Project Phase 1 Machine Learning

Name: Abdelrahman Mohamed

ID: 202101081

- ***Project Steps***

1- **Import libraries**
2- **Load the Data and Knew Some info about it**
3- **Implement KNN**
4- **Implement Naïve Bayes**
5- **Implement Logistic Regression**
6- **Model Evaluation (using ROC and AUC)**
7- **Compare the AUC's of the Three Algorithms**

This dataset contains data from a higher education institution on various variables related to undergraduate students, including demographics, social-economic factors, and academic performance, to investigate the impact of these factors on student dropout and academic success

1. Choose a dataset: I have choose data set about students information, the dataset has 35 columns, the target column called Target that has Two values (Graduate, Dropout)

2. Algorithm Implementation: I did the Three Algorithms and here's the summary

KNN Accuracy (1) = 84.30%

Naive Bayes Accuracy = 85.67%

Logistic regression Accuracy = 90.91%

3. Model Evaluation: I Graphed the 3 Algorithms ROC's and get the AUC's Scores for each algorithm

KNN AUC scores = 0.904

Naive Bayes AUC scores = 0.882

Logistic regression AUC scores = 0.962

4. Results Analysis: Interpret the results, and compare the performance of the three algorithms based on the ROC and AUC scores.

AUC Scores:

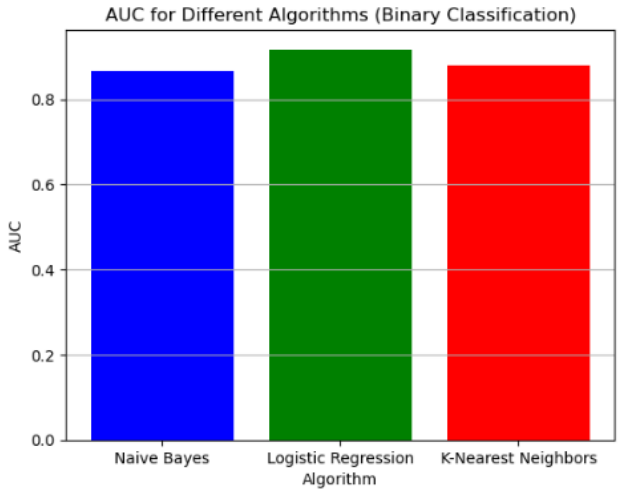
1- Logistic Regression has the highest AUC score (**0.962**), followed by K-Nearest Neighbors (**0.904**), and then Naive Bayes (**0.882**).
2- **A higher AUC score generally indicates better overall performance in terms of distinguishing between classes.**

- **Logistic Regression outperforms both K-Nearest Neighbors and Naive Bayes in terms of AUC scores, indicating that it is the most effective algorithm for the given classification task.**

- **From above we knew that the Logistic Regression is the best performance because it has the highest AUC score and also has the highest Algorithm Accuracy**

comparison Between AUC's of 3 algorithms

In [25]:
```
# Plot AUC values for different algorithms
plt.figure()
plt.bar(algorithms.keys(), auc_values, color=['blue', 'green', 'red'])
plt.xlabel('Algorithm')
plt.ylabel('AUC')
plt.title('AUC for Different Algorithms (Binary Classification)')
plt.grid(axis='y')
plt.show()
```



- FULL CODE

1- Import libraries

Import Needed Modules

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, classification_report
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve, auc
```

2- Load the Data and Knew Some info about it

Read the Data and Knew Some info about it

In [2]:

