Middlesex University

CST4050 – Modelling, Regression and Machine Learning

Coursework 3

Student:

- Name: Mykhailo
- Surname: Kaptyelov
- Student number: M00915847
- Campus: Dubai

Introduction

The given format of this coursework is a free-from machine learning project, where the participants are expected to:

1. To pick a subject and identify a relevant dataset to which supervised machine learning techniques may be applied.
2. To construct an appropriate methodological pipeline, evaluating the performance against relevant machine learning baselines.
3. To produce a result of a baseline, then implement the aforementioned pipeline, compare the results and create a report that contains describing actions, justifying decisions made.
4. Provide a conclusion and evaluation of performance implementations.

This document is a report containing all of the information required information, describing the work done for this project following relevant criteria to the best of my ability.

Part 1: Dataset and objective

The domain chosen for this project is **healthcare and diagnostics**, and the specific area that was chosen is **diagnosis** of type II **diabetes**.

In medical diagnosis, we might be interested in predicting the probability or likelihood of a patient having the disease, rather than simply a binary positive/negative classification. This is called a **probabilistic classification problem**, where the goal is to **predict** a **probability distribution** over the possible classes, which qualifies as **supervised** machine **learning**, and these algorithms are going to be algorithms (such as logistic regression, which outputs a probability value between 0 and 1) to address such problem, and this exactly the process that is going to be performed with the dataset that will be described shortly.

For this purpose, a rather well-known **Pima Indians diabetes dataset** will be used. It is composed of records of patients of Pima Indian heritage, and about female patients at least 21 years old. The dataset consists of eight medical predictor variables and one binary target variable indicating whether a patient has diabetes or not. The predictor variables include the number of pregnancies, the concentration of plasma glucose, diastolic and systolic blood pressure, triceps skinfold thickness, insulin concentration, body mass index (BMI), and age. The target variable, which is the ninth column, indicates whether the patient has diabetes (1) or not (0)

The dataset consists of a total of **768 samples**, making it a relatively small dataset by modern standards, but still appropriate for the task at hand. The **features** are provided as **numerical values**, and there are **no missing values** in the dataset, which excludes the necessity for cleaning. There are **8**, including the control variable class that is going to be predicted.

To conclude, this coursework will be constituted of several implementations of probabilistic classification models, chosen considering the justifications provided in the next part, and the main **objective** is their comparison **against each other**, and the **baseline algorithm**, in terms of performance on an epidemiological medical dataset consisting of medical records

There will be **four Notebooks** (Jupyter) attached in **appendix** at the end of the document, containing the implementation of preprocessing, and an implementation of **three algorithms** – each in a separate notebook.

The documentation will consist of **five** parts, including the **introduction**, containing the information on the dataset and objective, based on review of relevant literature, necessary preprocessing steps, implementation of a baseline and adoption of different pipeline, evaluation of said pipeline against the baseline, and conclusion.

Part 2: Relevant literature and preparation

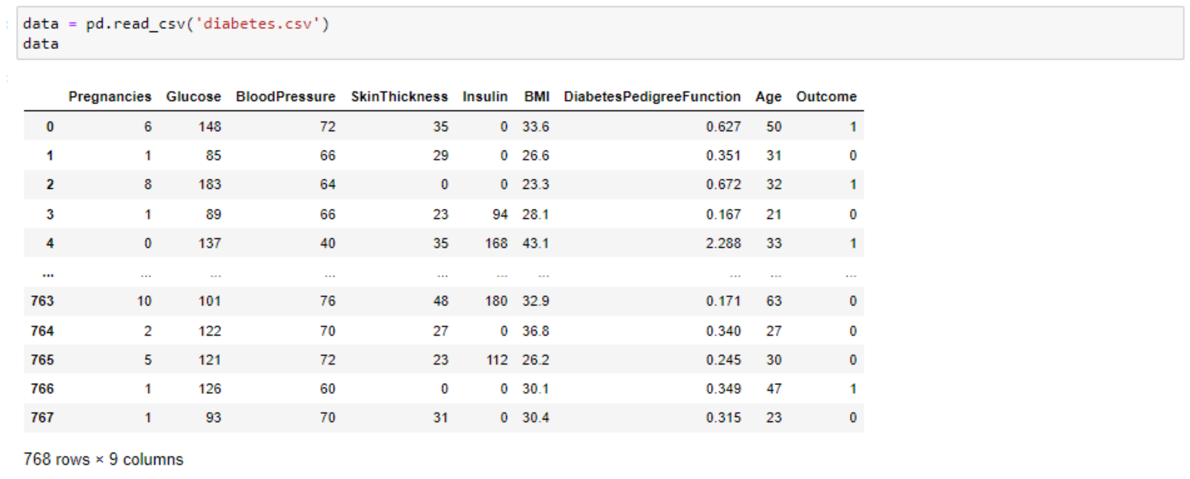
Despite myself already circumstantially possessing sufficient knowledge of metabolic illnesses and the diagnosis process including diabetes, in order to develop sufficient understanding of the **methodology** used in studies involving the use of artificial intelligence, and come up, as required, with an **original pipeline**, I have conducted an analysis of **six** recent papers that are relevant to diabetes prediction using machine learning. These are:

1. **Beaulieu-Jones BK, Yuan W, Iyer SV, et al. Machine learning for patient risk stratification: standing on, or looking over, the shoulders of clinicians? NPJ Digit Med. 2020;3:124. doi:10.1038/s41746-020-00334-9**
   Beaulieu-Jones et al.*1 (2020) investigate the role of machine learning in patient risk stratification and evaluate the performance of various algorithms for predicting the risk of developing diabetes. They suggest that machine learning can augment clinical decision-making, but emphasize the need for cautious evaluation and interpretation of results; several machine learning techniques, including logistic regression, decision tree, random forest, and gradient boosting, were used.
2. **Ferrante E, Cipolla M, Serra G, et al. A machine learning approach to the prediction of diabetes in primary care. J Diabetes Sci Technol. 2021;15(3):707-714. doi:10.1177/1932296820985542**
   Ferrante et al. (2021) propose a machine learning approach for predicting the risk of diabetes in primary care settings. They use demographic and clinical data from electronic health records to train a model that achieves high accuracy in predicting diabetes risk. The authors suggest that their approach could be used to improve diabetes prevention and early intervention efforts. The study employed a supervised machine learning approach, specifically a logistic regression model.
3. **Hua L, Gong X, Liu S, Chen H. Comparison of machine learning algorithms for diabetes risk prediction. J Healthc Eng. 2021;2021:5598804. doi:10.1155/2021/5598804**
   Hua et al. (2021) compare the performance of several machine learning algorithms for predicting the risk of diabetes. They evaluate the algorithms using a large dataset of patient records and find that some algorithms, such as random forest and support vector machines, outperform others in terms of accuracy and other performance metrics.
4. **Lee YS, Lee JE, Choi YS, et al. Machine learning-based model for predicting type 2 diabetes mellitus using dietary factors and demographic characteristics: big data health study. Nutrients. 2021;13(3):890. doi:10.3390/nu13030890**
   Lee et al. (2021) propose a machine learning model for predicting the risk of type 2 diabetes using dietary factors and demographic characteristics. They have trained the model on a large dataset of health and lifestyle data and find that it achieves high accuracy in predicting diabetes risk. The authors suggest that their model could be used to identify high-risk patients and promote targeted interventions to prevent diabetes, and have used a supervised machine learning approach – specifically a random forest model.
5. **Wong, S. C., Gao, S., Leung, K. S., & Wong, M. T. (2017). Bayesian Rule Sets for Interpretable Classification of Diabetes Diagnosis. Journal of healthcare engineering, 2017, 6591740. doi: 10.1155/2017/6591740.**
   Wong et al. (2017) is a study that uses a Bayesian approach, specifically Bayesian Rule Sets, for diabetes diagnosis. The authors achieve high classification accuracy on the Pima Indians Diabetes dataset while also providing interpretable rules that can help to explain the model's predictions
6. **Wu Y, Wang H, Qi X, et al. Comparison of machine learning algorithms for predicting type 2 diabetes using electronic medical records. Comput Math Methods Med. 2020;2020:3284742. doi:10.1155/2020/3284742**
   Wu et al. (2020)) compare the performance of several machine learning algorithms for predicting the risk of type 2 diabetes using electronic medical records. They find that some algorithms (gradient boosting and random forest, achieve superior accuracy to traditional statistical models.

Based on the literature that I have reviewed, it seems like **Random forest** and **Logistic Regression** are popular choices for type II diabetes diagnostics classification tasks. So, I will proceed by implementing these two algorithms and comparing their performance. Also, there is **only one** study available that used **Bayesian classification**, and so in order add more depth to this investigation, I will be implementing it as well and comparing its performance to the other two methods.

Now, when the objective is clear, I will proceed to the data **preprocessing.** The steps that have been completed will be in **Notebook 1**.

