Numerical Dataset

A) General information on dataset:

- **Dataset Name ->** Car price prediction
- **Number of features ->** 18
- **Number of rows ->** 18
- **Number of numerical features -> 5** (ID,Price,Prod. year,Cylinders,Airbags)
- **it has datatypes: float64(1), int64(4), object(13).**
- **Number of categorical features ->** 13 (Levy,Manufacturer,Model,Category,Leather interior,Fuel type,Engine volume,Mileage,Gear box type,Drive wheels,Doors,Wheel,Color)
- **Features labels ->** [‘ID’ , ’Price’, ’Levy’, ’Manufacturer’, ‘Model’, ‘Prod. year’, ‘Category’, ‘Leather interior’, ‘Fuel type’, ‘Engine volume’, ‘Mileage’, ‘Cylinders’, ‘Gear box type’, ‘Drive wheels’, ‘Doors’, ‘Wheel’, ‘Color’, ‘Airbags’]
- **Total number of samples(before preprocessing and cleaning) ->** 19237 entries.
- **Total number of samples(after preprocessing and cleaning) ->** 13164
- **Number of features after preprocessing and cleaning ->** 38
- **The number of samples used for training ->** 10531 (80% of the dataset)
- **The number of samples used for testing ->** 2633 (20% of the dataset)

B) Implementation details

- **-Hyperparameters used in our model:**
- **In train_test_split function:**

test_size = 0.2

➔ Which means that the test size takes 20% of our datasets.

random_state = 42

➔ Which means that we choose from the dataset by random but we specified 42 to keep the random generated images everytime we run the program.

- **In KNN we have:**
  -best_params:

  ➔ **n_neighbors =** 7
  ➔ **p =** 1
  ➔ **weights =** distance

C)Result details:

Linear Regression:

```
# Model training
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predictions
y_pred = lr.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)
print(f"Mean Absolute Error: {mae}")
print(f'Mean Squared Error: {mse}')
print(f"R-squared: {r2}")

lr_score = lr.score(X_test,y_test)
print(f'The Score of Model is :{lr_score}')
```

```
Mean Absolute Error: 6432.696726903207
Mean Squared Error: 69468901.78397739
R-squared: 0.3912666300984766
The Score of Model is :0.3912666300984766
```

The accuracy of the model is : 0.39

KNN:

- KNN

In [95]:
```
param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}
```

In [96]:
```
knn = KNeighborsRegressor()
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'n_neighbors': 7, 'p': 1, 'weights': 'distance'}
```

In [97]:
```
knn = KNeighborsRegressor(n_neighbors = 7, p = 1, weights = 'distance')
knn.fit(X_train, y_train)
# Make predictions on the test set
y_pred = knn.predict(X_test)
```

In [98]:
```
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}")
print(f'Mean Squared Error: {mse}')
print(f"R-squared: {r2}")
knn_score = knn.score(X_test,y_test)
print(f'The Score of Model is :{knn_score}')
```

```
Mean Absolute Error: 3306.582465837797
Mean Squared Error: 28467531.04815926
R-squared: 0.75054829337004
The Score of Model is :0.75054829337004
```

The accuracy of the model is : 0.75

Overall accuracy comparison:

Linear Regression -> 0.39

KNN -> 0.75

Image Dataset

A) General information on dataset:

- **Dataset Name ->** Oxford-IIIT Pet Dataset
- **Number of classes ->** 2
- **Classes labels ->** [‘Cats’ , ’Dogs’]
- **Total number of samples ->** 2000 Image
- **Cats number ->** 1000
- **Dogs number ->** 1000
- **The number of samples used for training ->** 1400 (70% of the dataset)
- **The number of samples used for testing ->** 600 (30% of the dataset)

B) Implementation details:

- **– Feature extraction information**
- **Feature extraction algorithm ->** HOG
- **The number of features extracted ->** 4098
- **The dimension of resulted features ->** (2000,4098)

- **-Hyperparameters used in our models**
- **In the hog extraction we modified:**

Orientations = 8

➔ Which means that he number of pins used are 8 pins.

- **In the train test split function**

test_size = 0.3

➔ Which means that the test size takes 30% of our datasets.

random_state = 42

➔ Which means that we choose from the dataset by random but we specified 42 to keep the random generated images everytime we run the program.

- **In the logistic regression model:**

max_iter = 1000

➔ **Which means that the max number of iterations allowed for the solver to converge and find the optimal solution.**

- **In the kmeans algorithm:**

n_clusters = 2

➔ **Which means that we have 2 clusters 1 for each class in our dataset.**

random_state = 42

- **Which means that we choose from the dataset by random but we specified 42 to keep the random generated images everytime we run the program.**

n_init = 20

- **Which means that we make sure that we make multiple runs with different intializations (20) and selecting the best among them.**

C) Results details:

- **Logistic Regression:**

- **Accuracy : 0.74**

```
# Labeling and splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(normalized_features, numeric_labels, test_size=0.3, random_state=42)

# Initialize and train the Logistic Regression model with an increased number of iterations
logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train, y_train)

# Predict on the test set
y_pred = logreg_model.predict(X_test)

# Evaluate the accuracy for Logistic Regression
accuracy_logreg = accuracy_score(y_test, y_pred)
print(f"Accuracy (Logistic Regression): {accuracy_logreg}")
```

```
Accuracy (Logistic Regression): 0.74
```

- **Classification report**

```
classification_report_logreg = classification_report(y_test, y_pred)
print("Classification Report (Logistic Regression):\n", classification_report_logreg)
```

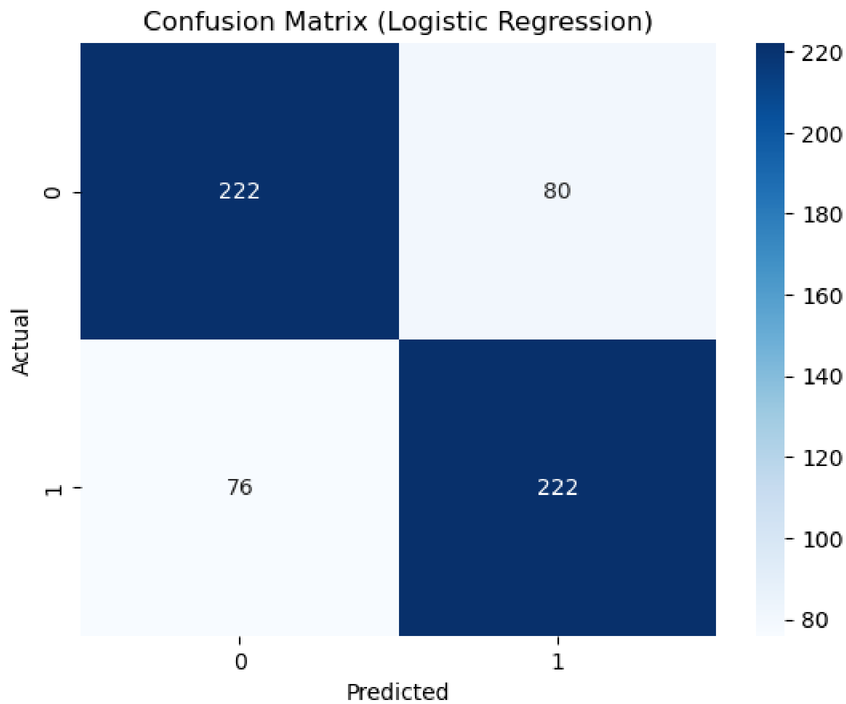
```
Classification Report (Logistic Regression):
              precision    recall  f1-score   support

           0       0.74      0.74      0.74       302
           1       0.74      0.74      0.74       298

    accuracy                           0.74       600
   macro avg       0.74      0.74      0.74       600
weighted avg       0.74      0.74      0.74       600
```

- **Confusion Matrix**

```
# Assuming y_test and y_pred are your true and predicted labels
conf_matrix_logreg = confusion_matrix(y_test, y_pred)

# Print the confusion matrix
print("Confusion Matrix (Logistic Regression):\n", conf_matrix_logreg)
true_negatives, false_positives, false_negatives, true_positives = conf_matrix_logreg[0][0], conf_matrix_logreg[0][1], conf_matri

accuracy = (true_positives + true_negatives) / sum(sum(conf_matrix_logreg))
print(f"Accuracy: {accuracy}")




# Plot the confusion matrix
sns.heatmap(conf_matrix_logreg, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix (Logistic Regression)')
plt.show()
```

```
Confusion Matrix (Logistic Regression):
 [[222  80]
 [ 76 222]]
Accuracy: 0.74
```

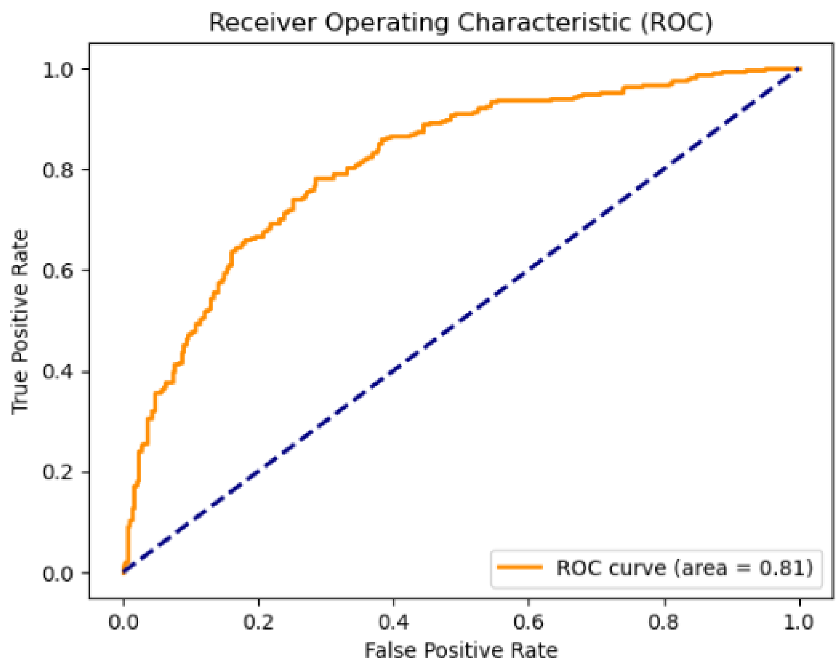
- **The plot for confusion matrix with seaborn**



- The first block represents -> True negative
- The second block represents -> False Positive
- The third block represents -> False Negative
- The fourth block represents -> True positive

- **ROC Curve**

```
# Predict probabilities on the test set
y_prob = logreg_model.predict_proba(X_test)[:, 1]

# Compute ROC curve and ROC area
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

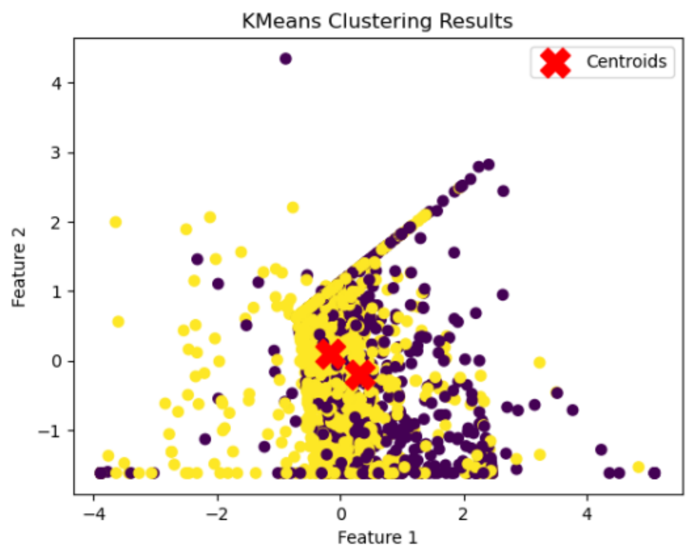


This plot shows the relationship between the true positive and the false positive rate.

Kmeans

- **Accuracy : 0.548 (almost 55%)**
- **Kmeans plot**

```
plt.scatter(normalized_features[:, 0], normalized_features[:, 1], c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red', marker='X', label='Centroids')
plt.title('KMeans Clustering Results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```



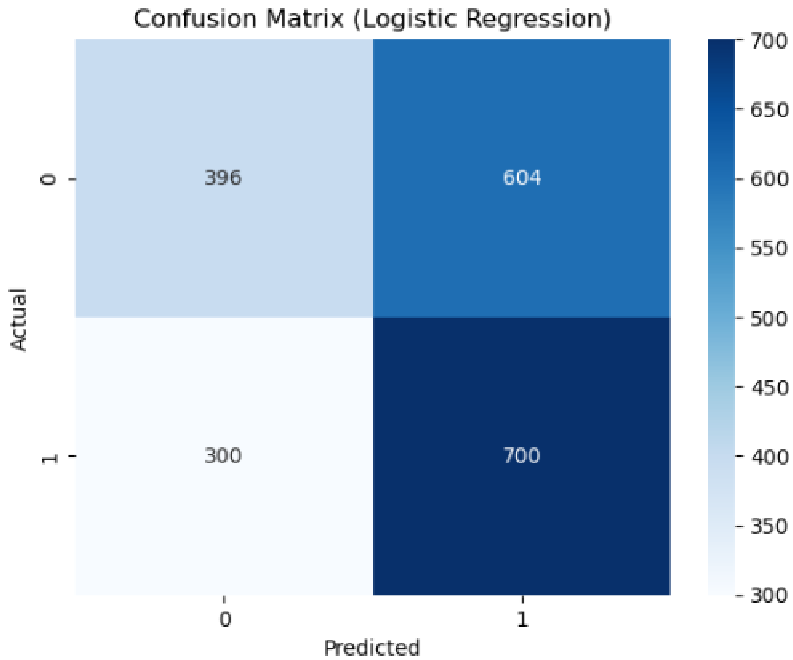
- **Confusion Matrix and accuracy**

```
predicted_labels = kmeans.labels_
```

```
conf_matrix = confusion_matrix(numeric_labels, predicted_labels)
print("Confusion Matrix:")
print(conf_matrix)
# Extract values from the confusion matrix
true_negatives, false_positives, false_negatives, true_positives = conf_matrix[0][0], conf_matrix[0][1], conf_matrix[1][0], conf_

# Calculate accuracy
accuracy = (true_positives + true_negatives) / sum(sum(conf_matrix))

# Print accuracy
print(f"Accuracy: {accuracy}")
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix (Logistic Regression)')
plt.show()
```

```
Confusion Matrix:
[[396 604]
 [300 700]]
Accuracy: 0.548
```

- **Confusion Matrix plot with seaborn**



- The first block represents -> True negative

- The second block represents -> False Positive

- The third block represents -> False Negative

- The fourth block represents -> True positive