```
df = pd.read_csv(r"Predict students' dropout and academic success.csv", delimiter=";")
df.head(10)
```

Out[2]:

	Marital status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Nacionality	Mother's qualification	Father's qualification	Mother's occupation	...	Curricular units 2nd sem (credited)	Curricular units 2nd sem (enrolled)	Cur uni
0	1	8	5	2	1	1	1	13	10	6	...	0	0	
1	1	6	1	11	1	1	1	1	3	4	...	0	6	
2	1	1	5	5	1	1	1	22	27	10	...	0	6	
3	1	8	2	15	1	1	1	23	27	6	...	0	6	
4	2	12	1	3	0	1	1	22	28	10	...	0	6	
5	2	12	1	17	0	12	1	22	27	10	...	0	5	
6	1	1	1	12	1	1	1	13	28	8	...	0	8	
7	1	9	4	11	1	1	1	22	27	10	...	0	5	
8	1	1	3	10	1	1	15	1	1	10	...	0	6	
9	1	1	1	10	1	1	1	1	14	5	...	0	6	

10 rows × 35 columns

In [3]:

```
# see all columns names
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4424 entries, 0 to 4423
Data columns (total 35 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   Marital status                                  4424 non-null   int64
 1   Application mode                                4424 non-null   int64
 2   Application order                               4424 non-null   int64
 3   Course                                          4424 non-null   int64
 4   Daytime/evening attendance                      4424 non-null   int64
 5   Previous qualification                          4424 non-null   int64
 6   Nacionality                                     4424 non-null   int64
 7   Mother's qualification                          4424 non-null   int64
 8   Father's qualification                          4424 non-null   int64
 9   Mother's occupation                             4424 non-null   int64
 10  Father's occupation                             4424 non-null   int64
 11  Displaced                                       4424 non-null   int64
 12  Educational special needs                       4424 non-null   int64
```

3- Implement KNN

KNN Implementation

In [7]:
```
# Read the dataset
df = pd.read_csv("Predict students' dropout and academic success.csv", delimiter=";")
df = df[df["Target"].isin(["Graduate", "Dropout"])]

df.head()
```

Out[7]:

	Marital status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Nacionality	Mother's qualification	Father's qualification	Mother's occupation	...	Curricular units 2nd sem (credited)	Curricular units 2nd sem (enrolled)	Cur un
0	1	8	5	2	1	1	1	13	10	6	...	0	0	
1	1	6	1	11	1	1	1	1	3	4	...	0	6	
2	1	1	5	5	1	1	1	22	27	10	...	0	6	
3	1	8	2	15	1	1	1	23	27	6	...	0	6	
4	2	12	1	3	0	1	1	22	28	10	...	0	6	

5 rows × 35 columns

In [8]:
```
# the target variable column is 'Target'
X = df.drop(columns=['Target'])
y = df['Target']

# Split the dataset into a train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
```

In [9]:
```
# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

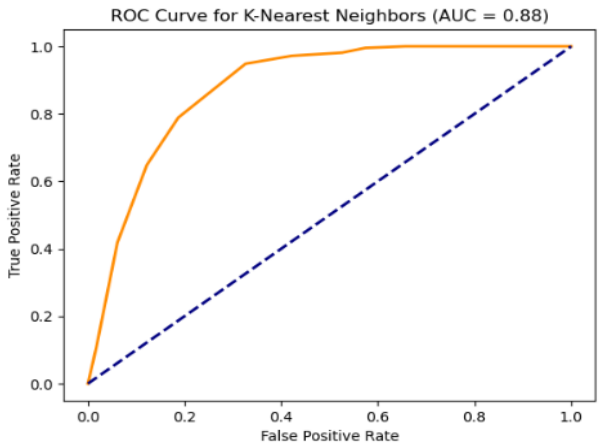
In [10]:
```
# Make the user intput the number of k
k = int(input("Enter the Number of Neighbors(k): "))
model = KNeighborsClassifier(n_neighbors=k)

# Fit the model
model.fit(X_train, y_train)

# Predict the responses for the test dataset
y_pred = model.predict(X_test)
```

```
Enter the Number of Neighbors(k): 10
```

In [11]:
```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
accuracy_percentage = accuracy * 100
print("Accuracy : {:.2f}%".format(accuracy_percentage))
```

```
Accuracy : 84.30%
```