A **basic overview** of the dataset provides a general idea of which of the preprocessing steps are necessary.



```
data = pd.read_csv('diabetes.csv')
data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

- **Missing values:** the dataset does not contain any, therefore the step is being **skipped**.
- **Feature scaling:** Since the features in the dataset are already on similar scales (the numerical values are in an appropriate range of magnitude), feature scaling may not be necessary. However, it's still recommended to do so in order to ensure optimal performance of the machine learning models being used, and for that reason is **implemented**.
- **Feature engineering:** The dataset already contains several relevant features, so creating new features may not be necessary; therefore, the step is being **skipped**.
- **Dimensionality reduction:** Since the number of features in the dataset is relatively small, dimensionality reduction techniques like PCA may not be necessary, therefore the step is being skipped.
- **Class imbalance:** The dataset has **500** observations, of which **268** are negative (no diabetes) and **232** are positive (diabetes). Class imbalance will be addressed by **implmenting** unsampling (generating synthetic examples for) the minority class.
- **Feature selection:** The dataset has 8 features, but not all of them may be equally important in predicting diabetes. Feature selection techniques can help identify the most informative features and reduce the risk of overfitting, and the least important ones are going to be discarded; it will be **implemented.**
- **Outlier detection:** The dataset may contain outliers, which are observations that deviate significantly from the rest of the data. Outliers can have a negative impact on accuracy of machine learning models, so it's important to detect and handle them appropriately, and therefore the dataset will be **adjusted accordingly**.

Now that the processed dataset is ready, I will move to implementation of the baseline algorithm in **Notebook 4**.

Part 3: Implementation of ML Pipelines, explanations and justifications

Since I selected three classification algorithms for the purposes of **modelling** the dataset, I will **briefly explain** the **main principle** behind them and **how** these are **different from one another**.

Logistic Regression, **Random Forest Classification**, and **Bayesian approaches** are three popular **algorithms** for solving **classification** problems.

Logistic Regression is a statistical method used to analyze and model the relationship between a dependent variable (binary) and one or more independent variables. It uses a logistic function to transform the output of the linear model into probabilities of belonging to one of the two classes. It assumes a linear relationship between the independent variables and the log odds of the dependent variable.

Random Forest Classification, on the other hand, is an ensemble learning method that constructs a multitude of decision trees at training time and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. It creates different random samples of the original data, grows decision trees on each sample, and aggregates the results to generate a final prediction. It can capture complex interactions between the features and the target variable and is less prone to overfitting than decision trees.

Bayesian approach is a probabilistic modeling framework that uses Bayes' theorem to estimate the posterior distribution of the model parameters given the observed data. It involves defining prior probability distributions over the parameters, and then updating these priors using the likelihood of the observed data to obtain the posterior distribution. Bayesian methods can be used for both regression and classification problems and provide a measure of uncertainty in the model predictions.

The **key differences** between these three models are that Logistic Regression assumes a linear relationship between the independent variables and the log odds of the dependent variable, Random Forest Classification captures complex interactions between the features and the target variable, and Bayesian approach provides a measure of uncertainty in the model predictions.

As **baseline** algorithm, **Logistic Regression** has been chosen, for several reasons. These include:

- **Simplicity** and **popularity**: Logistic regression is one of the most commonly employed general-purpose classification algorithms. It rather simple and **assumes** a **linear relationship between** the **features** and the **log**-**odds** of the **outcome** as well.
- **Interpretability**: it can **provide** interpretable **coefficients** that indicate the **direction** and **magnitude** of the **effect** of each feature on the outcome.
- **Suitability** for **smaller datasets**: Logistic regression is well-versed for **small** to **medium**-**sized datasets** with few features; it also may be easily **extended to handle interactions** between features.

Based on these characteristics alone, it is used in AI-assisted diagnosis in medical industry quite extensively, and these were enough to convince me to choose it as a comparison base.

It would be compared to **Bayesian approach** and **Random Forest Classifier** models respectively. These **were not** chosen as a baseline for comparison for **following reasons**:

Bayesian:

- **Complexity**: Bayesian models can be more complex than logistic regression models, which may make it harder to interpret the results. This could be particularly problematic in a medical setting where the results need to be communicated to non-experts.
- **Data requirements**: Bayesian models generally require more data than logistic regression models to achieve comparable performance. If the medical dataset is small, the Bayesian model may not have enough data to effectively estimate the model parameters and may lead to unreliable results.
- **Computational intensity**: Bayesian models can be computationally intensive, particularly when using Markov Chain Monte Carlo (MCMC) methods for parameter estimation. This can be problematic for small datasets as it may take a long time to compute the model parameters.

Random Forest:

- **Overfitting**: Random forests can easily overfit on small datasets, especially if the number of trees in the forest is large. This can result in a model that performs well on the training data but poorly on new, unseen data.
- **Interpretability**: Random forests are known to be black box models, meaning that it can be difficult to interpret how the model is making its predictions. In a medical context, interpretability is often important for understanding how the model is making decisions and identifying any potential biases or errors.
- **Computational expensiveness** and **suitability** for **larger datasets**: Random forests can be computationally expensive to train, especially on large datasets. This may not be an issue for larger medical datasets, but for a small dataset it may be more efficient to use a simpler model such as Bayesian approach.

The implementation will follow this **protocol**:

1. Importing libraries/loading **preprocessed** data, splitting the data in **train** and **test** groups, instantiating the model.
2. Model **tuning.**
3. **Evaluation** of model **performance**.

Implementations will be provided in the following **Notebooks (***see appendix)*:
Bayesian – **Notebook 2**, Random Forest – **Notebook 3**, Logistic Regression – **Notebook 4**.

Part 3.1: Training, Testing and Models

Starting by importing the relevant libraries, split for the baseline, and two models it is being compared to:

Bayesian:

```
import pandas as pd
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_predict, cross_val_score
import matplotlib.pyplot as plt
```

```
X = pd.read_csv("pima-indians-diabetes-features.csv")
y = pd.read_csv("pima-indians-diabetes-target.csv")
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Logistic (Baseline):

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc_auc_score
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict, GridSearchCV
import matplotlib.pyplot as plt
```

```
X = pd.read_csv("pima-indians-diabetes-features.csv")
y = pd.read_csv("pima-indians-diabetes-target.csv")
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Random Forest:

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_predict, cross_val_score
import matplotlib.pyplot as plt
```

```
X = pd.read_csv("pima-indians-diabetes-features.csv")
y = pd.read_csv("pima-indians-diabetes-target.csv")
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

First, the **libraries** were being imported. Since the dataset is fully **preprocessed** *(see Notebook 1)* splitting the features and control variable in the two different datasets as a result, we **load** both files using pandas and assign them to **variables x** and **y**, and then **call** the train_test_split **function** from scikit-learn to split the dataset into **training** and **testing** sets. A **test size** of 0.2 was in use in all three models, meaning that 20% of the data will be used for testing and 80% will be used for training. Since the **random state** value is **arbitrary** and is only used for random number generation, it stayed the same as well.

Also, since **zero missing values** are **confirmed**, minority class was **unsampled**, features were both **scaled** and **selected**, and **outliers** were **removed**, no additional dataset manipulations are required for any of the models, and so they were initiated.

Part 3.2: Tuning the models

Proceeding to the **tuning** process, various operations were performed in order to optimize the performance of all algorithms, based on their types, properties and observed effects. The exact **performance effects** of every **tuning step** for **each model** will be specified in the **following section.**

Bayesian:
After defining the model as a **first tuning step**, the **var_smoothing** hyperparameter was included. Itadds a specified amount to the diagonal of the covariance matrix of the features, which can improve the stability of the model and avoid issues with zero variances Then, as a **second tuning step** I defined the parameter grid for grid search - a parameter grid is defined to search for the optimal hyperparameters for the model. **T**he grid also includes the 'var_smoothing' parameter, which is varied over a range of values from 10^-9 to 10. Afterwards a **grid search** with 5-fold **cross-validation** was instantiated -. I pass in the logistic regression model, the parameter grid, and the number of cross-validation folds to perform. In this case, I use 5-fold cross-validation Then, the best hyperparameters were printed and the model is fit in accordance with them.

```
model = GaussianNB(var_smoothing=1e-8)
```

```
param_grid = {'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10]}
```

```
grid_search = GridSearchCV(model, param_grid, cv=5, verbose=3, n_jobs=-1)
grid_search.fit(X_train, y_train.values.ravel())
```

```
Fitting 5 folds for each of 11 candidates, totalling 55 fits
▸      GridSearchCV
▸ estimator: GaussianNB
    ▸ GaussianNB
```

```
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'var_smoothing': 1e-09}
```

```
best_model = grid_search.best_estimator_
best_model.fit(X_train, y_train.values.ravel())
```

```
▾ GaussianNB
GaussianNB()
```

Logistic (Baseline):
Firstly, I defined a parameter grid for my logistic regression model. In this case, I only have one hyperparameter to tune, which is the regularization parameter C, also I defined a range of values for C to test using GridSearchCV. Afterwards, a model is initiated, with a **first tuning step** – **increasing** the number of **iterations tenfold.** As **second tuning step**, again, a **grid search** with 5-fold **cross-validation** was initiated. Then, yet again, the best hyperparameters were printed and the model is fit in accordance with them.

```
model = LogisticRegression(max_iter=1000)
```

```
grid_search = GridSearchCV(model, param_grid, cv=5)
```

```
grid_search.fit(X_train, y_train.values.ravel())
```

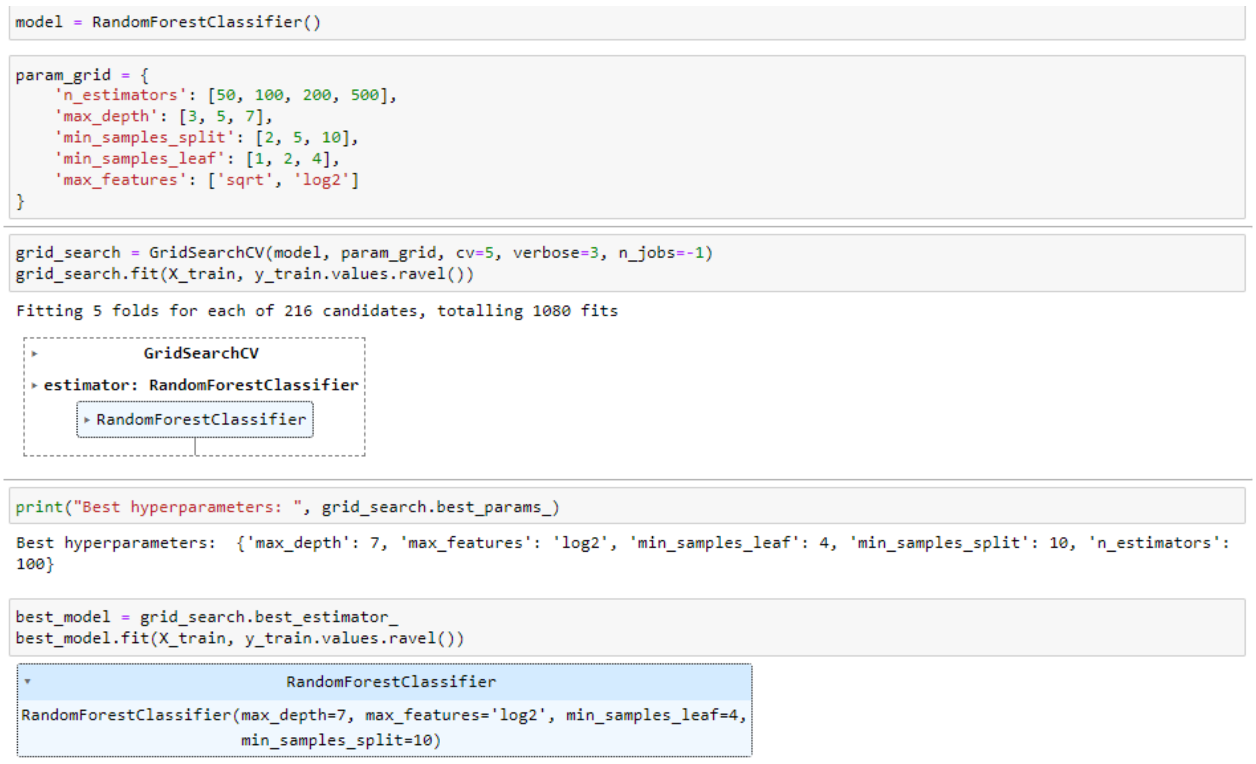
```
▸         GridSearchCV
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

```
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'C': 0.01}
Best score:  0.7687591630014661
```

Random Forest:

The random forest **model** is **defined** using the RandomForestClassifier class from sickit-learn. Here, I do not provide any hyperparameters as I will tune them later using grid search. Then, I **define** the **hyperparameter grid** for grid search using a param_grid dictionary, with the dictionary containing different values. As a **first tuning step**, I **adjusted** the "**number** of **trees** in the **forest**" by **modifying** the **(n_estimators)**. Then, yet again, as a **second tuning step,** I perform the **grid search** itself, using the GridSearchCV class from sickit-learn (I provide the random forest model and the hyperparameter grid to the GridSearchCV constructor and then I also provide the number of cross-validation folds as 5, verbose as 3 to print progress, and n_jobs as -1 to use all available CPU cores). Then, I fit the grid search object on the training data to find the best hyperparameters. Finally, the best hyperparameters were printed and the model is fit in accordance with them.



```
model = RandomForestClassifier()
```

```
param_grid = {
    'n_estimators': [50, 100, 200, 500],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}
```

```
grid_search = GridSearchCV(model, param_grid, cv=5, verbose=3, n_jobs=-1)
grid_search.fit(X_train, y_train.values.ravel())
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

```
▸        GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

```
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'max_depth': 7, 'max_features': 'log2', 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators':
100}
```

```
best_model = grid_search.best_estimator_
best_model.fit(X_train, y_train.values.ravel())
```

```
▾                    RandomForestClassifier
RandomForestClassifier(max_depth=7, max_features='log2', min_samples_leaf=4,
                       min_samples_split=10)
```

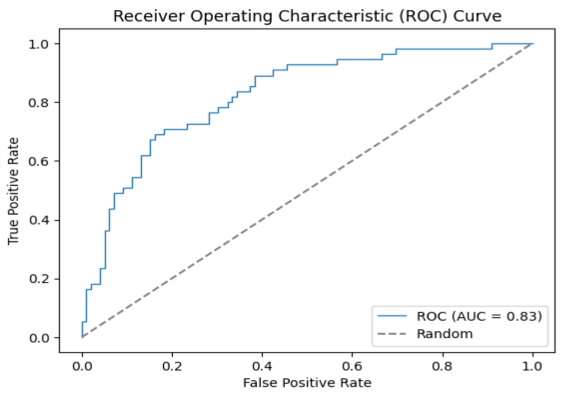
Afterwards, I made predictions on the test set using the predict method of the fitted model and store the predicted values in y_pred, based on test data split stored in x_test, in all **three models.**

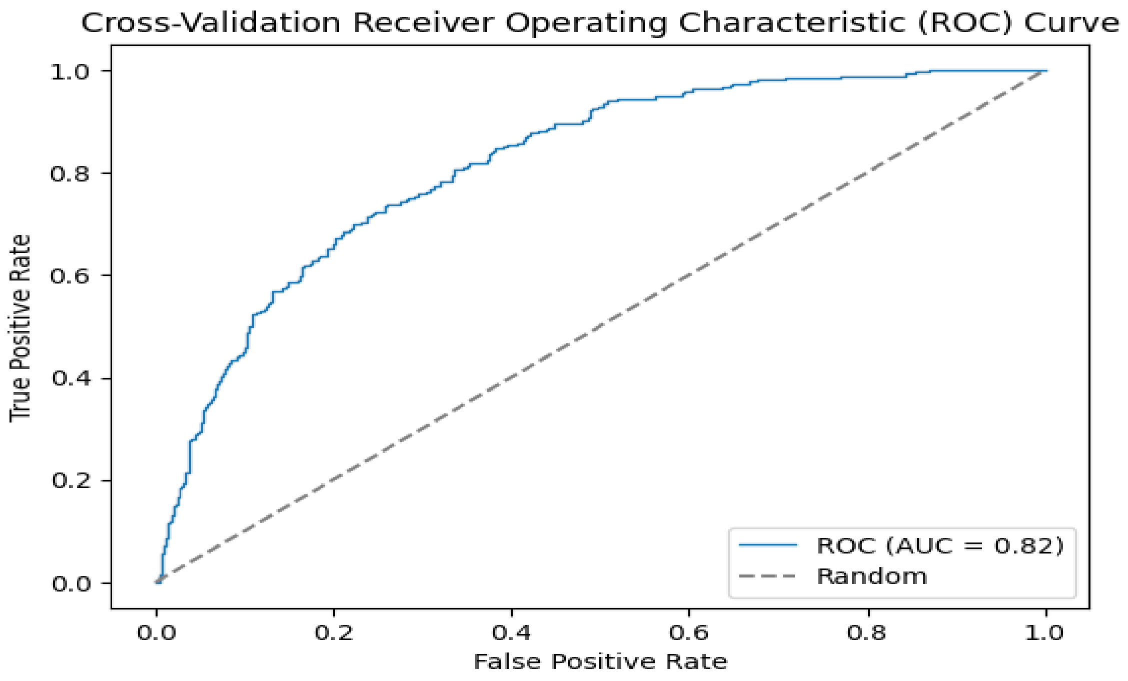
Part 3.3: Performance metrics

Now, I evaluated the performance of all models, using a **confusion matrix** that shows the number of true positives, true negatives, false positives, and false negatives, The **Receiver Operating Characteristic** (ROC), that shows the true positive rate (sensitivity) versus the false positive rate (1-specificity) for different threshold values, then I evaluate the performance of the models using **cross-validation**, which provides an estimate of the model's performance on unseen data, while also plotting the **ROC curve** for the **cross-validation** results, and finally making a **comparison** of the **cross-validation** scores, **mean cross-validation** score, and **standard deviation** of **cross-validation** scores **before** and **after** making the **change**, which allows for determining if the change improved the performance of the model.

Bayesian:

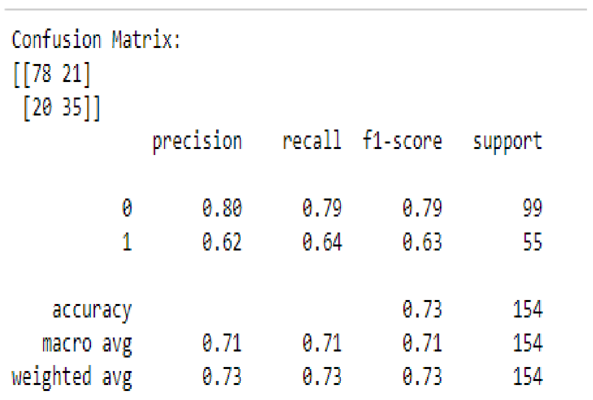
```
Confusion Matrix:
 [[79 20]
 [16 39]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.80      0.81        99
           1       0.66      0.71      0.68        55

    accuracy                           0.77       154
   macro avg       0.75      0.75      0.75       154
weighted avg       0.77      0.77      0.77       154
```



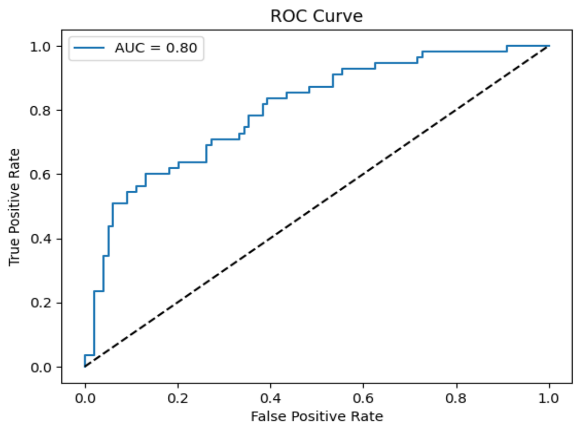


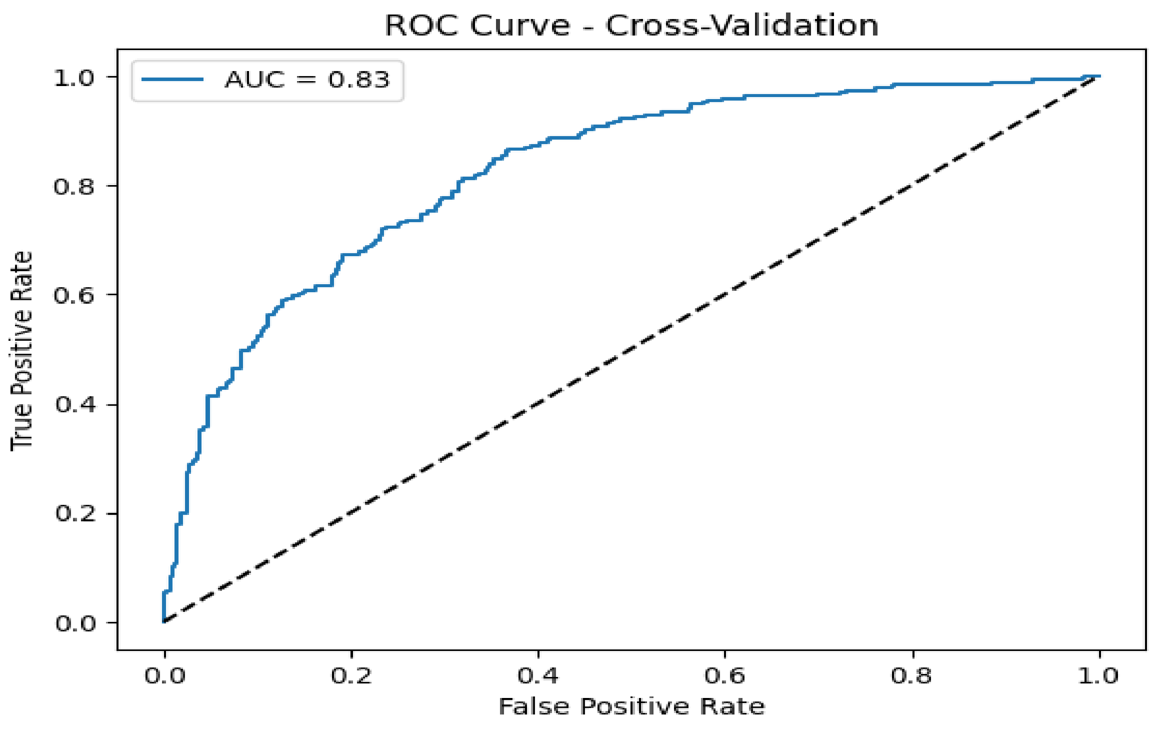
```
Cross-validation scores:  [0.75324675 0.72727273 0.74675325 0.78431373 0.74509804]
Mean cross-validation score:  0.7513368983957219
Standard deviation of cross-validation scores:  0.018601807594185577
```

Logistic (Baseline):



```
Confusion Matrix:
[[78 21]
 [20 35]]
              precision    recall  f1-score   support

           0       0.80      0.79      0.79        99
           1       0.62      0.64      0.63        55

    accuracy                           0.73       154
   macro avg       0.71      0.71      0.71       154
weighted avg       0.73      0.73      0.73       154
```



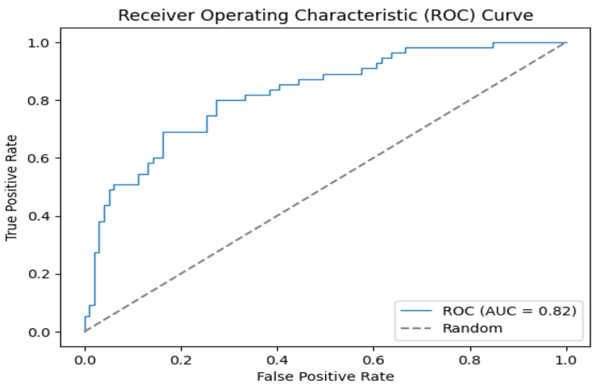


```
Cross-validation scores:  [0.77272727 0.74675325 0.75324675 0.81045752 0.77777778]
Mean cross-validation score:  0.7721925133689839
Standard deviation of cross-validation scores:  0.022368682173990264
```

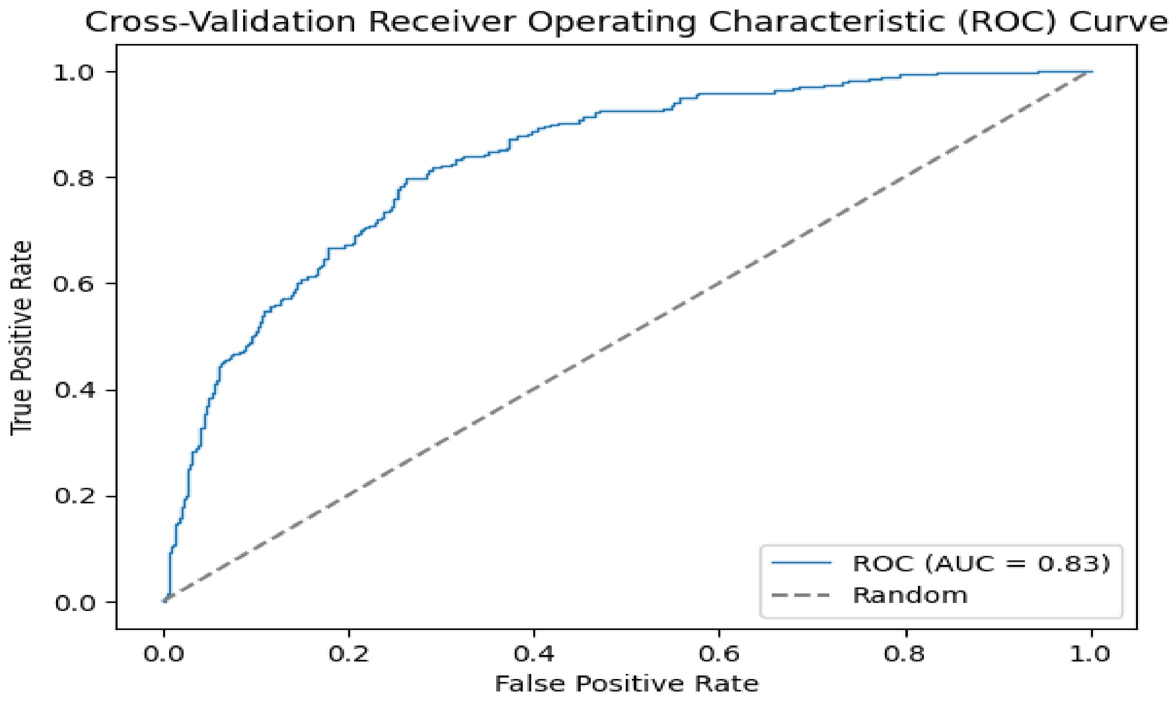
Increasing the number of **iterations tenfold** has resulted in a slight improvement in the mean cross-validation score from 0.763 to 0.772, and a decrease in the standard deviation of cross-validation scores from 0.042 to 0.022. This suggests that increasing the number of iterations has helped the model to better capture the patterns in the data and has improved its overall performance.

Random Forest:

```
Confusion Matrix:
 [[79 20]
 [17 38]]
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.80      0.81        99
           1       0.66      0.69      0.67        55

    accuracy                           0.76       154
   macro avg       0.74      0.74      0.74       154
weighted avg       0.76      0.76      0.76       154
```



w



```
Cross-validation scores:  [0.76623377 0.72077922 0.75324675 0.83006536 0.77777778]
Mean cross-validation score:  0.7696205755029284
Standard deviation of cross-validation scores:  0.03573559100445814
```

After **adjusting** the “**number** of **trees** in the **forest**” by **modifying** the **(n_estimators)** mean cross-validation score increased from 0.763 to 0.770, and the standard deviation decreased from 0.043 to 0.036. This suggests that the additional tuning step helped to fine-tune the model and improve its performance.

Part 4: Performance evaluation and comparison

Based on the performance metrics here is a comparison of the three models:

Bayesian:

- The model has an **accuracy** of **77**%, with a **precision** of **66**% for **class 1** and **83**% for **class 0**.
- The **cross-validation** scores have a **mean** of **75.13**% with a **standard deviation** of **1.86**%.
- The **ROC-AUC** score is 0**.83**.
- The confusion matrix indicates that the model predicted 79 true negatives and 39 true positives, with **20 false negatives** and **16 false positives**.

Random Forest:

- The model has an **accuracy** of **76**%, with a precision of **66**% for **class 1** and **82**% for **class 0.**
- The **cross-validation** scores have a **mean** of **76.96**% with a standard deviation of **3.57**%.
- The **ROC-AUC** score is **0.82**.
- The confusion matrix indicates that the model predicted **79 true negatives** and **38 true positives**, with **20 false negatives** and **17 false positives**.

Logistic Regression (Baseline):

- The model has an accuracy of **73**%, with a precision of **62**% for **class 1** and **80**% for **class 0.**
- The **cross-validation** scores have a **mean** of **77.22**% with a **standard deviation** of **2.24**%.
- The **ROC-AUC** score is 0**.80**.
- The confusion matrix indicates that the model predicted **78 true negatives** and **35 true positives**, with **21 false negatives** and **20 false positives**.

In terms of accuracy and precision, the Bayesian model performs slightly better than the Random Forest and Logistic Regression models. However, the Random Forest model has the highest ROC-AUC score and a slightly higher mean cross-validation score compared to the other two models.

Therefore, if the goal is to **maximize** overall **predictive power**, the **Random Forest model** may be the **best choice**. However, if **precision** is the most important metric, the **Bayesian model** may be the **best choice**.

Considering the size and the type of the dataset, it appears that the random forest algorithm had the highest mean cross-validation score and the highest cross-validation ROC, indicating that it is the best-performing algorithm in terms of predictive accuracy on this dataset. Logistic regression has also performed surprisingly well, with a slightly lower mean cross-validation score and cross-validation ROC compared to the random forest. The Bayesian implementation had the lowest mean cross-validation score and cross-validation ROC, suggesting that it did not perform as well on this dataset – however, it is only in terms of opportunity cost, given the complexity and computational demands of the algorithm. In terms of pure accuracy, Bayesian approach is **superior** to **both** the **baseline** algorithm and **random forest**.

Part 5: Conclusion and afterthoughts

The Bayesian implementation involved Bayesian logistic regression, which estimates the probability of the outcome using a Bayesian approach. Bayesian methods can be useful when prior information about the problem is available or when it is important to estimate the uncertainty around the estimated parameters. However. despite the accuracy of the Bayesian approach, the implementation had the lowest mean cross-validation score and cross-validation ROC, suggesting that it did not perform as well on this dataset relative to the aforementioned opportunity cost.
Random Forest algorithm is an ensemble learning method that builds multiple decision trees and combines their outputs to make a final prediction. This method is often effective on complex datasets with non-linear relationships between the features and outcome. In this case, the Pima Indians Diabetes dataset has many features that may interact with each other in complex ways to predict diabetes status, making the random forest algorithm well-suited for the task.

Logistic regression is a linear model that estimates the probability of a binary outcome based on one or more predictor variables. It assumes a linear relationship between the features and the log-odds of the outcome and can be useful when the relationship between the features and outcome is approximately linear or can be approximated as linear. I also attribute the rather unexpected success of Logistic Regression to good data preprocessing and the simplicity of the dataset.

In summary, the performance results of the three models suggest that the random forest algorithm is the best-performing algorithm on the Pima Indians Diabetes dataset, likely due to its ability to handle non-linear relationships between the features and outcome, considering all angles.

Considering this and the performance data acquired I have come to hold an opinion that baseline is objectively inferior as medical diagnosis algorithm compared to Random Forest and Bayesian classifications (despite the computational demand) in terms of the main goals of AI-assisted healthcare and most medical epidemiological research, which I identify as accuracy and predictive power.

Appendix

**1 et al. – “and others”.*

In [30]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
from imblearn.pipeline import Pipeline
from sklearn.utils import resample
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.ensemble import IsolationForest
```

In [31]:
```
data = pd.read_csv('diabetes.csv')
data
```

Out[31]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag
0	6	148	72	35	0	33.6	0.627	5
1	1	85	66	29	0	26.6	0.351	3
2	8	183	64	0	0	23.3	0.672	3
3	1	89	66	23	94	28.1	0.167	2
4	0	137	40	35	168	43.1	2.288	3
...	...	...	...	...	...	...	...	.
763	10	101	76	48	180	32.9	0.171	6
764	2	122	70	27	0	36.8	0.340	2
765	5	121	72	23	112	26.2	0.245	3
766	1	126	60	0	0	30.1	0.349	4
767	1	93	70	31	0	30.4	0.315	2

768 rows × 9 columns

Missing values: This code will first check if there are any missing values in the dataset, and if there are none, it will display a message.

In [32]:
```
missing = data.isnull().sum().sum()

if missing == 0:
    print("There are no missing values in the dataset.")
    fig, ax = plt.subplots()
    ax.text(0.5, 0.5, "No missing values", ha="center", va="center", fontsize=20, colo
    ax.axis("off")
    plt.show()
else:
    print(f"There are {missing} missing values in the dataset.")
    fig, ax = plt.subplots()
    ax.text(0.5, 0.5, "Missing values", ha="center", va="center", fontsize=20, color='
    ax.axis("off")
    plt.show()
```

```
There are no missing values in the dataset.
```

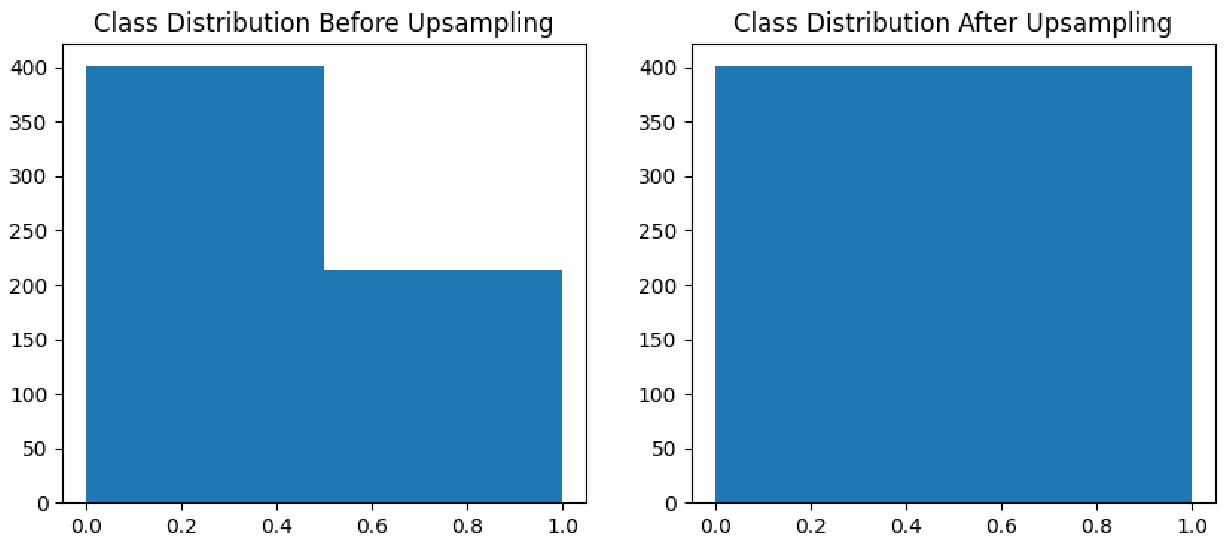
In [33]:
```
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
```

Upsampling the minority class.

In [34]:
```
X_upsampled, y_upsampled = resample(X_train[y_train == 1], y_train[y_train == 1],
                                    replace=True, n_samples=X_train[y_train == 0].shap
X_train_balanced = pd.concat([X_train[y_train == 0], X_upsampled])
y_train_balanced = pd.concat([y_train[y_train == 0], y_upsampled])
```

Plotting the results of unsampling.

In [35]:
```
fig, ax = plt.subplots(1, 2, figsize=(10, 4))
ax[0].hist(y_train, bins=2)
ax[0].set_title("Class Distribution Before Upsampling")
ax[1].hist(y_train_balanced, bins=2)
ax[1].set_title("Class Distribution After Upsampling")
plt.show()
```



The implementation of feature scaling.

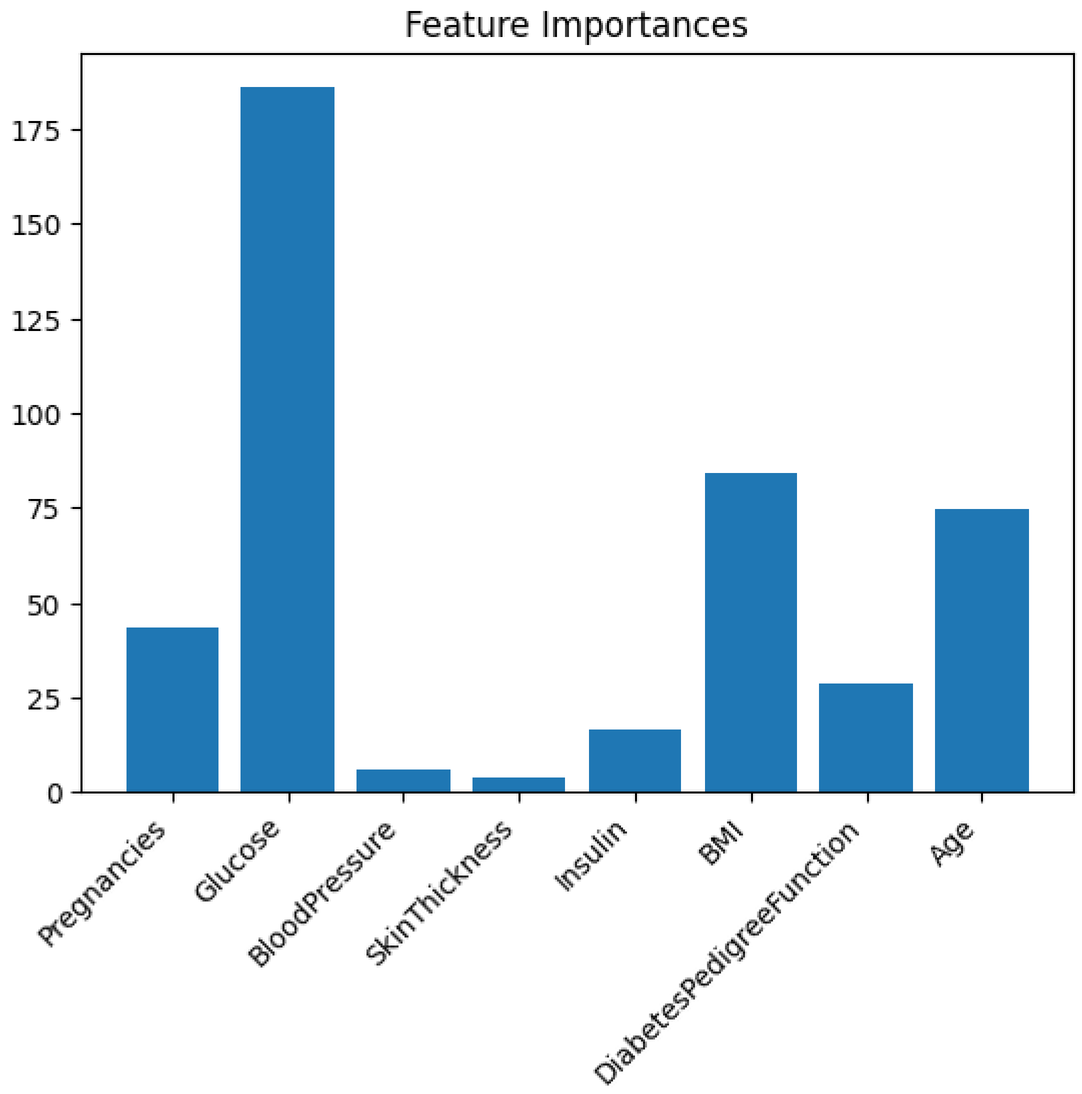
In [36]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_balanced)
X_test_scaled = scaler.transform(X_test)
```

Implementing the feature selection - the top 6 features with the highest F-value are selected for the final model.

In [37]:
```
selector = SelectKBest(f_classif, k=6)
X_train_selected = selector.fit_transform(X_train_scaled, y_train_balanced)
X_test_selected = selector.transform(X_test_scaled)
```

A bar plot of the feature importances after feature selection.

In [38]:
```
fig, ax = plt.subplots()
ax.bar(np.arange(X_train.shape[1]), selector.scores_)
ax.set_xticks(np.arange(X_train.shape[1]))
ax.set_xticklabels(X_train.columns, rotation=45, ha='right')
ax.set_title("Feature Importances")
plt.show()
```

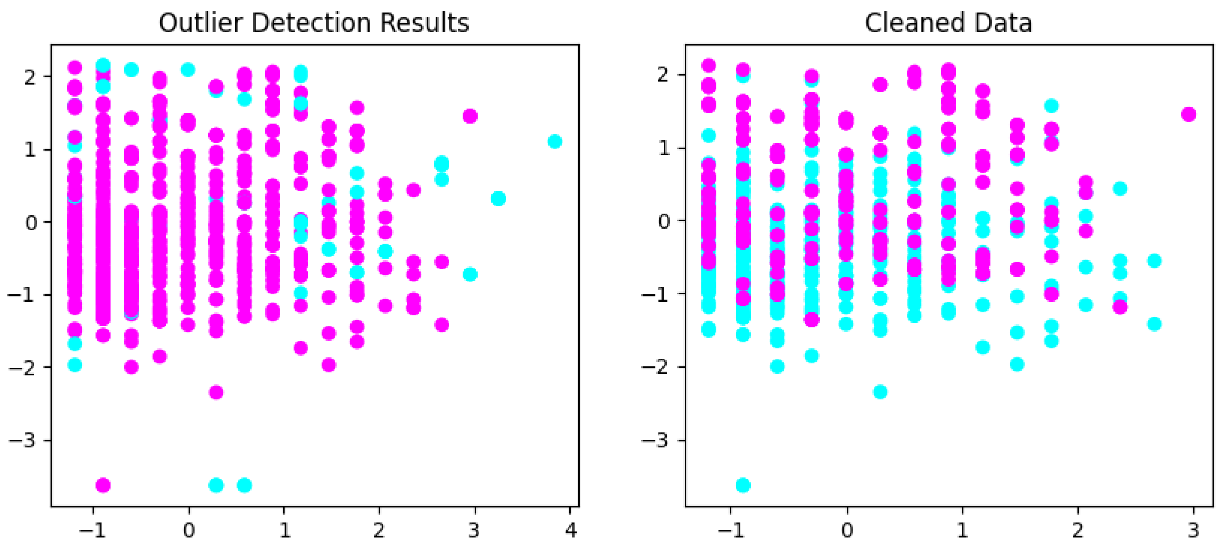


Detecting outliers using Isolation Forest.

In [39]:
```
outlier_detector = IsolationForest(contamination=0.1, random_state=42)
outlier_detector.fit(X_train_selected)
is_inlier = outlier_detector.predict(X_train_selected)
X_train_clean = X_train_selected[is_inlier == 1]
y_train_clean = y_train_balanced[is_inlier == 1]
```

Outlier detection results before and after cleaning.

In [40]:
```
fig, ax = plt.subplots(1, 2, figsize=(10, 4))
ax[0].scatter(X_train_selected[:, 0], X_train_selected[:, 1], c=is_inlier, cmap='cool'
ax[0].set_title("Outlier Detection Results")
ax[1].scatter(X_train_clean[:, 0], X_train_clean[:, 1], c=y_train_clean, cmap='cool')
ax[1].set_title("Cleaned Data")
plt.show()
```



Convert NumPy arrays to pandas dataframes in order to save the processed dataset to a single .csv file

In [41]:
```
X_train_clean_df = pd.DataFrame(X_train_clean, columns=X.columns[selector.get_support(
y_train_clean_df = pd.DataFrame(y_train_clean, columns=["target"])
```

In [42]:
```
processed_df = pd.concat([X_train_clean_df, y_train_clean_df], axis=1)
processed_df.to_csv("pima-indians-diabetes-processed.csv", index=False)
```

In [121…
```
import pandas as pd
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_predict,
import matplotlib.pyplot as plt
```

In [122…
```
X = pd.read_csv("pima-indians-diabetes-features.csv")
y = pd.read_csv("pima-indians-diabetes-target.csv")
```

In [123…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

In [124…
```
model = GaussianNB(var_smoothing=1e-8)
```

In [125…
```
param_grid = {'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1,
```

In [126…
```
grid_search = GridSearchCV(model, param_grid, cv=5, verbose=3, n_jobs=-1)
grid_search.fit(X_train, y_train.values.ravel())
```

