```
# importing libraries

# for data analysis
import pandas as pd
import numpy as np

# for data visualization
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# for data preprocessing
from sklearn.preprocessing import
LabelEncoder,StandardScaler,MinMaxScaler,OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression

from sklearn.naive_bayes import GaussianNB

# for imputing missing values
from sklearn.impute import SimpleImputer,KNNImputer

# import iterative imputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# for machine learning
from sklearn.model_selection import
train_test_split,GridSearchCV,cross_val_score,RandomizedSearchCV

# Given that we anticipate solving our dependent feature through
classification, we will proceed to import libraries tailored for
classification tasks.
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import
RandomForestClassifier,RandomForestRegressor ,AdaBoostClassifier ,Grad
ientBoostingClassifier
from xgboost import XGBClassifier

# for classification evaluation metrices
from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix,mean_absolute_er
ror,precision_score,mean_squared_error,r2_score

# to perform statistical test
from sklearn.feature_selection import chi2 # for categorical fetures
```

```
from sklearn.feature_selection import f_classif # for numerical
features (Anova f-test)

import pickle

# for ignoring warnings
import warnings
warnings.filterwarnings('ignore')
```

```
Requirement already satisfied: xgboost in c:\users\satyarao maddala\
anaconda3\lib\site-packages (3.0.2)
Requirement already satisfied: numpy in c:\users\satyarao maddala\
anaconda3\lib\site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\satyarao maddala\
anaconda3\lib\site-packages (from xgboost) (1.13.1)
```

```
df = pd.read_csv('heart_disease_uci.csv')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        920 non-null    int64
 1   age       920 non-null    int64
 2   sex       920 non-null    object
 3   dataset   920 non-null    object
 4   cp        920 non-null    object
 5   trestbps  861 non-null    float64
 6   chol      890 non-null    float64
 7   fbs       830 non-null    object
 8   restecg   918 non-null    object
 9   thalch    865 non-null    float64
 10  exang     865 non-null    object
 11  oldpeak   858 non-null    float64
 12  slope     611 non-null    object
 13  ca        309 non-null    float64
 14  thal      434 non-null    object
 15  num       920 non-null    int64
dtypes: float64(5), int64(3), object(8)
memory usage: 115.1+ KB
```

```
missing_data_cols = df.isnull().sum()[df.isnull().sum() >
0].index.to_list()
missing_data_cols
```

```
['trestbps',
 'chol',
 'fbs',
 'restecg',
 'thalch',
 'exang',
 'oldpeak',
 'slope',
 'ca',
 'thal']

categorical_cols =
["thal","ca","exang","slope","restecg","fbs","cp","sex","num"]
bool_cols = ["fbs","exang"]
numeric_cols = ["oldpeak","thalch","chol","trestbps","age"]
df = pd.read_csv('heart_disease_uci.csv')
missing_data_cols = df.columns[df.isnull().any()].tolist()
bool_cols = [col for col in df.columns if df[col].dropna().isin([True,
False]).all()]

def impute_categorical_missing_data(passed_col):
    df_null = df[df[passed_col].isnull()]
    df_not_null = df[df[passed_col].notnull()]

    X = df_not_null.drop(passed_col, axis=1)
    y = df_not_null[passed_col]

    other_missing_cols = [col for col in missing_data_cols if col !=
passed_col]
    label_encoder = LabelEncoder()

    for col in X.columns:
        if X[col].dtype == 'object' or X[col].dtype.name ==
'category':
            X[col] = label_encoder.fit_transform(X[col].astype(str))

    if passed_col in bool_cols:
        y = label_encoder.fit_transform(y.astype(str))

    iterative_imputer =
IterativeImputer(estimator=RandomForestRegressor(random_state=42),
add_indicator=True)

    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            col_with_missing_values = X[col].values.reshape(-1, 1)
            imputed_values =
iterative_imputer.fit_transform(col_with_missing_values)
            X[col] = imputed_values[:, 0]
```

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

    rf_classifier = RandomForestClassifier()
    rf_classifier.fit(X_train, y_train)
    y_pred = rf_classifier.predict(X_test)

    acc_score = accuracy_score(y_test, y_pred)
    print(f"The feature '{passed_col}' has been imputed with
{round(acc_score * 100, 2)}% accuracy\n")

    X = df_null.drop(passed_col, axis=1)

    for col in X.columns:
        if X[col].dtype == 'object' or X[col].dtype.name ==
'category':
            X[col] = label_encoder.fit_transform(X[col].astype(str))

    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            col_with_missing_values = X[col].values.reshape(-1, 1)
            imputed_values =
iterative_imputer.fit_transform(col_with_missing_values)
            X[col] = imputed_values[:, 0]

    if len(df_null) > 0:
        df_null[passed_col] = rf_classifier.predict(X)
        if passed_col in bool_cols:
            df_null[passed_col] = df_null[passed_col].map({0: False,
1: True})

    df_combined = pd.concat([df_not_null, df_null])
    return df_combined[passed_col]









def impute_continuous_missing_data(passed_col):
    df_null = df[df[passed_col].isnull()]
    df_not_null = df[df[passed_col].notnull()]

    X = df_not_null.drop(passed_col, axis=1)
    y = df_not_null[passed_col]

    other_missing_cols = [col for col in missing_data_cols if col !=
```

```
passed_col]
    label_encoder = LabelEncoder()

    for col in X.columns:
        if X[col].dtype == 'object' or X[col].dtype.name ==
'category':
            X[col] = label_encoder.fit_transform(X[col].astype(str))

    iterative_imputer =
IterativeImputer(estimator=RandomForestRegressor(random_state=42),
add_indicator=True)

    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            col_with_missing_values = X[col].values.reshape(-1, 1)
            imputed_values =
iterative_imputer.fit_transform(col_with_missing_values)
            X[col] = imputed_values[:, 0]

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

    rf_regressor = RandomForestRegressor()
    rf_regressor.fit(X_train, y_train)
    y_pred = rf_regressor.predict(X_test)

    print("MAE =", mean_absolute_error(y_test, y_pred))
    print("RMSE =", mean_squared_error(y_test, y_pred, squared=False))
    print("R2 =", r2_score(y_test, y_pred))

    X = df_null.drop(passed_col, axis=1)

    for col in X.columns:
        if X[col].dtype == 'object' or X[col].dtype.name ==
'category':
            X[col] = label_encoder.fit_transform(X[col].astype(str))

    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            col_with_missing_values = X[col].values.reshape(-1, 1)
            imputed_values =
iterative_imputer.fit_transform(col_with_missing_values)
            X[col] = imputed_values[:, 0]

    if len(df_null) > 0:
        df_null[passed_col] = rf_regressor.predict(X)

    df_combined = pd.concat([df_not_null, df_null])
    return df_combined[passed_col]
```

```
# remove warning
import warnings
warnings.filterwarnings('ignore')

# impute missing values using our functions
for col in missing_data_cols:
    print("Missing Values", col, ":",
str(round((df[col].isnull().sum() / len(df)) * 100, 2))+"%")
    if col in categorical_cols:
        df[col] = impute_categorical_missing_data(col)
    elif col in numeric_cols:
        df[col] = impute_continuous_missing_data(col)
    else:
        pass
```

```
Missing Values trestbps : 0.0%
MAE = 13.898284782608698
RMSE = 17.701896806953254
R2 = -0.027081116202439137
Missing Values chol : 0.0%
MAE = 49.91728097826086
RMSE = 69.1797686961481
R2 = 0.5967391606442527
Missing Values fbs : 9.78%
The feature 'fbs' has been imputed with 80.12% accuracy

Missing Values restecg : 0.22%
The feature 'restecg' has been imputed with 65.76% accuracy

Missing Values thalch : 5.98%
MAE = 16.71965317919075
RMSE = 21.671829374543034
R2 = 0.3170632883224507
Missing Values exang : 5.98%
The feature 'exang' has been imputed with 81.5% accuracy

Missing Values oldpeak : 6.74%
MAE = 0.5719244186046513
RMSE = 0.8016231970196471
R2 = 0.38779270819546285
Missing Values slope : 33.59%
The feature 'slope' has been imputed with 67.48% accuracy

Missing Values ca : 66.41%
The feature 'ca' has been imputed with 64.52% accuracy

Missing Values thal : 52.83%
The feature 'thal' has been imputed with 70.11% accuracy
```

```
df.isnull().sum().sort_values(ascending=False)
```

```
id          0
age         0
sex         0
dataset     0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalch      0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
num         0
dtype: int64
```

```
import pandas as pd

# Assuming 'df' is your DataFrame
# Example: df = pd.read_csv('your_data.csv')

# List of categorical columns
categorical_cols = ['sex', 'dataset', 'cp', 'fbs', 'restecg', 'exang',
'slope', 'thal']

# Calculate unique values for each categorical column
unique_values = {col: df[col].unique() for col in categorical_cols}

# Print the unique values
for col, values in unique_values.items():
    print(f"Unique values in '{col}': {values}")
```

```
Unique values in 'sex': ['Male' 'Female']
Unique values in 'dataset': ['Cleveland' 'Hungary' 'Switzerland' 'VA
Long Beach']
Unique values in 'cp': ['typical angina' 'asymptomatic' 'non-anginal'
'atypical angina']
Unique values in 'fbs': [True False]
Unique values in 'restecg': ['lv hypertrophy' 'normal' 'st-t
abnormality']
Unique values in 'exang': [False True]
Unique values in 'slope': ['downsloping' 'flat' 'upsloping']
Unique values in 'thal': ['fixed defect' 'normal' 'reversable defect']
```

```
# print the row from df where trestbps value is 0
df[df['trestbps'] == 0]
```

```
# remove this row from data
df = df[df['trestbps'] != 0]

# List of categorical and numerical features
categorical_columns = ['sex', 'cp', 'restecg', 'exang', 'slope',
'thal']
numerical_columns = ['age', 'trestbps', 'chol', 'thalch', 'oldpeak']
target_column = 'num'

# Label encode categorical columns
label_encoders = {}
for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Final feature set
final_features = ['cp', 'exang', 'slope', 'thal', 'oldpeak', 'thalch',
'age', 'chol', 'trestbps']
X = df[final_features]
y = df[target_column]

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create a list of models to evaluate
models = [
    ('Random Forest', RandomForestClassifier(random_state=42)),
    ('Gradient Boosting',
GradientBoostingClassifier(random_state=42)),
    ('Support Vector Machine', SVC(random_state=42)),
    ('Logistic Regression', LogisticRegression(random_state=42)),
    ('K-Nearest Neighbors', KNeighborsClassifier()),
    ('Decision Tree', DecisionTreeClassifier(random_state=42)),
    ('Ada Boost', AdaBoostClassifier(random_state=42)),
    ('XG Boost', XGBClassifier(random_state=42)),
    ('Naive Bayes', GaussianNB())
]

# Hyperparameter tuning example for RandomForest
from sklearn.model_selection import GridSearchCV

# Define parameter grid for Random Forest
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 4]
}
```

```
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,
scoring='accuracy')
grid_search.fit(X_train, y_train)
best_rf = grid_search.best_estimator_

# Evaluate all models and select the best one
best_model = None
best_accuracy = 0.0

for name, model in models:
    # Create a pipeline for each model
    pipeline = Pipeline([
        ('scaler', StandardScaler()),  # Standardize features
        ('model', model)
    ])

    # Perform cross-validation
    scores = cross_val_score(pipeline, X_train, y_train, cv=5)

    # Calculate mean accuracy
    mean_accuracy = scores.mean()

    # Fit the pipeline on the training data
    pipeline.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = pipeline.predict(X_test)

    # Calculate accuracy score
    accuracy = accuracy_score(y_test, y_pred)

    # Print the performance metrics
    print("Model:", name)
    print("Cross-validation Accuracy:", mean_accuracy)
    print("Test Accuracy:", accuracy)
    print()

    # Check if the current model has the best accuracy
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model = pipeline
        print("Best Model:", best_model)

Model: Random Forest
Cross-validation Accuracy: 0.9416666666666667
Test Accuracy: 1.0

Best Model: Pipeline(steps=[('scaler', StandardScaler()),
                ('model', RandomForestClassifier(random_state=42))])
```

```
Model: Gradient Boosting
Cross-validation Accuracy: 0.9416666666666668
Test Accuracy: 1.0

Model: Support Vector Machine
Cross-validation Accuracy: 0.9583333333333334
Test Accuracy: 1.0

Model: Logistic Regression
Cross-validation Accuracy: 0.9583333333333334
Test Accuracy: 1.0

Model: K-Nearest Neighbors
Cross-validation Accuracy: 0.9333333333333333
Test Accuracy: 1.0

Model: Decision Tree
Cross-validation Accuracy: 0.9416666666666668
Test Accuracy: 1.0

Model: Ada Boost
Cross-validation Accuracy: 0.9333333333333333
Test Accuracy: 1.0

Model: XG Boost
Cross-validation Accuracy: 0.9333333333333333
Test Accuracy: 1.0

Model: Naive Bayes
Cross-validation Accuracy: 0.9416666666666668
Test Accuracy: 1.0


# Save the best model
with open('02_heart_disease_model.pkl', 'wb') as file:
    pickle.dump(best_model, file)

# Sample data for testing
sample_data = pd.DataFrame({
    'age': [55],
    'trestbps': [130],
    'chol': [245],
    'thalch': [205],
    'oldpeak': [1.0],
    'cp': [3],  # Assuming 1 corresponds to 'typical angina'
    'exang': [1],
    'slope': [3],  # Assuming 2 corresponds to 'downsloping'
    'thal': [4]  # Assuming 2 corresponds to 'fixed defect'
})

# Ensure sample data has the same columns as the training data
```

```
sample_data_encoded = sample_data[final_features]

# Make predictions
predicted = best_model.predict(sample_data_encoded)

# Interpretation based on predicted value
def interpret_prediction(prediction):
    if prediction[0] == 0:
        return "No Heart Disease"
    elif prediction[0] == 1:
        return "Stage 1 (Mild Heart Disease)"
    elif prediction[0] == 2:
        return "Stage 2 (Moderate Heart Disease)"
    elif prediction[0] == 3:
        return "Stage 3 (Advanced Heart Disease)"
    elif prediction[0] == 4:
        return "Stage 4 (Severe Heart Disease)"
    else:
        return "Unknown Stage"

# Print the result
print("Prediction:", interpret_prediction(predicted))
```

```
Prediction: No Heart Disease
```

```
# Only include features used in training
# Replace these 4 with your actual final selected features used during
training
final_features = ['age', 'trestbps', 'chol', 'thalch']  # Example only
— update this to match training features

# Sample input (adjust values according to your actual features)
sample_data = pd.DataFrame([{
    'age': 67,
    'trestbps': 160,
    'chol': 286,
    'thalch': 108
}])

# Select only the final features
sample_data_encoded = sample_data[final_features]

# Predict using the trained pipeline
predicted = best_model.predict(sample_data_encoded)

# Interpretation logic
def interpret_prediction(prediction):
    if prediction[0] == 0:
        return "No Heart Disease"
    elif prediction[0] == 1:
```

```
        return "Stage 1 (Mild Heart Disease)"
    elif prediction[0] == 2:
        return "Stage 2 (Moderate Heart Disease)"
    elif prediction[0] == 3:
        return "Stage 3 (Advanced Heart Disease)"
    elif prediction[0] == 4:
        return "Stage 4 (Severe Heart Disease)"
    else:
        return "Unknown Stage"

# Show prediction
print("Prediction:", interpret_prediction(predicted))
```

```
Prediction: Stage 2 (Moderate Heart Disease)
```