- KNN ROC




```
AUC for K-Nearest Neighbors: 0.8793835476627884
Confusion Matrix for K-Nearest Neighbors:
 [[171  59]
 [ 28 185]]
Optimal Cutoff Threshold for K-Nearest Neighbors: 0.9483568075117371
```

4- Implement Naïve Bayes

Naive Bayes Implementation

In [12]:
```
# Read the dataset
df = pd.read_csv("Predict students' dropout and academic success.csv", delimiter=";")
df = df[df["Target"].isin(["Graduate", "Dropout"])]

df.head()
```

Out[12]:

	Marital status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Nacionality	Mother's qualification	Father's qualification	Mother's occupation	...	Curricular units 2nd sem (credited)	Curricular units 2nd sem (enrolled)	Cur uni (evalu
0	1	8	5	2	1	1	1	13	10	6	...	0	0	
1	1	6	1	11	1	1	1	1	3	4	...	0	6	
2	1	1	5	5	1	1	1	22	27	10	...	0	6	
3	1	8	2	15	1	1	1	23	27	6	...	0	6	
4	2	12	1	3	0	1	1	22	28	10	...	0	6	

5 rows × 35 columns

In [13]:
```
# Extract features and target variable
X = df.drop('Target', axis=1)
y = df['Target']
```

In [14]:
```
# Split the dataset into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
```

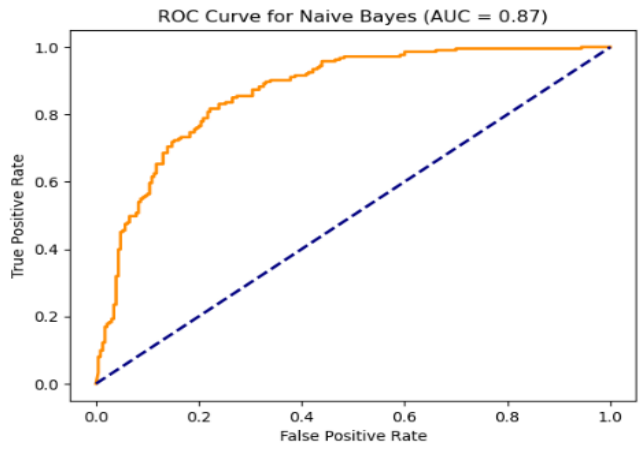
In [15]:
```
# Create a Naive Bayes classifier
clf = GaussianNB()

# Train the classifier on the training data
clf.fit(X_train, y_train)
```

Out[15]:
```
GaussianNB()
```

In [16]:
```
# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy * 100:.2f}%")

```

```
Accuracy: 85.67%
```

- Naïve Bayes ROC




```
AUC for Naive Bayes: 0.8665748111859564
Confusion Matrix for Naive Bayes:
 [[151  79]
 [ 21 192]]
Optimal Cutoff Threshold for Naive Bayes: 0.8169014084507042
```

5- Implement Logistic Regression

Logistic Regression Implementation

In [17]:
```
# Read the dataset
df = pd.read_csv("Predict students' dropout and academic success.csv", delimiter=";")
df = df[df["Target"].isin(["Graduate", "Dropout"])]
df.head()
```

Out[17]:

	Marital status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Nacionality	Mother's qualification	Father's qualification	Mother's occupation	...	Curricular units 2nd sem (credited)	Curricular units 2nd sem (enrolled)	Cur un (evalu
0	1	8	5	2	1	1	1	13	10	6	...	0	0	
1	1	6	1	11	1	1	1	1	3	4	...	0	6	
2	1	1	5	5	1	1	1	22	27	10	...	0	6	
3	1	8	2	15	1	1	1	23	27	6	...	0	6	
4	2	12	1	3	0	1	1	22	28	10	...	0	6	

5 rows × 35 columns

In [18]:
```
# Extract features and target variable
X = df.drop('Target', axis=1)
y = (df['Target'] == 'Graduate').astype(int)  # assuming 'Graduate' is the positive class

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
```

In [19]:
```
# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

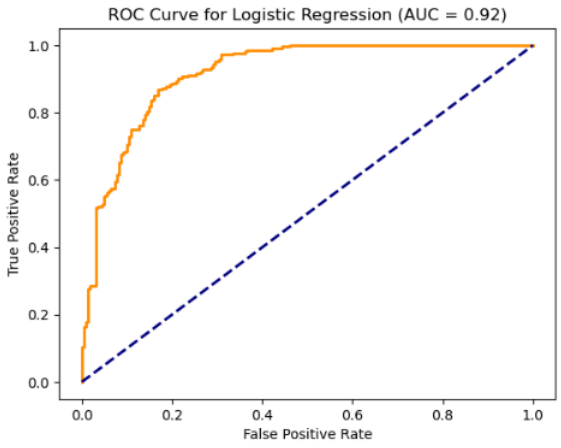
In [20]:
```
# Create a logistic regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)
```

Out[20]:
```
LogisticRegression()
```

In [21]:
```
# Make predictions on the test data
y_pred_prob = model.predict_proba(X_test)[:, 1]  # Probabilities of belonging to class 1

# Apply the threshold of 0.5 to make binary predictions
y_pred = (y_pred_prob > 0.5).astype(int)  # positive

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 90.91%
```

- Logistic Regression ROC




```
AUC for Logistic Regression: 0.9173504796897326
Confusion Matrix for Logistic Regression:
 [[181  49]
 [ 23 190]]
Optimal Cutoff Threshold for Logistic Regression: 0.8685446009389671
```

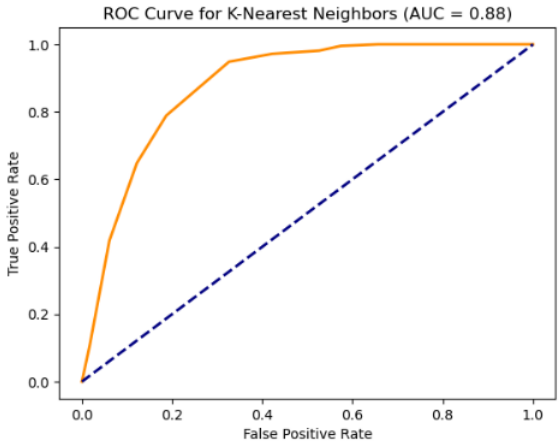
6- Model Evaluation (using ROC and AUC)

Model Evaluation (using ROC and AUC)

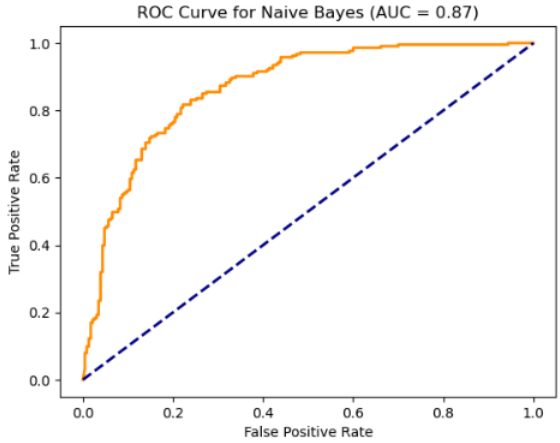
In [24]:

```
# Define the algorithms
algorithms = {
    'Naive Bayes': GaussianNB(),
    'Logistic Regression': LogisticRegression(),
    'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=k)
}

# Initialize lists to store AUC values for each algorithm
auc_values = []

for algorithm_name, algorithm in algorithms.items():
    # Train the classifier
    algorithm.fit(X_train, y_train)

    # Predict probabilities for the positive class
    y_scores = algorithm.predict_proba(X_test)[:, 1]

    # Calculate AUC
    auc = roc_auc_score(y_test, y_scores)

    # Calculate the confusion matrix
    y_pred = algorithm.predict(X_test)
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Determine the optimal Cutoff Threshold
    fpr, tpr, _ = roc_curve(y_test, y_scores)
    optimal_threshold = tpr[np.argmax(tpr - fpr)]

