Assignment 4: Pipelines and Hyperparameter Tuning

- **Full Name** = Sudarshan Naicker
- **UCID** = 30162797 ***

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  \
0     28    1   2     130.0  132.0  0.0      2.0    185.0    0.0
0.0
1     29    1   2     120.0  243.0  0.0      0.0    160.0    0.0
0.0
2     29    1   2     140.0    NaN  0.0      0.0    170.0    0.0
0.0
3     30    0   1     170.0  237.0  0.0      1.0    170.0    0.0
0.0
4     31    0   2     100.0  219.0  0.0      1.0    150.0    0.0
0.0
..   ...  ...  ..       ...    ...  ...      ...      ...    ...
...
289   52    1   4     160.0  331.0  0.0      0.0     94.0    1.0
2.5
290   54    0   3     130.0  294.0  0.0      1.0    100.0    1.0
0.0
291   56    1   4     155.0  342.0  1.0      0.0    150.0    1.0
3.0
292   58    0   2     180.0  393.0  0.0      0.0    110.0    1.0
1.0
293   65    1   4     130.0  275.0  0.0      1.0    115.0    1.0
1.0

     slope  ca  thal  num
0      NaN NaN   NaN    0
1      NaN NaN   NaN    0
2      NaN NaN   NaN    0
3      NaN NaN   6.0    0
4      NaN NaN   NaN    0
..     ...  ..   ...  ...
289    NaN NaN   NaN    1
290    2.0 NaN   NaN    1
291    2.0 NaN   NaN    1
```

```
292     2.0 NaN   7.0    1
293     2.0 NaN   NaN    1

[294 rows x 14 columns]
```

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** ....

The columns, slope, ca and thal have a large number of missing values, this means that these features wont be able to give much information to the model to predict the target variable. This could also lead to bias as some values are present and most are missing, hence it is better to remove the entire column.

```
# 1.1
# Add necessary code here.

sixty = df.isnull().mean() * 100 # to check the percentage of missing values in each column
print(sixty)
df_removed = df.drop(columns=['slope', 'ca', 'thal']) #  Droping these columns with > 60% missing values
df_removed



age             0.000000
sex             0.000000
```

```
cp           0.000000
trestbps     0.340136
chol         7.823129
fbs          2.721088
restecg      0.340136
thalach      0.340136
exang        0.340136
oldpeak      0.000000
slope       64.625850
ca          98.979592
thal        90.476190
num          0.000000
dtype: float64

     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
..   ...  ...  ..       ...    ...  ...      ...      ...    ...
...  ...
289   52    1   4     160.0  331.0  0.0      0.0     94.0    1.0
2.5    1
290   54    0   3     130.0  294.0  0.0      1.0    100.0    1.0
0.0    1
291   56    1   4     155.0  342.0  1.0      0.0    150.0    1.0
3.0    1
292   58    0   2     180.0  393.0  0.0      0.0    110.0    1.0
1.0    1
293   65    1   4     130.0  275.0  0.0      1.0    115.0    1.0
1.0    1

[294 rows x 11 columns]
```

Answer:

- **1.2** ....

I have chosen the mean strategy since all the values are numerical and maintains the variance of the dataset. It works by adding all the non-missing values in a column then divide the sum by the total number of non-missing values for that columns and replaces all the missing values in the column with the mean value, this is repeated for every columns with missing values.

```python
# 1.2
# Add necessary code here.

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')

df_changed = pd.DataFrame(imputer.fit_transform(df_removed),
columns=df_removed.columns)


print(f'{df_removed.isnull().sum()}\n') # Check the missing values
before simple imputer
print(df_changed.isnull().sum()) # Check the missing values after
simple imputer

df_changed
```

```
age          0
sex          0
cp           0
trestbps     1
chol        23
fbs          8
restecg      1
thalach      1
exang        1
oldpeak      0
num          0
dtype: int64

age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
num         0
dtype: int64

      age  sex   cp  trestbps        chol  fbs  restecg  thalach
exang  \
0    28.0  1.0  2.0     130.0  132.000000  0.0      2.0    185.0
0.0
1    29.0  1.0  2.0     120.0  243.000000  0.0      0.0    160.0
0.0
```

```
2    29.0  1.0  2.0     140.0  250.848708  0.0       0.0    170.0
0.0
3    30.0  0.0  1.0     170.0  237.000000  0.0       1.0    170.0
0.0
4    31.0  0.0  2.0     100.0  219.000000  0.0       1.0    150.0
0.0
..    ...  ...  ...       ...         ...  ...       ...      ...    ..
.
289  52.0  1.0  4.0     160.0  331.000000  0.0       0.0     94.0
1.0
290  54.0  0.0  3.0     130.0  294.000000  0.0       1.0    100.0
1.0
291  56.0  1.0  4.0     155.0  342.000000  1.0       0.0    150.0
1.0
292  58.0  0.0  2.0     180.0  393.000000  0.0       0.0    110.0
1.0
293  65.0  1.0  4.0     130.0  275.000000  0.0       1.0    115.0
1.0

     oldpeak  num
0        0.0  0.0
1        0.0  0.0
2        0.0  0.0
3        0.0  0.0
4        0.0  0.0
..       ...  ...
289      2.5  1.0
290      0.0  1.0
291      3.0  1.0
292      1.0  1.0
293      1.0  1.0

[294 rows x 11 columns]
```

Answer:

- **1.3** ....

Categorical features are cp and restecg we are using oneHotEncoder as it is used to convert categorical variables into numerical values. Numerical features are age, trestbps, chol, thalach, oldpeak. we use standardScaler as it converts numerical data to have a mean of zero and a standard deviation of one this will help ml models to better understand the input variables as they are scaled to a standard range. Binary features are sex, fbs, exangnot. They are not processed as they are simple with only 2 different values, 1 or 0.

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

```
y = df_changed['num']
X = df_changed.drop(columns=['num'])

categorical_features = ['cp', 'restecg']
numerical_features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
binary_features = ['sex', 'fbs', 'exang']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'),
categorical_features)],
        remainder='passthrough'
    )

print("Numerical features:", numerical_features)
print("Categorical (non-binary) features:", categorical_features)
print("Binary features:", binary_features)
```

```
Numerical features: ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
Categorical (non-binary) features: ['cp', 'restecg']
Binary features: ['sex', 'fbs', 'exang']
```

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 the previous assignment. **(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** …………………

Logistic Regression

- **Use Case**: Mainly used to predict a binary output variable from one or more input features.
- **Strength**: Linear models are simple, interpretable, and fast to train.
- **Weakness**: May not perform well on complex or non-linear data.
- **Rationale**: Since we do have 3 binary features, I have included the LR model.

Random Forest

- **Use Case**: An ensemble learning method that operates by constructing a collection of decision trees and outputting the result by either averaging (Regression) or majority voting (Classification) of all trees.
- **Strength**:
  - Capable of handling both linear and non-linear data.
  - By averaging the results of each tree, we reduce the amount of overfitting.
- **Weakness**:
  - More complex, leading to longer training times.
  - Doesn't tend to perform well on very high dimensional, sparse data, such as text data.
- **Rationale**: Since the trees are averaged out and make it perform way better than an individual tree, hence I am choosing RF.

Support Vector Classifier

- **Use Case**: Support vectors are the data points that lie closest to the decision boundary. SVC is a machine learning model that can perform linear or non-linear classification, regression, and even detect outliers.
- **Strength**:
  - Effective in high-dimensional spaces.
  - Can use different kernel functions for a better fit.
- **Weakness**:
  - Complex and requires careful selection of the kernel and regularization parameters.
- **Rationale**: Since we can select various parameters for a better fit, I am using the SVC model.

```python
# 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

lr_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())
])

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

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

Answer:

- **2.2** ....

```python
# 2.2
# Add necessary code here.

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import cross_val_score

lr_param_grid = {
    'classifier__C': [0.1, 1, 10],
    'classifier__penalty': ['l1'],
    'classifier__solver': ['liblinear', 'saga'],
    'classifier__max_iter': [1000, 5000]
}

rf_param_grid = {
    'classifier__n_estimators': [100, 200, 300],
    'classifier__max_depth': [5, 10, 15]
}

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

lr_grid_search = GridSearchCV(lr_pipeline, lr_param_grid, cv=5,
scoring='accuracy')
rf_grid_search = GridSearchCV(rf_pipeline, rf_param_grid, cv=5,
scoring='accuracy')
svc_grid_search = GridSearchCV(svc_pipeline, svc_param_grid, cv=5,
scoring='accuracy')

lr_grid_search.fit(X, y)
rf_grid_search.fit(X, y)
svc_grid_search.fit(X, y)

#f1 Score
f1_scores_lr = cross_val_score(lr_pipeline, X, y, cv=5,
scoring='f1').mean()
f1_scores_rf = cross_val_score(rf_pipeline, X, y, cv=5,
scoring='f1').mean()
f1_scores_svc = cross_val_score(svc_pipeline, X, y, cv=5,
scoring='f1').mean()


# Reporting F1 scores and accuracy scores
print("Best parameters for LR:", lr_grid_search.best_params_)
print("Best score for LR:", lr_grid_search.best_score_)
print(f"F1 Score for LR: {f1_scores_lr:.4f}")

print("\nBest parameters for RF:", rf_grid_search.best_params_)
print("Best score for RF:", rf_grid_search.best_score_)
print(f"F1 Score for RF: {f1_scores_rf:.4f}")

print("\nBest parameters for SVC:", svc_grid_search.best_params_)
print("Best score for SVC:", svc_grid_search.best_score_)
print(f"F1 Score for SVC: {f1_scores_svc:.4f}")
```

```
Best parameters for LR: {'classifier__C': 1, 'classifier__max_iter':
1000, 'classifier__penalty': 'l1', 'classifier__solver': 'saga'}
Best score for LR: 0.8196376388077148
F1 Score for LR: 0.7306

Best parameters for RF: {'classifier__max_depth': 5,
'classifier__n_estimators': 100}
Best score for RF: 0.7784921098772648
F1 Score for RF: 0.6780

Best parameters for SVC: {'classifier__C': 0.1, 'classifier__kernel':
'linear'}
Best score for SVC: 0.819812974868498
F1 Score for SVC: 0.6972
```

Answer:

- **2.3** ....

The meta-model's role is to combine the predictions of all estimators to make a final prediction. Logistic Regression is simple, interpretable, and efficient in combining predictions from various models without adding computational overhead.

```
# 2.3
# Add necessary code here.

from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.metrics import accuracy_score, f1_score
import numpy as np

stacking_clf = StackingClassifier(
    estimators=[
        ('lr', lr_pipeline),
        ('rf', rf_pipeline),
        ('svc', svc_pipeline)
    ],
    final_estimator= LogisticRegression()
)

cv = StratifiedKFold(n_splits=5)
accuracy_scores = cross_val_score(stacking_clf, X, y, cv=cv,
scoring='accuracy')
f1_scores = cross_val_score(stacking_clf, X, y, cv=cv, scoring='f1')

# Calculate mean and standard deviation for both scores
accuracy_mean, accuracy_std = np.mean(accuracy_scores),
np.std(accuracy_scores)
f1_mean, f1_std = np.mean(f1_scores), np.std(f1_scores)

#results
print(f"Accuracy Scores for individual folds: {accuracy_scores}")
print(f"Accuracy: {accuracy_mean:.2f} ± {accuracy_std:.2f}")
print(f"F1 Score: {f1_mean:.2f} ± {f1_std:.2f}")
```

```
Accuracy Scores for individual folds[0.81355932 0.76271186 0.83050847
0.83050847 0.75862069]
Accuracy: 0.80 ± 0.03
F1 Score: 0.70 ± 0.10
```

Answer:

- **2.4** ....

The stacking classifier shows a mean accuracy of 0.80 which is better compared to the accuracy scores of RF is 0.77 but the BEST score for LR and SVC is 0.81 which is greater than 0.8. However that is the best score of all 5 folds when you compare the each of the folds for stacking_clf we can see that 0.8474 was the best score this means that the overall mean of the LR and SVC would be lower. hence the stacking classifier has performed better in terms of accuracy scores than each of the individual models.

For the F1 scores, the stacking classifier's mean F1 score of 0.70 which is better compared to the mean f1 scores of RF and SVC which is 0.68 and 0.69 respectively. However LR's f1 score is 0.7306 but is notably higher. The improvement in the F1 score for the stacking classifier suggests it may be more effective at balancing false positives and false negatives than the RF and SVC. LR model perform better as it is a more simple model, though the stacking classifier uses the LR as meta-model it is trained on the predictions made by the base estimators, the lr_pipeline uses raw feature making it less complicated to make prediction.

Overall The Stacking Classifier is a better model as the Accuracy scores are higher than their individual models and the F1 score is also mostly better this is beacuse The stacking model benefits from the diversity of the base estimators, each bringing its strengths and reducing the impact of any one model's weaknesses.

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: