Assignment 4: Pipelines and Hyperparameter Tuning

- **Full Name** = Alessandro Baldassarre
- **UCID** = 30163507

In this assignment, you will be putting together everything you have learned so far. You will need to find your own dataset, do all the appropriate preprocessing, test different supervised learning models, and evaluate the results. More details for each step can be found below. You will also be asked to describe the process by which you came up with the code. More details can be found below. Please cite any websites or AI tools that you used to help you with this assignment.

For this assignment, in addition to your .ipynb file, please also attach a PDF file. To generate this PDF file, you can use the print function (located under the "File" within Jupyter Notebook). Name this file ENGG444_Assignment##_yourUCID.pdf (this name is similar to your main .ipynb file). We will evaluate your assignment based on the two files and you need to provide both.

Question	Point(s)
1. Preprocessing Tasks	
1.1	2
1.2	2
1.3	4
2. Pipeline and Modeling	
2.1	3
2.2	6
2.3	5
2.4	3
3. Bonus Question	**2**
Total	**25**

0. Dataset

This data is a subset of the **Heart Disease Dataset**, which contains information about patients with possible coronary artery disease. The data has **14 attributes** and **294 instances**. The attributes include demographic, clinical, and laboratory features, such as age, sex, chest pain type, blood pressure, cholesterol, and electrocardiogram results. The last attribute is the **diagnosis of heart disease**, which is a categorical variable with values from 0 (no presence) to 4 (high presence). The data can be used for **classification** tasks, such as predicting the presence or absence of heart disease based on the other attributes.

```
import pandas as pd

# Define the data source link
_link = 'https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.hungarian.data'

# Read the CSV file into a Pandas DataFrame, considering '?' as missing values
df = pd.read_csv(_link, na_values='?',
                 names=['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs',
                        'restecg', 'thalach', 'exang', 'oldpeak', 'slope',
                        'ca', 'thal', 'num'])

# Display the DataFrame
display(df)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	th
0	28	1	2	130.0	132.0	0.0	2.0	185.0	0.0	0.0	NaN	NaN	N
1	29	1	2	120.0	243.0	0.0	0.0	160.0	0.0	0.0	NaN	NaN	N
2	29	1	2	140.0	NaN	0.0	0.0	170.0	0.0	0.0	NaN	NaN	N
3	30	0	1	170.0	237.0	0.0	1.0	170.0	0.0	0.0	NaN	NaN	(
4	31	0	2	100.0	219.0	0.0	1.0	150.0	0.0	0.0	NaN	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	
289	52	1	4	160.0	331.0	0.0	0.0	94.0	1.0	2.5	NaN	NaN	N
290	54	0	3	130.0	294.0	0.0	1.0	100.0	1.0	0.0	2.0	NaN	N
291	56	1	4	155.0	342.0	1.0	0.0	150.0	1.0	3.0	2.0	NaN	N
292	58	0	2	180.0	393.0	0.0	0.0	110.0	1.0	1.0	2.0	NaN	'
293	65	1	4	130.0	275.0	0.0	1.0	115.0	1.0	1.0	2.0	NaN	N

294 rows × 14 columns

Next steps: Generate code with df View recommended plots

1. Preprocessing Tasks

- **1.1** Find out which columns have more than 60% of their values missing and drop them from the data frame. Explain why this is a reasonable way to handle these columns. **(2 Points)**
- **1.2** For the remaining columns that have some missing values, choose an appropriate imputation method to fill them in. You can use the `SimpleImputer` class from `sklearn.impute` or any other method you prefer. Explain why you chose this method and how it affects the data. **(2 Points)**
- **1.3** Assign the `num` column to the variable `y` and the rest of the columns to the variable `X`. The `num` column indicates the presence or absence of heart disease based on the angiographic disease status of the patients. Create a `ColumnTransformer` object that applies different preprocessing steps to different subsets of features. Use `StandardScaler` for the numerical features, `OneHotEncoder` for the categorical features, and `passthrough` for the binary features. List the names of the features that belong to each group and explain why they need different transformations. You will use this `ColumnTransformer` in a pipeline in the next question. **(4 Points)**

Answer:

- **1.1** .....
- This is a reasonable way to handle these columns because a column that contains more than 60% NaN values is lacking too much data to provide reliable results and will negatively impact the model. The high amounts of missing data could lead to biased models. Providing a ton of missing values to the model can also hinder its performance. If NaN values are not dropped, then imputing them is another solution but when there are so many imputation could get complex and some imputation strategies might not be justified. Also removing NaN values just makes the model simpler as well which in turn makes it more manageable.

```
# 1.1
# Add necessary code here.

missing_data = df.columns[df.isnull().mean() > 0.6]

df = df.drop(columns=missing_data)

missing_data, df.head()
```

```
(Index(['slope', 'ca', 'thal'], dtype='object'),
    age  sex  cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  num
 0   28    1   2     130.0  132.0  0.0      2.0    185.0    0.0      0.0    0
 1   29    1   2     120.0  243.0  0.0      0.0    160.0    0.0      0.0    0
 2   29    1   2     140.0    NaN  0.0      0.0    170.0    0.0      0.0    0
 3   30    0   1     170.0  237.0  0.0      1.0    170.0    0.0      0.0    0
 4   31    0   2     100.0  219.0  0.0      1.0    150.0    0.0      0.0    0)
```

Answer:

- **1.2** .....

- I chose two separate methods for the different types of data.
  - For numeric data, mean or median are two good choices for imputing missing values. I chose median because it is preferable for data that can be skewed and/or sensitive to outliers such as blood pressure or cholestorol. Mean is a better choice for data that follows a normal distribution and does not contain significant outliers. Since blood pressures can spike and what not I figured it would be a better option to go with median.
  - As for categorical data, I chose to go with most frequent because it is straightforward and when there is a category that is most frequent it just makes sense (is intuitive) for the missing value to be part of that majority. For example, if it seems that males more often get heart disease than females then for the sex attribute most frequent would impute the values with male which is a valid assumption.

```
# 1.2
# Add necessary code here.

from sklearn.impute import SimpleImputer

# Separate columns by type
numeric_columns = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
categorical_columns = ['sex', 'cp', 'fbs', 'restecg', 'num', 'exang']

# Apply Median Imputation for numeric columns
median_imputer = SimpleImputer(strategy='median')
df[numeric_columns] = median_imputer.fit_transform(df[numeric_columns])

# Apply Most Frequent Imputation for categorical columns
freq_imputer = SimpleImputer(strategy='most_frequent')
df[categorical_columns] = freq_imputer.fit_transform(df[categorical_columns])

df
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	num
0	28.0	1.0	2.0	130.0	132.0	0.0	2.0	185.0	0.0	0.0	0.0
1	29.0	1.0	2.0	120.0	243.0	0.0	0.0	160.0	0.0	0.0	0.0
2	29.0	1.0	2.0	140.0	243.0	0.0	0.0	170.0	0.0	0.0	0.0
3	30.0	0.0	1.0	170.0	237.0	0.0	1.0	170.0	0.0	0.0	0.0
4	31.0	0.0	2.0	100.0	219.0	0.0	1.0	150.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
289	52.0	1.0	4.0	160.0	331.0	0.0	0.0	94.0	1.0	2.5	1.0
290	54.0	0.0	3.0	130.0	294.0	0.0	1.0	100.0	1.0	0.0	1.0
291	56.0	1.0	4.0	155.0	342.0	1.0	0.0	150.0	1.0	3.0	1.0
292	58.0	0.0	2.0	180.0	393.0	0.0	0.0	110.0	1.0	1.0	1.0
293	65.0	1.0	4.0	130.0	275.0	0.0	1.0	115.0	1.0	1.0	1.0

294 rows × 11 columns

Next steps: Generate code with df View recommended plots

Answer:

- **1.3** .....
  - Standard Scaler for numerical features 'num':
    - Scaling the numeric data is important because it will normalize the data and ensure that every feature contributes equally to the distance computations in models like KNN or SVM and will also optimize for algorithms like gradient descent.
  - OneHotEncoder for categorical features 'cat':
    - OneHotEncoder will transform every categorical feature with n possible values into n binary features with only one active feature. This is required for categorical data since models generally will require numerical inputs so this conversion of the categorical data is necessary to be able to provide this data to the model.
  - Passthrough for binary features 'bin':

- Passthrough is used for binary feature data since it is already in numerical format and only consists of two categories it can be passed through, hence the name 'passthrough', unchanged since this data does not require any transformations.

```
# 1.3
# Add necessary code here.
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

target = 'num'

y = df[target]               # Assigning num to y
X = df.drop(target, axis=1)  # Assigning everything else to X

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

# Lists of column names
numerical_features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
categorical_features = ['cp', 'restecg']
binary_features = ['sex', 'fbs', 'exang']

# Creating the ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),  # Standard Scaler for Numerical Features
        ('cat', OneHotEncoder(), categorical_features), # OneHotEncoder for categorical features
        ('bin', 'passthrough', binary_features)         # Passthrough for binary features
    ]
)
```

2. Pipeline and Modeling

- **2.1** Create **three** `Pipeline` objects that take the column transformer from the previous question as the first step and add one or more models as the subsequent steps. You can use any models from `sklearn` or other libraries that are suitable for binary classification. For each pipeline, explain **why** you selected the model(s) and what are their **strengths and weaknesses** for this data set. **(3 Points)**
- **2.2** Use `GridSearchCV` to perform a grid search over the hyperparameters of each pipeline and find the best combination that maximizes the cross-validation score. Report the best parameters and the best score for each pipeline. Then, update the hyperparameters of each pipeline using the best parameters from the grid search. **(6 Points)**
- **2.3** Form a stacking classifier that uses the three pipelines from the previous question as the base estimators and a meta-model as the `final_estimator`. You can choose any model for the meta-model that is suitable for binary classification. Explain **why** you chose the meta-model and how it combines the predictions of the base estimators. Then, use `StratifiedKFold` to perform a cross-validation on the stacking classifier and present the accuracy scores and F1 scores for each fold. Report the mean and the standard deviation of each score in the format of `mean ± std`. For example, `0.85 ± 0.05`. Interpret the results and compare them with the baseline scores from **question 2.2**. **(5 Points)**
- **2.4**: Interpret the final results of the stacking classifier and compare its performance with the individual models. Explain how stacking classifier has improved or deteriorated the prediction accuracy and F1 score, and what are the possible reasons for that. **(3 Points)**

Answer:

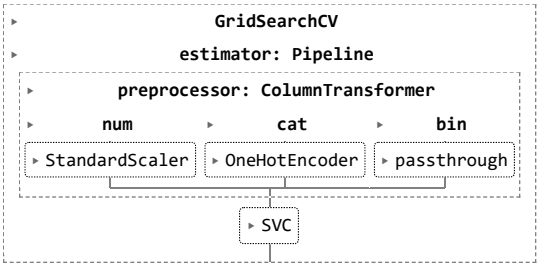
- **2.1** .....

```
# 2.1
# Add necessary code here.
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Creating pipelines
lr_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('classifier', LogisticRegression())])

rf_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('classifier', RandomForestClassifier())])

svm_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('classifier', SVC())])
```

Answer:

- **2.2** .....

```
# 2.2
# Add necessary code here.
from sklearn.metrics import make_scorer, accuracy_score, f1_score

# Define parameter grids
param_grid_lr = {'classifier__C': [0.01, 0.1, 1, 10]}

param_grid_rf = {'classifier__n_estimators': [50, 100, 200],
                 'classifier__max_depth': [None, 10, 20]}

param_grid_svm = {'classifier__C': [0.1, 1, 10],
                  'classifier__kernel': ['linear', 'rbf']}

scoring = {
    'accuracy': make_scorer(accuracy_score), # Scoring based on accuracy_score
    'f1_score': make_scorer(f1_score)        # Scoring based on F1_score
}

# Perform Grid Search
grid_search_lr = GridSearchCV(lr_pipeline, param_grid_lr, cv=5, scoring=scoring, refit='f1_score', n_jobs=-1)
grid_search_rf = GridSearchCV(rf_pipeline, param_grid_rf, cv=5, scoring=scoring, refit='f1_score', n_jobs=-1)
grid_search_svm = GridSearchCV(svm_pipeline, param_grid_svm, cv=5, scoring=scoring, refit='f1_score', n_jobs=-1)

# Fit the Random Forest model
grid_search_rf.fit(X_train, y_train)

# Fit the Logistic Regression model
grid_search_lr.fit(X_train, y_train)

# Fit the SVM model
grid_search_svm.fit(X_train, y_train)
```




```
# 2.2 continued
# Get the best parameters for Random Forest based on F1 score
best_params_rf = grid_search_rf.best_params_

# Get the best parameters for Logistic Regression based on F1 score
best_params_lr = grid_search_lr.best_params_

# Get the best parameters for SVM based on F1 score
best_params_svm = grid_search_svm.best_params_
```

```
# Access the results for Random Forest with both scoring metrics
results_rf = grid_search_rf.cv_results_

# Access the results for Logistic Regression with both scoring metrics
results_lr = grid_search_lr.cv_results_

# Access the results for SVM with both scoring metrics
results_svm = grid_search_svm.cv_results_

# Print the results for Random Forest
print("Random Forest Results:")
print("Accuracy scores:", results_rf['mean_test_accuracy'])               # Print mean_test_accuracy
print("F1 scores:", results_rf['mean_test_f1_score'])                     # Print mean F1 scores
print("\nBest Parameters for Random Forest based on F1:", best_params_rf)  # Print best parameters based on F1 score


# Print the results for Logistic Regression
print("\nLogistic Regression Results:")
print("Accuracy scores:", results_lr['mean_test_accuracy'])                   # Print mean_test_accuracy
print("F1 scores:", results_lr['mean_test_f1_score'])                         # Print mean F1 scores
print("\nBest Parameters for Logistic Regression based on F1:", best_params_lr) # Print best parameters based on F1 score


# Print the results for SVM
print("\nSVM Results:")
print("Accuracy scores:", results_svm['mean_test_accuracy'])      # Print mean_test_accuracy
print("F1 scores:", results_svm['mean_test_f1_score'])            # Print mean F1 scores
print("\nBest Parameters for SVM based on F1:", best_params_svm)  # Print best parameters based on F1 score
```

```
Random Forest Results:
Accuracy scores: [0.80425532 0.8        0.80851064 0.81276596 0.8        0.80425532
 0.8        0.80851064 0.8       ]
F1 scores: [0.71423829 0.71320172 0.71895585 0.72160401 0.70577054 0.71563772
 0.70463027 0.72503217 0.70852534]

Best Parameters for Random Forest based on F1: {'classifier__max_depth': 20, 'classifier__n_estimators': 100}

Logistic Regression Results:
Accuracy scores: [0.79574468 0.8212766  0.82978723 0.82553191]
F1 scores: [0.62234129 0.73042437 0.75242662 0.74802867]

Best Parameters for Logistic Regression based on F1: {'classifier__C': 1}

SVM Results:
Accuracy scores: [0.82553191 0.76595745 0.8        0.81276596 0.80851064 0.77021277]
F1 scores: [0.75070334 0.55458003 0.70892711 0.73109337 0.71831127 0.67628796]

Best Parameters for SVM based on F1: {'classifier__C': 0.1, 'classifier__kernel': 'linear'}
```

Answer:

- **2.3** .....

```
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.linear_model import LogisticRegression
import numpy as np

# Define the stacking classifier
```

Answer:

- **2.4** .....
  - The stacking classifier showed an improvement in performance in comparison to the models. This improvement can be shown in both the accuracy and F1 score. Although the improvement is not drastic it is still an improvement from the mean of the accuracy scores and f1 scores provided by each model.
    - In terms of accuracy, the stacking classifier achieves the greatest accuracy with a value of 0.83. This suggests the LR model was effective at using the different strengths of the base models which led to a more accurate generalization.
    - In terms of f1 score, the stacking classifier achieves the greatest f1 score with a value of 0.74. This means the stacking classifier predicts more accurately and maintains good balance between precision and recall.
  - Some reasons for improvement include:
    - The stacking classifier benefits from the diversity of the base models (the random forest, the logistic regressor, and the support vector machine). Each model has its own individual strengths and weaknesses which are combined by the stacking classifier to capture complex patterns in the data which could have been missed by any of the base models.
    - The logistic regression meta-model is effective at weighing the predictions from the base models, which resulted in the improvements of accuracy and f1 score. It appears to find a meaningful way to combine the various outcomes of the base models into its own final outcome which outperforms each base models individual predictions.

Bonus Question: The stacking classifier has achieved a high accuracy and F1 score, but there may be still room for improvement. Suggest **two** possible ways to improve the modeling using the stacking classifier, and explain **how** and **why** they could improve the performance. **(2 points)**

Answer:

1. Using different base models
   - Adding more base models to the stacking classifier or potentially replacing the ones that already exist with other base models that could improve performance. This could increase the diversity of predictions of the stacking classifier which could result in the model capturing even more patterns in the data. Potential models to add could include Gradient Boosting.
   - This would improve performance because the key to stacking is diversity among the base models. Diversity results in the stacking classifier capturing various aspects of the data, this in turn means the model can then make use of a wider range of information when predicting.
2. Tuning the meta-model and preprocessing