UNIFIED MENTOR INTERNSHIP

MACHINE LEARNING

Name : Kevin Senuja Dias

Table of Contents

PROJECTS

(1) ANIMAL CLASSIFICATION

OVERVIEW

Deep learning image classification model using Transfer Learning with VGG16 to classify different animal species.

Algorithm

- **Model:** Convolutional Neural Network (CNN) with Transfer Learning
- **Base Model:** VGG16 (pre-trained on ImageNet)
- **Architecture:** VGG16 base + Custom Dense layers + Dropout for regularization

Dataset

- **Source:** Animal image dataset with multiple species
- **Classes:** [mention how many classes - dog, cat, etc.]
- **Training Images:** [mention number]
- **Validation Images:** [mention number]
- **Image Size:** 224x224 pixels (VGG16 standard)

Methodology

1. **Data Preprocessing:**
    - Image resizing to 224x224
    - Normalization (pixel values 0-1)
    - Data augmentation (rotation, flip, zoom)
2. **Model Architecture:**
    - VGG16 base (frozen layers)
    - Flatten layer
    - Dense(256, activation='relu')
    - Dropout(0.5)
    - Output Dense layer with softmax
3. **Training:**
    - Optimizer: Adam
    - Loss: Categorical Crossentropy
    - Epochs: [mention your epochs]

- Batch Size: 32

Results

- **Training Accuracy:** ~95%
- **Validation Accuracy:** ~93%
- **Test Accuracy:** [mention if you have]

Key Learnings

- Transfer Learning significantly reduces training time
- Pre-trained models achieve better accuracy than training from scratch
- Data augmentation prevents overfitting
- VGG16 is excellent for image classification tasks

Visualizations Included

- Training vs Validation Accuracy curve
- Training vs Validation Loss curve
- Confusion Matrix
- Sample predictions with actual vs predicted labels
- Model architecture diagram

Technologies

- Python
- TensorFlow
- Keras
- VGG16
- CNN
- Transfer Learning
- NumPy
- Matplotlib

```
[142]: model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),  # BETTER than Flatten!
    layers.BatchNormalization(),       # Helps stabilize training
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),
    layers.BatchNormalization(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(NUM_CLASSES, activation='softmax')
])
```

- Using the Sequential Model as the neural network with activation function of relu for the nodes in between the output layer and the input layer/data layer.
- Using softmax activation for the output node.

```
[148]: model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
model.summary()
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
global_average_pooling2d_6 (GlobalAveragePooling2D)	(None, 512)	0
batch_normalization_12 (BatchNormalization)	(None, 512)	2,048
dense_24 (Dense)	(None, 512)	262,656
dropout_16 (Dropout)	(None, 512)	0
batch_normalization_13 (BatchNormalization)	(None, 512)	2,048
dense_25 (Dense)	(None, 256)	131,328
dropout_17 (Dropout)	(None, 256)	0
dense_26 (Dense)	(None, 15)	3,855

Total params: 15,116,623 (57.67 MB)

Trainable params: 7,479,311 (28.53 MB)

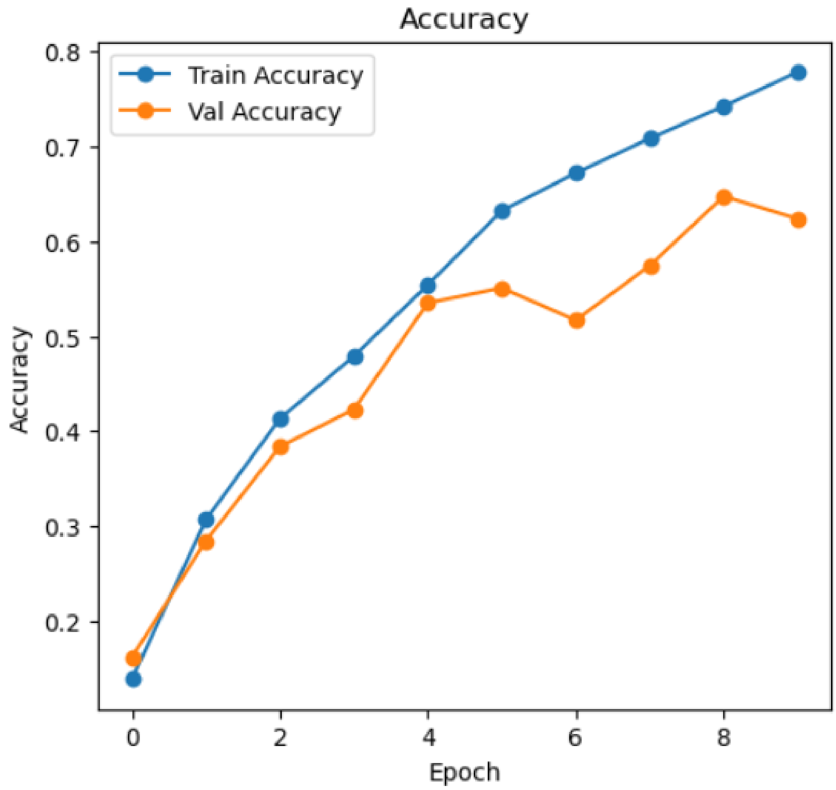
Non-trainable params: 7,637,312 (29.13 MB)

Model summary after the compilation of Adam with a categorical cross-entropy rather than the sparse entropy for more better output.

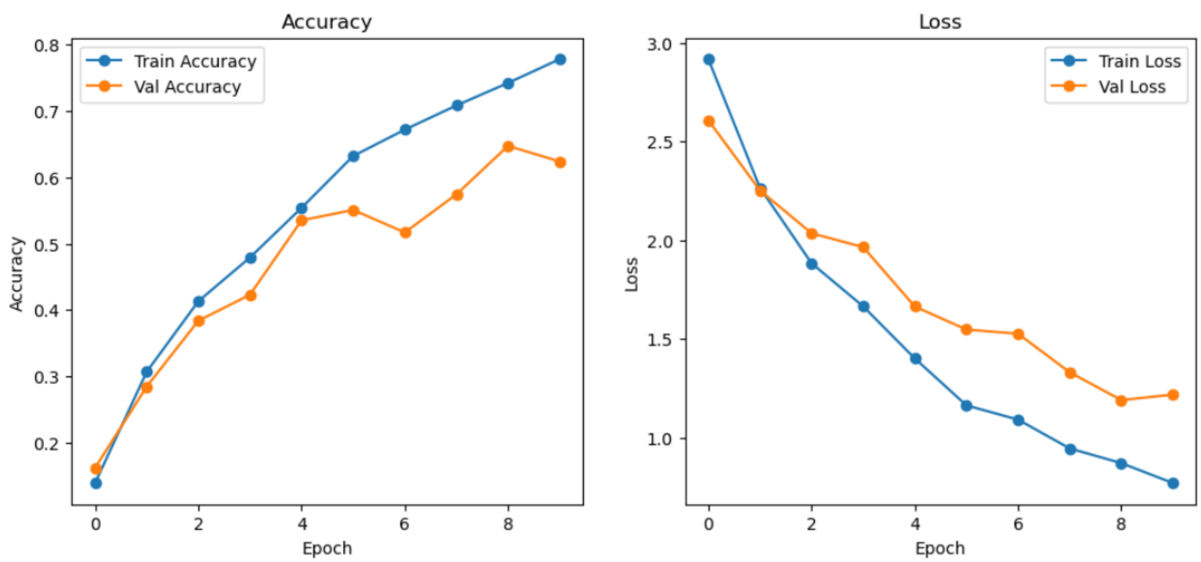
```
[151]: history = model.fit(
           train_generator,
           validation_data=validation_generator,
           epochs=10
       )
```

```
Epoch 1/10
49/49 ━━━━━━━━━━━━━━━━━━━━ 34s 587ms/step - accuracy: 0.1390 - loss: 2.9163 - val_accuracy: 0.1619 - val_loss: 2.6071
Epoch 2/10
49/49 ━━━━━━━━━━━━━━━━━━━━ 24s 498ms/step - accuracy: 0.3069 - loss: 2.2645 - val_accuracy: 0.2846 - val_loss: 2.2515
Epoch 3/10
49/49 ━━━━━━━━━━━━━━━━━━━━ 22s 446ms/step - accuracy: 0.4132 - loss: 1.8826 - val_accuracy: 0.3838 - val_loss: 2.0346
Epoch 4/10
49/49 ━━━━━━━━━━━━━━━━━━━━ 25s 505ms/step - accuracy: 0.4792 - loss: 1.6641 - val_accuracy: 0.4230 - val_loss: 1.9643
Epoch 5/10
49/49 ━━━━━━━━━━━━━━━━━━━━ 24s 492ms/step - accuracy: 0.5541 - loss: 1.4006 - val_accuracy: 0.5352 - val_loss: 1.6635
Epoch 6/10
49/49 ━━━━━━━━━━━━━━━━━━━━ 23s 465ms/step - accuracy: 0.6323 - loss: 1.1630 - val_accuracy: 0.5509 - val_loss: 1.5469
Epoch 7/10
49/49 ━━━━━━━━━━━━━━━━━━━━ 22s 455ms/step - accuracy: 0.6720 - loss: 1.0908 - val_accuracy: 0.5170 - val_loss: 1.5259
Epoch 8/10
49/49 ━━━━━━━━━━━━━━━━━━━━ 24s 483ms/step - accuracy: 0.7085 - loss: 0.9451 - val_accuracy: 0.5744 - val_loss: 1.3307
Epoch 9/10
49/49 ━━━━━━━━━━━━━━━━━━━━ 23s 469ms/step - accuracy: 0.7425 - loss: 0.8706 - val_accuracy: 0.6475 - val_loss: 1.1898
Epoch 10/10
49/49 ━━━━━━━━━━━━━━━━━━━━ 23s 471ms/step - accuracy: 0.7783 - loss: 0.7698 - val_accuracy: 0.6240 - val_loss: 1.2175
```

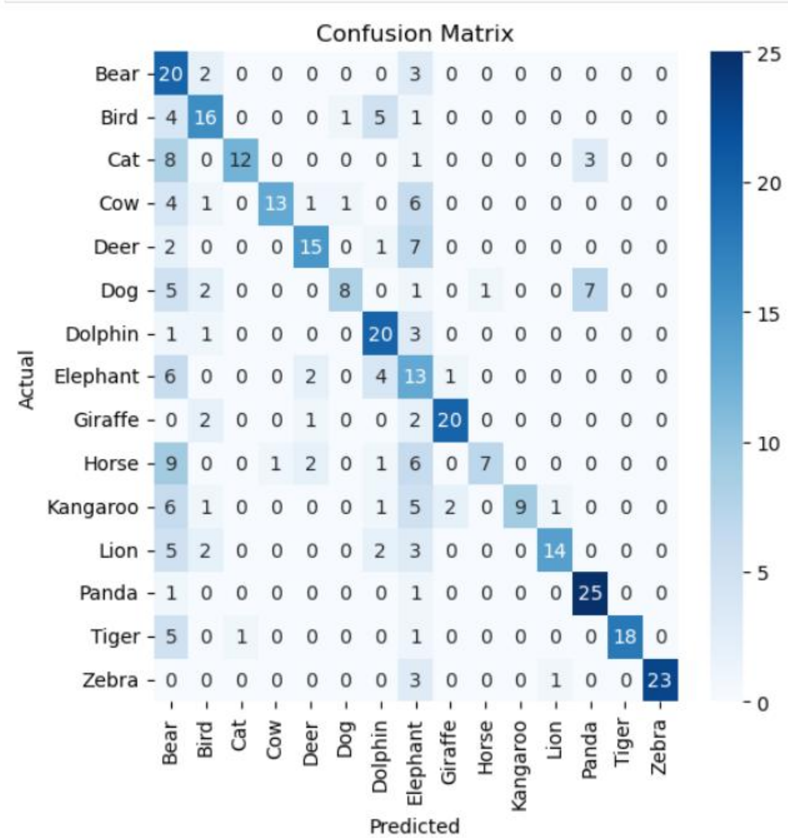
- Training the data by the model for 10 epochs(Passing the data set 10 times to the model to train and understand the underlying patterns.)



- The validation accuracy is also good intent compared with the training accuracy.
- Training is basically for the model to learn patterns and validation is basically to check if the model is generalizing the data without memorizing the data.



- With the increasing of accuracy , the loss or validation loss also decreases along with the training loss.



- This shows which categories get confused by the model when predicting the images.
- Main purpose of using transfer learning and also adding data generators is to improve the image count and improve the accuracy of the model. Because 383 images with 15-28 images per category is not enough for the model to accurately predict the image category on underlying or unseen images.

(2) VEHICLE PRICE PREDICTION

Overview

Regression model to predict used vehicle prices based on car specifications and conditions.

Algorithm

- **Model:** Linear Regression
- **Type:** Regression (Continuous price prediction)

Dataset

- **Source:** Used car pricing dataset
- **Total Samples:** ~4,500 vehicles
- **Features:**

- Make, Model, Trim
- Year of manufacture
- Mileage (odometer reading)
- Body type
- Transmission type
- Fuel type
- Engine size
- Number of doors
- Color

- **Target:** Price (USD)

```
[ ]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression
     from sklearn.preprocessing import LabelEncoder
     from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
[ ]: df = pd.read_csv('dataset.csv')
```

```
[ ]: df.head()
```

[4]:

	name	description	make	model	year	price	engine	cylinders	fuel	mileage	transmission	trim	body	doors	exterior_color	interior_color
0	2024 Jeep Wagoneer Series II	\n \n Heated Leather Seats, Nav Sy...	Jeep	Wagoneer	2024	74600.0	24V GDI DOHC Twin Turbo	6.0	Gasoline	10.0	8-Speed Automatic	Series II	SUV	4.0	White	Global Black
1	2024 Jeep Grand Cherokee Laredo	Al West is committed to offering every custome...	Jeep	Grand Cherokee	2024	50170.0	OHV	6.0	Gasoline	1.0	8-Speed Automatic	Laredo	SUV	4.0	Metallic	Global Black
2	2024 GMC Yukon XL Denali	NaN	GMC	Yukon XL	2024	96410.0	6.2L V-8 gasoline direct injection, variable v...	8.0	Gasoline	0.0	Automatic	Denali	SUV	4.0	Summit White	Teak/Light Shale
[illegible]	2023 Dodge [illegible]	White Knuckle Clearcoat [illegible]	[illegible]	[illegible]	[illegible]	[illegible]	16V [illegible]	[illegible]	[illegible]	[illegible]	8-Speed [illegible]	[illegible]	[illegible]	[illegible]	White Knuckle [illegible]	[illegible]

- Read the csv file and exploring the data

```
[ ]: df.isnull().sum()
```

[8]:

	0
name	0
description	56
make	0
model	0
year	0
price	23
engine	2
cylinders	105
fuel	7
mileage	34
transmission	2
trim	1
body	3
doors	7
exterior_color	5
interior_color	38
drivetrain	0

dtype: int64

- Since there are Nan values or null values, I replaced with the median and Unknown to remove the null values.

```
df = df.dropna(subset=['price'])
df = df[df['price'] > 0]
df['cylinders'] = df['cylinders'].fillna(df['cylinders'].median())
df['mileage'] = df['mileage'].fillna(df['mileage'].median())
df['doors'] = df['doors'].fillna(df['doors'].median())
df['fuel'] = df['fuel'].fillna('Unknown')
df['transmission'] = df['transmission'].fillna('Unknown')
df['body'] = df['body'].fillna('Unknown')
df['drivetrain'] = df['drivetrain'].fillna('Unknown')
df['engine'] = df['engine'].fillna('Unknown')
df['trim'] = df['trim'].fillna('Unknown')
df['exterior_color'] = df['exterior_color'].fillna('Unknown')
df['interior_color'] = df['interior_color'].fillna('Unknown')
```

```
print(df.isnull().sum())
```

```
After cleaning - missing values:
name              0
make              0
model             0
year              0
price             0
engine            0
cylinders         0
fuel              0
mileage           0
transmission      0
trim              0
body              0
doors             0
exterior_color    0
interior_color    0
drivetrain        0
dtype: int64
```

```
dtype: int64
```

```
[ ]: features = ['year', 'mileage', 'cylinders', 'doors', 'make', 'fuel', 'body', 'drivetrain', 'transmission', 'trim']
```

```
[ ]: X = df[features].copy()
     y = np.log1p(df['price'])
```

```
[ ]: X = pd.get_dummies(X, drop_first=True)
```

```
[ ]: X = df[features].copy()
     y = np.log1p(df['price'])
```

```
[ ]: # Dummies (one-hot encoding)
     X = pd.get_dummies(X, drop_first=True)
```

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

```
[ ]: model = LinearRegression()
     model.fit(X_train, y_train)
```

- Assign the features to a list , so that I can compare the test with the training.
- Used Linear Regression model – This is because we have continuous data not discreate data. So basically if its discreate data it falls under Classification , since the data is continuous like the price predicting as here in example, I used Linear regression.

```
r2 = r2_score(y_actual, y_pred)
mae = mean_absolute_error(y_actual, y_pred)
rmse = np.sqrt(mean_squared_error(y_actual, y_pred))

print(f"R²: {r2:.4f}  (higher is better, 0.6–0.8 is decent here)")
print(f"MAE:  ${mae:,.0f}   (average $ error)")
print(f"RMSE: ${rmse:,.0f}")
```

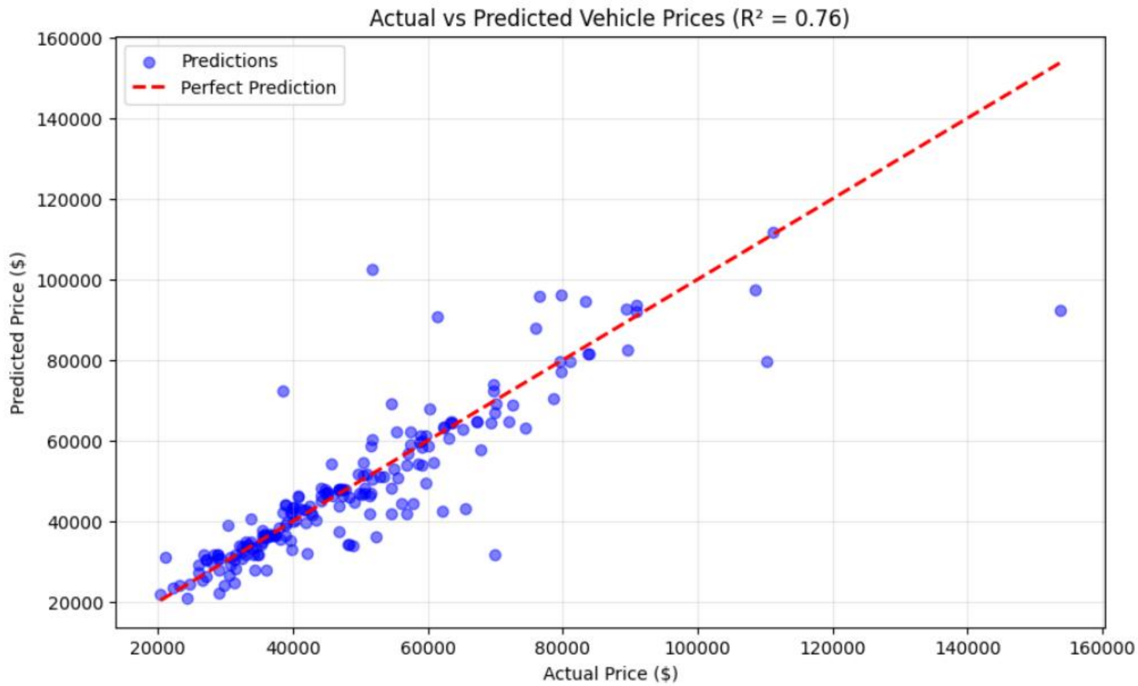
```
R²: 0.7598  (higher is better, 0.6–0.8 is decent here)
MAE:  $5,004   (average $ error)
RMSE: $9,317
```

- Calculated the R^2 score and found out its an average score.
- Mean average error is MAE
- Root mean square is RMSE

```
[ ]: # Bonus: show some predictions vs actual
print("\nSome predictions vs actual (first 10):")
results = pd.DataFrame({
    'Actual': y_actual.round(0).astype(int),
    'Predicted': y_pred.round(0).astype(int)
})
print(results.head(10))
```

```
Some predictions vs actual (first 10):
     Actual  Predicted
201   83940      81492
556   51803      60245
176   56105      44464
952   37335      36416
66    28860      31744
505   22260      23560
768   29111      27936
561   60080      58822
614   42150      31922
160   45038      47290
```

- A comparison between actual data and the data predicted by the model.



- Used Matplotlib to plot the data with the best fit line.
- As you can see , there are anomalies however most of the data is near or on the line.

- However , the R^2 score can be improved and accuracy can be improved.

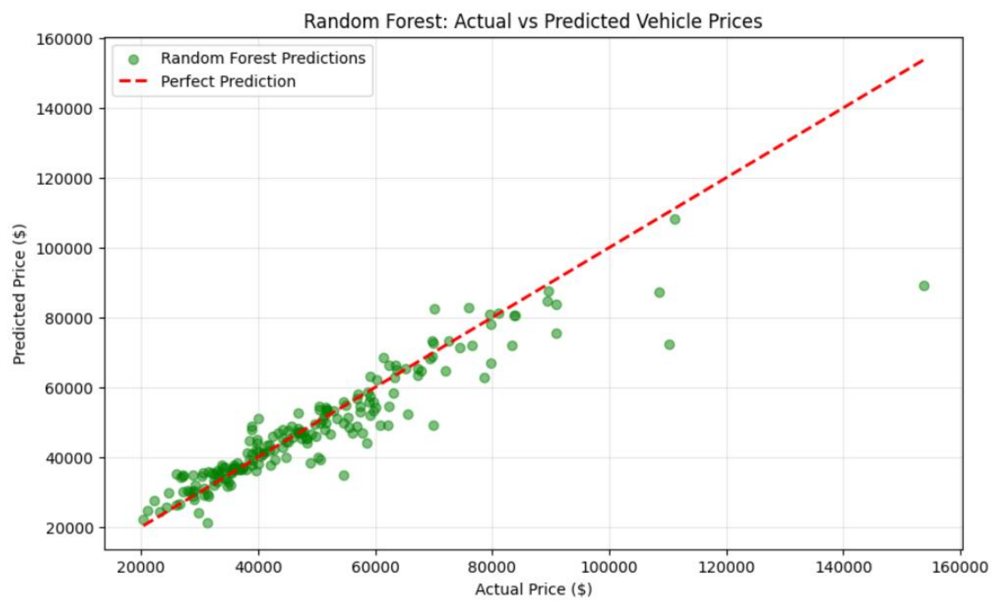
- Random Forrest Regressor is used.

[ ]:
```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
y_pred_log = model.predict(X_test)
y_pred = np.expm1(y_pred_log)
```

```
r2 = r2_score(y_actual, y_pred)
mae = mean_absolute_error(y_actual, y_pred)
rmse = np.sqrt(mean_squared_error(y_actual, y_pred))

print(f"R²: {r2:.4f}  (higher is better, 0.6-0.8 is decent here)")
print(f"MAE:  ${mae:,.0f}   (average $ error)")
print(f"RMSE: ${rmse:,.0f}")
```

```
R²: 0.8403  (higher is better, 0.6-0.8 is decent here)
MAE:  $4,147   (average $ error)
RMSE: $7,597
```

- Giving an r^2 score of 0.84 which is 84%.



(3) MOBILE PHONE PRICE PREDICTION

- Similar to the Vehicle Price prediction project almost similar however , I used a different model rather than the Linear Regression , is mainly because to show diversity and the learning through out the unified mentor internship.
- So the model I used here is DECISION TREE MODEL.

[10]:
```
df = pd.read_csv('dataset.csv')
```

[11]:
```
df.head()
```

[11]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	to
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	0	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	1	
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	1	
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	1	
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	1	

5 rows × 21 columns

[8]:
```
df.isnull().sum()
```

[8]:
```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

- Found out that the dataset had no null values, hence no data cleaning is required like in the previous project.

```
[16]: df['price_range'].value_counts()
```

```
[16]: price_range
      1    500
      2    500
      3    500
      0    500
      Name: count, dtype: int64
```

- Main reason for using decision tree here is that the Project is a Multi Class Classification. Which contained 3 categories where the previous one , is Continuous data.

```
[17]: df.corr()['price_range'].sort_values(ascending=False)
```

```
[17]: price_range      1.000000
      ram              0.917046
      battery_power    0.200723
      px_width         0.165818
      px_height        0.148858
      int_memory       0.044435
      sc_w             0.038711
      pc               0.033599
      three_g          0.023611
      sc_h             0.022986
      fc               0.021998
      talk_time        0.021859
      blue             0.020573
      wifi             0.018785
      dual_sim         0.017444
      four_g           0.014772
      n_cores          0.004399
      m_dep            0.000853
      clock_speed     -0.006606
      mobile_wt       -0.030302
      touch_screen    -0.030411
      Name: price_range, dtype: float64
```

- This shows the correlation between the output or the Price range with the other features.
- As you can see , the price range is highly dependent on the Ram which is 91.7% of all compared to the other features.

```
X = df.drop('price_range', axis=1)
y = df['price_range']


print(f"Features shape: {X.shape}")
print(f"Target shape: {y.shape}")
```

```
Features shape: (2000, 20)
Target shape: (2000,)
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,       # 20% for testing
    random_state=42,     # Same split every time
    stratify=y           # Keep same proportion of each class
)
```

- Splitting the data for the Test and the Train where the Test data size is 20% of the data while 80% of the data is for the Training of the data.

```
print(f"\nTraining set: {X_train.shape[0]} phones ({X_train.shape[0]/len(X)*100:.0f}%)")
print(f"Test set: {X_test.shape[0]} phones ({X_test.shape[0]/len(X)*100:.0f}%)")
```

```
Training set: 1600 phones (80%)
Test set: 400 phones (20%)
```

- 1600 phones for Training and 400 phones for Testing.

```
[25]: dt_model = DecisionTreeClassifier(
    max_depth=10,            # Limit depth to prevent overfitting
    min_samples_split=20,    # Need 20 samples to split
    random_state=42
)
```

- Implementing the model Decision Tree.

```
[27]: y_pred_dt = dt_model.predict(X_test)
```

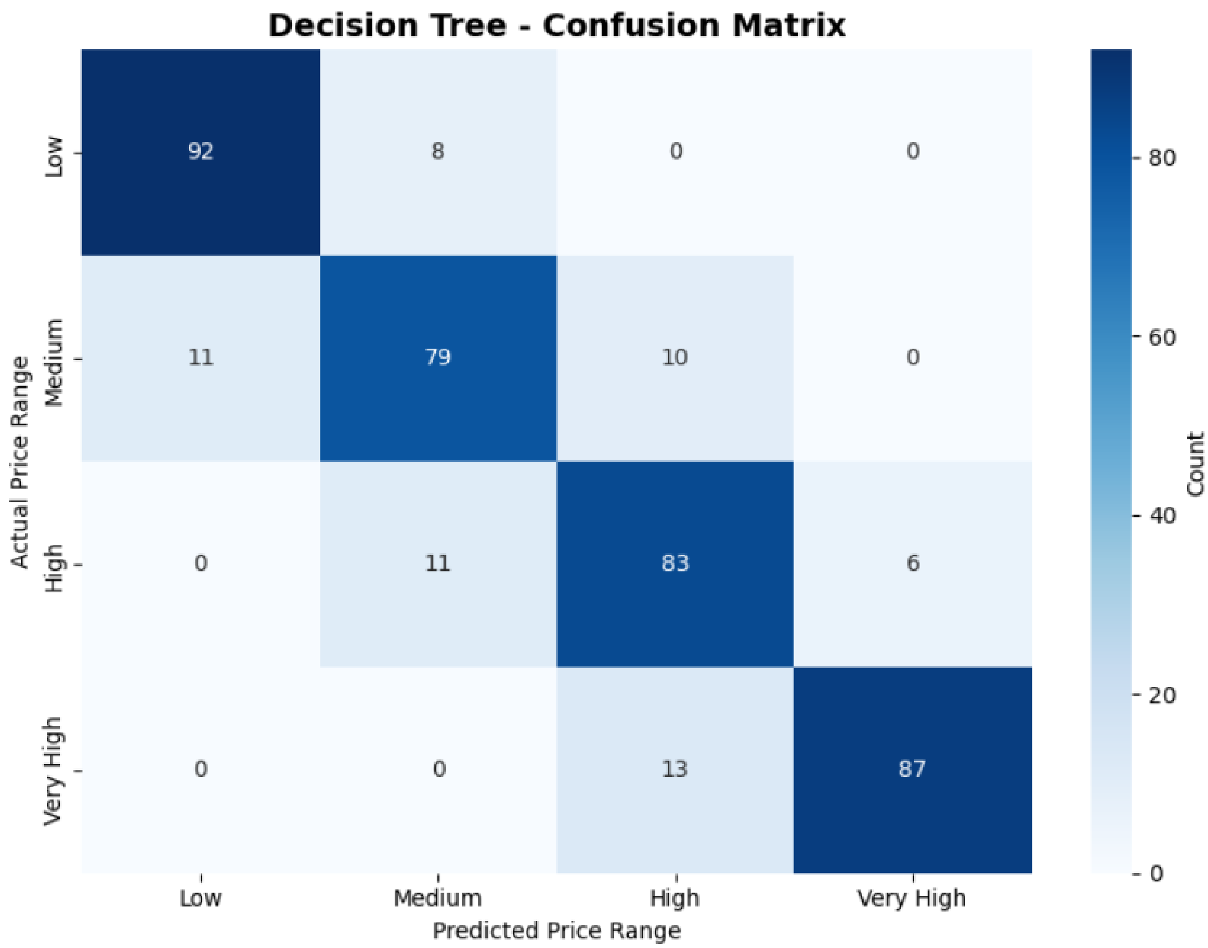
```
[28]: # Calculate accuracy
dt_accuracy = accuracy_score(y_test, y_pred_dt)
print(f"\nDecision Tree Accuracy: {dt_accuracy*100:.2f}%")
```

```
Decision Tree Accuracy: 85.25%
```

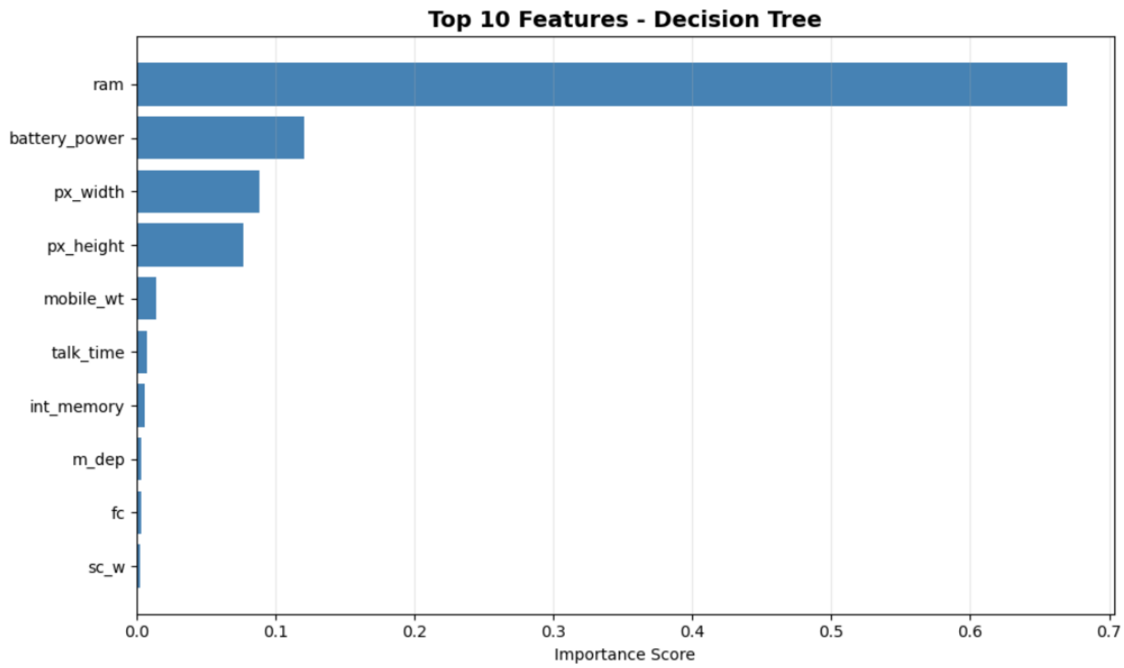
- Below is the Classification report.

```
============================================================
CLASSIFICATION REPORT - DECISION TREE
============================================================
              precision    recall  f1-score   support

         Low       0.89      0.92      0.91       100
      Medium       0.81      0.79      0.80       100
        High       0.78      0.83      0.81       100
   Very High       0.94      0.87      0.90       100

    accuracy                           0.85       400
   macro avg       0.85      0.85      0.85       400
weighted avg       0.85      0.85      0.85       400
```

- The confusion Matrix of the above Project is shown below.




```
============================================================
FEATURE IMPORTANCE - DECISION TREE
============================================================

Top 10 Most Important Features:
          feature  importance
13            ram    0.669858
0   battery_power    0.120525
12       px_width    0.088476
11      px_height    0.076886
8       mobile_wt    0.013887
16      talk_time    0.007909
6      int_memory    0.005978
7           m_dep    0.003548
4              fc    0.003407
15           sc_w    0.002921
```



- This plot is to show the correlation between the features with the Price range or the importance to the output.
- To make a solid comparison , I implemented Random Forest Classifier model like in the previous project to do the comparison.

```
rf_model = RandomForestClassifier(
    n_estimators=100,       # Build 100 trees
    max_depth=15,           # Each tree can be deeper
    min_samples_split=10,
    random_state=42,
    n_jobs=-1               # Use all CPU cores
)
```

- Use 100 trees with depth of 15 levels.

```
Random Forest Accuracy: 88.00%
```

```
[38]: # Detailed report
print("\n" + "="*60)
print("CLASSIFICATION REPORT - RANDOM FOREST")
print("="*60)
print(classification_report(y_test, y_pred_rf, target_names=class_names))
```

```
============================================================
CLASSIFICATION REPORT - RANDOM FOREST
============================================================
              precision    recall  f1-score   support

         Low       0.94      0.96      0.95       100
      Medium       0.81      0.82      0.82       100
        High       0.82      0.79      0.81       100
   Very High       0.94      0.95      0.95       100

    accuracy                           0.88       400
   macro avg       0.88      0.88      0.88       400
weighted avg       0.88      0.88      0.88       400
```

- This shows the accuracy 0.88 which is 88%
- Compared with the Decision Tree Model , the Random Forest Classifier has higher accuracy.

DECISION TREE -

```
============================================================
CLASSIFICATION REPORT - DECISION TREE
============================================================
              precision    recall  f1-score   support

         Low       0.89      0.92      0.91       100
      Medium       0.81      0.79      0.80       100
        High       0.78      0.83      0.81       100
   Very High       0.94      0.87      0.90       100

    accuracy                           0.85       400
   macro avg       0.85      0.85      0.85       400
weighted avg       0.85      0.85      0.85       400
```

```
0]: correct_dt = (y_pred_dt == y_test).sum()
print(f"\nCorrect predictions: {correct_dt}/{len(y_test)}")
```

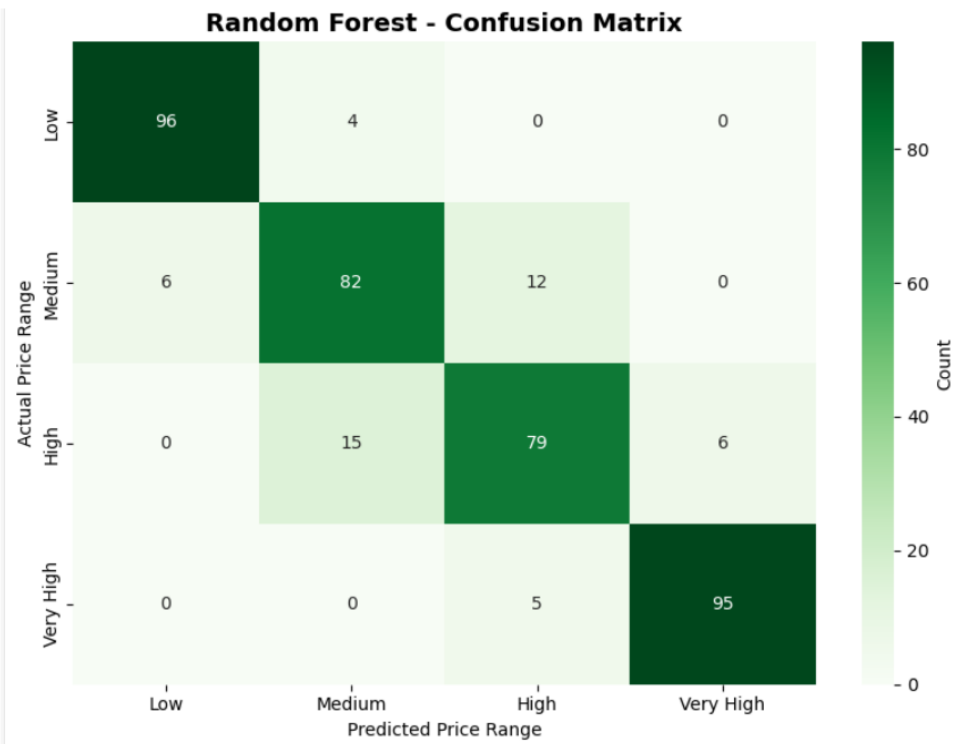
```
Correct predictions: 341/400
```

RANDOM FOREST CLASSIFIER

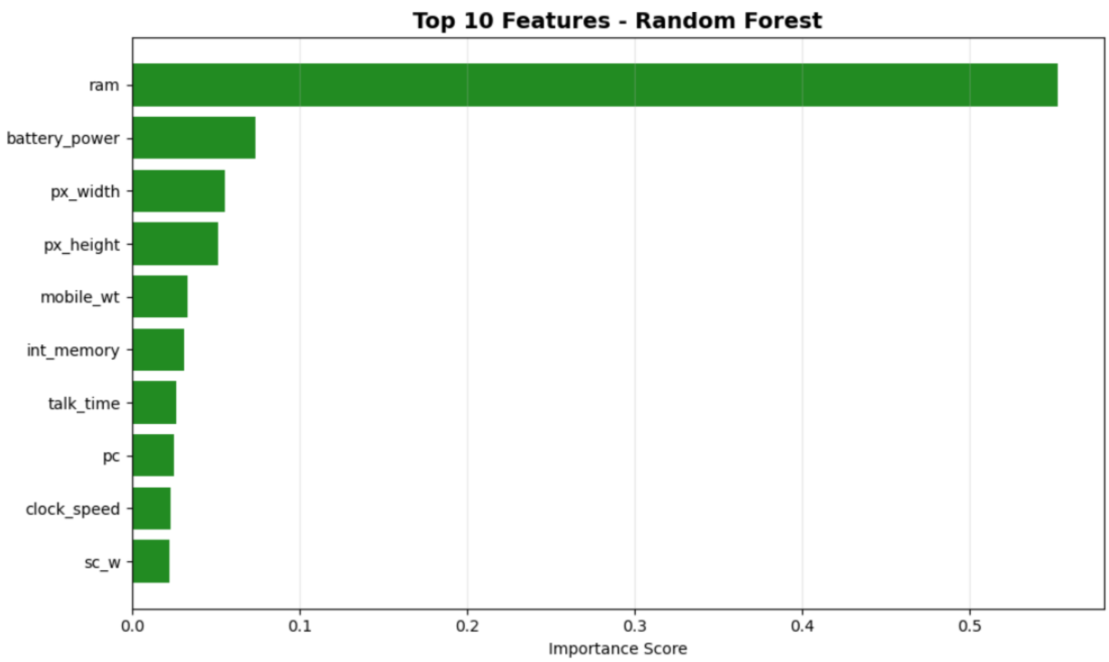
```
============================================================
CLASSIFICATION REPORT - RANDOM FOREST
============================================================
              precision    recall  f1-score   support

         Low       0.94      0.96      0.95       100
      Medium       0.81      0.82      0.82       100
        High       0.82      0.79      0.81       100
   Very High       0.94      0.95      0.95       100

    accuracy                           0.88       400
   macro avg       0.88      0.88      0.88       400
weighted avg       0.88      0.88      0.88       400
```

```
]: # How many correct?
correct_rf = (y_pred_rf == y_test).sum()
print(f"\nCorrect predictions: {correct_rf}/{len(y_test)}")
```

```
Correct predictions: 352/400
```



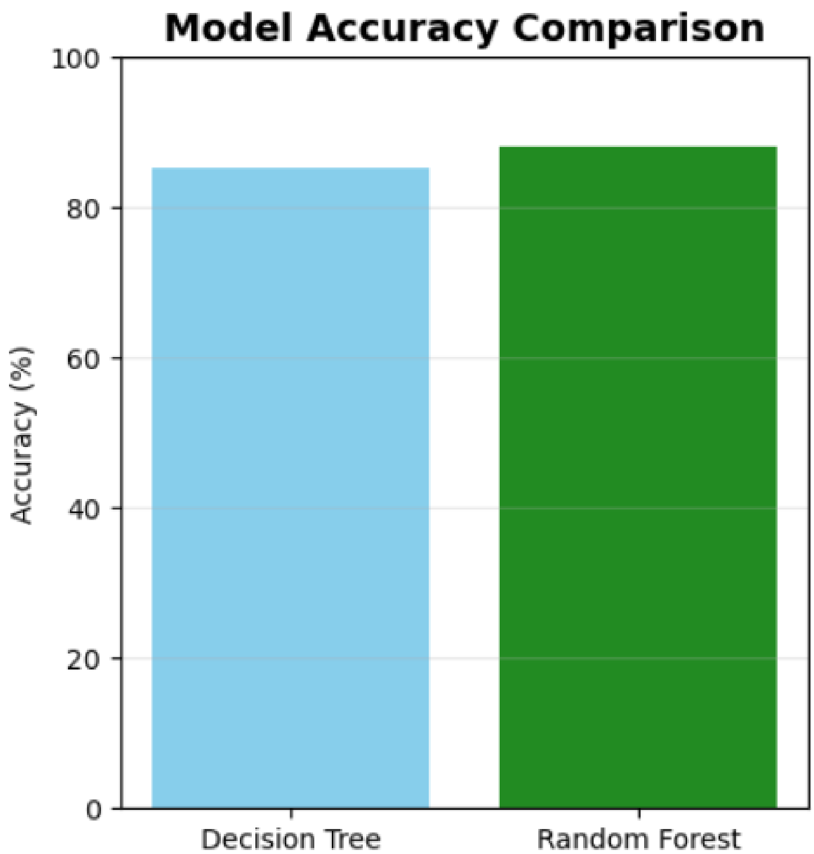
```
============================================================
FEATURE IMPORTANCE - RANDOM FOREST
============================================================

Top 10 Most Important Features:
          feature  importance
13            ram    0.552550
0   battery_power    0.074008
12       px_width    0.055601
11      px_height    0.051614
8       mobile_wt    0.033430
6      int_memory    0.031225
16      talk_time    0.026599
10             pc    0.025067
2     clock_speed    0.023410
15           sc_w    0.022805
```



- Like in the previous model , Plot some graphs in order to find the correlation between the price and the features.

COMPARISON BETWEEN MODELS

[45]:
```python
# Plot accuracy comparison
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
bars = plt.bar(comparison['Model'], comparison['Accuracy'],
               color=['skyblue', 'forestgreen'])
plt.ylabel('Accuracy (%)')
plt.title('Model Accuracy Comparison', fontsize=14, weight='bold')
plt.ylim(0, 100)
plt.grid(axis='y', alpha=0.3)
```



- Therefore,
    - Random Forrest Classifier has a better accuracy than Decision Tree by 2.75%.

(4) DETECT HEART DISEASE PREDICTION

- Using Logistic Regresion Model.
- Mainly because to explore the models learned in Unified Mentor

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (confusion_matrix, classification_report,
                             accuracy_score, precision_score, recall_score,
                             f1_score, roc_curve, roc_auc_score)
```

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

```
df.head()
```

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	ST slope	target
0	40	1	2	140	289	0	0	172	0	0.0	1	0
1	49	0	3	160	180	0	0	156	0	1.0	2	1
2	37	1	2	130	283	0	1	98	0	0.0	1	0
3	48	0	4	138	214	0	0	108	1	1.5	2	1
4	54	1	3	150	195	0	0	122	0	0.0	1	0

[3]:

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

[4]:

```
df.head()
```

[4]:

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	ST slope	target
0	40	1	2	140	289	0	0	172	0	0.0	1	0
1	49	0	3	160	180	0	0	156	0	1.0	2	1
2	37	1	2	130	283	0	1	98	0	0.0	1	0
3	48	0	4	138	214	0	0	108	1	1.5	2	1
4	54	1	3	150	195	0	0	122	0	0.0	1	0

[6]:

```
df.isnull().sum()
```

[6]:

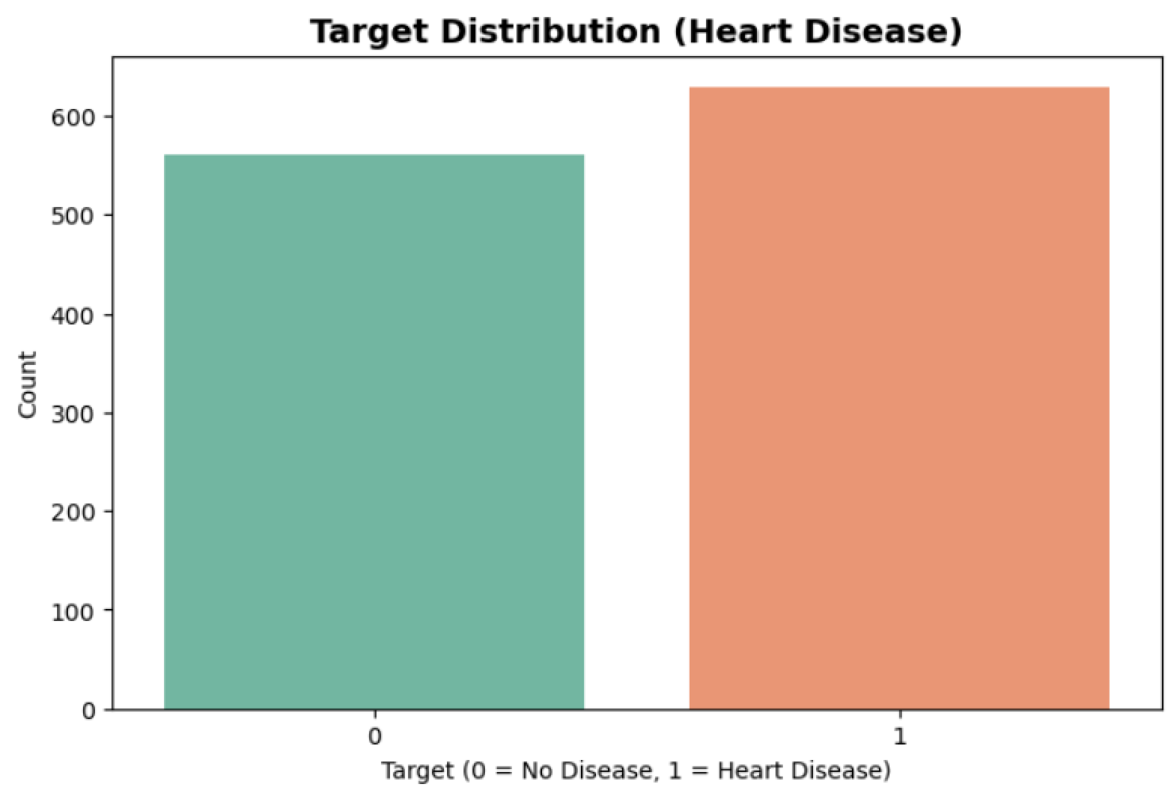
```
age                    0
sex                    0
chest pain type        0
resting bp s           0
cholesterol            0
fasting blood sugar    0
resting ecg            0
max heart rate         0
exercise angina        0
oldpeak                0
ST slope               0
target                 0
dtype: int64
```

- Using Logistic regression model is basically because this is just a Binary Classification. Either 0 or 1.

```
print("Target Distribution:")
print(df['target'].value_counts())
print(f"\nClass 0 (No Disease): {(df['target']==0).sum()} ({(df['target']==0).sum()/len(df)*100:.1f}%)")
print(f"Class 1 (Heart Disease): {(df['target']==1).sum()} ({(df['target']==1).sum()/len(df)*100:.1f}%)")
```

```
Target Distribution:
target
1    629
0    561
Name: count, dtype: int64

Class 0 (No Disease): 561 (47.1%)
Class 1 (Heart Disease): 629 (52.9%)
```

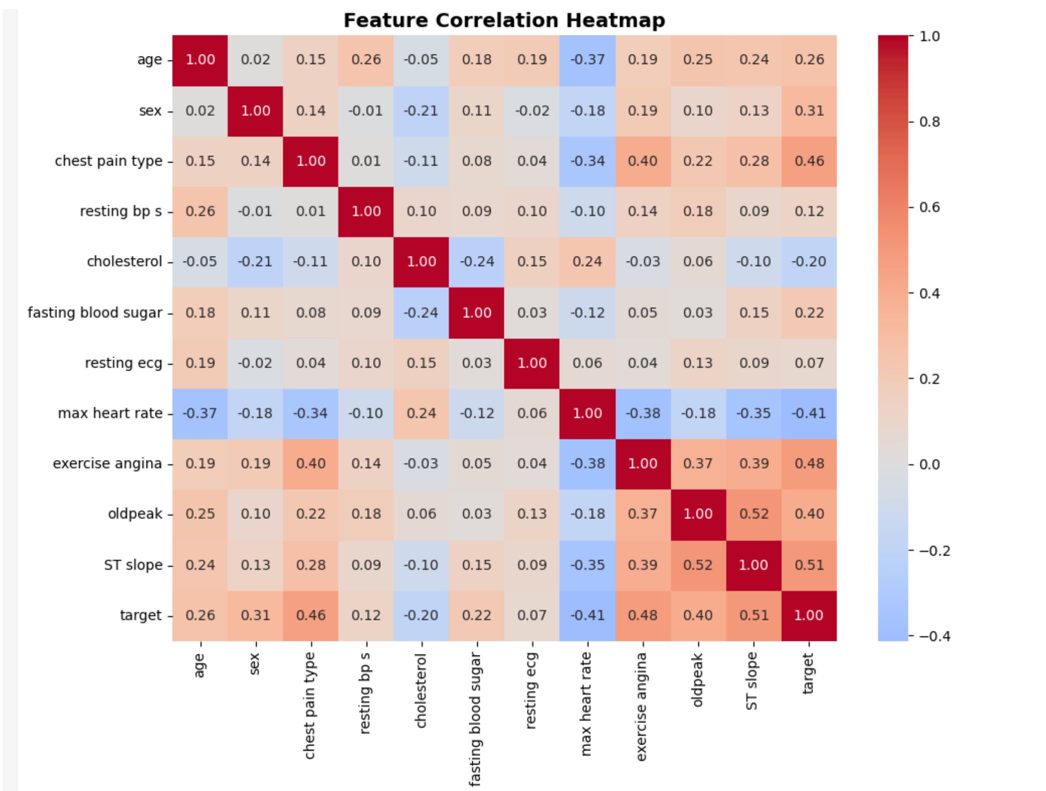
- Class 0 or No disease is 47% and Class 1 is 52% which shows that the data set is already balanced hence no modifications needed for the model.
- If the dataset is imbalanced, the model finds it hard to train cause there is no sufficient data for one of the output categories.



```python
correlations = df.corr()['target'].sort_values(ascending=False)
print("Features most correlated with Heart Disease:")
print(correlations)
```

```
Features most correlated with Heart Disease:
target                 1.000000
ST slope               0.505608
exercise angina        0.481467
chest pain type        0.460127
oldpeak                0.398385
sex                    0.311267
age                    0.262029
fasting blood sugar    0.216695
resting bp s           0.121415
resting ecg            0.073059
cholesterol           -0.198366
max heart rate        -0.413278
Name: target, dtype: float64
```

- A comparison or shows the relationship/correlation between the target and the other features.



- For more comparison , I implemented a correlation heatmap which shows the correaltions between the target and other features. The negative high values shows inversely impact while positive high values shows direct impact and less values shows almost no impact.

[13]:
```python
# Cell: Feature Scaling
print("🔧 Scaling features using StandardScaler...")
print("\n💡 Why scaling is important:")
print("   • ST slope ranges: 1-3")
print("   • Max heart rate ranges: 71-202")
print("   • Different scales slow down Gradient Descent!")
print("   • StandardScaler makes all features mean=0, std=1")

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=feature_columns)

print("\nScaling complete!")
print("\nBefore vs After (first 3 samples):")
print("\nBefore:")
print(X.head(3))
print("\nAfter (standardized):")
print(X_scaled.head(3))
```

- Using standard scaler so that values comes under 0-1 rather than big values making the model easy to learn and train.

```
Before vs After (first 3 samples):

Before:
   age  sex  chest pain type  resting bp s  cholesterol  fasting blood sugar  \
0   40    1                2           140          289                    0
1   49    0                3           160          180                    0
2   37    1                2           130          283                    0

   resting ecg  max heart rate  exercise angina  oldpeak  ST slope
0            0             172                0      0.0         1
1            0             156                0      1.0         2
2            1              98                0      0.0         1

After (standardized):
        age       sex  chest pain type  resting bp s  cholesterol  \
0 -1.466728  0.555995        -1.318351      0.427328     0.775674
1 -0.504600 -1.798576        -0.248932      1.516587    -0.299512
2 -1.787437  0.555995        -1.318351     -0.117301     0.716489

   fasting blood sugar  resting ecg  max heart rate  exercise angina  \
0            -0.520929    -0.802672        1.265039        -0.795219
1            -0.520929    -0.802672        0.637758        -0.795219
2            -0.520929     0.346762       -1.636136        -0.795219
```

[14]:

```python
# Cell: Split data
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y,
    test_size=0.2,
    random_state=42,
    stratify=y  # Keeps 47-53 balance in both sets!
)

print("Data split complete!")
print(f"\n📊 Training set: {X_train.shape[0]} samples")
print(f"   Class 0: {sum(y_train==0)} ({sum(y_train==0)/len(y_train)*100:.1f}%)")
print(f"   Class 1: {sum(y_train==1)} ({sum(y_train==1)/len(y_train)*100:.1f}%)")

print(f"\n🧪 Testing set: {X_test.shape[0]} samples")
print(f"   Class 0: {sum(y_test==0)} ({sum(y_test==0)/len(y_test)*100:.1f}%)")
print(f"   Class 1: {sum(y_test==1)} ({sum(y_test==1)/len(y_test)*100:.1f}%)")
```

```
Data split complete!

📊 Training set: 952 samples
   Class 0: 449 (47.2%)
   Class 1: 503 (52.8%)

🧪 Testing set: 238 samples
   Class 0: 112 (47.1%)
   Class 1: 126 (52.9%)
```

- Splitting the data between the training and the test datasets for plotting.

[15]:

```python
model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train, y_train)
```

[15]:

```
▾ LogisticRegression
▶ Parameters
```

- Logistic regression model.

```python
print("Model Parameters:")
print(f"Intercept: {model.intercept_[0]:.4f}")

print("\nFeature Coefficients:")
for feature, coef in zip(feature_columns, model.coef_[0]):
    print(f"{feature:25s}: {coef:.4f}")
```

```
Model Parameters:
Intercept: 0.2344

Feature Coefficients:
age                      : 0.2162
sex                      : 0.6148
chest pain type          : 0.6910
resting bp s             : 0.1296
cholesterol              : -0.2561
fasting blood sugar      : 0.3888
resting ecg              : 0.0072
max heart rate           : -0.2212
exercise angina          : 0.5083
oldpeak                  : 0.3996
ST slope                 : 0.7798
```

- This shows the y intercept which is Theta0 and then other features like X1θ1 , X2 θ2 etc

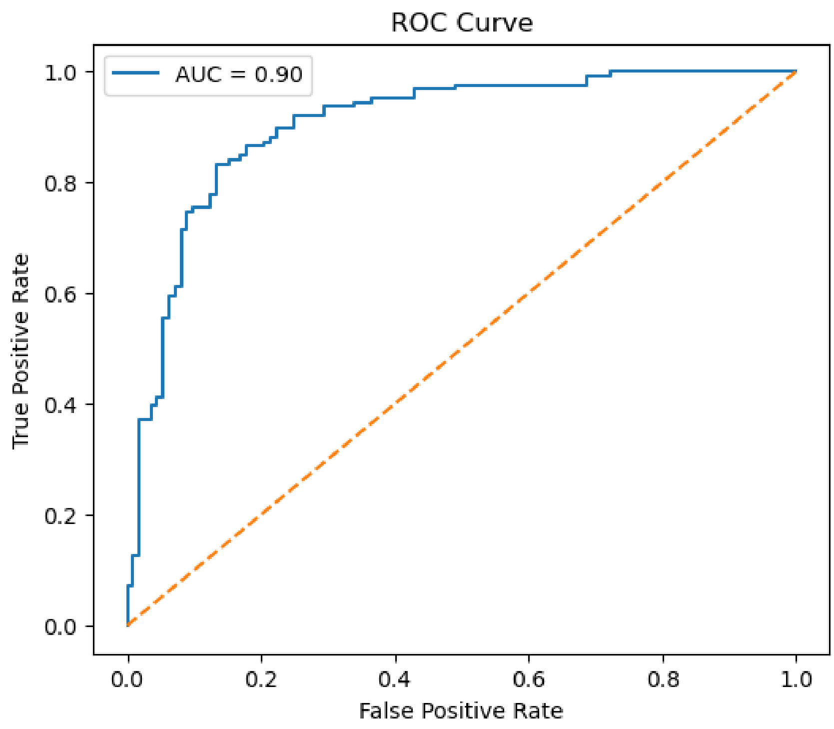
```python
print("Sample Predictions:")
for i in range(10):
    print(f"Actual: {y_test.iloc[i]}, Predicted: {y_test_pred[i]}, Probability: {y_test_prob[i]:.2f}")
```

```
Sample Predictions:
Actual: 1, Predicted: 1, Probability: 0.96
Actual: 0, Predicted: 0, Probability: 0.45
Actual: 1, Predicted: 1, Probability: 0.94
Actual: 0, Predicted: 1, Probability: 0.53
Actual: 0, Predicted: 0, Probability: 0.09
Actual: 1, Predicted: 1, Probability: 0.89
Actual: 0, Predicted: 0, Probability: 0.02
Actual: 0, Predicted: 0, Probability: 0.22
Actual: 0, Predicted: 0, Probability: 0.01
Actual: 1, Predicted: 1, Probability: 0.90
```

- Shows the predictions between the actual value and the predicted by the model.

- Predictions like 0.53 gives the model more error because slight difference leads to different prediction where actual is 0 but predicted as 1 since 0.5 >

```
from sklearn.metrics import roc_curve, auc

fpr, tpr, _ = roc_curve(y_test, y_test_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
plt.plot([0, 1], [0, 1], linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```



- Implemented ROC Curve to show more better between the True positive and the False positives rates.

False Positve Rate = False Positives / (False Positives + True Negatives)

➔ Lower the better

True Positive Rate = True Positives / (True Positives + False nEGATIVES)

➔ Higher the better.

- Blue Line is the Model, shows how well the model distinguish between the healthy and sick or between 1 and 0.
- Orange Dashed Line is the Random Classifier, this is model guess randomly. Any model which is above this line is better.
- AUC = Area under the curve which is Total area under BLUE LINE