    # Append AUC to the list
    auc_values.append(auc)

    # Plot the ROC curve
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC = {roc_auc:.2f}')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for {algorithm_name} (AUC = {auc:.2f})')
    plt.show()

    # Print AUC, Confusion Matrix, and Cutoff Threshold
    print(f"AUC for {algorithm_name}:", auc)
    print(f"Confusion Matrix for {algorithm_name}:\n", conf_matrix)
    print(f"Optimal Cutoff Threshold for {algorithm_name}:", optimal_threshold)
```

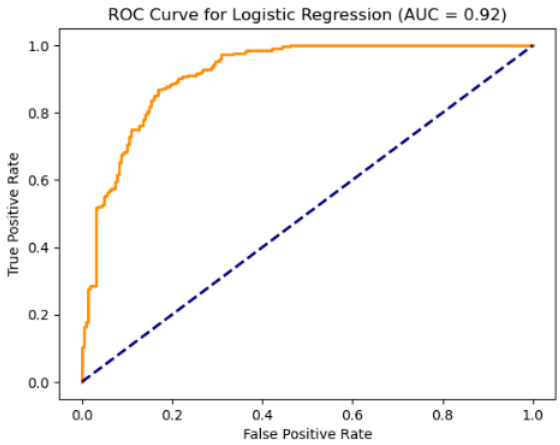
- ROC’s for the three Algorithms




```
AUC for K-Nearest Neighbors: 0.8793835476627884
Confusion Matrix for K-Nearest Neighbors:
 [[171  59]
 [ 28 185]]
Optimal Cutoff Threshold for K-Nearest Neighbors: 0.9483568075117371
```




```
AUC for Naive Bayes: 0.8665748111859564
Confusion Matrix for Naive Bayes:
 [[151  79]
 [ 21 192]]
Optimal Cutoff Threshold for Naive Bayes: 0.8169014084507042
```

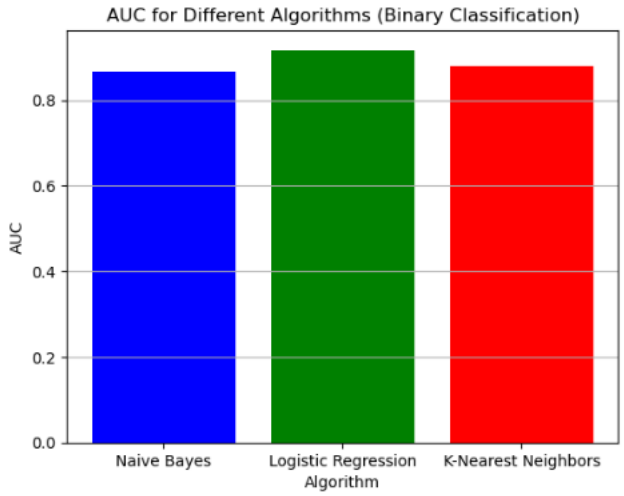



```
AUC for Logistic Regression: 0.9173504796897326
Confusion Matrix for Logistic Regression:
 [[181  49]
 [ 23 190]]
Optimal Cutoff Threshold for Logistic Regression: 0.8685446009389671
```

7- Compare the AUC's of the Three Algorithms

comparison Between AUC's of 3 algorithms

In [25]:
```
# Plot AUC values for different algorithms
plt.figure()
plt.bar(algorithms.keys(), auc_values, color=['blue', 'green', 'red'])
plt.xlabel('Algorithm')
plt.ylabel('AUC')
plt.title('AUC for Different Algorithms (Binary Classification)')
plt.grid(axis='y')
plt.show()
```



From above we knew that the Logistic Regression is the best performance because it has the highest AUC score because a higher AUC score generally indicates better overall performance in terms of distinguishing between classes and also has the highest Algorithm Accuracy