# Imoprt libraries

In [169…

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**from** plotly **import** express **as** px

In [170…

*# Prepocessing the data*

**from** sklearn.preprocessing **import** StandardScaler,MinMaxScaler,LabelEncoder

**from** sklearn.impute **import** SimpleImputer,KNNImputer

In [171…

*# Machine Learning*

**from** sklearn.model\_selection **import** train\_test\_split, cross\_val\_score

# Load the dataset

In [172…

df **=** pd**.**read\_csv("heart.csv") df**.**head()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[172]: | **age** | **sex** | **cp** | **trestbps** | **chol** | **fbs** | **restecg** | **thalach** | **exang** | **oldpeak** | **slope** | **ca** | **thal** | **target** |
|  | **0** 52 | 1 | 0 | 125 | 212 | 0 | 1 | 168 | 0 | 1.0 | 2 | 2 | 3 | 0 |
|  | **1** 53 | 1 | 0 | 140 | 203 | 1 | 0 | 155 | 1 | 3.1 | 0 | 0 | 3 | 0 |
|  | **2** 70 | 1 | 0 | 145 | 174 | 0 | 1 | 125 | 1 | 2.6 | 0 | 0 | 3 | 0 |
|  | **3** 61 | 1 | 0 | 148 | 203 | 0 | 1 | 161 | 0 | 0.0 | 2 | 1 | 3 | 0 |
|  | **4** 62 | 0 | 0 | 138 | 294 | 1 | 1 | 106 | 0 | 1.9 | 1 | 3 | 2 | 0 |

# Exploratory Data Analysis (EDA)

In [173…

<class 'pandas.core.frame.DataFrame'> RangeIndex: 1025 entries, 0 to 1024 Data columns (total 14 columns):

df**.**info()

# Column Non-Null Count Dtype

1. age 1025 non-null int64
2. sex 1025 non-null int64
3. cp 1025 non-null int64
4. trestbps 1025 non-null int64
5. chol 1025 non-null int64
6. fbs 1025 non-null int64
7. restecg 1025 non-null int64
8. thalach 1025 non-null int64
9. exang 1025 non-null int64
10. oldpeak 1025 non-null float64
11. slope 1025 non-null int64
12. ca 1025 non-null int64
13. thal 1025 non-null int64
14. target 1025 non-null int64 dtypes: float64(1), int64(13)

memory usage: 112.2 KB

df**.**shape

In [174… Out[174]:

df**.**describe()**.**T

In [175…

(1025, 14)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[175]: |  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
|  | **age** | 1025.0 | 54.434146 | 9.072290 | 29.0 | 48.0 | 56.0 | 61.0 | 77.0 |
|  | **sex** | 1025.0 | 0.695610 | 0.460373 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 |
|  | **cp** | 1025.0 | 0.942439 | 1.029641 | 0.0 | 0.0 | 1.0 | 2.0 | 3.0 |
|  | **trestbps** | 1025.0 | 131.611707 | 17.516718 | 94.0 | 120.0 | 130.0 | 140.0 | 200.0 |
|  | **chol** | 1025.0 | 246.000000 | 51.592510 | 126.0 | 211.0 | 240.0 | 275.0 | 564.0 |
|  | **fbs** | 1025.0 | 0.149268 | 0.356527 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
|  | **restecg** | 1025.0 | 0.529756 | 0.527878 | 0.0 | 0.0 | 1.0 | 1.0 | 2.0 |
|  | **thalach** | 1025.0 | 149.114146 | 23.005724 | 71.0 | 132.0 | 152.0 | 166.0 | 202.0 |
|  | **exang** | 1025.0 | 0.336585 | 0.472772 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |
|  | **oldpeak** | 1025.0 | 1.071512 | 1.175053 | 0.0 | 0.0 | 0.8 | 1.8 | 6.2 |
|  | **slope** | 1025.0 | 1.385366 | 0.617755 | 0.0 | 1.0 | 1.0 | 2.0 | 2.0 |
|  | **ca** | 1025.0 | 0.754146 | 1.030798 | 0.0 | 0.0 | 0.0 | 1.0 | 4.0 |
|  | **thal** | 1025.0 | 2.323902 | 0.620660 | 0.0 | 2.0 | 2.0 | 3.0 | 3.0 |
|  | **target** | 1025.0 | 0.513171 | 0.500070 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 |

In [176… Out[176]:

In [177… Out[177]:

In [178… Out[178]:

(29, 77)

df['age']**.**min(), df['age']**.**max()

df**.**isnull()**.**sum()

age 0

sex 0

cp 0

trestbps 0

chol 0

fbs 0

restecg 0

thalach 0

exang 0

oldpeak 0

slope 0

ca 0

thal 0

target 0

dtype: int64

df**.**duplicated()**.**sum()

723

In [179…

duplicated\_rows **=** df[df**.**duplicated(keep**=False**)]

print("Duplicated Rows (all instances):") print(duplicated\_rows)

Duplicated Rows (all instances):

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | age | sex | cp | trestbps | | chol | fbs | restecg | thalach | exang | oldpeak | \ |
| 0 | 52 | 1 | 0 | 125 | | 212 | 0 | 1 | 168 | 0 | 1.0 |  |
| 1 | 53 | 1 | 0 | 140 | | 203 | 1 | 0 | 155 | 1 | 3.1 |  |
| 2 | 70 | 1 | 0 | 145 | | 174 | 0 | 1 | 125 | 1 | 2.6 |  |
| 3 | 61 | 1 | 0 | 148 | | 203 | 0 | 1 | 161 | 0 | 0.0 |  |
| 4 | 62 | 0 | 0 | 138 | | 294 | 1 | 1 | 106 | 0 | 1.9 |  |
| ... | ... | ... | .. | ... | | ... | ... | ... | ... | ... | ... |  |
| 1020 | 59 | 1 | 1 | 140 | | 221 | 0 | 1 | 164 | 1 | 0.0 |  |
| 1021 | 60 | 1 | 0 | 125 | | 258 | 0 | 0 | 141 | 1 | 2.8 |  |
| 1022 | 47 | 1 | 0 | 110 | | 275 | 0 | 0 | 118 | 1 | 1.0 |  |
| 1023 | 50 | 0 | 0 | 110 | | 254 | 0 | 0 | 159 | 0 | 0.0 |  |
| 1024 | 54 | 1 | 0 | 120 | | 188 | 0 | 1 | 113 | 0 | 1.4 |  |
|  | slope | ca | thal | | target | | | | | | | |
| 0 | 2 | 2 | 3 | | 0 | | | | | | | |
| 1 | 0 | 0 | 3 | | 0 | | | | | | | |
| 2 | 0 | 0 | 3 | | 0 | | | | | | | |
| 3 | 2 | 1 | 3 | | 0 | | | | | | | |
| 4 | 1 | 3 | 2 | | 0 | | | | | | | |
| ... | ... | .. | ... | | ... | | | | | | | |
| 1020 | 2 | 0 | 2 | | 1 | | | | | | | |
| 1021 | 1 | 1 | 3 | | 0 | | | | | | | |
| 1022 | 1 | 1 | 2 | | 0 | | | | | | | |
| 1023 | 2 | 0 | 2 | | 1 | | | | | | | |
| 1024 | 1 | 1 | 3 | | 0 | | | | | | | |

[1025 rows x 14 columns]

In [180…

*# Calculate Q1 (25th percentile) and Q3 (75th percentile)*

Q1 **=** df['age']**.**quantile(0.25) Q3 **=** df['age']**.**quantile(0.75)

*# Calculate IQR (Interquartile Range)*

IQR **=** Q3 **-** Q1

*# Define lower and upper bounds for outliers*

lower\_bound **=** Q1 **-** 1.5 **\*** IQR upper\_bound **=** Q3 **+** 1.5 **\*** IQR

*# Identify outliers*

outliers **=** df[(df['age'] **<** lower\_bound) **|** (df['age'] **>** upper\_bound)]

print("Outliers detected:") print(outliers)

Outliers detected:

Empty DataFrame

Columns: [age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, c a, thal, target]

Index: []

It's a good sign ,there is no null values and outliers.

# Visualization

In [181…

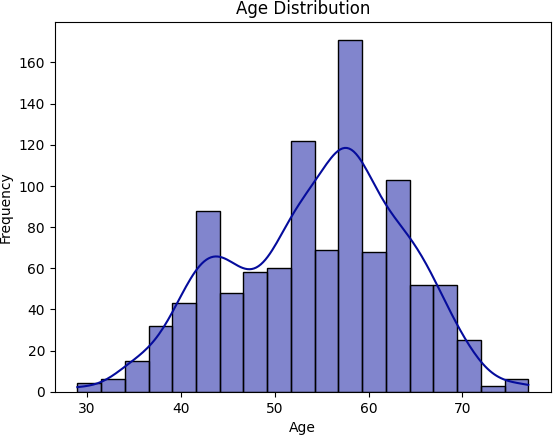
*# Define custom colors*

custom\_colors **=** ["#050C9C", "#3572EF", "#3ABEF9"]

*# Plot the histogram with custom colors*

sns**.**histplot(df['age'], kde**=True**, color**=**"#050C9C", palette**=**custom\_colors) plt**.**title("Age Distribution")

plt**.**xlabel("Age") plt**.**ylabel("Frequency") plt**.**show()



The age column distribution seems to be normaly distributed.

In [182…

*# Plot the mean, Median and mode of age column* sns**.**histplot(df['age'], kde**=True**) plt**.**axvline(df['age']**.**mean(), color**=**'Red') plt**.**axvline(df['age']**.**median(), color**=** 'Green') plt**.**axvline(df['age']**.**mode()[0], color**=**'Blue') plt**.**title("Central Tendency of Age") plt**.**xlabel("Age")

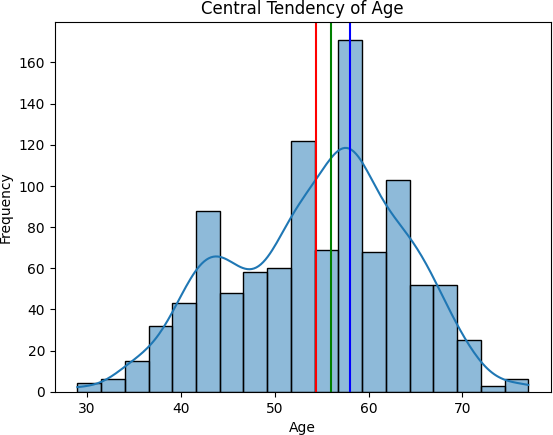
plt**.**ylabel("Frequency") plt**.**show()

*# print the value of mean, median and mode of age column*

print('Mean', df['age']**.**mean())

print('Median', df['age']**.**median())

print('Mode', df['age']**.**mode())



Mean 54.43414634146342

Median 56.0

Mode 0 58

Name: age, dtype: int64

In [183…

*# plot the histogram of age column*

df['sex'] **=** df['sex']**.**map({1: 'male', 0: 'female'}) fig **=** px**.**histogram(

df, x**=**'age', color**=**'sex',

title**=**'Age Distribution by Sex', width**=**600,

height**=**500

)

fig**.**show()

In [184…

*# Find the values of sex column*

df['sex']**.**value\_counts()

Out[184]:

In [185…

sex

male 713

female 312

Name: count, dtype: int64

*# Find the values count of age column grouping by sex column*

df**.**groupby('sex')['age']**.**value\_counts()

Out[185]:

In [186…

sex age

|  |  |  |
| --- | --- | --- |
| female | 62 | 24 |
|  | 58 | 21 |
|  | 63 | 17 |
|  | 54 | 15 |
|  | 55 | 15  .. |
| male | 69 | 6 |
|  | 29 | 4 |
|  | 34 | 3 |
|  | 37 | 3 |
|  | 77 | 3 |

Name: count, Length: 73, dtype: int64

*# value count of cp column*

df['cp']**.**value\_counts()

Out[186]:

In [187…

cp

0 497

2 284

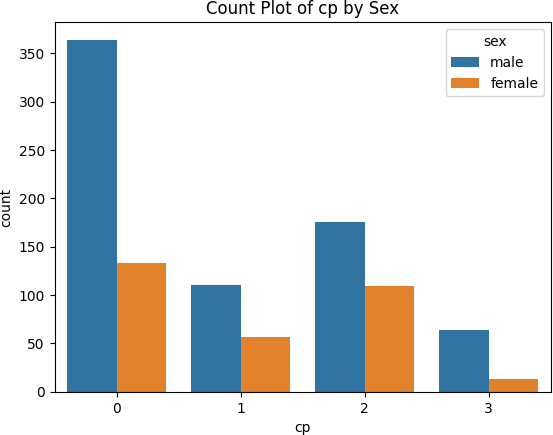
1 167

3 77

Name: count, dtype: int64

*# Creating the count plot of cp column*

sns**.**countplot(df, x**=**'cp', hue**=**'sex')**.**set\_title('Count Plot of cp by Sex') plt**.**show()



In [188…

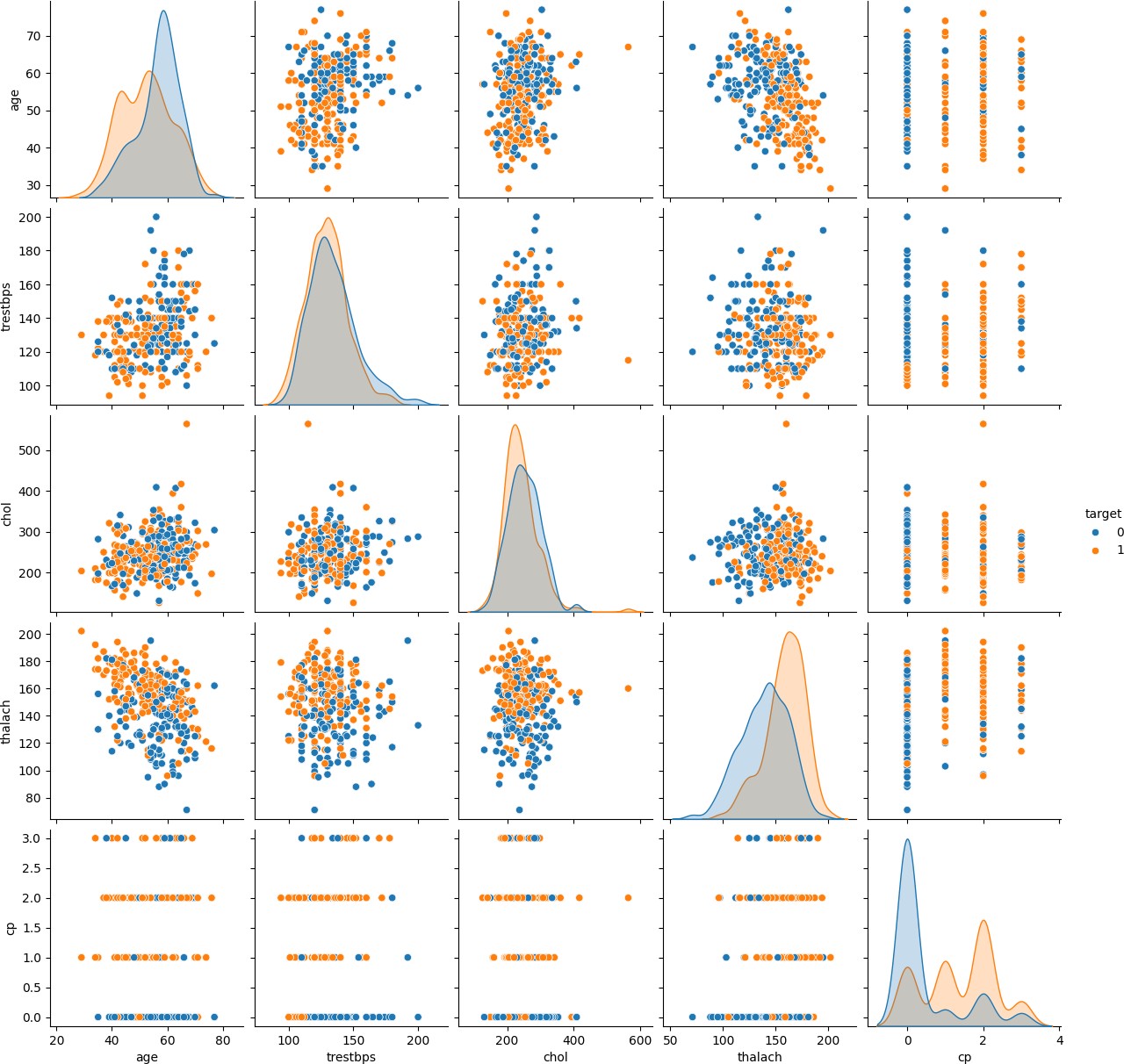
*# Draw the plot of age column group by cp column*

fig **=** px**.**histogram(data\_frame**=**df, x**=**'age', color**=**'cp',title**=** 'Age distribution by CP ', width**=**600 , height**=**500)

fig**.**show()

In [189…

sns**.**pairplot(df[['age','trestbps','chol','thalach','target','cp']], hue **=** 'target') plt**.**show()



# Machine learning models

In [190… Out[190]:

In [191… Out[191]:

In [192…

Out[192]:

Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',

df**.**columns

'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'], dtype='object')

df['target']**.**unique()

array([0, 1])

df['target']**.**value\_counts()

target

1 526

0 499

Name: count, dtype: int64

The Target column is my predicted attribute. I will use this column to predict the heart disease. The unique values in this column are: [0,1] which states that the presence of heart diseases.

0 = no heart disease. 1 = Heart Disease.

In [193…

*# split the data into X and y* X**=** df**.**drop('target', axis**=**1) y **=** df['target'] Label\_Encoder **=** LabelEncoder()

**for** col **in** X**.**columns:

**if** X[col]**.**dtype **==** 'object' **or** X[col]**.**dtype **==** 'category': X[col] **=** Label\_Encoder**.**fit\_transform(X[col])

**else**:

**pass**

*# split the data into train and test*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2, random\_state**=**10

|  |  |  |
| --- | --- | --- |
|  | | Enlist all the models that I will use to predict the heart disease. These models should be classifiers for multi\_class classification- |
| a.logistic regression. |
| b.KNN  c.SVM |
| d.Decision Tree  e.Random Forest |
| In | [194… | X\_test**.**head() |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[194]: | **age** | **sex** | **cp** | **trestbps** | **chol** | **fbs** | **restecg** | **thalach** | **exang** | **oldpeak** | **slope** | **ca** | **thal** |
|  | **909** 50 | 1 | 0 | 144 | 200 | 0 | 0 | 126 | 1 | 0.9 | 1 | 0 | 3 |
|  | **748** 44 | 1 | 2 | 120 | 226 | 0 | 1 | 169 | 0 | 0.0 | 2 | 0 | 2 |
|  | **919** 38 | 1 | 3 | 120 | 231 | 0 | 1 | 182 | 1 | 3.8 | 1 | 0 | 3 |
|  | **975** 39 | 1 | 0 | 118 | 219 | 0 | 1 | 140 | 0 | 1.2 | 1 | 0 | 3 |
|  | **246** 54 | 1 | 1 | 192 | 283 | 0 | 0 | 195 | 0 | 0.0 | 2 | 1 | 3 |

In [195…

*# improt All models.*

**from** sklearn.linear\_model **import** LogisticRegression **from** sklearn.neighbors **import** KNeighborsClassifier **from** sklearn.svm **import** SVC

**from** sklearn.tree **import** DecisionTreeClassifier, plot\_tree

**from** sklearn.ensemble **import** RandomForestClassifier

*#importing pipeline*

**from** sklearn.pipeline **import** Pipeline

*# import metrics*

**from** sklearn.metrics **import** accuracy\_score, confusion\_matrix, classification\_report, mea

**from** sklearn.model\_selection **import** train\_test\_split,GridSearchCV, cross\_val\_score

In [196…

*# Define the models to evaluate*

models **=** [

('Logistic Regression', LogisticRegression(random\_state**=**101)), ('KNeighbors Classifier', KNeighborsClassifier()),

('Decision Tree Classifier', DecisionTreeClassifier(random\_state**=**101)),

('Random Forest', RandomForestClassifier(random\_state**=**101)), ('Support Vector Machine', SVC(random\_state**=**101))

]

best\_model **= None**

best\_accuracy **=** 0.0

*# Iterate over the models and evaluate their performance*

**for** name, model **in** models:

*# Create a pipeline for each model*

pipeline **=** Pipeline([

('imputer', SimpleImputer(strategy**=**'most\_frequent')), ('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

*# Retrieve the best model*

print("Best Model:", best\_model)

Model: Logistic Regression

Cross Validation Accuracy: 0.8426829268292682 Test Accuracy: 0.8634146341463415

Model: KNeighbors Classifier

Cross Validation Accuracy: 0.7134146341463415 Test Accuracy: 0.7317073170731707

Model: Decision Tree Classifier

Cross Validation Accuracy: 0.9902439024390244 Test Accuracy: 0.9853658536585366

Model: Random Forest

Cross Validation Accuracy: 0.9902439024390244 Test Accuracy: 1.0

Model: Support Vector Machine

Cross Validation Accuracy: 0.6878048780487804 Test Accuracy: 0.6829268292682927

Best Model: Pipeline(steps=[('imputer', SimpleImputer(strategy='most\_frequent')), ('model', RandomForestClassifier(random\_state=101))])

Evaluation of the model

In [197…

df**.**head()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[197]: | **age** | **sex** | **cp** | **trestbps** | **chol** | **fbs** | **restecg** | **thalach** | **exang** | **oldpeak** | **slope** | **ca** | **thal** | **target** |
| **0** | 52 | male | 0 | 125 | 212 | 0 | 1 | 168 | 0 | 1.0 | 2 | 2 | 3 | 0 |
| **1** | 53 | male | 0 | 140 | 203 | 1 | 0 | 155 | 1 | 3.1 | 0 | 0 | 3 | 0 |
| **2** | 70 | male | 0 | 145 | 174 | 0 | 1 | 125 | 1 | 2.6 | 0 | 0 | 3 | 0 |
| **3** | 61 | male | 0 | 148 | 203 | 0 | 1 | 161 | 0 | 0.0 | 2 | 1 | 3 | 0 |
| **4** | 62 | female | 0 | 138 | 294 | 1 | 1 | 106 | 0 | 1.9 | 1 | 3 | 2 | 0 |

In [198…

categorical\_cols **=** ['cp', 'slope','thal', 'exang', 'restecg','fbs', 'ca', 'sex']

In [199…

**def** evaluate\_classification\_models(X, y, categorical\_columns):

*# Encode categorical columns* X\_encoded **=** X**.**copy() label\_encoders **=** {}

**for** col **in** categorical\_columns: label\_encoders[col] **=** LabelEncoder()

X\_encoded[col] **=** label\_encoders[col]**.**fit\_transform(X[col])

*# Split data into train and test sets*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X\_encoded, y, test\_size**=**0.3, ran

*# Define models*

models **=** {

"Logistic Regression": LogisticRegression(), "KNN": KNeighborsClassifier(),

"SVM": SVC(),

"Decision Tree": DecisionTreeClassifier(), "Random Forest": RandomForestClassifier()

}

*# Train and evaluate models*

results **=** {} best\_model **= None** best\_accuracy **=** 0.0

**for** name, model **in** models**.**items(): model**.**fit(X\_train, y\_train) y\_pred **=** model**.**predict(X\_test)

accuracy **=** accuracy\_score(y\_test, y\_pred) results[name] **=** accuracy

**if** accuracy **>** best\_accuracy:

best\_accuracy **=** accuracy best\_model **=** name

**return** results, best\_model

*# Example usage:*

results, best\_model **=** evaluate\_classification\_models(X, y, categorical\_cols) print("Model accuracies:", results)

print("Best model:", best\_model)

Model accuracies: {'Logistic Regression': 0.8474025974025974, 'KNN': 0.7207792207792207, 'SVM': 0.6883116883116883, 'Decision Tree': 0.9577922077922078, 'Random Forest': 1.0}

Best model: Random Forest

# Conclusion

After thorough data exploration, we trained several machine learning models including Logistic Regression, Decision Tree, Random Forest,SVM and KNN. The models achieved the following accuracies:

Logistic Regression :0.83 Decision Tree : 0.95 Random Forest : 0.99 KNN : 0.72

SVM: 0.69

Among these models, Random Forest performed the best with an accuracy of 99% making it the top choice for predicting heart diseases based on the given features.