```
Fitting 5 folds for each of 11 candidates, totalling 55 fits
```

Out[126]:
```
▸ GridSearchCV
▸ estimator: GaussianNB
    ▸ GaussianNB
```

In [127…
```
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'var_smoothing': 1e-09}
```

In [128…
```
best_model = grid_search.best_estimator_
best_model.fit(X_train, y_train.values.ravel())
```

Out[128]:
```
▾ GaussianNB
GaussianNB()
```

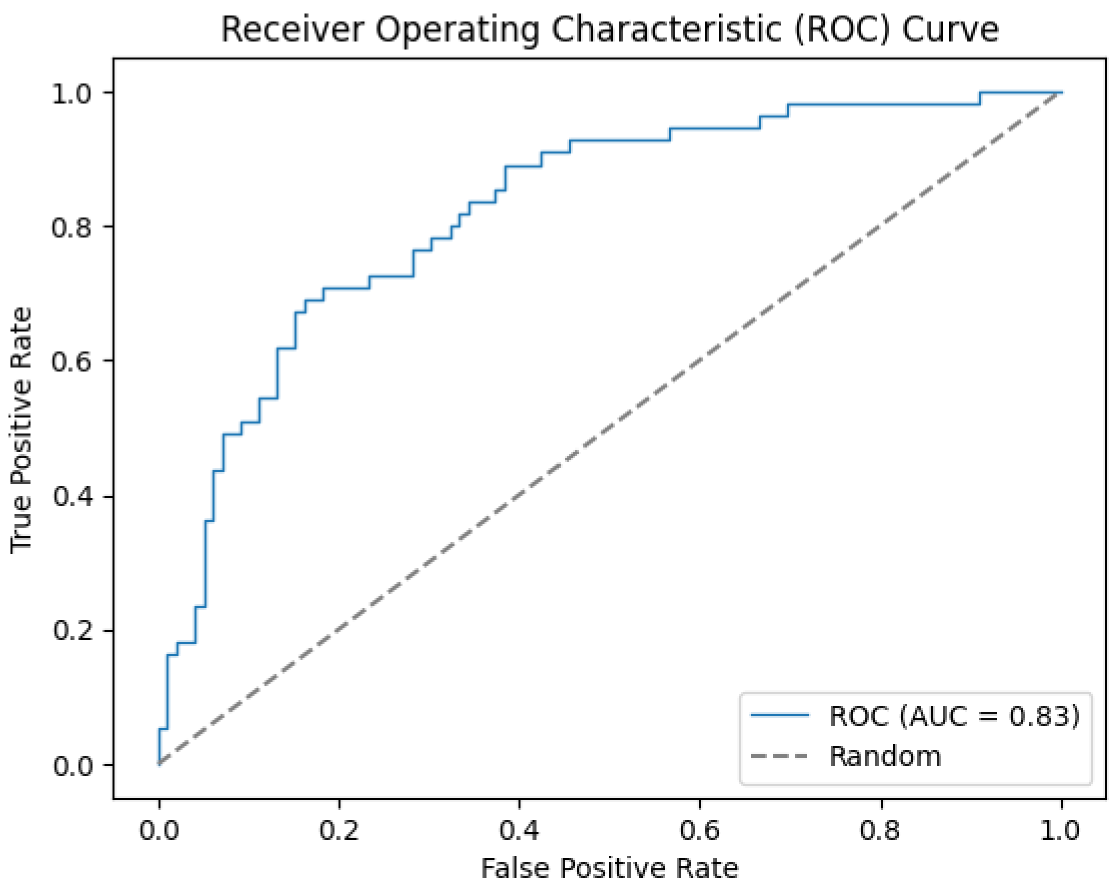
In [129…
```
y_pred = best_model.predict(X_test)
```

In [130…
```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[79 20]
 [16 39]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.80      0.81        99
           1       0.66      0.71      0.68        55

    accuracy                           0.77       154
   macro avg       0.75      0.75      0.75       154
weighted avg       0.77      0.77      0.77       154
```

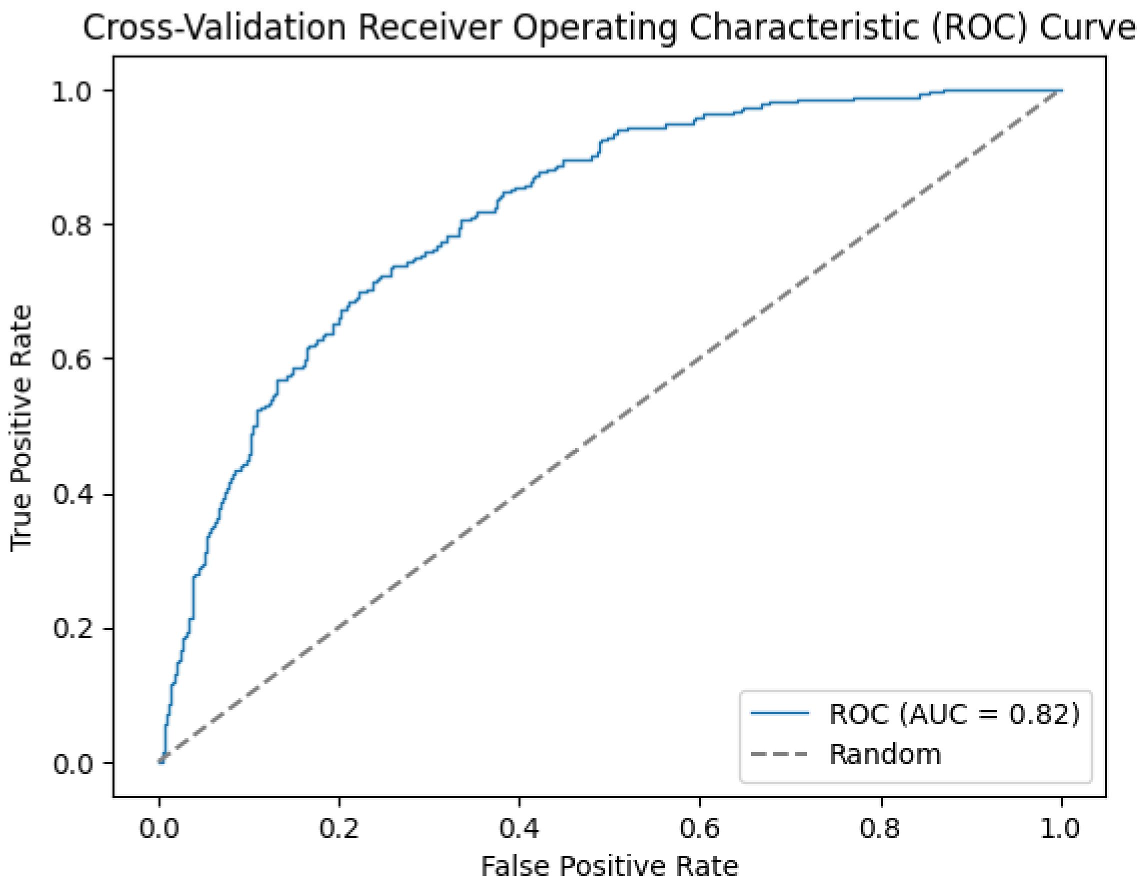
In [131…

```
y_pred_proba = best_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, lw=1, label='ROC (AUC = %0.2f)' % (roc_auc))
plt.plot([0, 1], [0, 1], '--', color='gray', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



In [132…

```
cv_scores = cross_val_score(best_model, X, y.values.ravel(), cv=5)
y_pred_cv = cross_val_predict(best_model, X, y.values.ravel(), cv=5, method='predict_
fpr_cv, tpr_cv, thresholds_cv = roc_curve(y.values.ravel(), y_pred_cv)
roc_auc_cv = auc(fpr_cv, tpr_cv)
plt.plot(fpr_cv, tpr_cv, lw=1, label='ROC (AUC = %0.2f)' % (roc_auc_cv))
plt.plot([0, 1], [0, 1], '--', color='gray', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Cross-Validation Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



In [133…

```
print("Cross-validation scores: ", cv_scores)
print("Mean cross-validation score: ", np.mean(cv_scores))
print("Standard deviation of cross-validation scores: ", np.std(cv_scores))
```

```
Cross-validation scores:  [0.75324675 0.72727273 0.74675325 0.78431373 0.74509804]
Mean cross-validation score:  0.7513368983957219
Standard deviation of cross-validation scores:  0.018601807594185577
```

In [112…
```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_predict,
import matplotlib.pyplot as plt
```

In [113…
```
X = pd.read_csv("pima-indians-diabetes-features.csv")
y = pd.read_csv("pima-indians-diabetes-target.csv")
```

In [114…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

In [115…
```
model = RandomForestClassifier()
```

In [116…
```
param_grid = {
    'n_estimators': [50, 100, 200, 500],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}
```

In [117…
```
grid_search = GridSearchCV(model, param_grid, cv=5, verbose=3, n_jobs=-1)
grid_search.fit(X_train, y_train.values.ravel())
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

Out[117]:
```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [118…
```
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'max_depth': 7, 'max_features': 'log2', 'min_samples_leaf':
4, 'min_samples_split': 10, 'n_estimators': 100}
```

In [119…
```
best_model = grid_search.best_estimator_
best_model.fit(X_train, y_train.values.ravel())
```

Out[119]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=7, max_features='log2', min_samples_leaf=4,
                       min_samples_split=10)
```

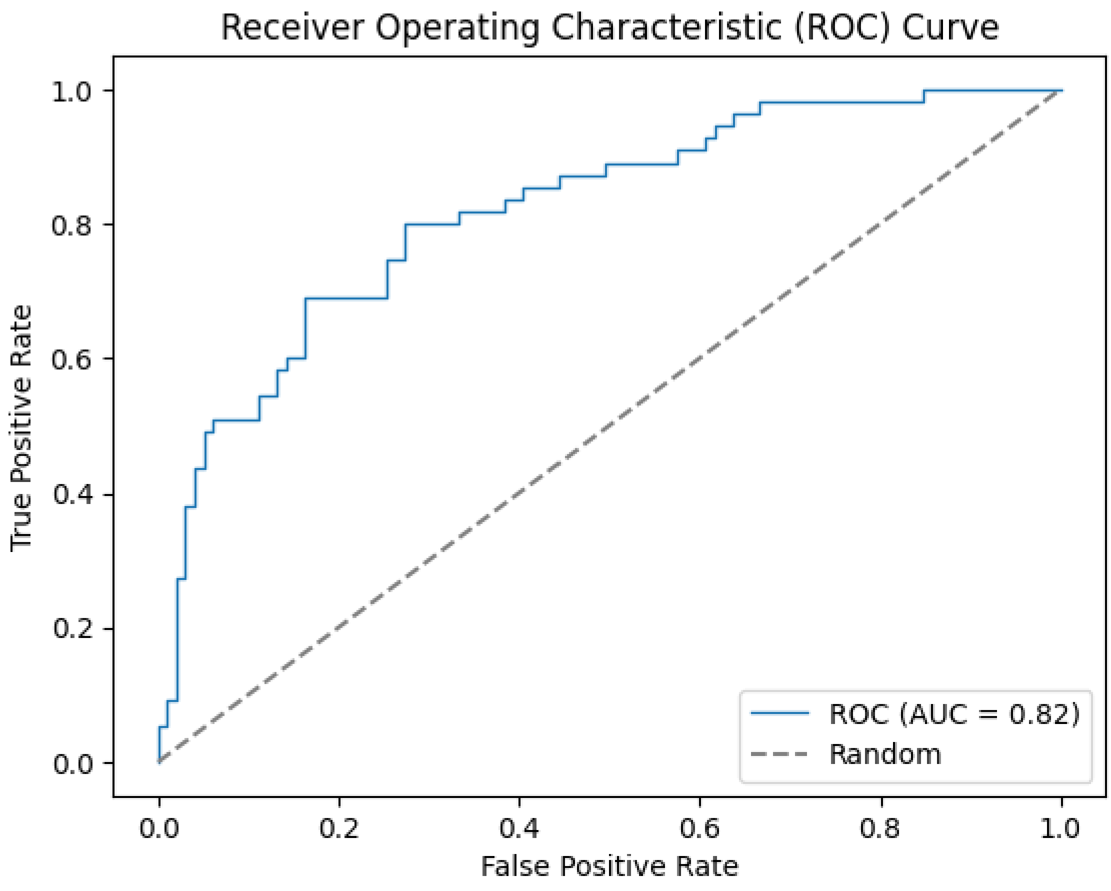
In [120…
```
y_pred = best_model.predict(X_test)
```

In [121…
```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[79 20]
 [17 38]]
Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.80      0.81        99
           1       0.66      0.69      0.67        55

    accuracy                           0.76       154
   macro avg       0.74      0.74      0.74       154
weighted avg       0.76      0.76      0.76       154
```

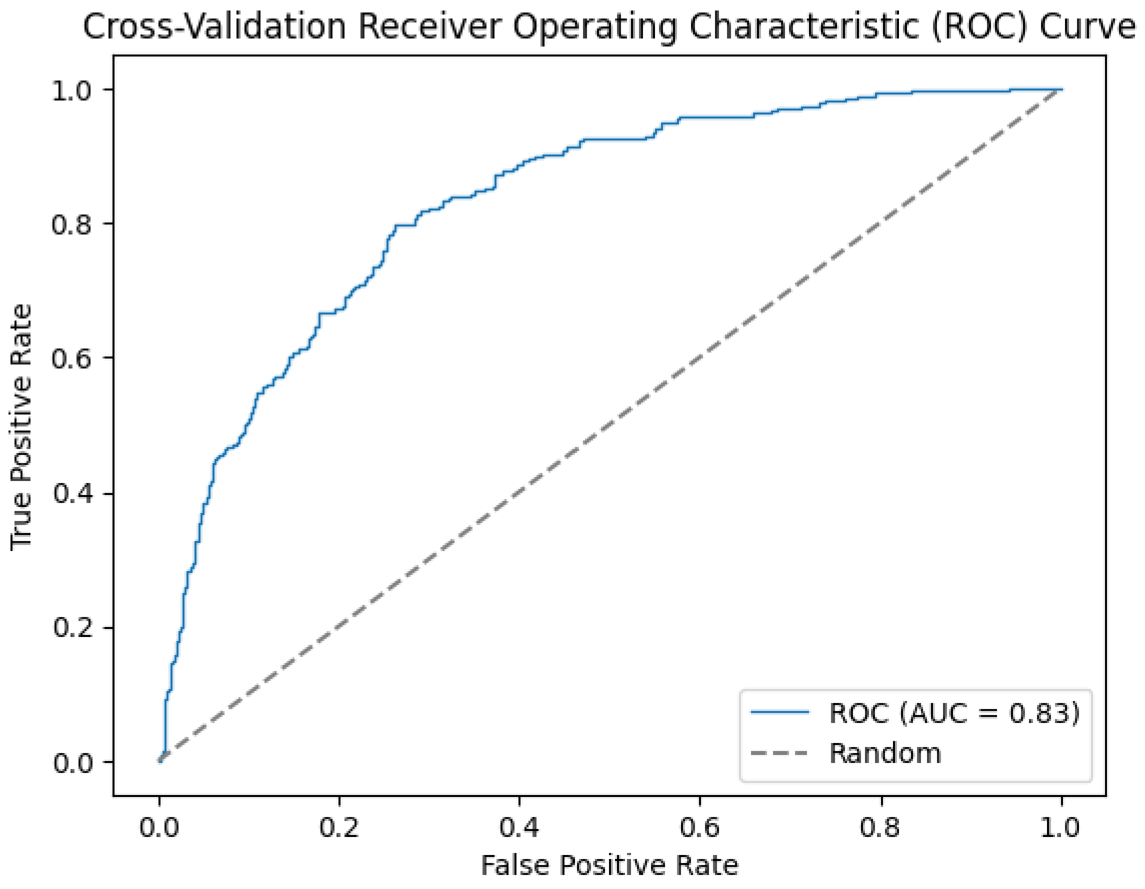
In [122…

```
y_pred_proba = best_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, lw=1, label='ROC (AUC = %0.2f)' % (roc_auc))
plt.plot([0, 1], [0, 1], '--', color='gray', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



In [123…

```
cv_scores = cross_val_score(best_model, X, y.values.ravel(), cv=5)
y_pred_cv = cross_val_predict(best_model, X, y.values.ravel(), cv=5, method='predict_
fpr_cv, tpr_cv, thresholds_cv = roc_curve(y.values.ravel(), y_pred_cv)
roc_auc_cv = auc(fpr_cv, tpr_cv)
plt.plot(fpr_cv, tpr_cv, lw=1, label='ROC (AUC = %0.2f)' % (roc_auc_cv))
plt.plot([0, 1], [0, 1], '--', color='gray', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Cross-Validation Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



In [124…

```
print("Cross-validation scores: ", cv_scores)
print("Mean cross-validation score: ", np.mean(cv_scores))
print("Standard deviation of cross-validation scores: ", np.std(cv_scores))
```

```
Cross-validation scores:  [0.76623377 0.72077922 0.75324675 0.83006536 0.77777778]
Mean cross-validation score:  0.7696205755029284
Standard deviation of cross-validation scores:  0.03573559100445814
```

In [67]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc_au
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predi
import matplotlib.pyplot as plt
```

In [68]:
```
X = pd.read_csv("pima-indians-diabetes-features.csv")
y = pd.read_csv("pima-indians-diabetes-target.csv")
```

In [69]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

In [70]:
```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}
```

In [71]:
```
model = LogisticRegression(max_iter=1000)
```

In [72]:
```
grid_search = GridSearchCV(model, param_grid, cv=5)
```

In [73]:
```
grid_search.fit(X_train, y_train.values.ravel())
```

Out[73]:
```
▸ GridSearchCV
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

In [74]:
```
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

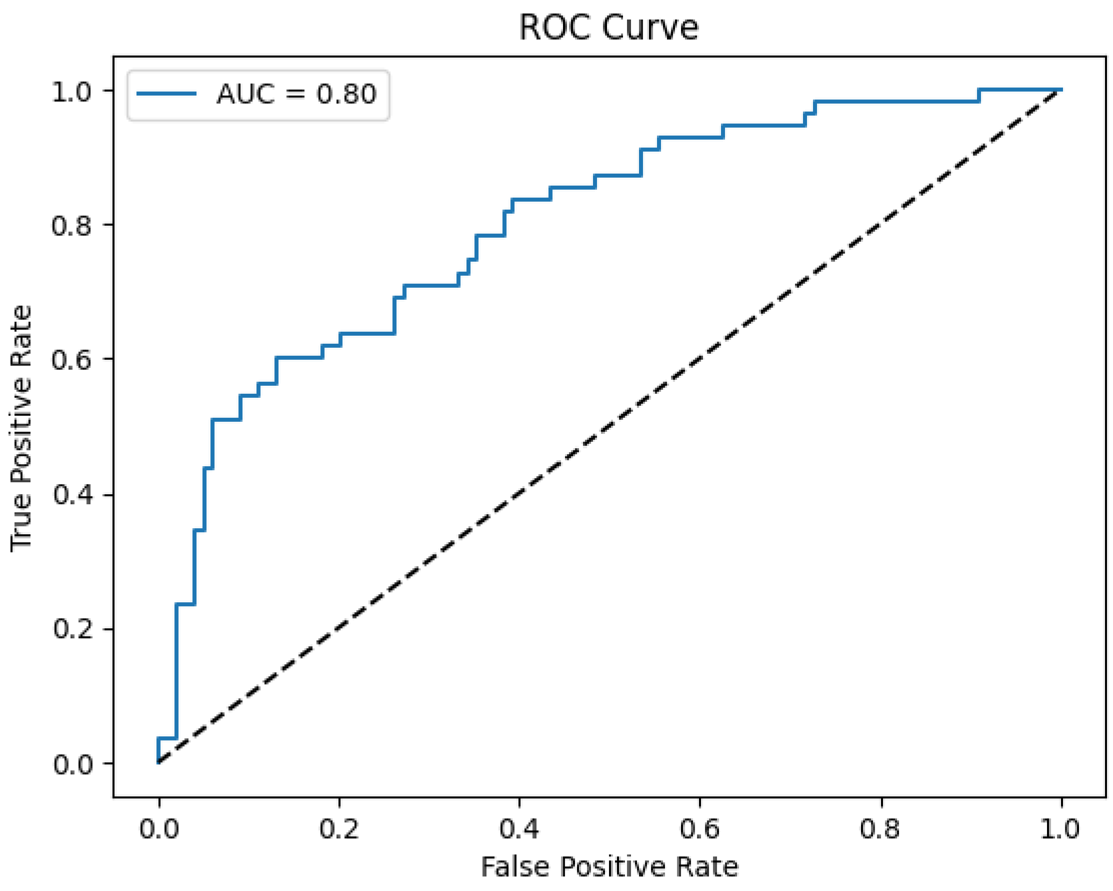
```
Best parameters:  {'C': 0.01}
Best score:  0.7687591630014661
```

In [75]:
```
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

In [76]:
```
confusion = confusion_matrix(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[78 21]
 [20 35]]
              precision    recall  f1-score   support

           0       0.80      0.79      0.79        99
           1       0.62      0.64      0.63        55

    accuracy                           0.73       154
   macro avg       0.71      0.71      0.71       154
weighted avg       0.73      0.73      0.73       154
```

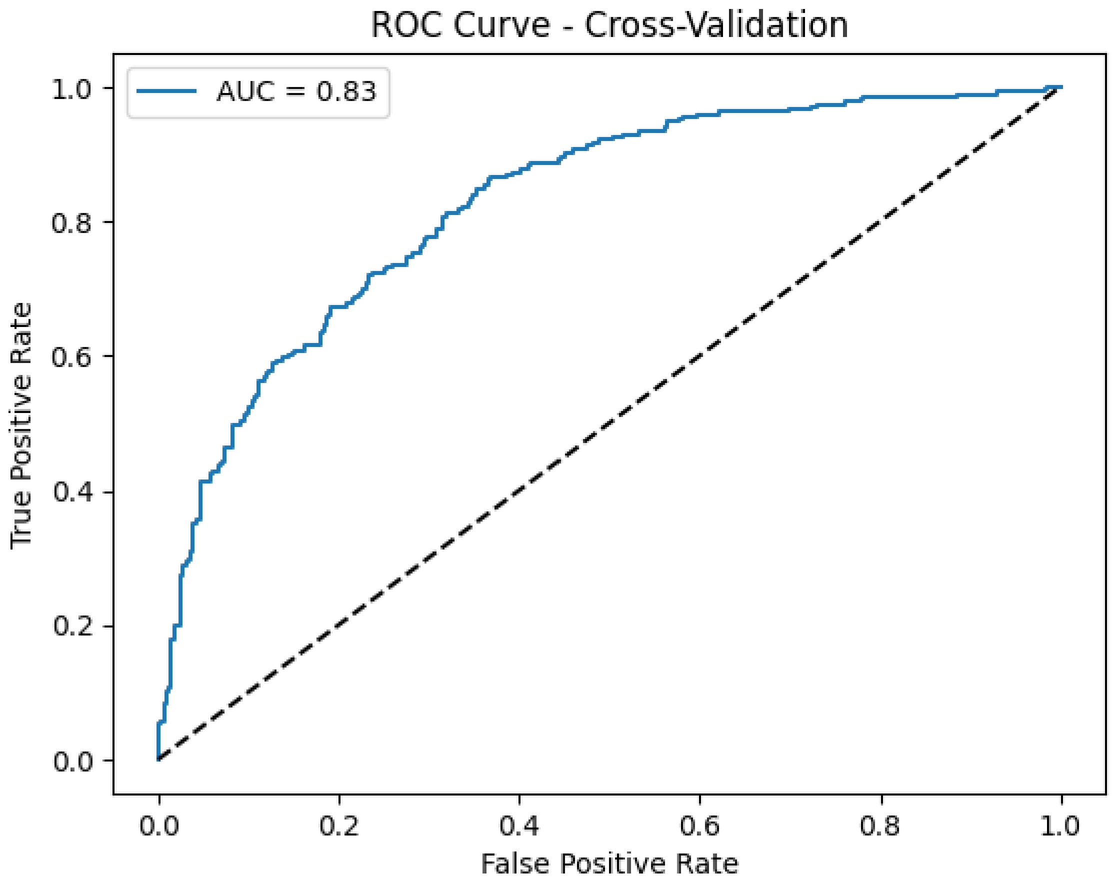
In [77]:
```
y_prob = best_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```



In [78]:
```
y_pred_cv = cross_val_predict(best_model, X, y.values.ravel(), cv=5, method='predict_
fpr_cv, tpr_cv, thresholds_cv = roc_curve(y.values.ravel(), y_pred_cv)
roc_auc_cv = roc_auc_score(y.values.ravel(), y_pred_cv)
print(f"Cross-Validation Accuracy: {grid_search.best_score_:.2f}")
plt.plot(fpr_cv, tpr_cv, label=f"AUC = {roc_auc_cv:.2f}")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Cross-Validation")
plt.legend()
plt.show
```

```
Cross-Validation Accuracy: 0.77
```

Out[78]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [79]:
```
cv_scores = cross_val_score(model, X, y.values.ravel(), cv=5)

print("Cross-validation scores: ", cv_scores)
print("Mean cross-validation score: ", np.mean(cv_scores))
print("Standard deviation of cross-validation scores: ", np.std(cv_scores))
```

```
Cross-validation scores:  [0.77272727 0.74675325 0.75324675 0.81045752 0.77777778]
Mean cross-validation score:  0.7721925133689839
Standard deviation of cross-validation scores:  0.022368682173990264
```