68886236,
         1.61484498, -0.66211455],
       ...,
       [-1.08942364,  1.05095599,  0.88235345, ...,  1.61313327,
        -1.14758469, -0.4643109 ],
       [ 0.89182804, -1.06404037, -1.03687382, ..., -0.63349628,
        -1.14758469,  0.61840379],
       [-0.70283795, -0.91053257, -0.69286138, ..., -1.18884291,
        -0.42695086,  0.93072534]])
```

```
[-0.86391532,  0.86333535,  0.86424753,  1.01338174,  0.84412268,
  0.07403092,  0.3640603 , -0.9111696 , -0.78726777, -0.4643109 ],
[ 0.50524235, -0.91053257, -0.67475546, -0.5687194 , -0.52847274,
  0.17457524,  0.63341573, -1.0878708 , -0.66716214, -0.05829289],
[-0.07463619, -0.29650137, -0.31263711, -0.35049855,  0.17498241,
  1.38110705,  0.6783083 , -0.81019748, -0.78726777, -0.87032891],
[ 1.92272323, -1.96803075, -2.0508052 , -2.05989517, -2.46726378,
 -1.7357668 ,  0.85787858, -1.01214171, -1.02747905,  0.27485009],
[-0.70283795,  0.91450462,  0.86424753,  0.92245638,  0.84412268,
  0.07403092, -1.3418574 ,  0.52768304,  0.05347169, -1.10977543],
[ 0.73075067, -1.72924084, -1.59815726, -1.45978785, -1.36918744,
  0.27511955,  0.85787858, -0.9111696 , -1.02747905,  0.60799307],
[-0.89613079,  0.50515048,  0.70129427,  0.75879075,  0.89559501,
  0.67729682,  0.63341573, -1.16359988,  1.49473934,  0.21238578],
[-0.33235999,  0.26636057,  0.30296409,  0.17686849,  0.51813127,
  0.67729682,  0.54363059, -0.50728114, -0.78726777, -1.21388261],
[-0.73505342,  1.05095599,  1.06341263,  1.04975188,  0.86128012,
 -0.32814635, -1.70099796,  1.8150775 ,  1.73495061,  0.07704645],
[-0.10685167, -1.02992753, -1.03687382, -0.93242081, -0.40837064,
  1.68274   ,  1.12723401, -1.03738474, -0.78726777,  0.11868932],
[-0.15517488,  0.24930415, -0.0953661 , -0.27775827,  0.17498241,
  0.77784114,  0.63341573, -0.63349628,  0.17357733, -1.33881123],
[-1.55654802,  0.40281195,  0.71940019,  0.86790117,  0.8098078 ,
  0.37566387,  0.81298601, -1.18884291,  1.49473934, -0.47472162],
[ 0.58578103,  0.16402203,  0.23054042,  0.32234906,  0.15782497,
 -0.22760203, -0.21954312,  0.90632847, -0.30684522,  0.86826103]])
```

Train Logistic Regression Model

In [60]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()  # Initialize a Logistic Regression model
model.fit(X_train_scaled, y_train)  # Train the model on the scaled training dat
```

Out[60]:
```
LogisticRegression()
```

Predict and Evaluate on Test Set

In [67]:
```
y_pred = model.predict(X_test_scaled)  # Predict the target variable for the tes
y_pred
```

Out[67]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1], dtype=int64)
```

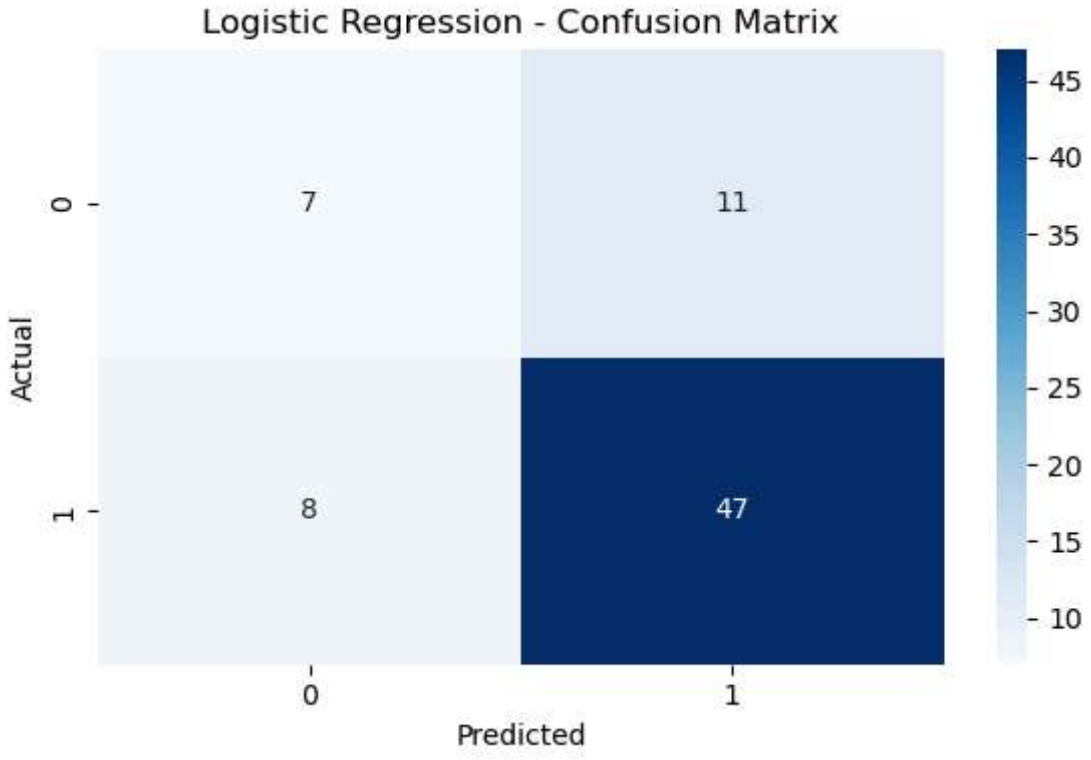
In [69]:
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sc
print(" ◆  Classification Report:\n")  # Print the classification report heading
print(classification_report(y_test, y_pred))  # Print the classification report
print(f" ◆  Accuracy: {accuracy_score(y_test, y_pred):.2f}")  # Print the accura
```

- Classification Report:

	precision	recall	f1-score	support
0	0.47	0.39	0.42	18
1	0.81	0.85	0.83	55
accuracy			0.74	73
macro avg	0.64	0.62	0.63	73
weighted avg	0.73	0.74	0.73	73

- Accuracy: 0.74

In [71]:
```
plt.figure(figsize=(6, 4))  # Create a figure for the plot with specific size
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title("Logistic Regression - Confusion Matrix")  # Set the title of the plot
plt.xlabel("Predicted")  # Label the x-axis as 'Predicted'
plt.ylabel("Actual")  # Label the y-axis as 'Actual'
plt.tight_layout()  # Adjust layout to ensure everything fits well in the plot
plt.show()  # Show the plot
```

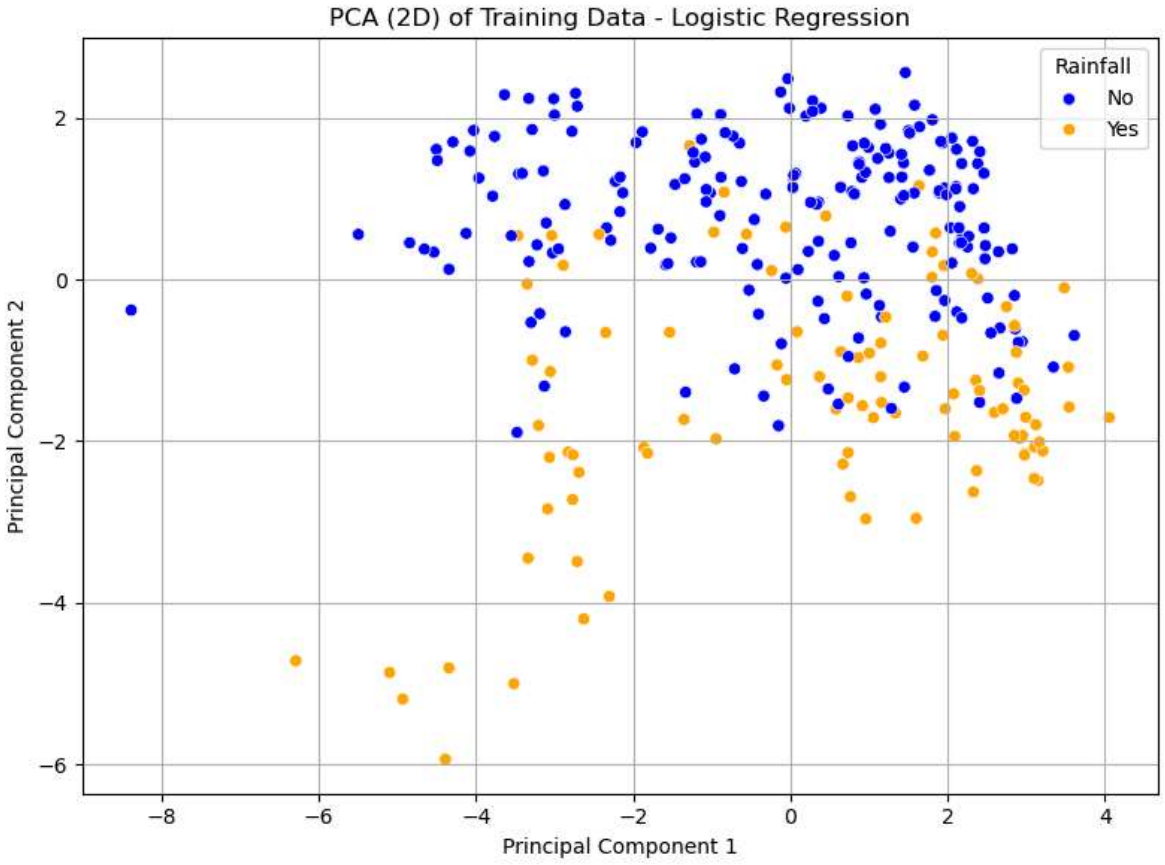


PCA Visualization of Training Data

In [76]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)  # Initialize PCA with 2 components (2D reduction)
X_train_2D = pca.fit_transform(X_train_scaled)  # Apply PCA on the scaled traini

plt.figure(figsize=(8, 6))  # Create a figure for the plot with specific size
sns.scatterplot(x=X_train_2D[:, 0], y=X_train_2D[:, 1], hue=y_train, palette=['c
plt.title("PCA (2D) of Training Data - Logistic Regression")  # Set the title of
plt.xlabel("Principal Component 1")  # Label the x-axis as 'Principal Component
plt.ylabel("Principal Component 2")  # Label the y-axis as 'Principal Component
plt.legend(title="Rainfall", labels=["No", "Yes"])  # Add a legend to the plot
plt.grid(True)  # Show a grid on the plot
```

```
plt.tight_layout()  # Adjust layout to ensure everything fits well in the plot
plt.show()  # Show the plot
```

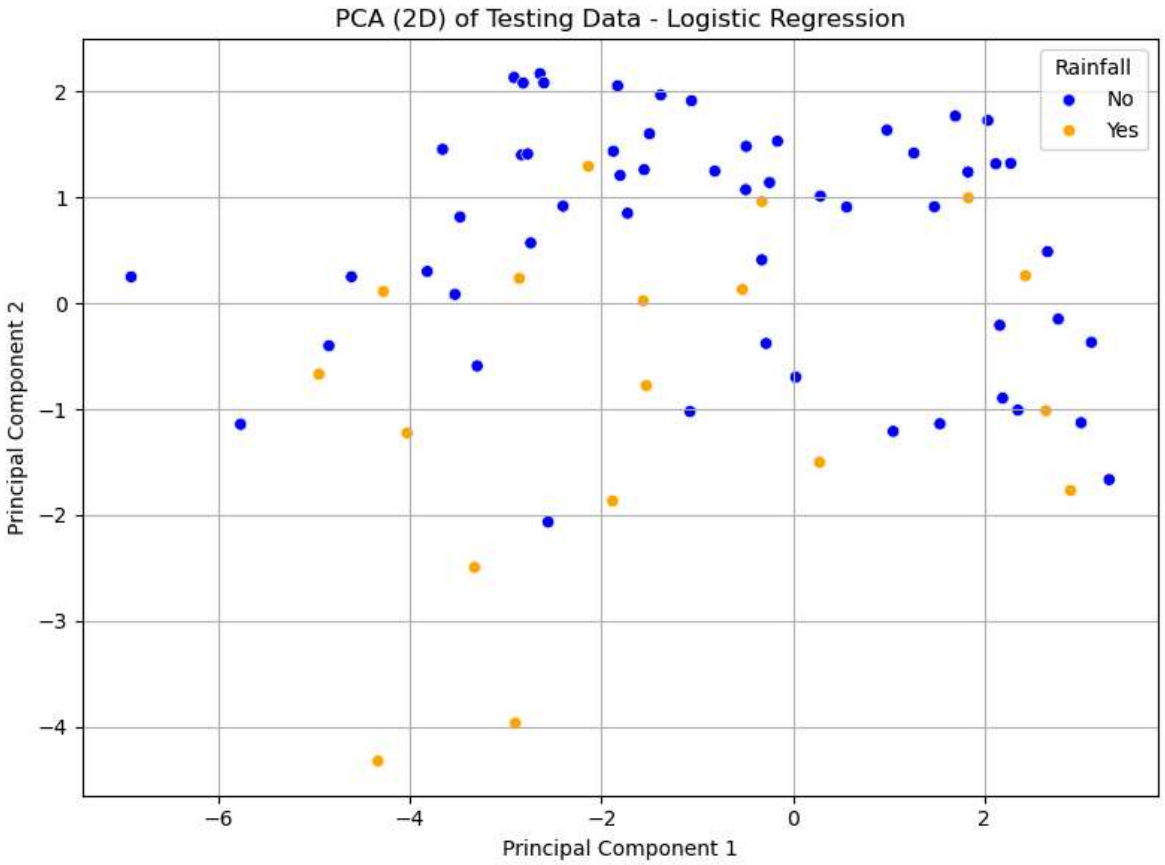


PCA Visualization of Testing Data

In [79]:
```
pca = PCA(n_components=2)

# Fit PCA on training data (important to avoid data leakage) and transform test
pca.fit(X_train_scaled)
X_test_2D = pca.transform(X_test_scaled)  # Apply PCA on scaled testing data

# Plot the 2D PCA representation of testing data
plt.figure(figsize=(8, 6))  # Create a figure for the plot with specific size
sns.scatterplot(x=X_test_2D[:, 0], y=X_test_2D[:, 1], hue=y_test, palette=['oran
plt.title("PCA (2D) of Testing Data - Logistic Regression")  # Title
plt.xlabel("Principal Component 1")  # X-axis label
plt.ylabel("Principal Component 2")  # Y-axis label
plt.legend(title="Rainfall", labels=["No", "Yes"])  # Legend
plt.grid(True)  # Show grid
plt.tight_layout()  # Adjust layout
plt.show()  # Show the plot
```



Predict on a New Instance

In [82]:
```
new_instance = [[1015.5, 21.5, 20.1, 18.7, 16.3, 88, 55, 6.2, 100, 10.5]]  # Def
new_instance_scaled = scaler.transform(new_instance)  # Scale the new instance u
new_prediction = model.predict(new_instance_scaled)[0]  # Make a prediction on t
print(f"\n🔮 Prediction for new data: {'Rain' if new_prediction == 1 else 'No Ra
```

```
🔮 Prediction for new data: No Rain
C:\Anaconda\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have v
alid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

In [ ]: