Name - Aritra Mukherjee

ID: UMIP270247

Machine Learning Intern

Unified Mentor Pvt. Limited

Projects List :

1. Animal Classification
2. Forest cover prediction
3. Heart Disease
4. Mobile Phone Pricing
5. Thyroid cancer
6. Vehicle Price Prediction

Main GitHub Repo- UMIP270247---All-Projects

CONTINUE

Project - 1

Animal Classifier Using Deep Learning & Transfer Learning.

Presented by – Aritra Mukherjee

ID: UMIP270247

Machine Learning Intern
At - Unified Mentor Pvt. Limited

Introduction To Project – Animal Classification

Animal classification is an essential task in the field of computer vision, where the goal is to identify the category of an animal based on an input image. This project focuses on building an intelligent system that can automatically classify animals into 15 predefined categories using image data.

By training a model on a diverse dataset of animal images, we aim to develop a solution that understands visual patterns and features—such as shapes, colors, and textures—to accurately predict the animal shown in the image.

This project not only helps in applying core deep learning concepts but also demonstrates how AI can be used in real-world scenarios such as wildlife conservation, educational tools, and image-based search systems.

GOAL

In this project, our goal is to develop an intelligent image classification system capable of recognizing different animals from input images.

My aim to create a solution that can:

- Automatically analyze an image and identify the animal shown.
- Provide accurate predictions in real time
- Offer a smooth and interactive user experience for classification

Dataset Overview for Classification

Dataset Structure

• The dataset comprises 15 folders, each representing a distinct class for classification tasks.

Image Dimensions

• All images in the dataset are standardized to dimensions of 224 x 224 x 3 pixels, ideal for deep learning applications.

Suitability for Deep Learning

• The dataset is specifically designed for image classification using deep learning and transfer learning methods.

Class Representation

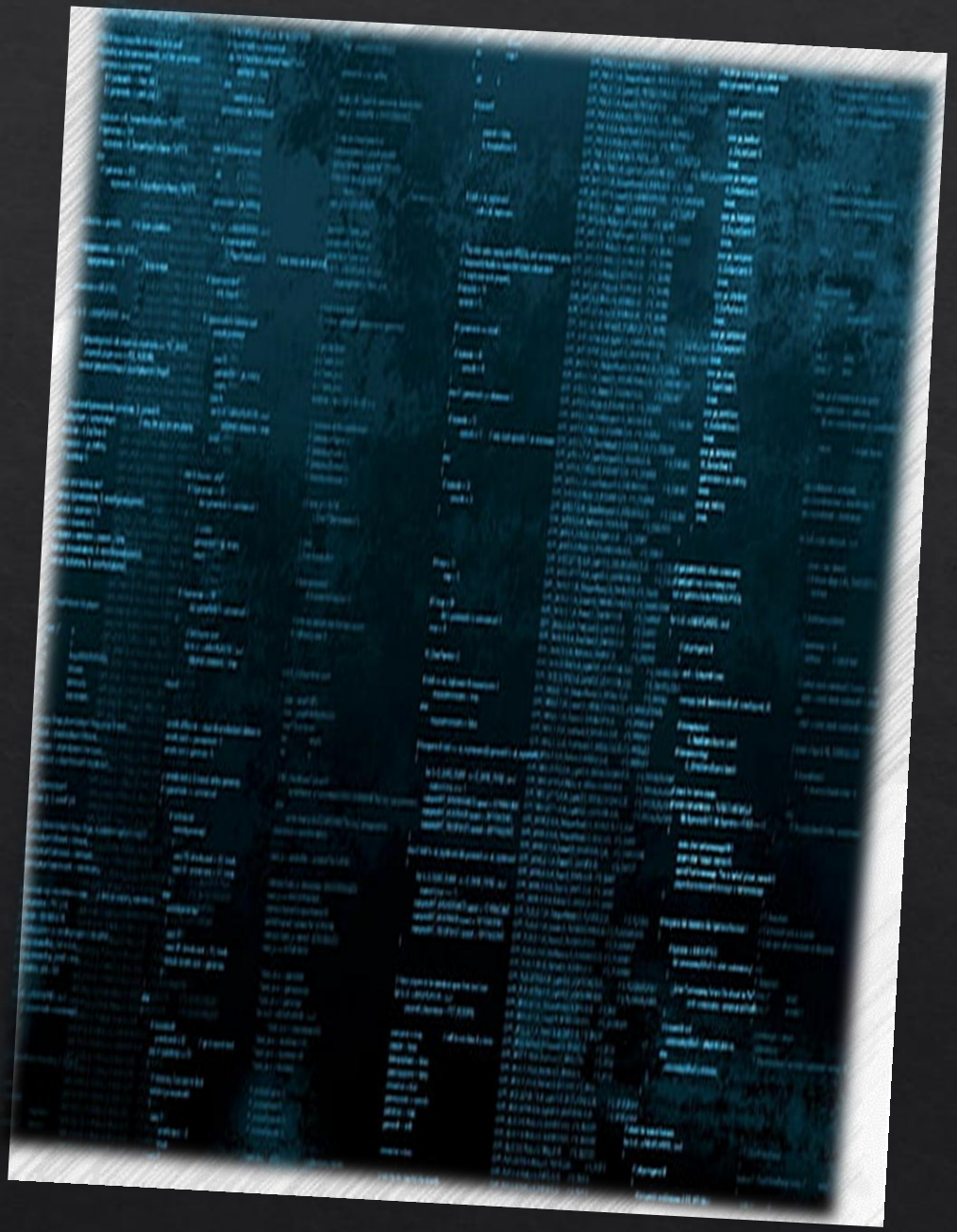
• Each folder's name corresponds to a specific class label, facilitating organized data handling.

Image Classification Potential

• This dataset provides a robust foundation for training models on various animal classifications.

Animal Classes in the Dataset

1. Bear
2. Dog
3. Cat
4. Cow
5. Deer
6. Bird
7. Dolphin
8. Elephant
9. Giraffe
10. Horse
11. Kangaroo
12. Panda
13. Tiger
14. Zebra
15. Lion



Tech Stack

Technologies & Tools Used

- **Python**

The primary programming language used for data handling, model training, and app logic.

- **TensorFlow & Keras**

Deep learning libraries used to build and train the convolutional neural network (CNN) model.

- **Transfer Learning (MobileNetV2)**

A lightweight, pretrained model used to enhance performance and reduce training time.

- **Gradio**

A user-friendly library to create an interactive web interface for real-time predictions.

- **Hugging Face Spaces**

Used for deploying the model and app to the web, making it publicly accessible without requiring backend infrastructure.

Import Required Libraries

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import gradio as gr
import matplotlib.pyplot as plt
```

Prepare the Data with Augmentation

```python
train_datagen = keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2  # 20% for validation
)

train_df = train_datagen.flow_from_directory(
    'dataset',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

validation_df = train_datagen.flow_from_directory(
    'dataset',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 1561 images belonging to 15 classes.
Found 383 images belonging to 15 classes.
```

Load Pretrained MobileNetV2

```
base_model = MobileNetV2(
    weights='imagenet',
    include_top=False,
    input_shape=(224, 224, 3)
    )
base_model.trainable = False  # Freeze base model
```

Loads MobileNetV2
(pretrained on ImageNet) without the top layer and freezes it
so only the new layers will train.

Build the Full Model

```
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.3),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(15, activation='softmax')
])
```

Compile the Model

```
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0005),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Define Training Callbacks

```python
callbacks = [
    EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2)
]
```

Train the Model

```python
# Model Training
history = model.fit(
    train_df,
    validation_data=validation_df,
    epochs=30,
    callbacks=callbacks
)
```

Train the Model

```
Epoch 1/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 43s 767ms/step - accuracy: 0.3389 - loss: 2.2444 - val_accuracy: 0.7520 - val_loss: 0.8392 - learning_rate: 5.0000e-04
Epoch 2/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 37s 760ms/step - accuracy: 0.7442 - loss: 0.8127 - val_accuracy: 0.8198 - val_loss: 0.5716 - learning_rate: 5.0000e-04
Epoch 3/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 38s 774ms/step - accuracy: 0.8304 - loss: 0.5554 - val_accuracy: 0.8407 - val_loss: 0.5144 - learning_rate: 5.0000e-04
Epoch 4/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 39s 799ms/step - accuracy: 0.8512 - loss: 0.4733 - val_accuracy: 0.8329 - val_loss: 0.5467 - learning_rate: 5.0000e-04
Epoch 5/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 39s 804ms/step - accuracy: 0.8543 - loss: 0.4394 - val_accuracy: 0.8355 - val_loss: 0.5133 - learning_rate: 5.0000e-04
Epoch 6/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 39s 801ms/step - accuracy: 0.8689 - loss: 0.4131 - val_accuracy: 0.8251 - val_loss: 0.5762 - learning_rate: 5.0000e-04
Epoch 7/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 39s 805ms/step - accuracy: 0.9191 - loss: 0.2671 - val_accuracy: 0.8512 - val_loss: 0.4820 - learning_rate: 5.0000e-04
Epoch 8/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 41s 833ms/step - accuracy: 0.9066 - loss: 0.2887 - val_accuracy: 0.8433 - val_loss: 0.5225 - learning_rate: 5.0000e-04
Epoch 9/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 41s 840ms/step - accuracy: 0.9080 - loss: 0.2980 - val_accuracy: 0.8433 - val_loss: 0.4931 - learning_rate: 5.0000e-04
Epoch 10/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 39s 793ms/step - accuracy: 0.9205 - loss: 0.2536 - val_accuracy: 0.8773 - val_loss: 0.3965 - learning_rate: 2.5000e-04
Epoch 11/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 41s 833ms/step - accuracy: 0.9200 - loss: 0.2279 - val_accuracy: 0.8512 - val_loss: 0.4738 - learning_rate: 2.5000e-04
Epoch 12/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 38s 780ms/step - accuracy: 0.9361 - loss: 0.2210 - val_accuracy: 0.8616 - val_loss: 0.4208 - learning_rate: 2.5000e-04
Epoch 13/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 613ms/step - accuracy: 0.9422 - loss: 0.1820 - val_accuracy: 0.8460 - val_loss: 0.4786 - learning_rate: 1.2500e-04
Epoch 14/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 28s 582ms/step - accuracy: 0.9433 - loss: 0.1697 - val_accuracy: 0.8642 - val_loss: 0.4319 - learning_rate: 1.2500e-04
Epoch 15/30
49/49 ━━━━━━━━━━━━━━━━━━━━ 28s 582ms/step - accuracy: 0.9461 - loss: 0.1769 - val_accuracy: 0.8642 - val_loss: 0.4407 - learning_rate: 6.2500e-05
```

Accuracy – 94%

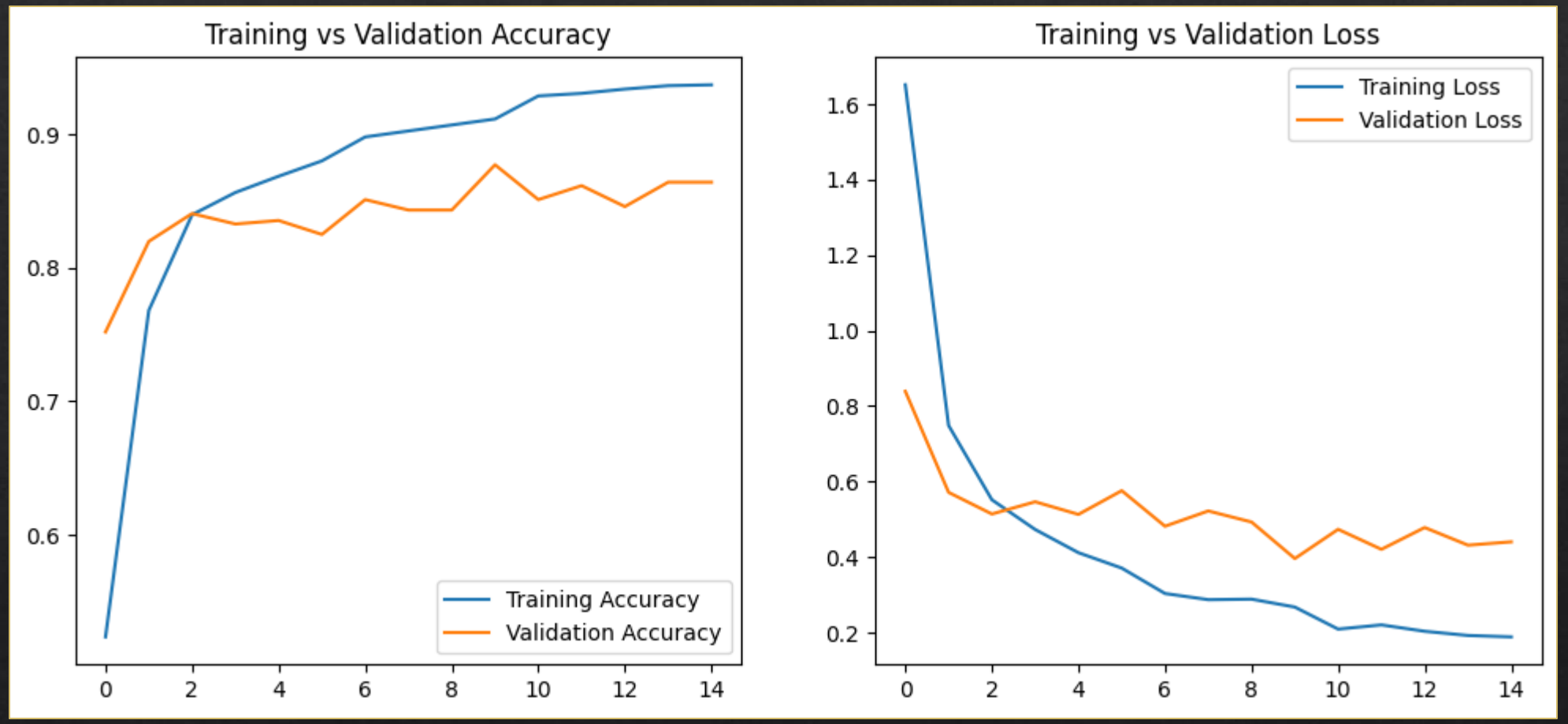
Save the Trained Model

```
model.save('animal_classifier_model.h5')
```

Visualize Training History

```
def plot_training_history(history):
    plt.figure(figsize=(12, 5))

    # Accuracy Plot
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Training vs Validation Accuracy')
    plt.legend()

    # Loss Plot
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Training vs Validation Loss')
    plt.legend()

    plt.show()

plot_training_history(history)
```

Visualization



Prediction

Animal Classifier - Identify Your Wildlife Companion!

AritraMOfficial / Animal_Classifier 1

image

output

Predicted: Tiger (Confidence: 99.949996948242 19%)

Clear

Submit

Animal Classifier - Identify Your Wildlife Companion!

AritraMOfficial / Animal_Classifier 1

image

output

Predicted: Bird (Confidence: 99.98999786376953%)

Clear

Submit

Deploy Using Hugging Face Space

Live Demo - aritramofficial-animal-classifier

Conclusion

We successfully built an intelligent animal image classifier that predicts the animal in any given image with high accuracy.

It demonstrates the practical use of deep learning in solving real-world visual problems and can be applied in areas like wildlife monitoring, educational apps, and smart animal recognition systems.

Project - 2

Forest Cover Type Prediction Using Machine Learning

Presented by – Aritra Mukherjee

ID: UMIP270247

Machine Learning Intern
At - Unified Mentor Pvt. Limited

Introduction To Project – Forest Cover Type Prediction

The Forest Cover Type Prediction project uses geographical and cartographic features to predict the type of forest cover.

It is based on the Roosevelt National Forest in northern Colorado.

Helps in ecological planning, forest management, and environmental monitoring.

GOAL

Predicting Vegetation Types Based on Cartographic Features.

- Build a machine learning model to predict one of the **7 forest cover types** based on input features like elevation, slope, soil type, etc.

- Deploy a user-friendly **Gradio-based web application** where users can input data and get real-time predictions.

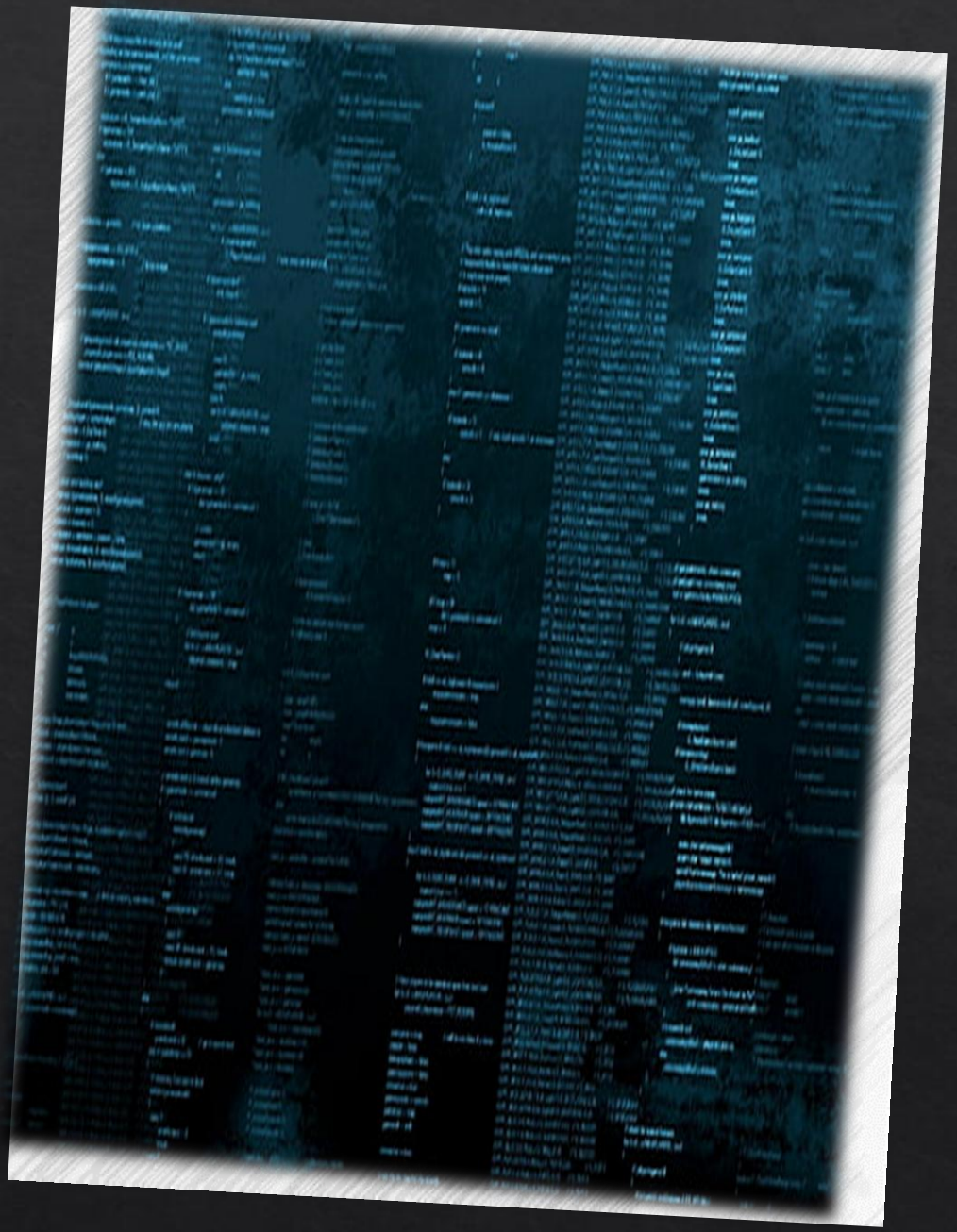
Dataset Overview for Classification

Dataset:
Forest Cover Type Dataset – Given By Unified Mentor Pvt. Lim

Features:
Numerical: Elevation, Slope, Aspect, Distances, Hillshades
Categorical (One-hot): 4 Wilderness Areas, 40 Soil Types

Target Classes (7 types):

1. Spruce/Fir
2. Lodgepole Pine
3. Ponderosa Pine
4. Cottonwood/Willow
5. Aspen
6. Douglas-fir
7. Krummholz

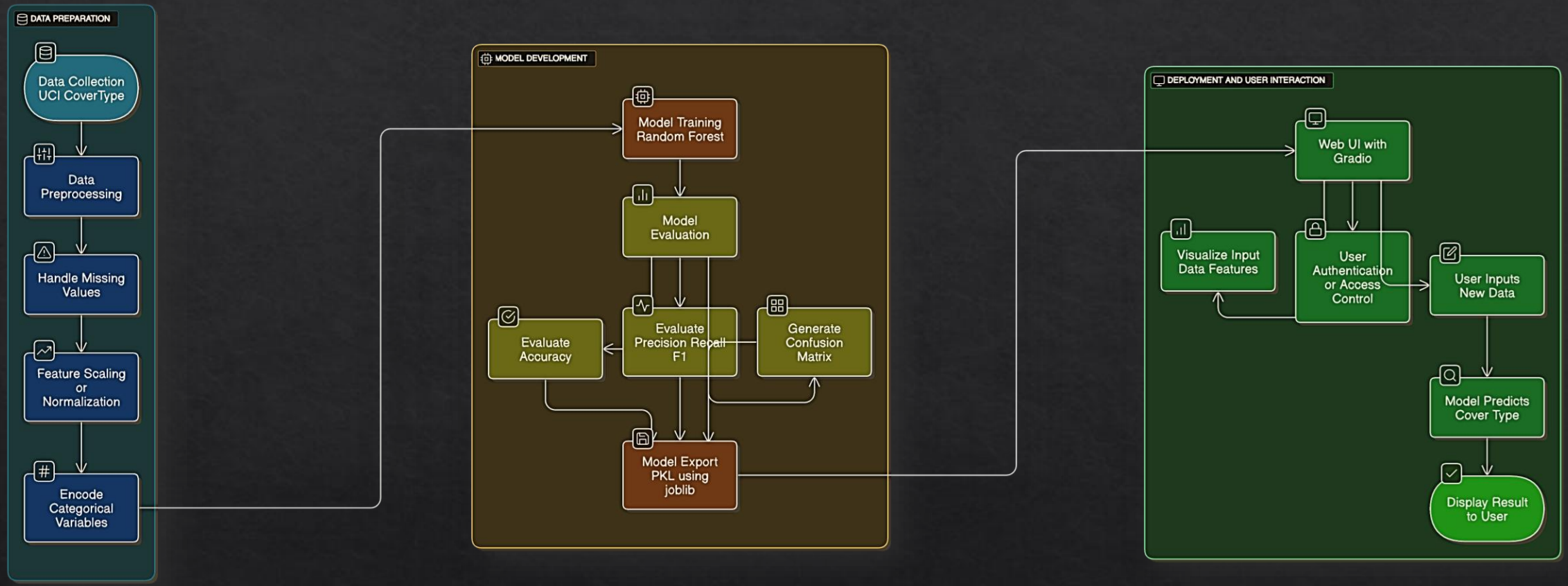


Tech Stack

Technologies & Tools Used

- Programming Language:
  - Python
- Libraries:
  - Scikit-learn – Random Forest Model
  - Joblib – Model serialization
  - NumPy – Data handling
  - Gradio – Web-based user interface
- Tools:
  - Jupyter Notebook (for training and analysis)
  - VS Code - Virtual Python environment
  - Gradio Blocks for deployment

Work Flowchart



Import Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import joblib
```

Load The Dataset

```
df = pd.read_csv("train.csv")
df.drop('Id', axis=1, inplace=True)

df.head()
```

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am	Hillshade_Noon	Hillshade_3pm
0	2596	51	3	258	0	510	221	232	148
1	2590	56	2	212	-6	390	220	235	151
2	2804	139	9	268	65	3180	234	238	135
3	2785	155	18	242	118	3090	238	238	122
4	2595	45	2	153	-1	391	220	234	150

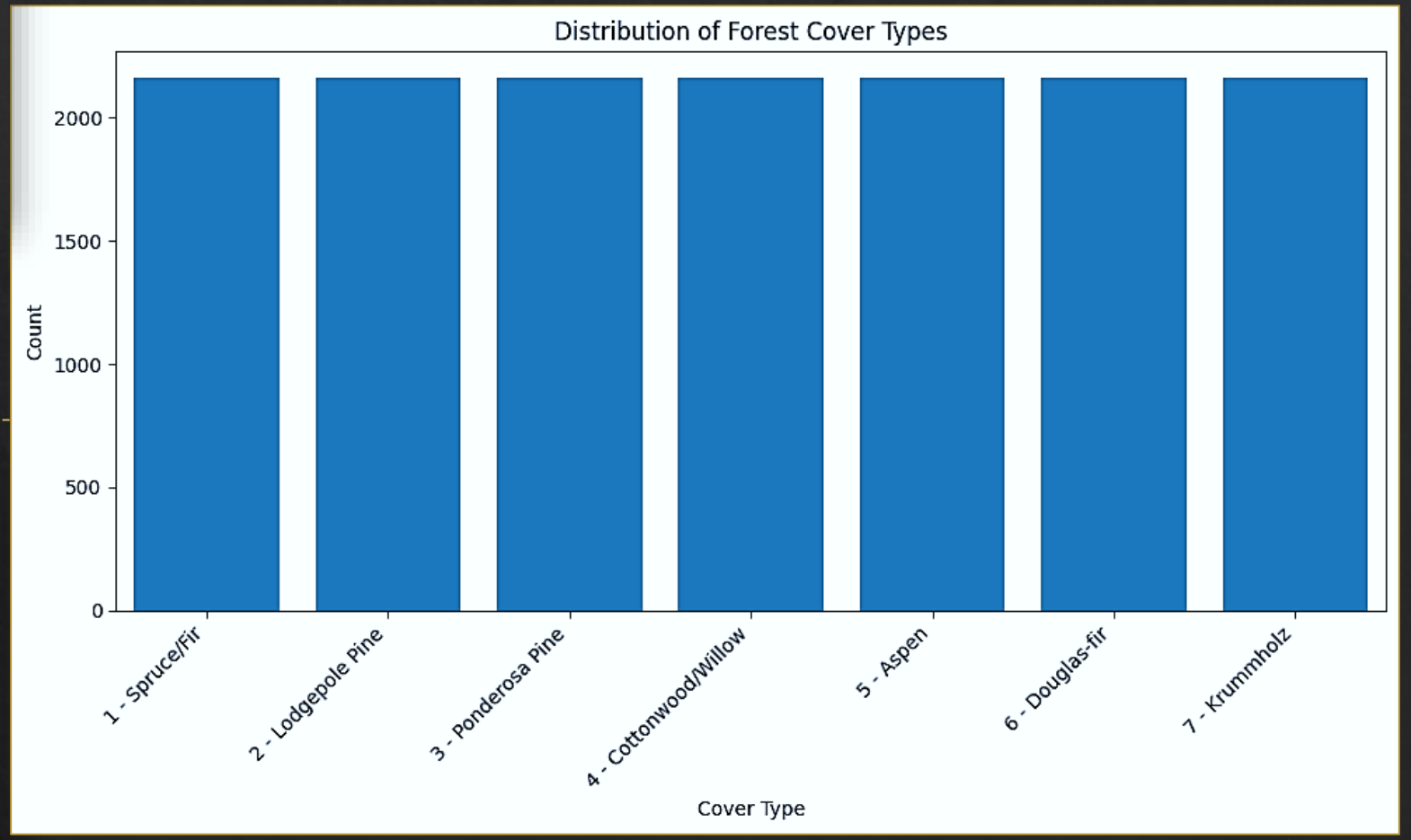
5 rows × 55 columns

...	Soil_Type32	Soil_Type33	Soil_Type34	Soil_Type35	Soil_Type36	Soil_Type37	Soil_Type38	Soil_Type39	Soil_Type40	Cover_Type
...	0	0	0	0	0	0	0	0	0	5
...	0	0	0	0	0	0	0	0	0	5
...	0	0	0	0	0	0	0	0	0	2
...	0	0	0	0	0	0	0	0	0	2
...	0	0	0	0	0	0	0	0	0	5

Visualization

- Distribution of Forest Cover Types

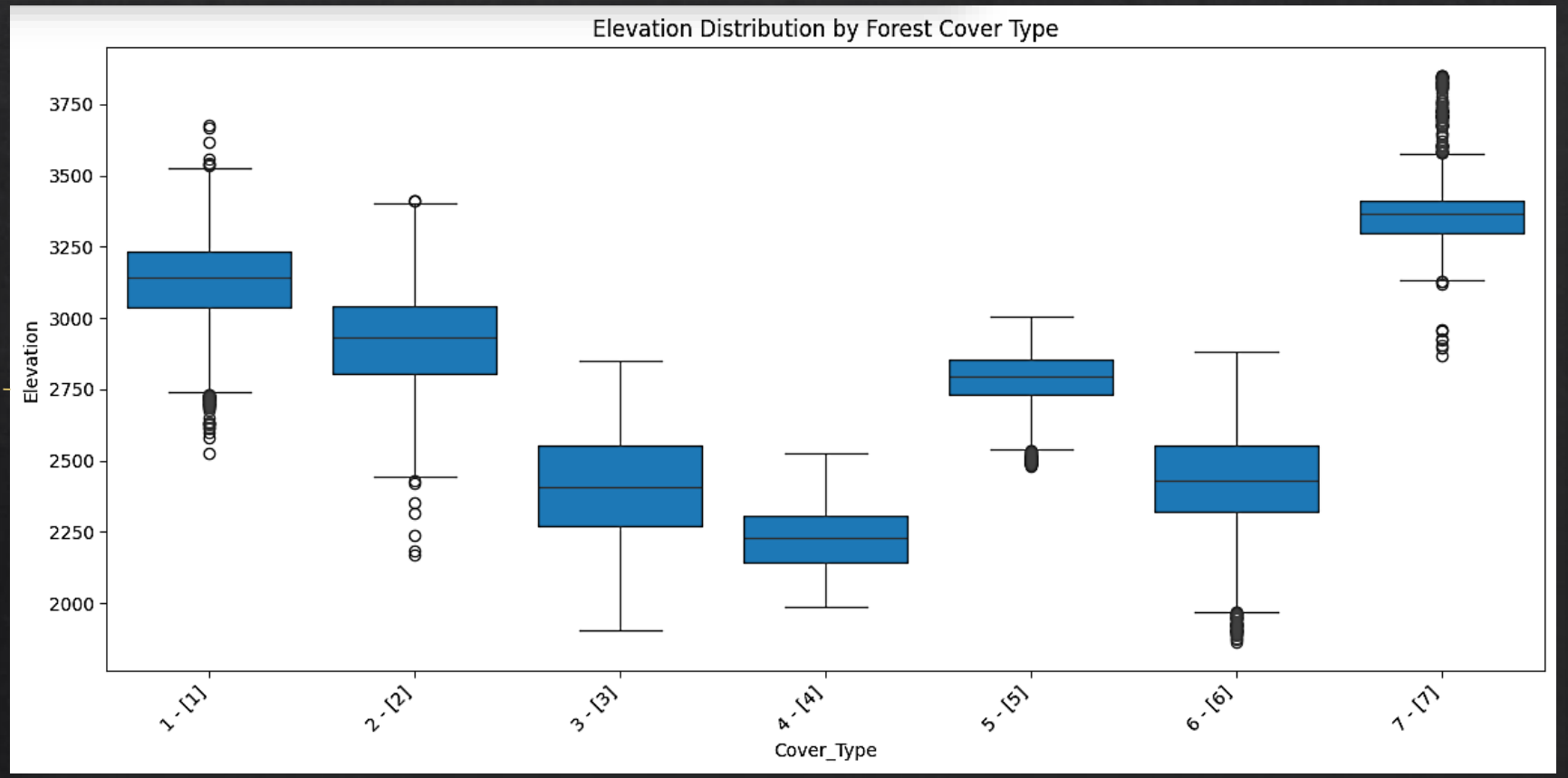
```
plt.figure(figsize=(10, 6))
cover_type_counts = df['Cover_Type'].value_counts().sort_index()
sns.barplot(x=cover_type_counts.index, y=cover_type_counts.values)
plt.title('Distribution of Forest Cover Types')
plt.xlabel('Cover Type')
plt.ylabel('Count')


cover_types = {
    1: 'Spruce/Fir',
    2: 'Lodgepole Pine',
    3: 'Ponderosa Pine',
    4: 'Cottonwood/Willow',
    5: 'Aspen',
    6: 'Douglas-fir',
    7: 'Krummholz'
}

plt.xticks(range(7), [f"{i+1} - {cover_types[i+1]}" for i in range(7)], rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Visualization

- Checks how Elevation relates to Cover Type

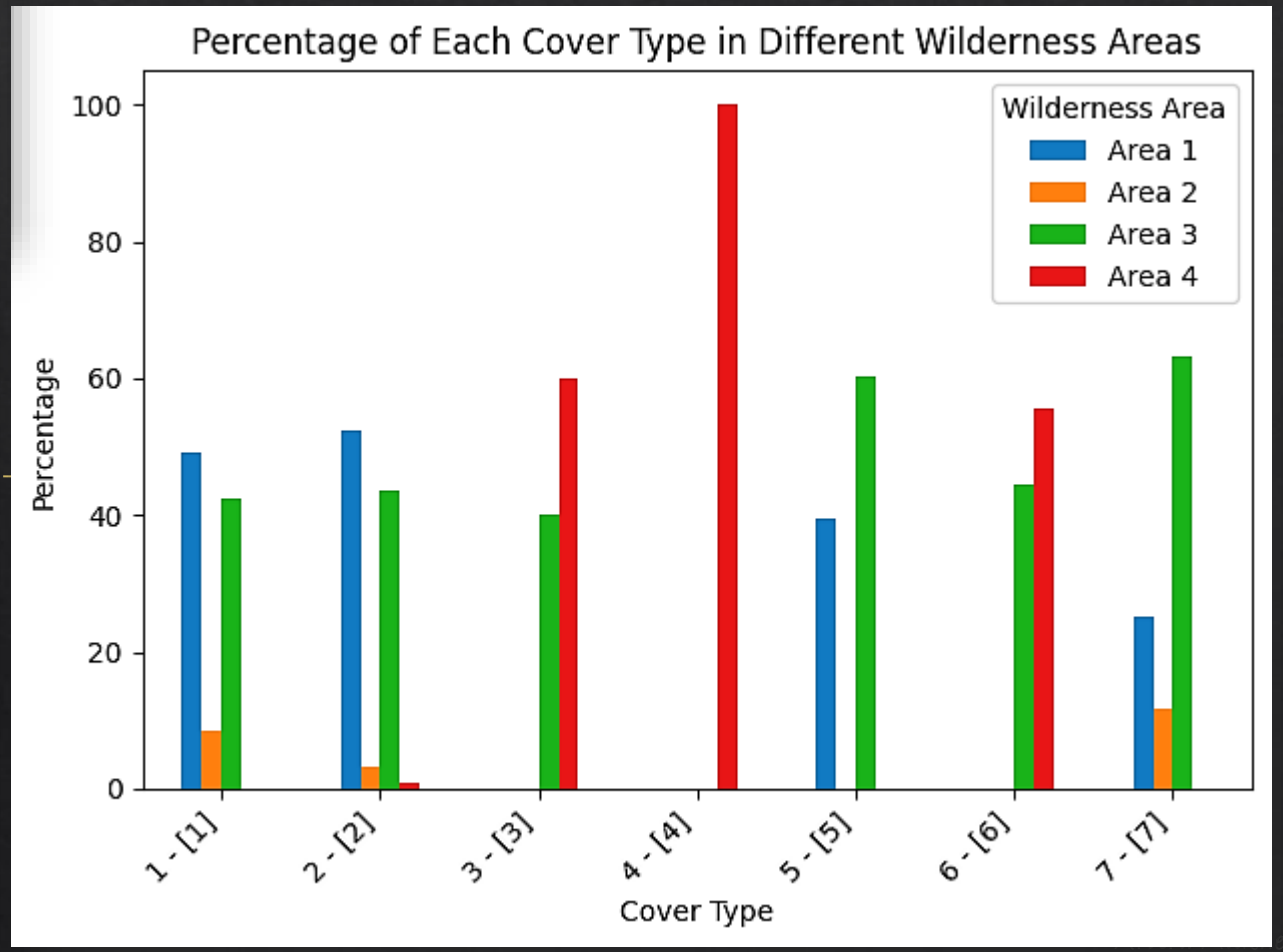
```
plt.figure(figsize=(12, 6))
sns.boxplot(x='Cover_Type', y='Elevation', data=df)
plt.title('Elevation Distribution by Forest Cover Type')
plt.xticks(range(7), [f"{i+1} - {[i+1]}" for i in range(7)], rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Visualization

- Check distribution of wilderness areas per cover type

```
wilderness_columns = [col for col in df.columns if col.startswith('Wilderness_Area')]
wilderness_dist = pd.DataFrame()

for i, col in enumerate(wilderness_columns):
    temp = df.groupby('Cover_Type')[col].mean() * 100
    wilderness_dist[f'Area {i+1}'] = temp

plt.figure(figsize=(12, 6))
wilderness_dist.plot(kind='bar')
plt.title('Percentage of Each Cover Type in Different Wilderness Areas')
plt.xlabel('Cover Type')
plt.ylabel('Percentage')
plt.xticks(range(7), [f"{i+1} - {[i+1]}" for i in range(7)], rotation=45, ha='right')
plt.legend(title='Wilderness Area')
plt.tight_layout()
plt.show()
```



Data Preprocessing

```python
# Splitting features and target
X = df.drop("Cover_Type", axis=1)
y = df["Cover_Type"]

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

# Train Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Save the Best Model

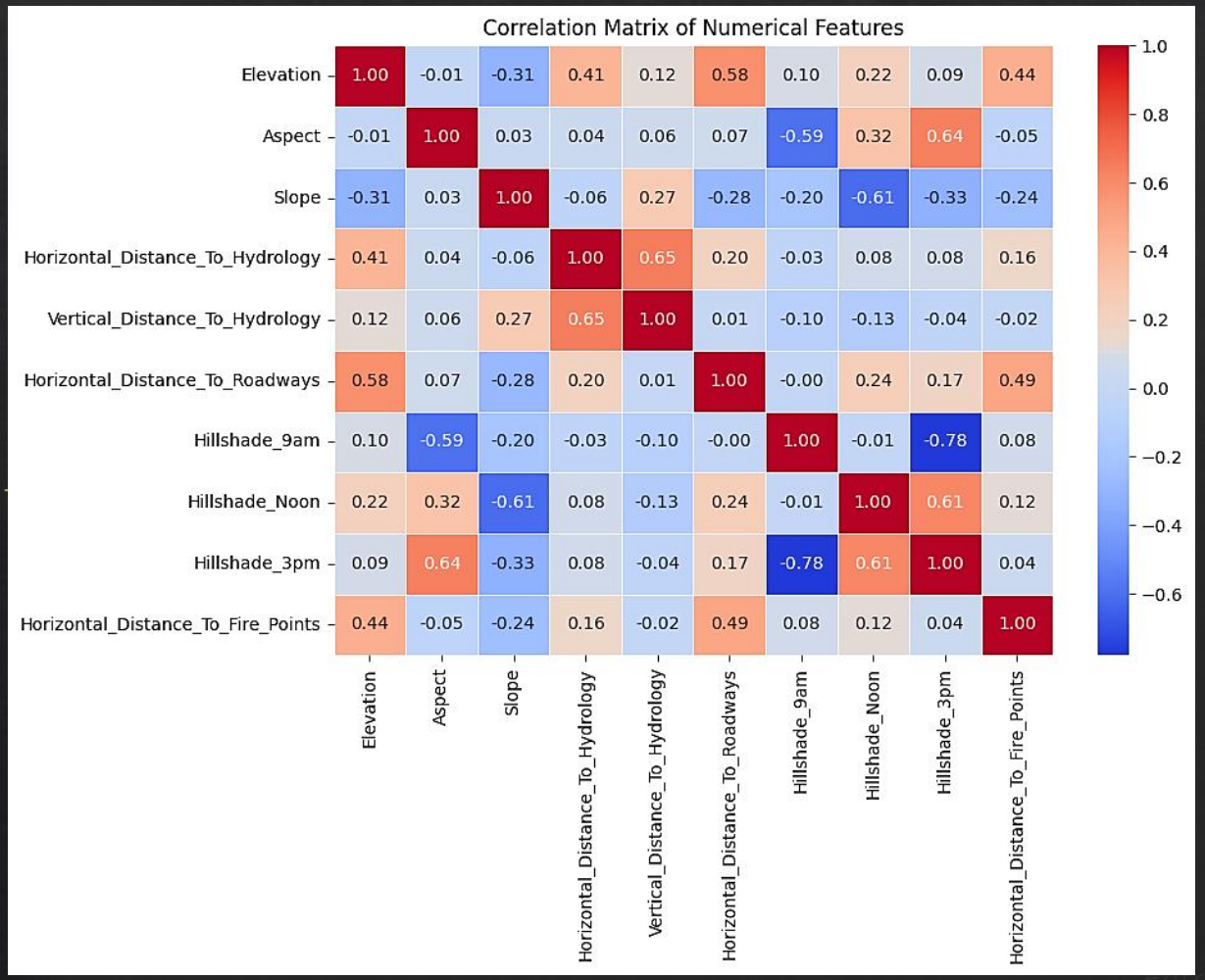
```python
joblib.dump(rf, "forest_cover_model.pkl")
```

Visualization

- Visualize correlations between numerical features

```
binary_cols = [col for col in X.columns if col.startswith('Wilderness_Area') or col.startswith('Soil_Type')]
numerical_cols = [col for col in X.columns if col not in binary_cols]
```

```
plt.figure(figsize=(10, 8))
correlation_matrix = df[numerical_cols].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix of Numerical Features')
plt.tight_layout()
plt.show()
```



Model Training and Tuning

```python
y_pred = rf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

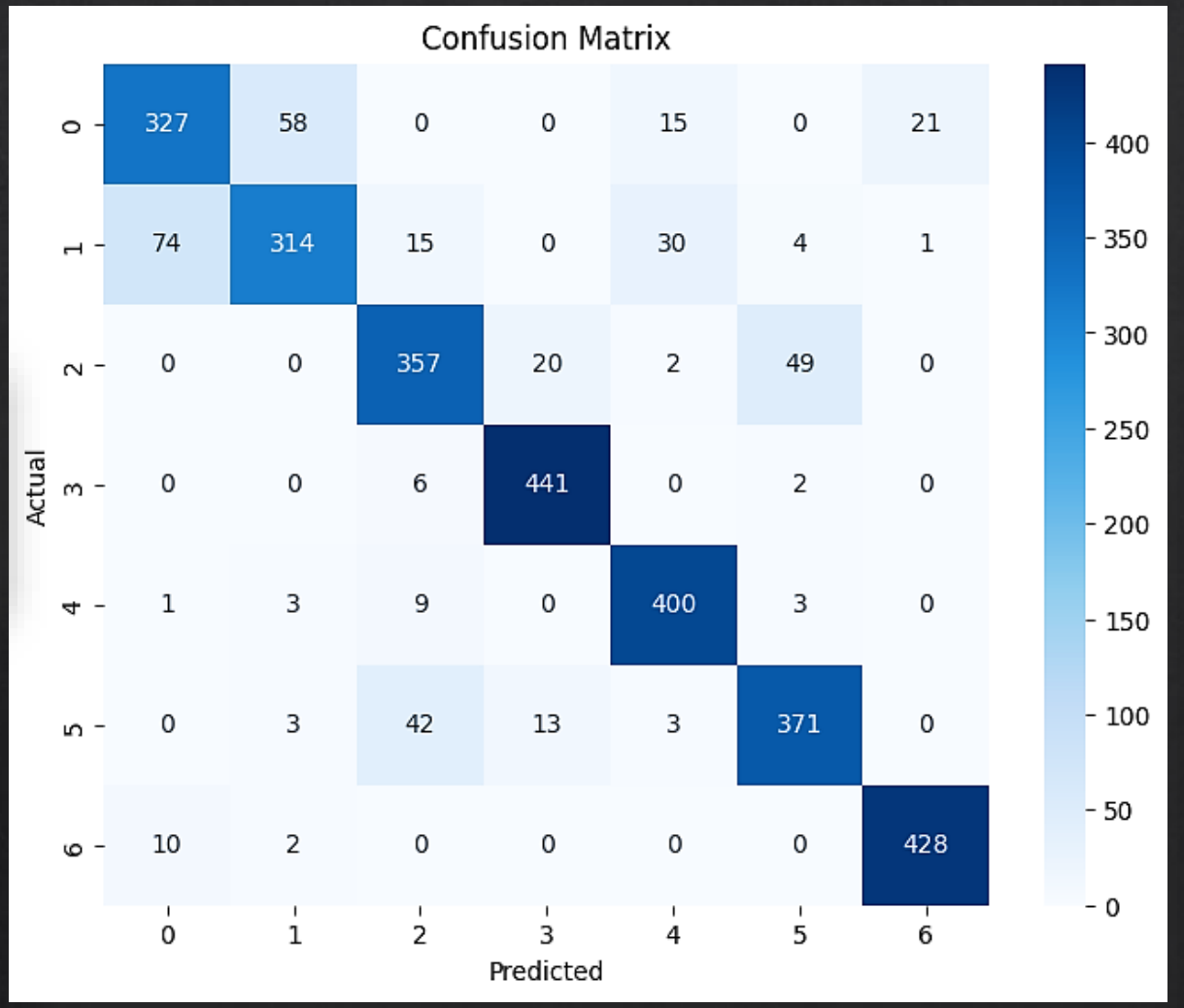
```
Accuracy: 0.8723544973544973

Classification Report:
              precision    recall  f1-score   support

           1       0.79      0.78      0.79       421
           2       0.83      0.72      0.77       438
           3       0.83      0.83      0.83       428
           4       0.93      0.98      0.96       449
           5       0.89      0.96      0.92       416
           6       0.86      0.86      0.86       432
           7       0.95      0.97      0.96       440

    accuracy                           0.87      3024
   macro avg       0.87      0.87      0.87      3024
weighted avg       0.87      0.87      0.87      3024
```

Accuracy – 87%

Visualization

Plot the - Confusion Matrix

```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



UI APP CODE

```python
import gradio as gr
import joblib
import numpy as np

model = joblib.load("forest_cover_model.pkl")

label_map = {
    1: "Spruce/Fir",
    2: "Lodgepole Pine",
    3: "Ponderosa Pine",
    4: "Cottonwood/Willow",
    5: "Aspen",
    6: "Douglas-fir",
    7: "Krummholz"
}

def predict_cover(elevation, aspect, slope, hor_dist_hydro, vert_dist_hydro,
                  hor_dist_road, hor_dist_fire, hillshade_9am, hillshade_noon, hillshade_3pm,
                  wilderness_choice, soil_choice):
```

```python
        ### Feature Information:
        - **Elevation**: Elevation in meters
        - **Aspect**: Aspect in degrees azimuth
        - **Slope**: Slope in degrees
        - **Horizontal/Vertical Distance to Hydrology**: Distance to nearest surface water features
        - **Horizontal Distance to Roadways**: Distance to nearest roadway
        - **Hillshade indices**: Hillshade index at 9am, noon, and 3pm on summer solstice
        - **Horizontal Distance to Fire Points**: Distance to nearest wildfire ignition points
        - **Wilderness Area**: 4 binary columns for wilderness area designation
        - **Soil Type**: 40 binary columns for soil type designation

        This application uses a Random Forest classifier trained on the forest cover type dataset.
        """)
        gr.Markdown("### Created by Aritra Mukherjee")
app.launch()
```

Prediction

Forest Cover Type Prediction

Predict the forest cover type for a 30m x 30m patch of land based on cartographic variables

Predict | About

Elevation (m) 2804
0 — 4000

Aspect (azimuth degrees) 139
0 — 360

Slope (degrees) 9
0 — 60

Horizontal Distance to Hydrology (m) 268
0 — 1500

Vertical Distance to Hydrology (m) 65
-500 — 700

Horizontal Distance to Roadways (m) 3180
0 — 8000

Hillshade 9am 234
0 — 255

Hillshade Noon 238
0 — 255

Hillshade 3pm 137
0 — 255

Horizontal Distance to Fire Points (m) 6121
0 — 8000

Wilderness Area (Select One)

- (•) Wilderness Area 1
- ( ) Wilderness Area 2
- ( ) Wilderness Area 3
- ( ) Wilderness Area 4

Soil Type (Select One)

( ) Soil Type 1	( ) Soil Type 2	( ) Soil Type 3	( ) Soil Type 4	( ) Soil Type 5	( ) Soil Type 6
( ) Soil Type 7	( ) Soil Type 8	( ) Soil Type 9	( ) Soil Type 10	( ) Soil Type 11	(•) Soil Type 12
( ) Soil Type 13	( ) Soil Type 14	( ) Soil Type 15	( ) Soil Type 16	( ) Soil Type 17	( ) Soil Type 18
( ) Soil Type 19	( ) Soil Type 20	( ) Soil Type 21	( ) Soil Type 22	( ) Soil Type 23	( ) Soil Type 24
( ) Soil Type 25	( ) Soil Type 26	( ) Soil Type 27	( ) Soil Type 28	( ) Soil Type 29	( ) Soil Type 30
( ) Soil Type 31	( ) Soil Type 32	( ) Soil Type 33	( ) Soil Type 34	( ) Soil Type 35	( ) Soil Type 36
( ) Soil Type 37	( ) Soil Type 38	( ) Soil Type 39	( ) Soil Type 40		

Predict Forest Cover Type

Prediction Result

Predicted Cover Type: Lodgepole Pine

Prediction

Forest Cover Type Prediction

Predict the forest cover type for a 30m x 30m patch of land based on cartographic variables

Predict | About

📘 About This Project

Forest Cover Type Prediction

This application predicts the type of forest cover using cartographic variables for a 30m x 30m patch of land in the Roosevelt National Forest of northern Colorado.

Forest Cover Types (target classes):

1. Spruce/Fir
2. Lodgepole Pine
3. Ponderosa Pine
4. Cottonwood/Willow
5. Aspen
6. Douglas-fir
7. Krummholz

Feature Information:

- **Elevation**: Elevation in meters
- **Aspect**: Aspect in degrees azimuth
- **Slope**: Slope in degrees
- **Horizontal/Vertical Distance to Hydrology**: Distance to nearest surface water features
- **Horizontal Distance to Roadways**: Distance to nearest roadway
- **Hillshade indices**: Hillshade index at 9am, noon, and 3pm on summer solstice
- **Horizontal Distance to Fire Points**: Distance to nearest wildfire ignition points
- **Wilderness Area**: 4 binary columns for wilderness area designation
- **Soil Type**: 40 binary columns for soil type designation

This application uses a Random Forest classifier trained on the forest cover type dataset.

Created by Aritra Mukherjee

GitHub Repo - Forest-Cover-Prediction-ml-app

Conclusion

- Successfully built a web app that predicts forest cover type using machine learning.
- Useful for forest conservation, geographical analysis, and environmental modeling.
- Scalable for further applications like wildfire risk prediction or ecological zoning.

Project - 3

Heart Disease Prediction Using Deep Learning

Presented by – Aritra Mukherjee

ID: UMIP270247

Machine Learning Intern
At - Unified Mentor Pvt. Limited

Introduction To Project – Heart Disease Prediction

This project aims to build an intelligent system capable of predicting the presence of heart disease in a patient based on clinical data.

It leverages modern machine learning and deep learning techniques, wrapped in a user-friendly interface developed using Streamlit.

Users can input medical parameters, and the model provides an instant prediction with confidence.

GOAL

- ✓ To predict whether a patient has heart disease using machine learning.
- ✓ Provide a simple and interactive tool for quick heart health assessment.
- ✓ Enhance early diagnosis and awareness through AI support.

Dataset Overview for Classification

- **Total Features**:
  11 input features + 1 target

- **Key Attributes**:
  - Age, Sex, Chest Pain Type
  - Resting Blood Pressure, Cholesterol
  - Fasting Blood Sugar, Resting ECG
  - Maximum Heart Rate, Exercise-Induced Angina
  - ST Depression (Oldpeak), ST Segment Slope
  - **Target**: 0 = Healthy, 1 = Heart Disease

Tech Stack

Technologies & Tools Used

- **Python**:
  Core language for implementation
- **Pandas, NumPy**:
  Data manipulation and preprocessing
- **Matplotlib, Seaborn**:
  Data visualization and analysis
- **Scikit-learn**:
  Feature scaling, model evaluation
- **TensorFlow / Keras**:
  Deep learning model (Sequential Neural Network)
- **Pickle & JSON**:
  Saving scaler and feature info
- **Streamlit**:
  Web-based user interface for prediction

Import Required Libraries

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pickle
import json
```

Explore The Dataset

```
df = pd.read_csv('dataset.csv')
```
✓ 0.1s

```
print('Null values:', df.isnull().sum().sum())
```
✓ 0.0s

```
Null values: 0
```

```
df.describe()
```
✓ 0.0s

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	ST slope	target
count	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000
mean	53.720168	0.763866	3.232773	132.153782	210.363866	0.213445	0.698319	139.732773	0.387395	0.922773	1.624370	0.528571
std	9.358203	0.424884	0.935480	18.368823	101.420489	0.409912	0.870359	25.517636	0.487360	1.086337	0.610459	0.499393
min	28.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	60.000000	0.000000	-2.600000	0.000000	0.000000
25%	47.000000	1.000000	3.000000	120.000000	188.000000	0.000000	0.000000	121.000000	0.000000	0.000000	1.000000	0.000000
50%	54.000000	1.000000	4.000000	130.000000	229.000000	0.000000	0.000000	140.500000	0.000000	0.600000	2.000000	1.000000
75%	60.000000	1.000000	4.000000	140.000000	269.750000	0.000000	2.000000	160.000000	1.000000	1.600000	2.000000	1.000000
max	77.000000	1.000000	4.000000	200.000000	603.000000	1.000000	2.000000	202.000000	1.000000	6.200000	3.000000	1.000000

```
df.info()
```
✓ 0.1s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1190 entries, 0 to 1189
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  1190 non-null   int64
 1   sex                  1190 non-null   int64
 2   chest pain type      1190 non-null   int64
 3   resting bp s         1190 non-null   int64
 4   cholesterol          1190 non-null   int64
 5   fasting blood sugar  1190 non-null   int64
 6   resting ecg          1190 non-null   int64
 7   max heart rate       1190 non-null   int64
 8   exercise angina      1190 non-null   int64
 9   oldpeak              1190 non-null   float64
 10  ST slope             1190 non-null   int64
 11  target               1190 non-null   int64
dtypes: float64(1), int64(11)
memory usage: 111.7 KB
```

Preprocessing (StandardScaler), model building (train_test_split)

```
X = df.drop(columns=['target'])
y = df['target']
```

✓ 0.0s

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

✓ 0.1s

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

✓ 0.0s

Model Saving

```
with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)
```

✓ 0.1s

```
feature_names = list(X.columns)
with open('feature_names.json', 'w') as f:
    json.dump(feature_names, f)
```

✓ 0.0s

Scaler saved with pickle as - scaler.pkl
Feature list saved as - feature_names.json

Sequential Model Configuration

```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])
```

✓ 0.0s

```
▷Initialize Reactive Jupyter | Sync all Stale code
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

✓ 0.0s

Train the Model

```
▷Initialize Reactive Jupyter | Sync all Stale code
history = model.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), epochs=50, batch_size=16)
✓ 9.4s
```

```
Epoch 1/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.5825 - loss: 0.6773 - val_accuracy: 0.8361 - val_loss: 0.4561
Epoch 2/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7921 - loss: 0.4730 - val_accuracy: 0.8403 - val_loss: 0.3891
Epoch 3/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8078 - loss: 0.4556 - val_accuracy: 0.8529 - val_loss: 0.3780
Epoch 4/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7985 - loss: 0.4219 - val_accuracy: 0.8487 - val_loss: 0.3692
Epoch 5/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8114 - loss: 0.4201 - val_accuracy: 0.8403 - val_loss: 0.3623
Epoch 6/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8274 - loss: 0.4088 - val_accuracy: 0.8487 - val_loss: 0.3577
Epoch 7/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8340 - loss: 0.3843 - val_accuracy: 0.8529 - val_loss: 0.3533
Epoch 8/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8375 - loss: 0.3555 - val_accuracy: 0.8571 - val_loss: 0.3477
Epoch 9/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8404 - loss: 0.3807 - val_accuracy: 0.8613 - val_loss: 0.3438
Epoch 10/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8493 - loss: 0.3684 - val_accuracy: 0.8697 - val_loss: 0.3413
```

• • •

```
Epoch 49/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8967 - loss: 0.2602 - val_accuracy: 0.8824 - val_loss: 0.3077
Epoch 50/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8800 - loss: 0.3040 - val_accuracy: 0.8782 - val_loss: 0.3082
```

Evaluate the Model

```
loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

✓ 0.0s

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8796 - loss: 0.3232
Test Accuracy: 87.82%
```

Accuracy – 88%

Save the Trained Model

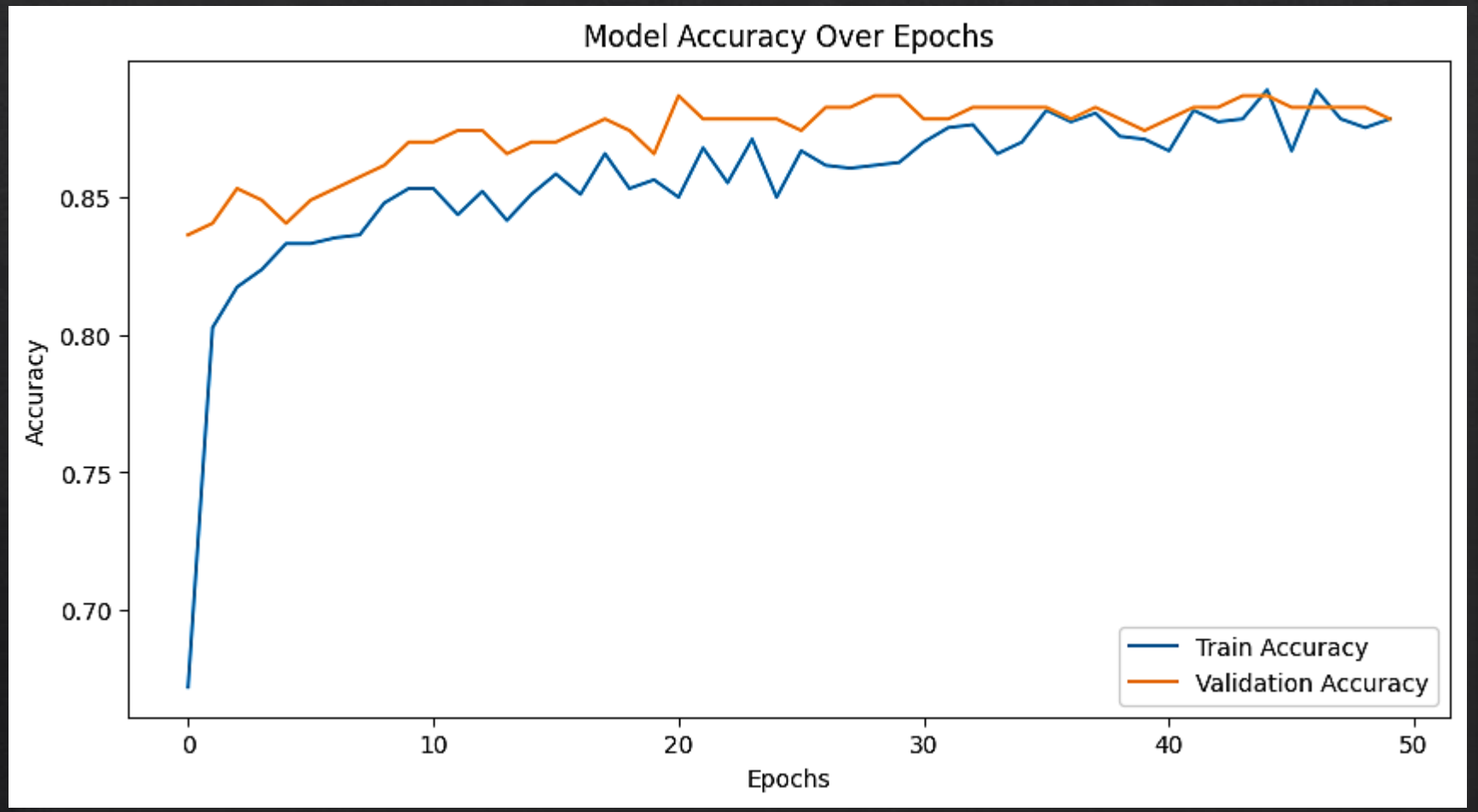
```
model.save('heart_disease_model.h5')
```

✓ 0.1s

Visualization

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy Over Epochs')
plt.show()
```

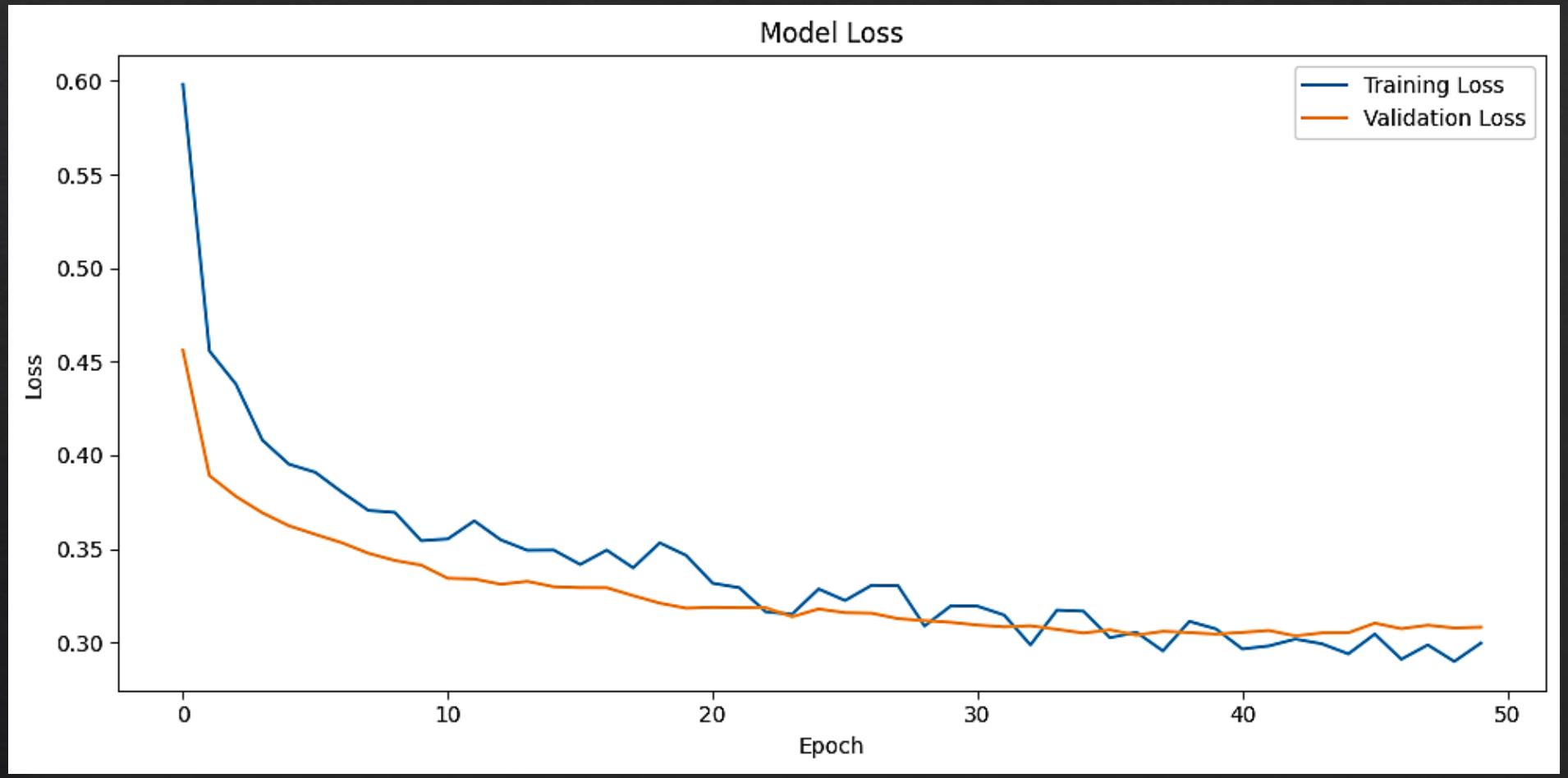
✓ 0.6s



Visualization

```
plt.figure(figsize=(10,5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

✓ 0.1s

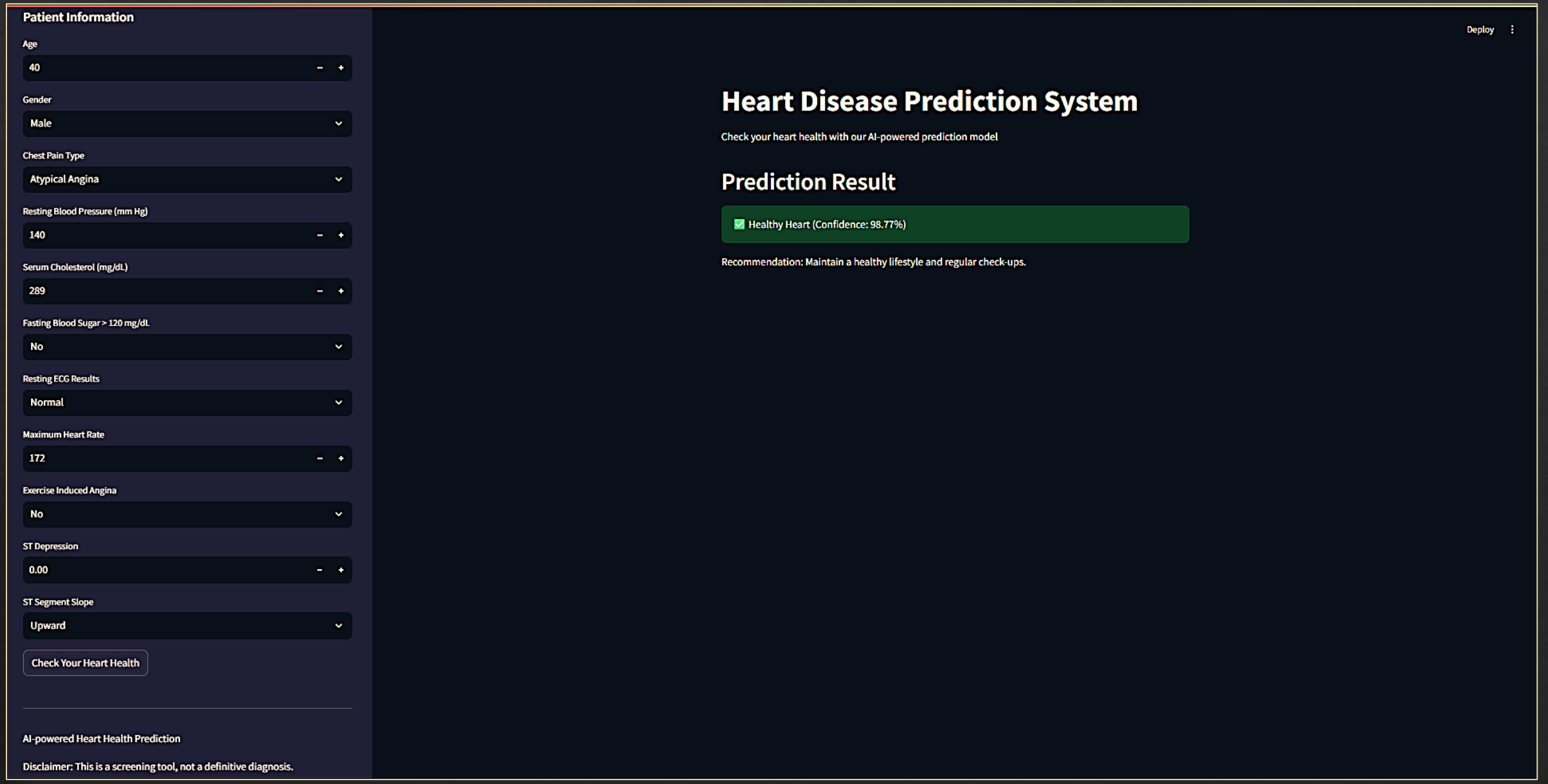


Frontend with Streamlit

```python
import streamlit as st
import pandas as pd
import numpy as np
import pickle
import json
from tensorflow.keras.models import load_model

# Page Configuration
st.set_page_config(page_title="Heart Disease Prediction", page_icon="❤️")

# Title
st.title("Heart Disease Prediction System")
st.write("Check your heart health with our AI-powered prediction model")

# Load the pre-trained model, scaler, and feature names
@st.cache_resource
def load_model_and_scaler():
    model = load_model('heart_disease_model.h5')

    with open('scaler.pkl', 'rb') as f:
        scaler = pickle.load(f)

    with open('feature_names.json', 'r') as f:
        feature_names = json.load(f)

    return model, scaler, feature_names

model, scaler, feature_names = load_model_and_scaler()
```

```python
    input_encoded = input_data.reindex(columns=feature_names, fill_value=0)
    input_scaled = scaler.transform(input_encoded)
    prediction = model.predict(input_scaled)
    probability = prediction[0][0]
    result = "Potential Heart Disease" if probability > 0.5 else "Healthy Heart"
    confidence = probability * 100 if probability > 0.5 else (1 - probability) * 100

    st.header("Prediction Result")
    if result == "Potential Heart Disease":
        st.error(f"⚠️ {result} (Confidence: {confidence:.2f}%)")
        st.write("Recommendation: Please consult with a healthcare professional.")
    else:
        st.success(f"✅ {result} (Confidence: {confidence:.2f}%)")
        st.write("Recommendation: Maintain a healthy lifestyle and regular check-ups.")

st.sidebar.markdown("---")
st.sidebar.write("AI-powered Heart Health Prediction")
st.sidebar.write("Disclaimer: This is a screening tool, not a definitive diagnosis.")
```

Prediction



GitHub Repo- Ai-Heart-Health

Conclusion

Developed an AI-driven heart disease prediction system.

Achieved accurate classification using deep learning.

Scalable and interactive via Streamlit for real-world usability.

Supports early screening and promotes health awareness.

Project - 4

Mobile Phone Price Predictor using Machine Learning

Presented by – Aritra Mukherjee

ID: UMIP270247

Machine Learning Intern
At - Unified Mentor Pvt. Limited

Introduction To Project – Phone Price Predictor

In today's competitive smartphone market, price estimation based on specifications helps manufacturers, retailers, and consumers make informed decisions.

This project aims to predict the price range of mobile phones using machine learning, based on features like RAM, battery, screen size, camera specs, etc.

A user-friendly Streamlit web application allows interactive predictions.

GOAL

Primary Objective:

To build a machine learning model that can accurately classify a smartphone into one of four price categories:

- Low (₹5,000–12,000)
- Medium (₹12,000–20,000)
- High (₹20,000–35,000)
- Very High (₹35,000+)

Secondary Goals:

- Create a clean and responsive web app for predictions.
- Visualize feature importance to understand key factors affecting price.

Dataset Overview

Dataset Details:

- Rows: 2000
- Target: price range (0–3)

Features: 20+

- RAM, internal memory, processor speed, battery power
- Camera specs (front & rear)
- Screen size & resolution
- Connectivity features (Bluetooth, Wi-Fi, 3G, 4G, Dual SIM)

Tech Stack

Technologies & Tools Used

Languages & Tools:

- Python – data processing, modelling
- Streamlit – web UI for predictions
- Pandas / NumPy – data handling
- Matplotlib / Seaborn – visualizations
- Scikit-learn – ML model, preprocessing
- Pickle – saving and loading trained models

Model Used:

- Random Forest Classifier
- Accuracy and feature importance indicate a tree-based approach

Load the Dataset

```
▷Initialize Reactive Jupyter | Sync all Stale code
df = pd.read_csv('dataset.csv')
df.head()
```

✓ 0.0s

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	wifi	price_range
0	842	0	2.2	0	1	0	7	0.6	188	2	...	1	1
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	0	2
2	563	1	0.5	1	2	1	41	0.9	145	5	...	0	2
3	615	1	2.5	0	0	0	10	0.8	131	6	...	0	2
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	0	1

Data Exploration and Analysis

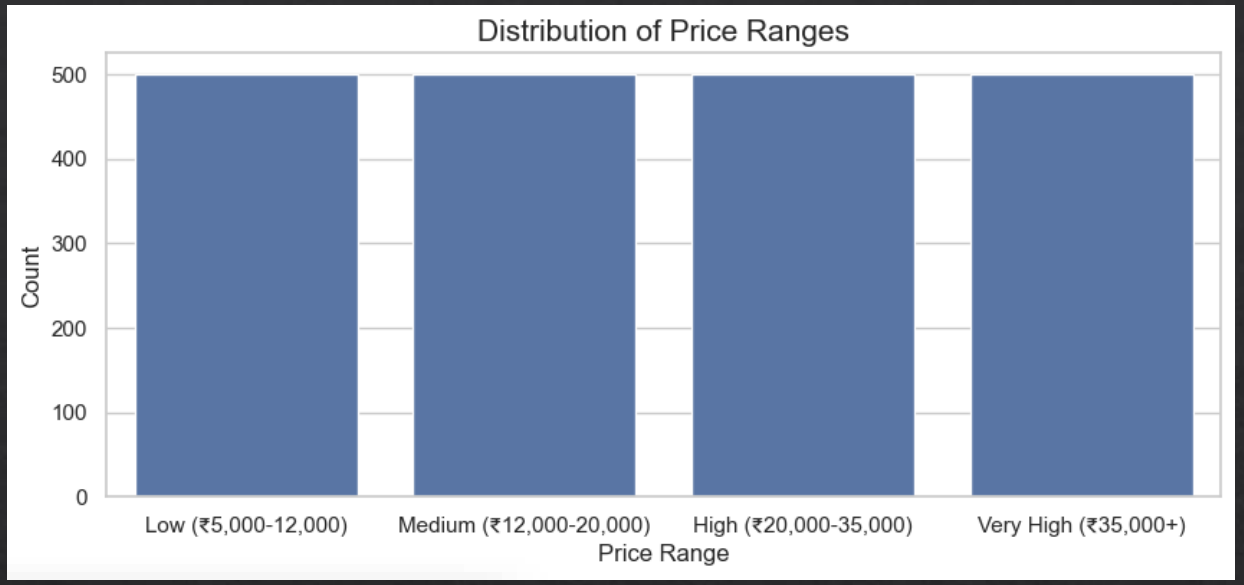
```
▷Initialize Reactive Jupyter | Sync all Stale code
print("Dataset shape:", df.shape)
print("\nDataset info:")
df.info()
print("\nSummary statistics:")
df.describe()
```

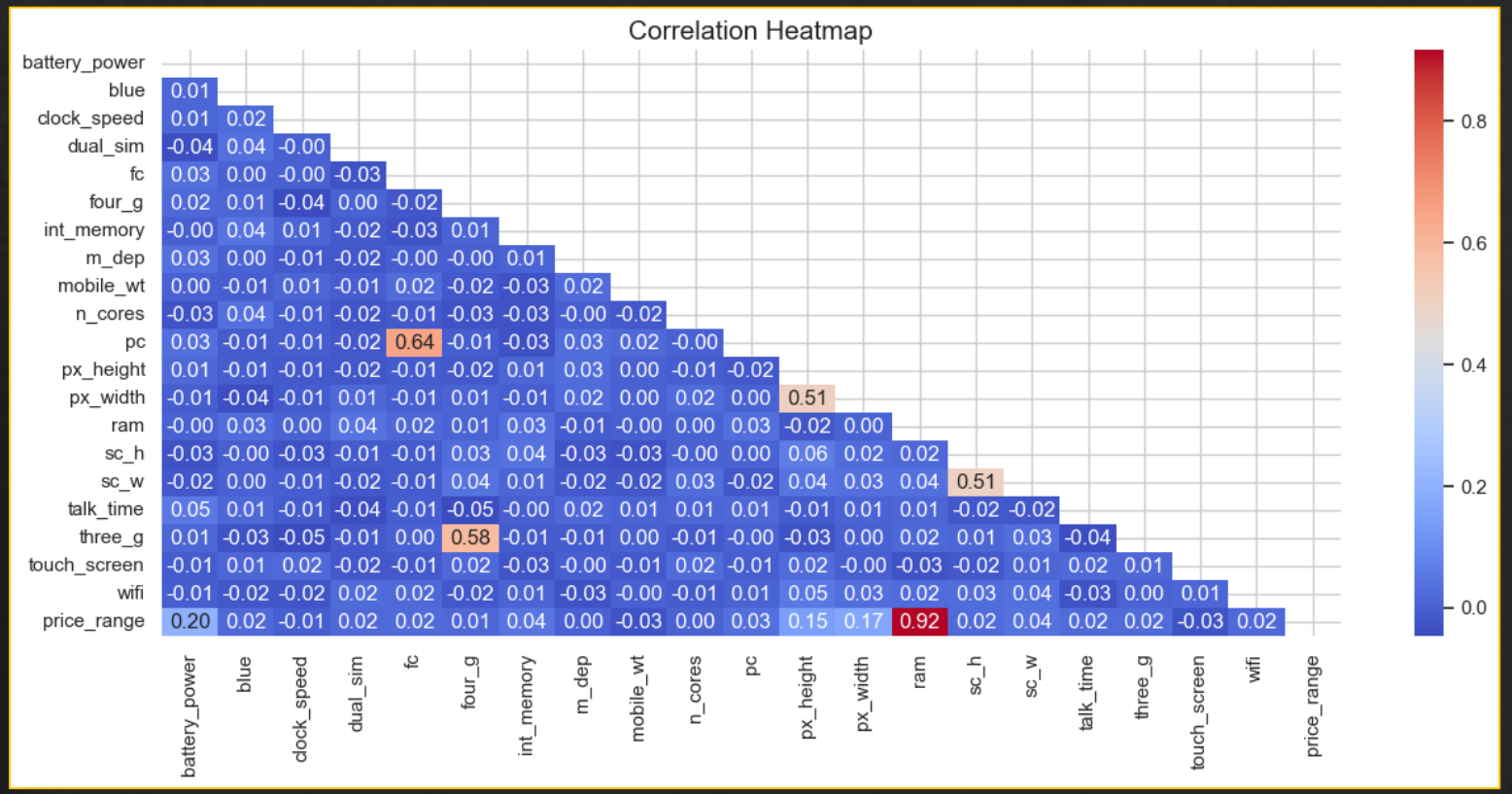
Summary statistics:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	...
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	4.520500	...
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	...
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	...
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...

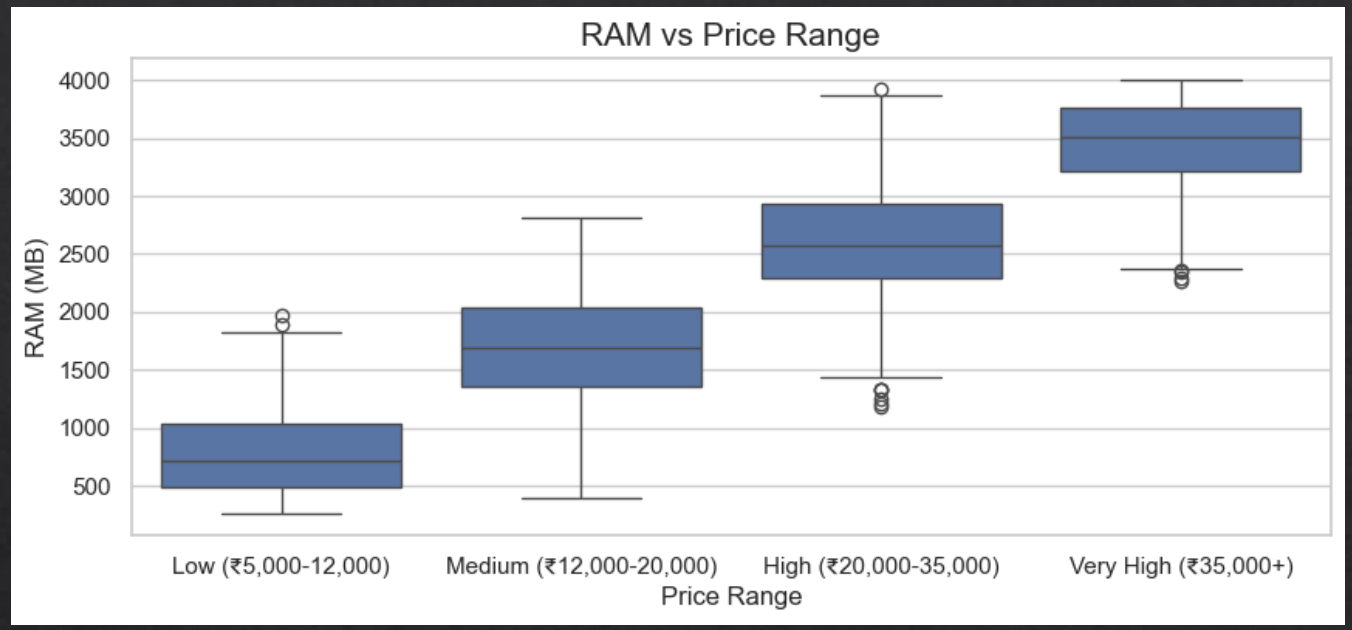
8 rows × 21 columns

Distribution of Price Ranges



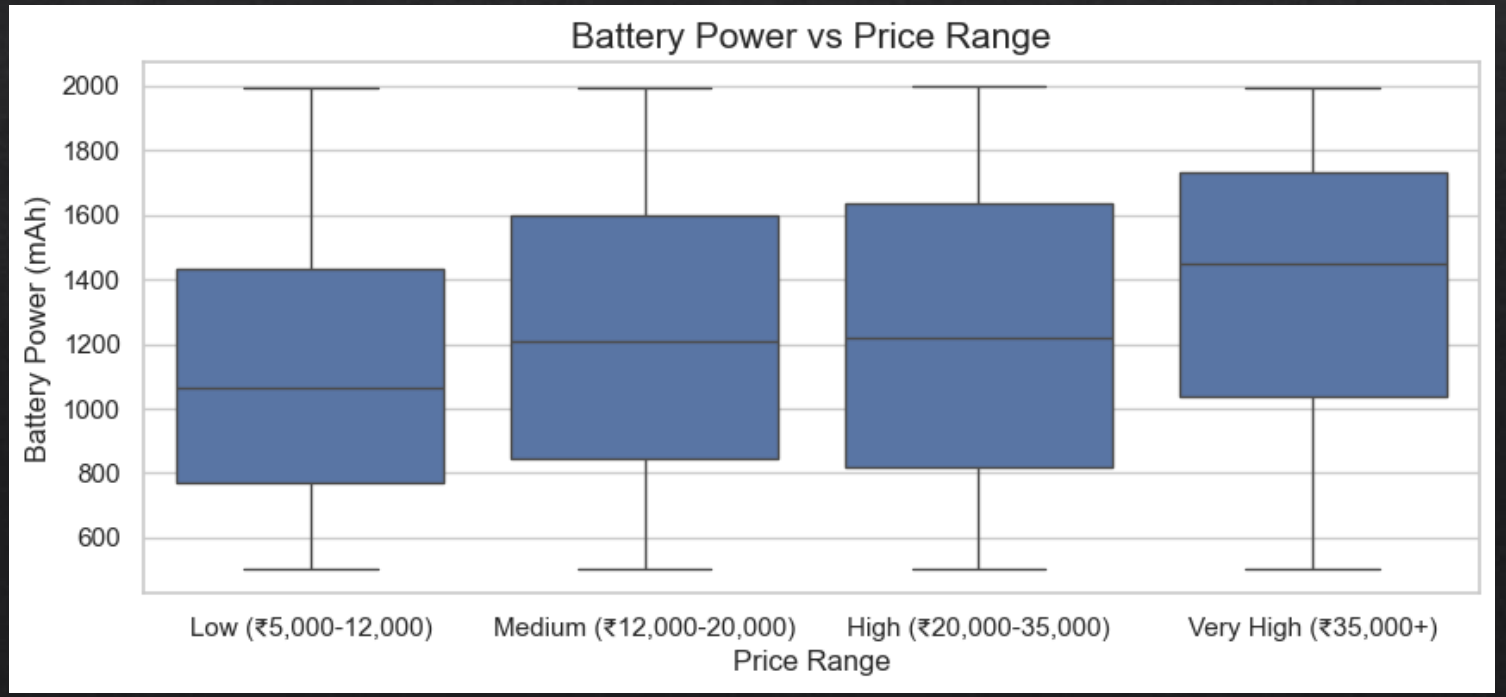


Correlation Heatmap

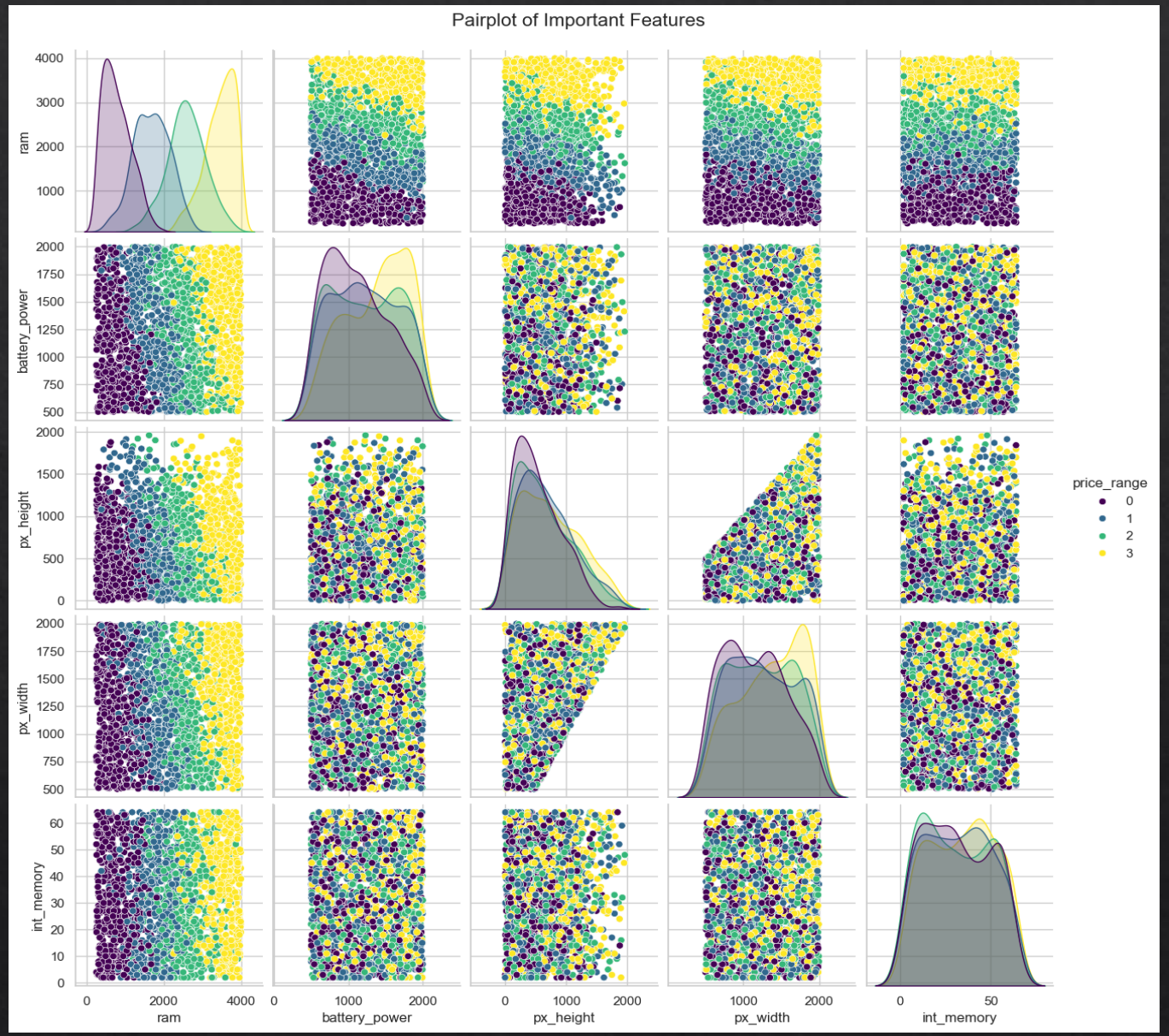


RAM vs Price Range

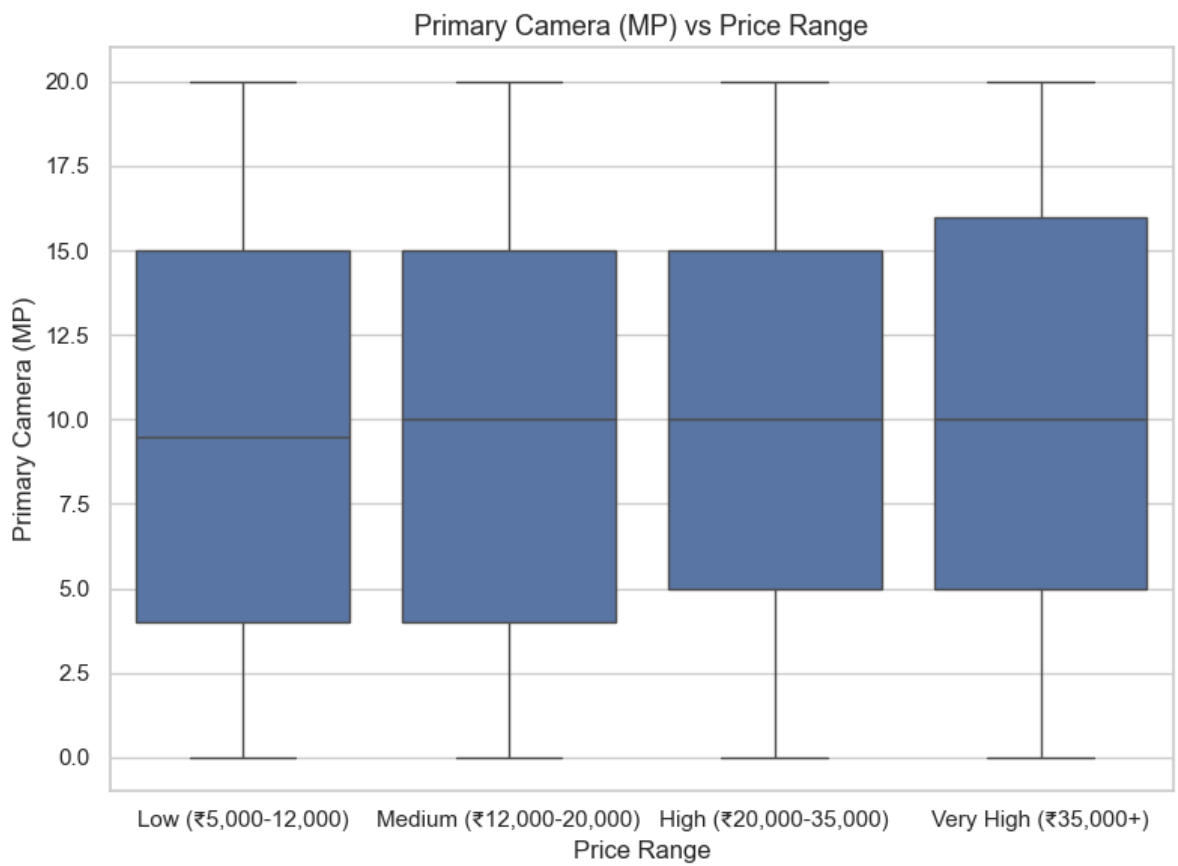
Battery Power vs Price Range

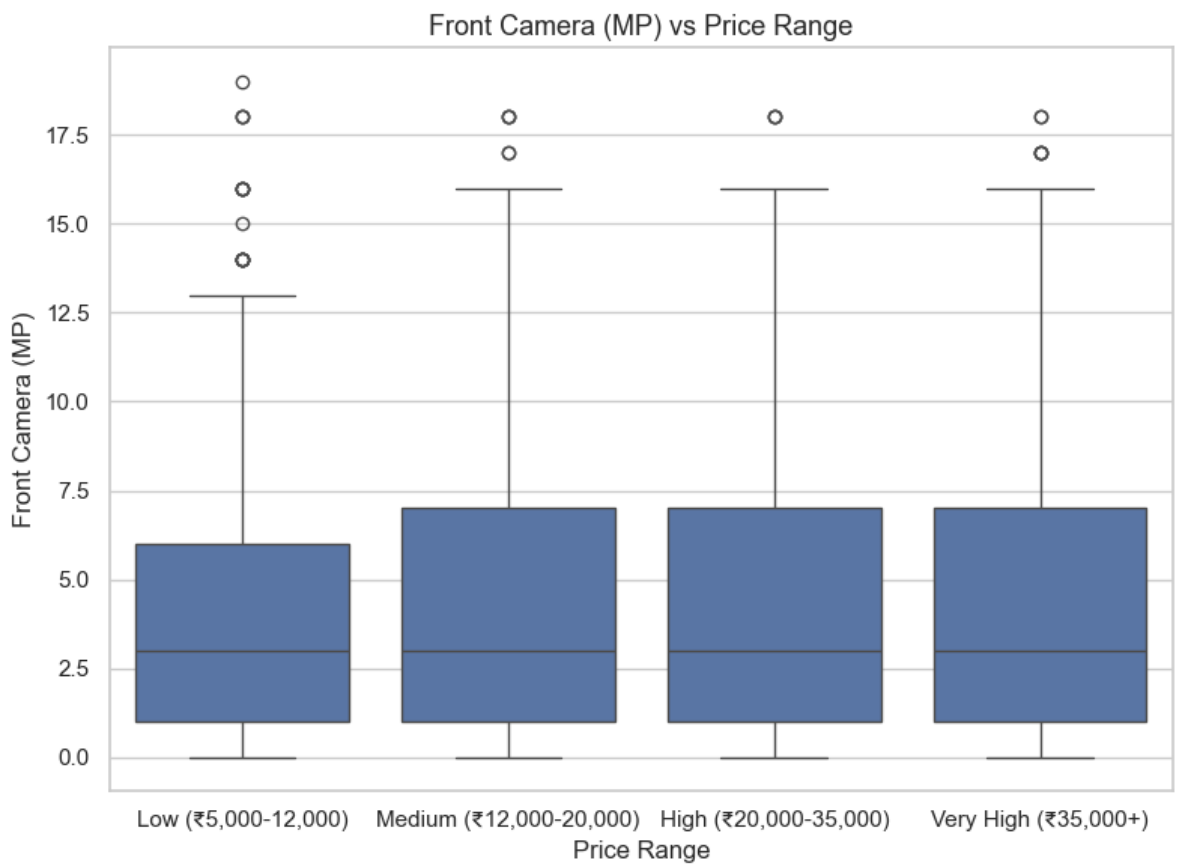


Pair plot of Important Features



The Price Different b/w Primary Camera (MP) & Front Camera(FC)





Feature Selection and Preprocessing

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

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

print(f"Training set shape: {X_train.shape}")
print(f"Testing set shape: {X_test.shape}")
```

```
Training set shape: (1600, 22)
Testing set shape: (400, 22)
```

```python
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


with open('scaler.pkl', 'wb') as file:
    pickle.dump(scaler, file)
```

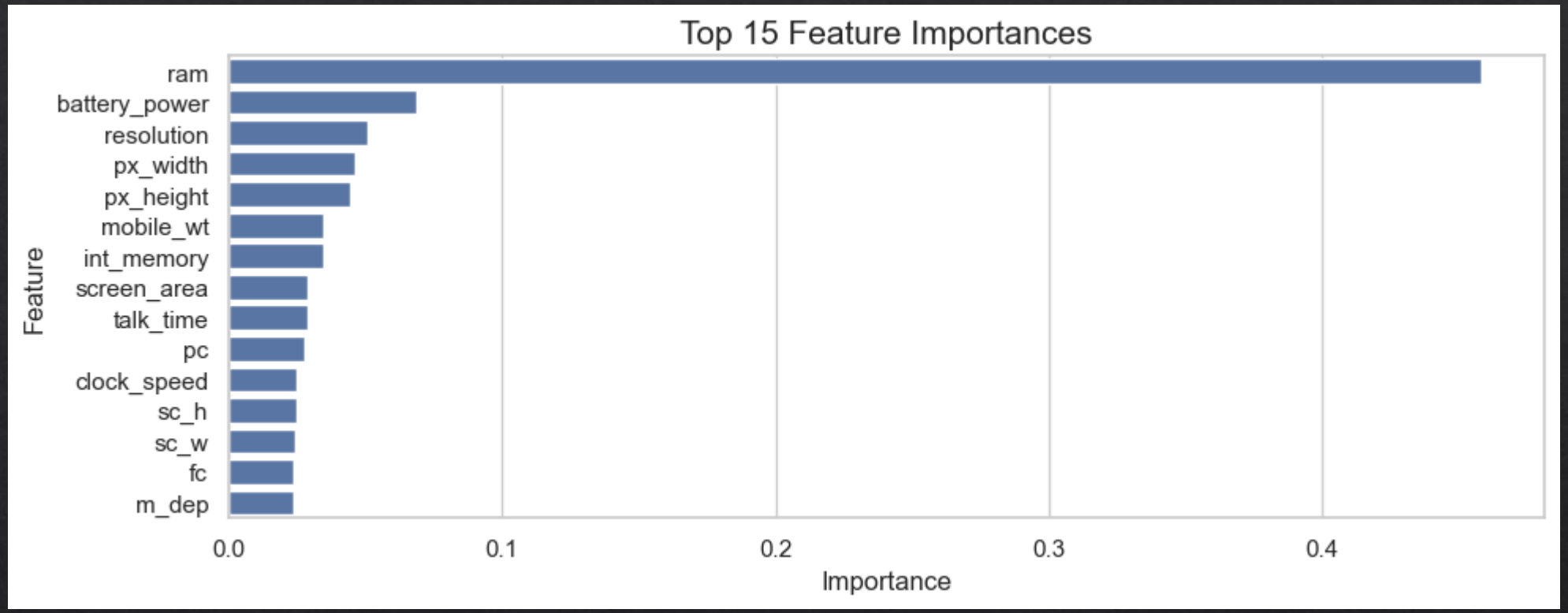
Model Training and Evaluation

```python
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

y_pred = rf_model.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

plt.figure(figsize=(10, 4))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix', fontsize=15)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(ticks=[0.5, 1.5, 2.5, 3.5], labels=[price_ranges[i] for i in range(4)])
plt.yticks(ticks=[0.5, 1.5, 2.5, 3.5], labels=[price_ranges[i] for i in range(4)])
plt.show()
```

```
Accuracy: 0.8975

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.94       105
           1       0.86      0.88      0.87        91
           2       0.83      0.87      0.85        92
           3       0.95      0.89      0.92       112

    accuracy                           0.90       400
   macro avg       0.89      0.90      0.90       400
weighted avg       0.90      0.90      0.90       400
```

Accuracy – 89.75%

Top 15 Feature Importances

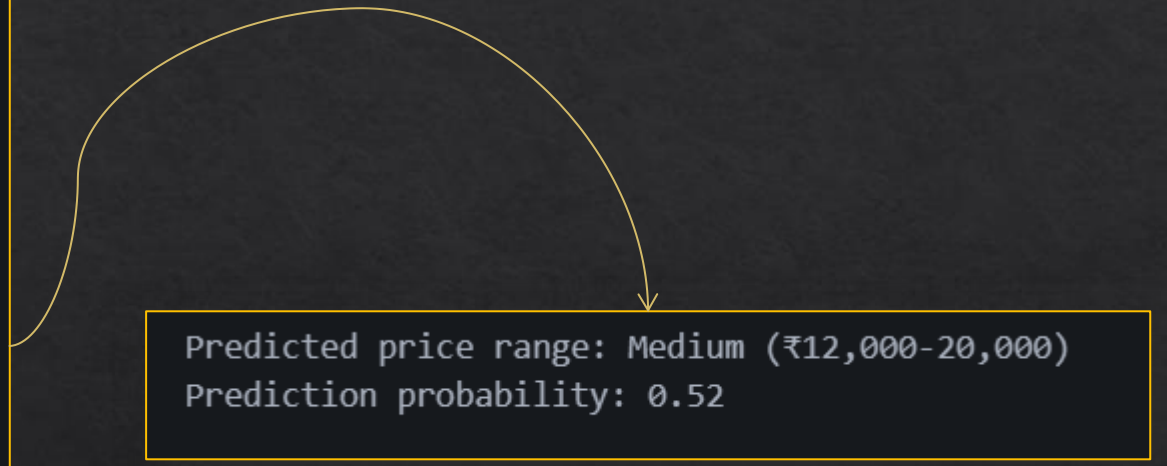


Save the Trained Model

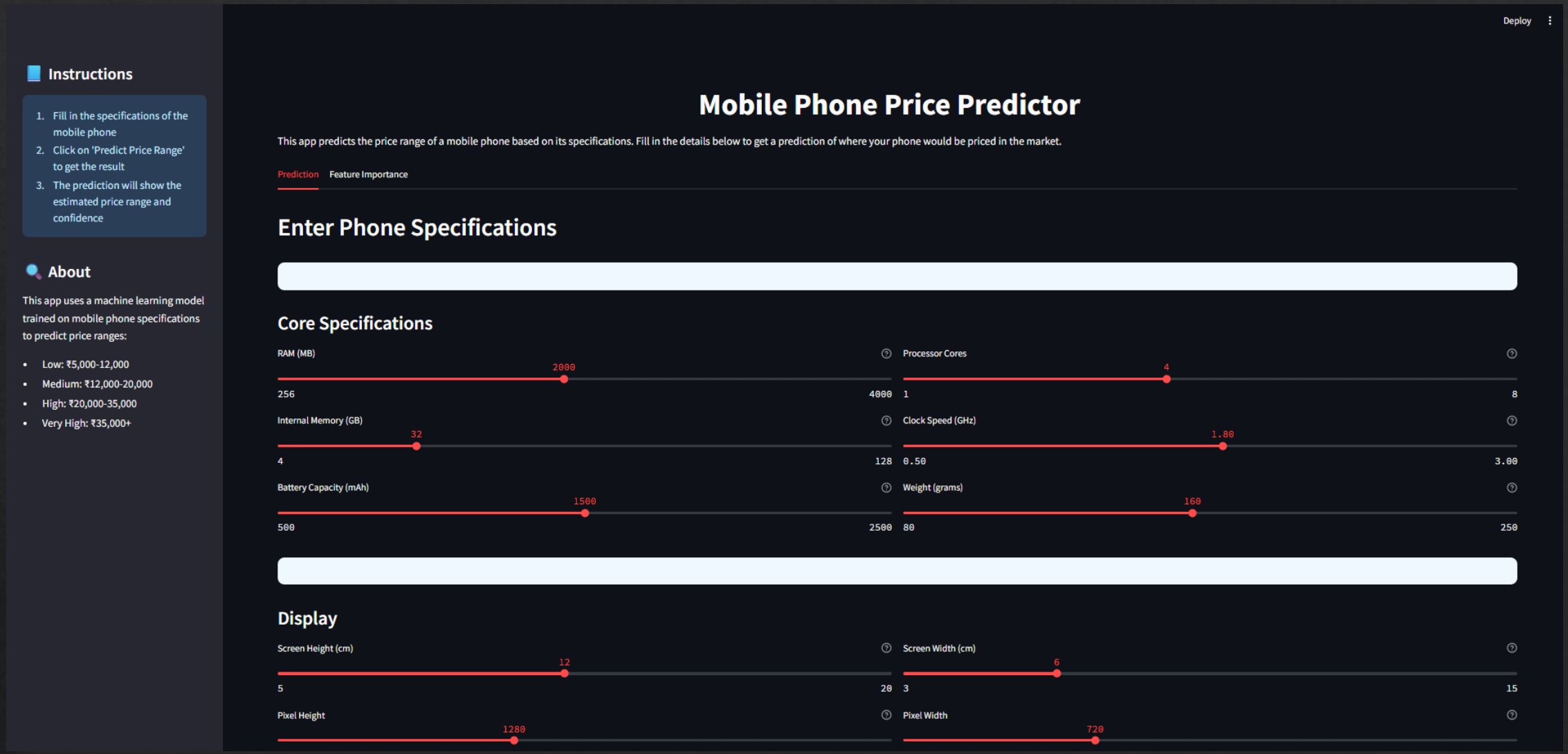
```
Model and feature names saved successfully!
```

Model Testing with Sample Data

```python
budget_phone = {
    'battery_power': 1000,
    'blue': 1,
    'clock_speed': 1.2,
    'dual_sim': 1,
    'fc': 5,
    'four_g': 0,
    'int_memory': 16,
    'm_dep': 0.7,
    'mobile_wt': 150,
    'n_cores': 4,
    'pc': 8,
    'px_height': 1280,
    'px_width': 720,
    'ram': 1500,
    'sc_h': 12,
    'sc_w': 6,
    'talk_time': 8,
    'three_g': 1,
    'touch_screen': 1,
    'wifi': 1,
    'screen_area': 12 * 6,
    'resolution': 1280 * 720
}

prediction, probability, price_range = predict_price_range(budget_phone)
print(f"Predicted price range: {price_range}")
print(f"Prediction probability: {probability:.2f}")
```



Prediction



Prediction

Camera

Primary Camera (MP)

12

0 20

Front Camera (MP)

5

0 20

Connectivity & Other Features

Bluetooth

No

WiFi

No

3G

No

4G

No

Dual SIM

No

Talk Time (hours)

12

2 24

Predict Price Range

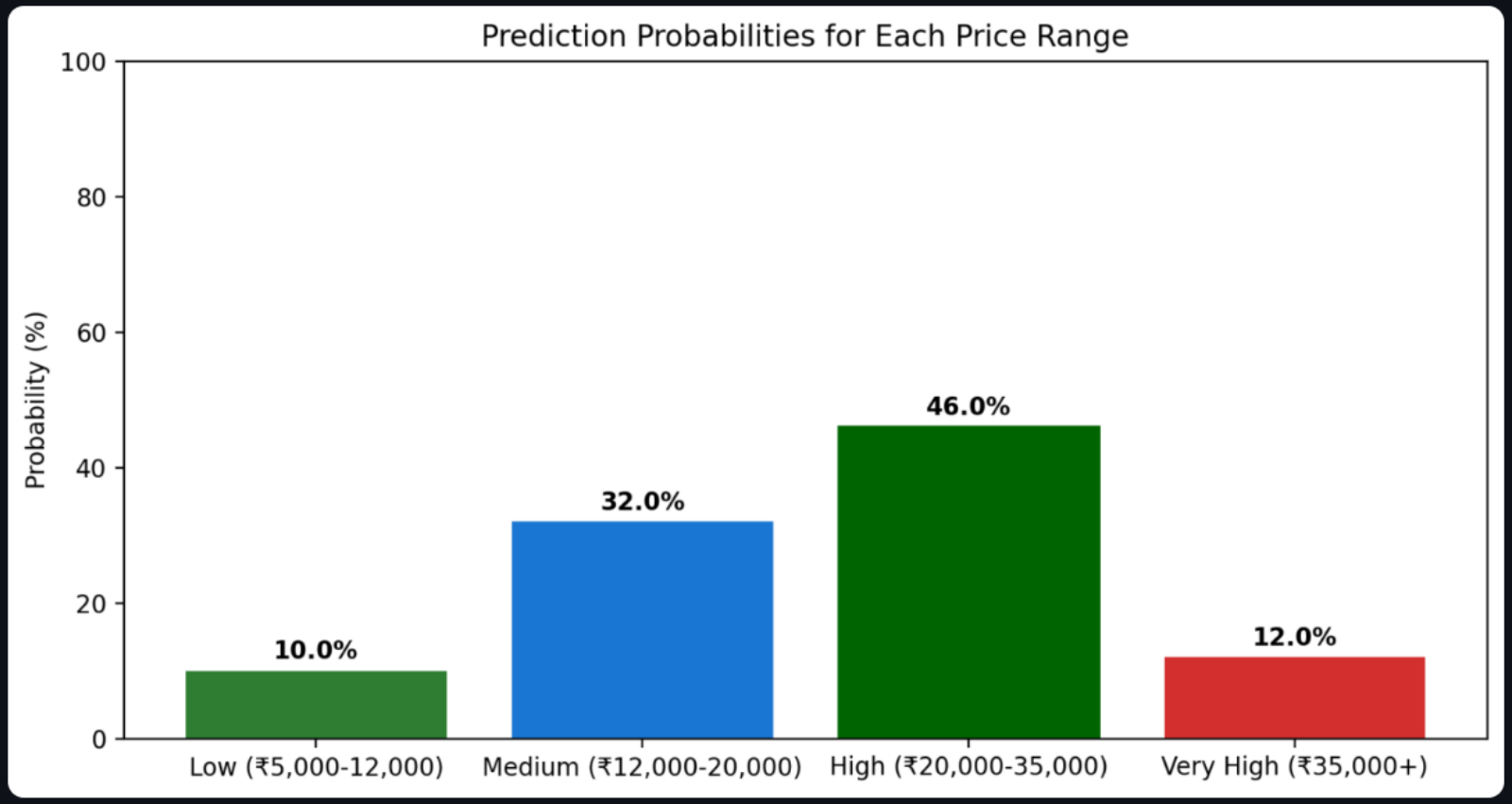
Prediction Result

Price Range: High (₹20,000-35,000)

Confidence: **46.0%**

Prediction

Confidence: **46.0%**



Mobile Price Prediction App | Created with Aritra Mukherjee.

GitHub Repo Link

GitHub Repo- Mobile-Price-Predictor-ML-App

Conclusion

Successfully developed an interactive ML-based app for smartphone price prediction.

Achieved clear classification of phones into pricing tiers based on hardware features.

Enables practical use in e-commerce, retail analysis, or product planning.

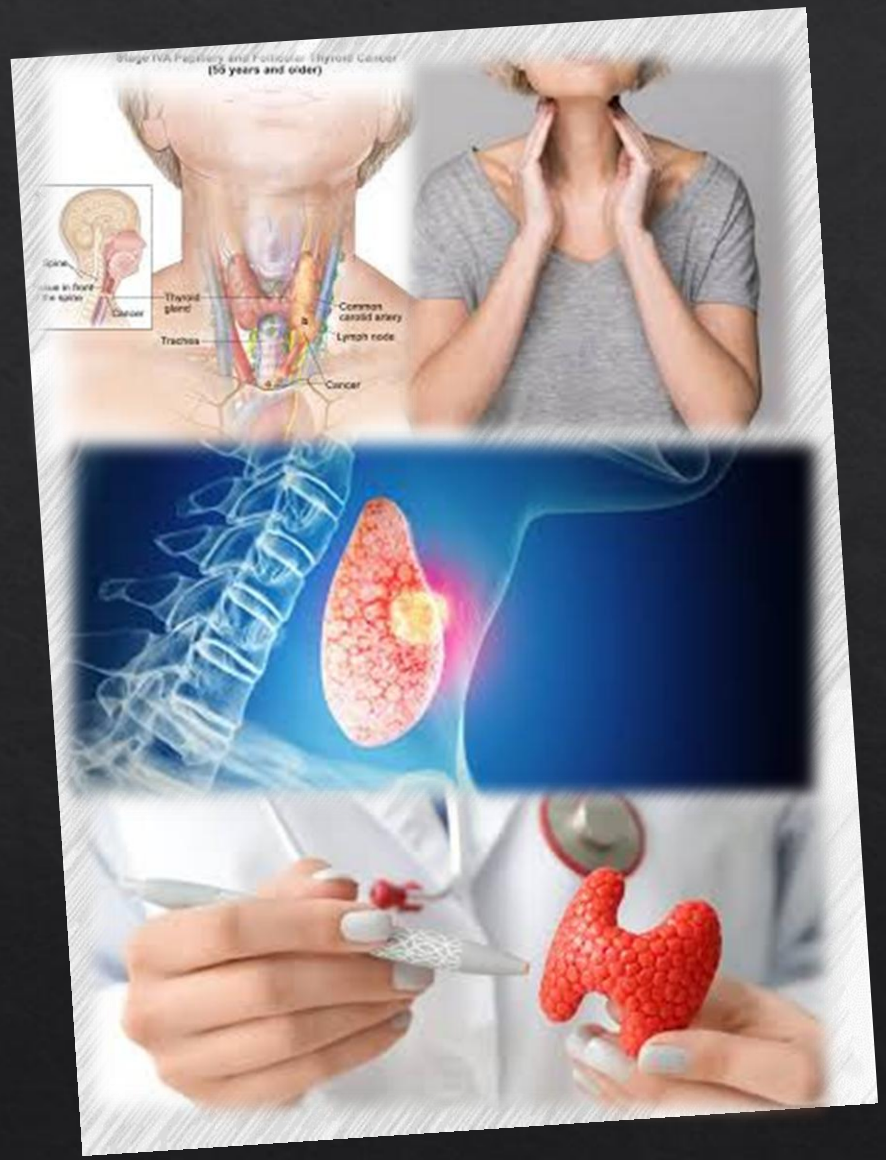
Project - 5

Thyroid Cancer Recurrence Prediction using Deep Learning"

Presented by – Aritra Mukherjee

ID: UMIP270247

Machine Learning Intern
At - Unified Mentor Pvt. Limited

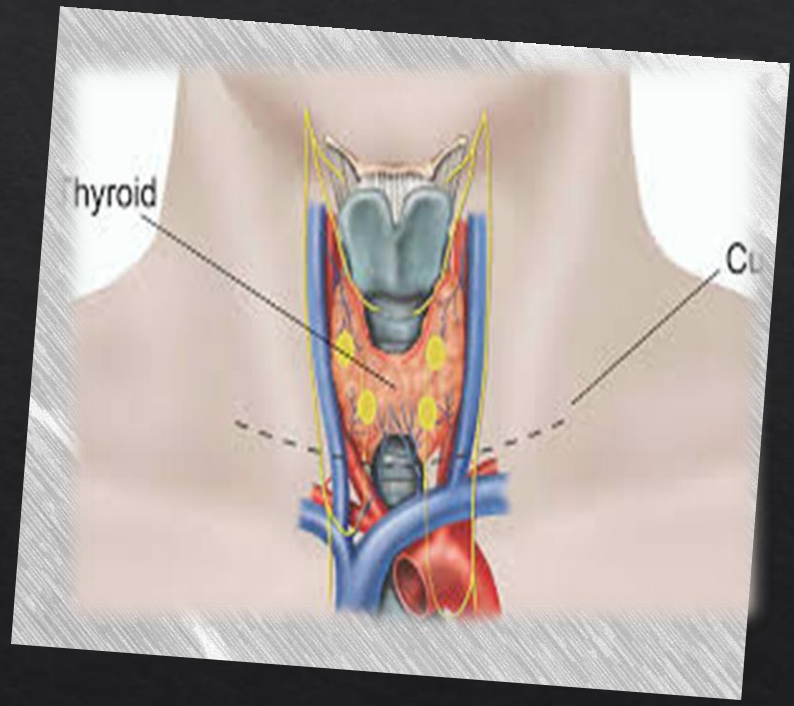


Introduction To Project – Thyroid Cancer

- Thyroid cancer is a significant health issue worldwide.
- Predicting recurrence can help doctors design better treatment plans.
- Our solution uses machine learning to support early prediction based on patient data.

GOAL

- ✓ To build an AI-based system that predicts the likelihood of thyroid cancer recurrence.
- ✓ Design an end-to-end ML pipeline (from preprocessing to prediction).
- ✓ Deploy the model using Streamlit for interactive usage by clinicians or researchers.



Dataset Overview for Classification

Collected from a historical thyroid cancer patient dataset – Given By Company Unified Mentor.

Target Variable:

- Recurrence **(Yes / No)**

Features:

- Demographic: Age, Gender
- Lifestyle & History: Smoking, Radiotherapy
- Clinical: Pathology, Thyroid Function, TNM staging, etc.

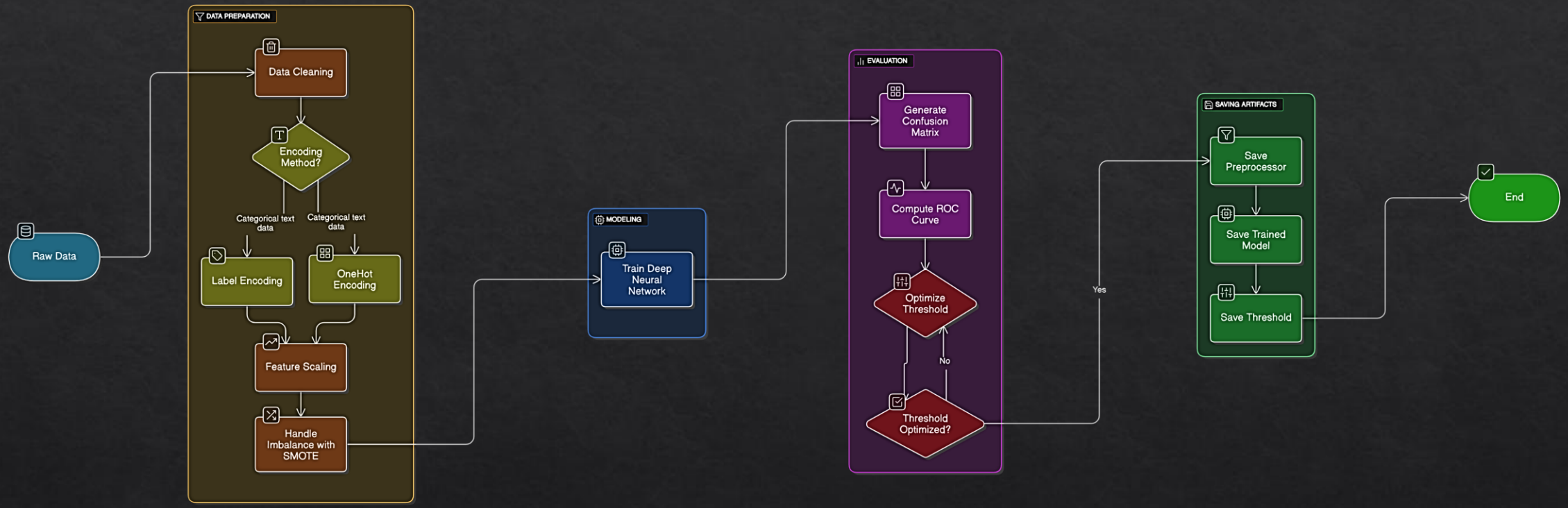
Class Imbalance Present (handled using SMOTE)

Tech Stack

Technologies & Tools Used

Layer	Tools Used
Language	Python
Data Handling	Pandas, NumPy
ML / DL	Scikit-learn, TensorFlow / Keras
Preprocessing	Column Transformer, Pipeline, SMOTE
Deployment	Streamlit
Model Storage	Pickle (.pkl), Keras (.h5)

Workflow Pipeline



Load Dataset and Display Basic Info

```
▷Initialize Reactive Jupyter | Sync all Stale code
data = pd.read_csv("dataset.csv")

print("Dataset Shape:", data.shape)
print("\nFirst 5 rows of the dataset:")
display(data.head())
print("\nData types:")
print(data.dtypes)
print("\nMissing values:")
print(data.isnull().sum())
```

✓ 0.2s

```
Dataset Shape: (383, 17)

First 5 rows of the dataset:
```

	Age	Gender	Smoking	Hx Smoking	Hx Radiothreapy	Thyroid Function	Physical Examination	Adenopathy	Pathology	Focality	Risk	T	N	M	Stage	Response	Recurred
0	27	F	No	No	No	Euthyroid	Single nodular goiter-left	No	Micropapillary	Uni-Focal	Low	T1a	N0	M0	I	Indeterminate	No
1	34	F	No	Yes	No	Euthyroid	Multinodular goiter	No	Micropapillary	Uni-Focal	Low	T1a	N0	M0	I	Excellent	No
2	30	F	No	No	No	Euthyroid	Single nodular goiter-right	No	Micropapillary	Uni-Focal	Low	T1a	N0	M0	I	Excellent	No
3	62	F	No	No	No	Euthyroid	Single nodular goiter-right	No	Micropapillary	Uni-Focal	Low	T1a	N0	M0	I	Excellent	No
4	62	F	No	No	No	Euthyroid	Multinodular goiter	No	Micropapillary	Multi-Focal	Low	T1a	N0	M0	I	Excellent	No

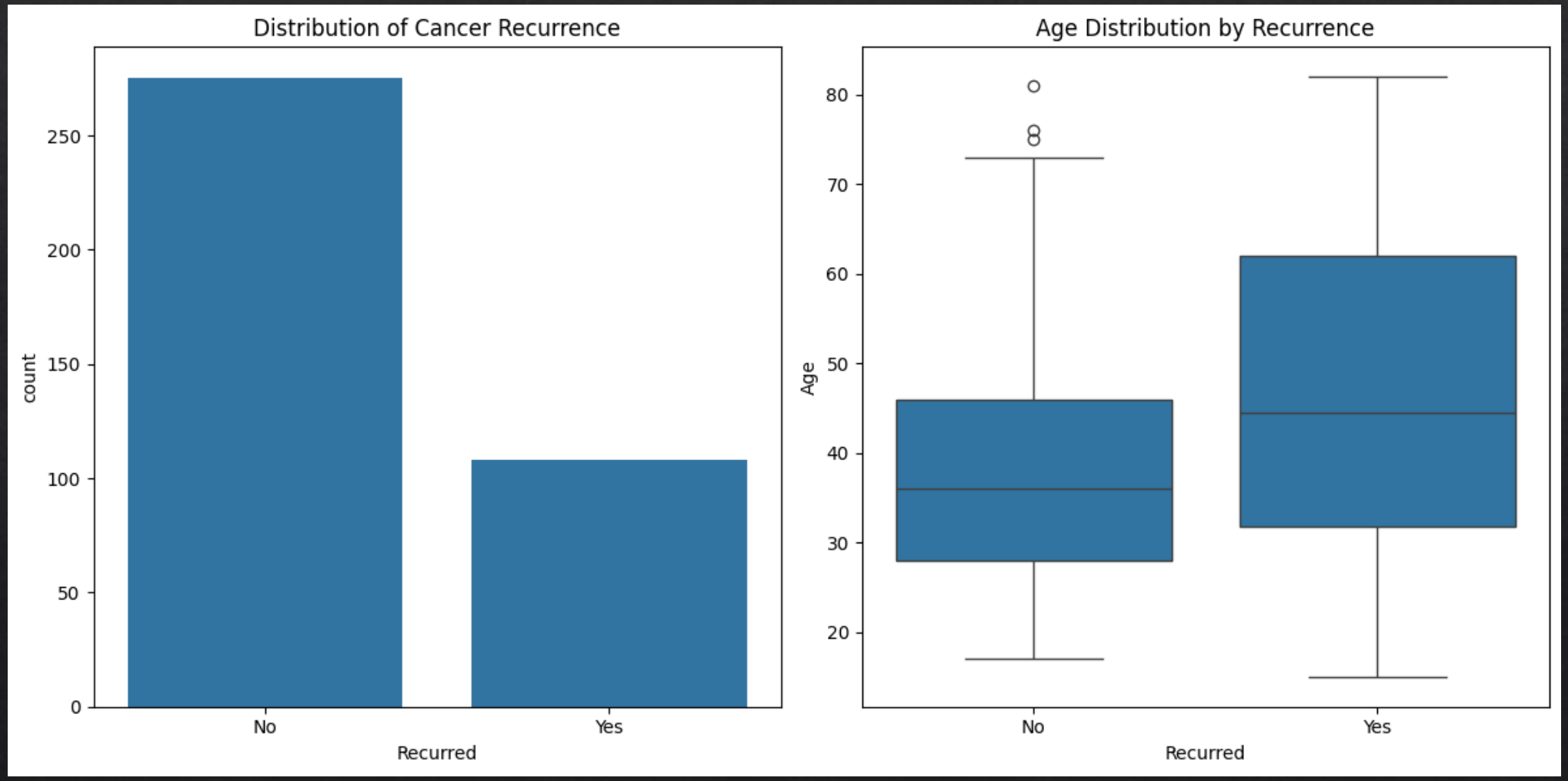
Data Preprocessing

```
if data['Age'].dtype == 'object':
    data['Age'] = pd.to_numeric(data['Age'], errors='coerce')

data = data.fillna(data.mode().iloc[0])

print("\nTarget variable distribution:")
print(data['Recurred'].value_counts())
print(data['Recurred'].value_counts(normalize=True))
```

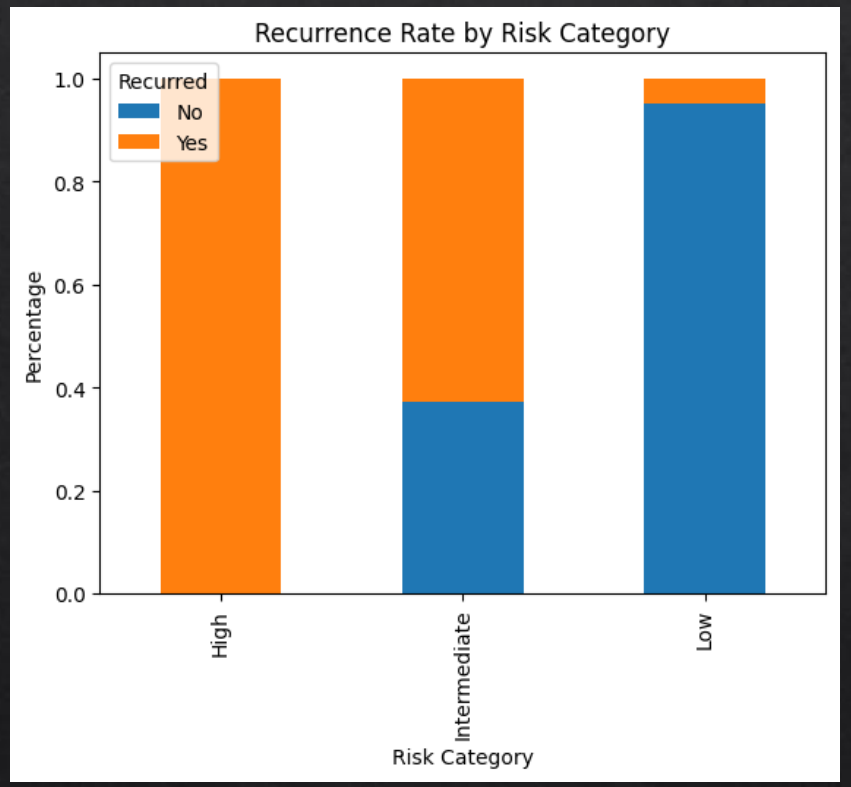
✓ 0.1s

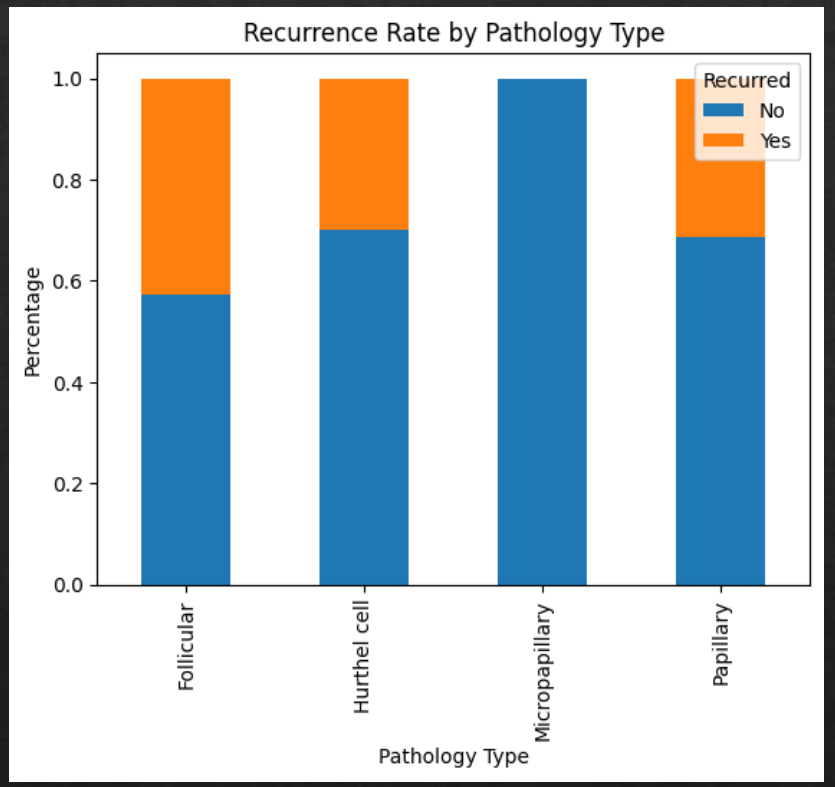
```
Target variable distribution:
Recurred
No     275
Yes    108
Name: count, dtype: int64
Recurred
No     0.718016
Yes    0.281984
Name: proportion, dtype: float64
```

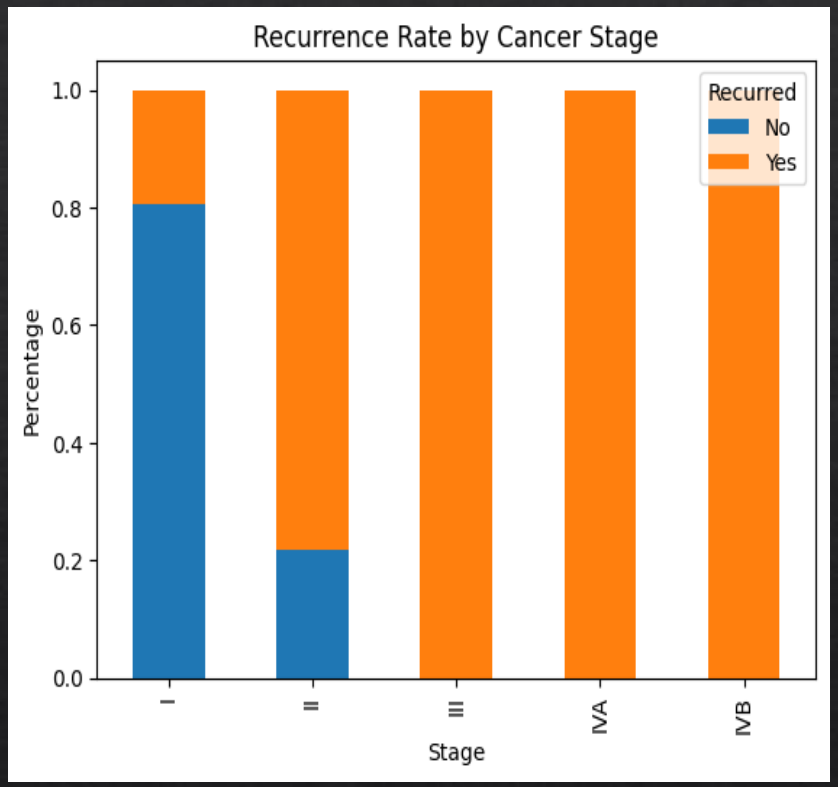
Data Visualization



Data Visualization







Feature Engineering

```
# Define target variable
X = data.drop('Recurred', axis=1)
y = data['Recurred'].map({'Yes': 1, 'No': 0})
```

```
# Create preprocessing pipelines for both numerical and categorical data
numerical_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')
```

(Preprocessing + SMOTE)

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])
```

```
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train_processed, y_train)
```

Model Architecture

```
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_processed.shape[1],)),
    Dropout(0.4),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

Model Architecture

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	7,168
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
dropout_2 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 1)	33

Total params: 17,537 (68.50 KB)

Trainable params: 17,537 (68.50 KB)

Non-trainable params: 0 (0.00 B)

Use Early stopping to prevent overfitting

```
early_stopping = EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)
```

```
history = model.fit(
    X_train_smote, y_train_smote,
    epochs=150,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stopping],
    verbose=1
)
```

Train the Model For 150 Epochs

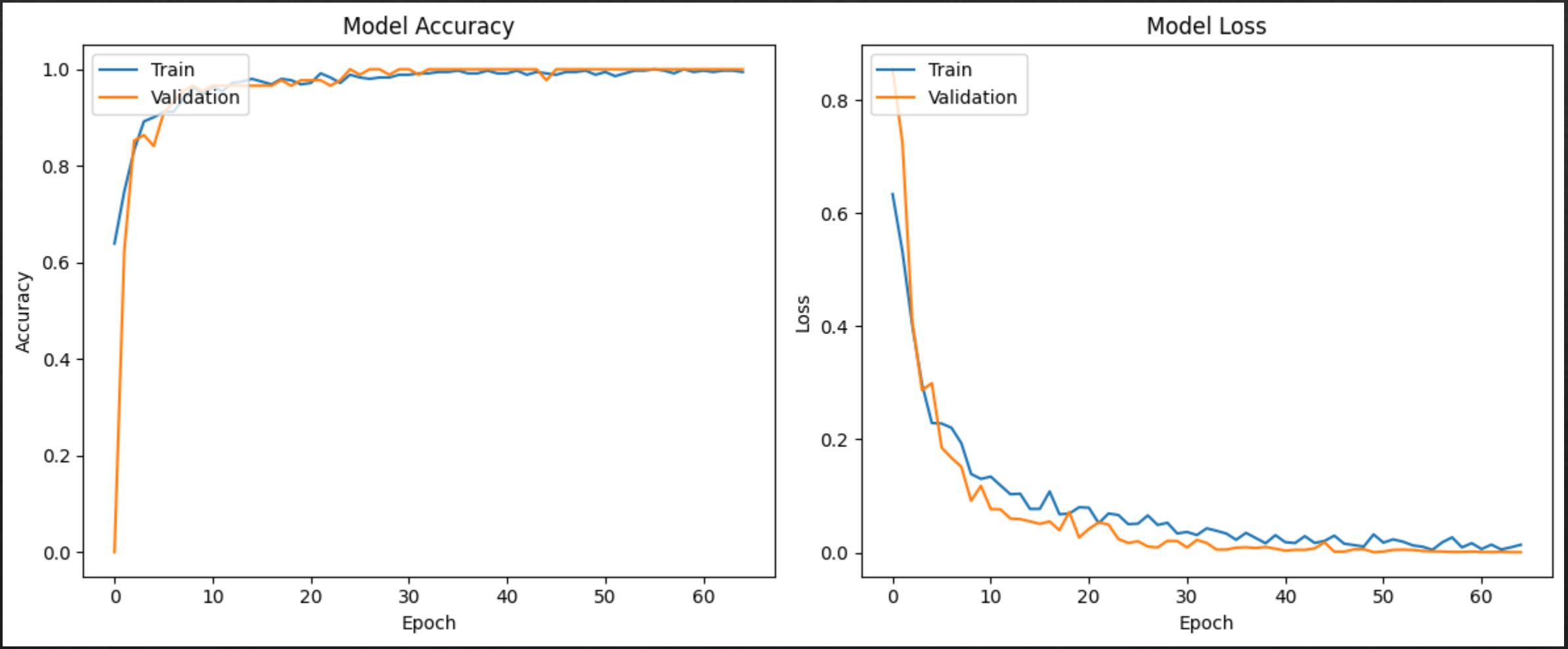
```
Epoch 1/150
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.6126 - loss: 0.6576 - val_accuracy: 0.0000e+00 - val_loss: 0.8545
Epoch 2/150
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7439 - loss: 0.5378 - val_accuracy: 0.6250 - val_loss: 0.7256
Epoch 3/150
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8305 - loss: 0.4227 - val_accuracy: 0.8523 - val_loss: 0.4100
Epoch 4/150
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8957 - loss: 0.3056 - val_accuracy: 0.8636 - val_loss: 0.2872
Epoch 5/150
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8983 - loss: 0.2330 - val_accuracy: 0.8409 - val_loss: 0.2994
Epoch 6/150
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9115 - loss: 0.2280 - val_accuracy: 0.9091 - val_loss: 0.1855
Epoch 7/150
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9079 - loss: 0.2172 - val_accuracy: 0.9318 - val_loss: 0.1675
Epoch 8/150
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9469 - loss: 0.1558 - val_accuracy: 0.9545 - val_loss: 0.1518
Epoch 9/150
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9465 - loss: 0.1911 - val_accuracy: 0.9659 - val_loss: 0.0915
Epoch 10/150
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9500 - loss: 0.1249 - val_accuracy: 0.9545 - val_loss: 0.1179
```

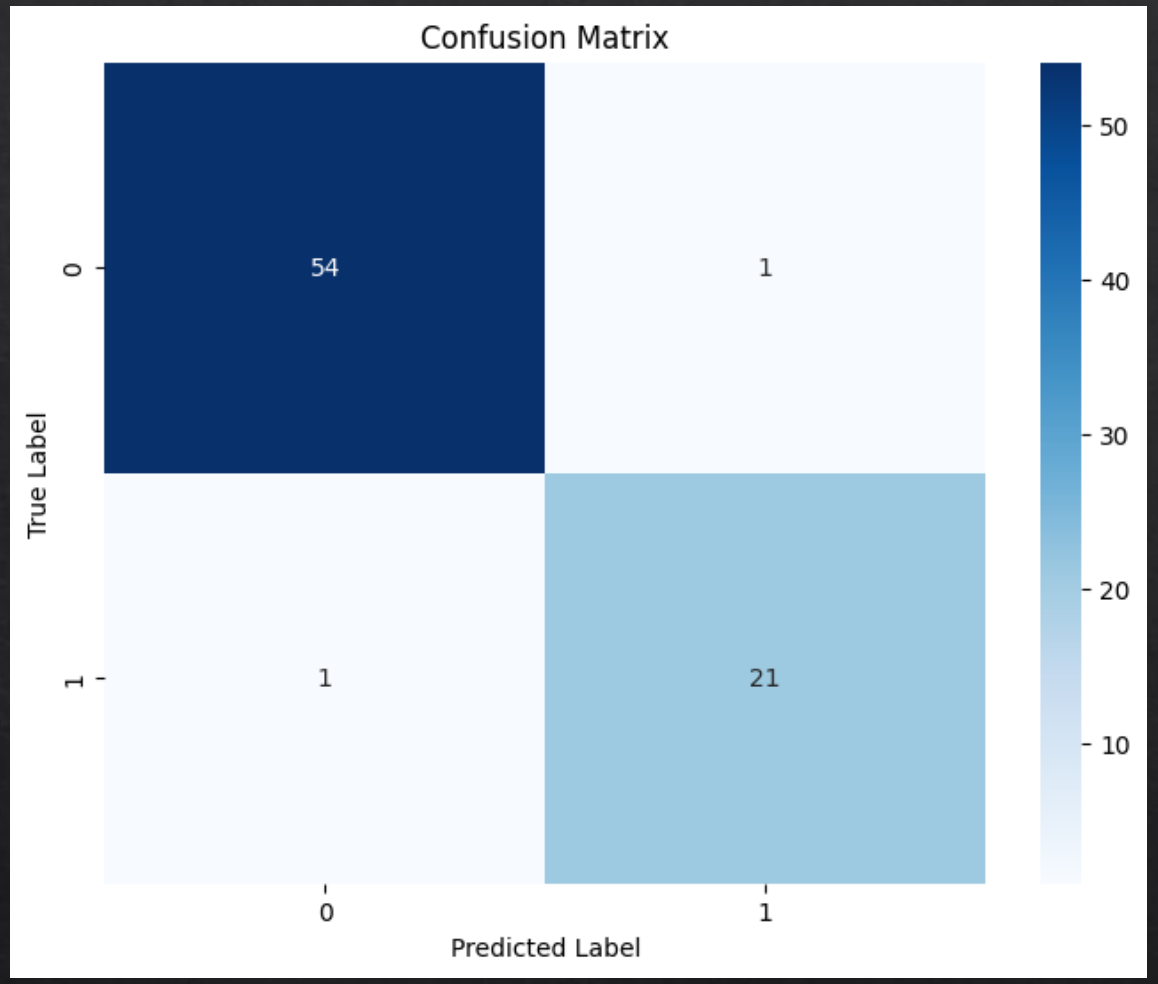
It's stop at 65 by getting 99% accuracy

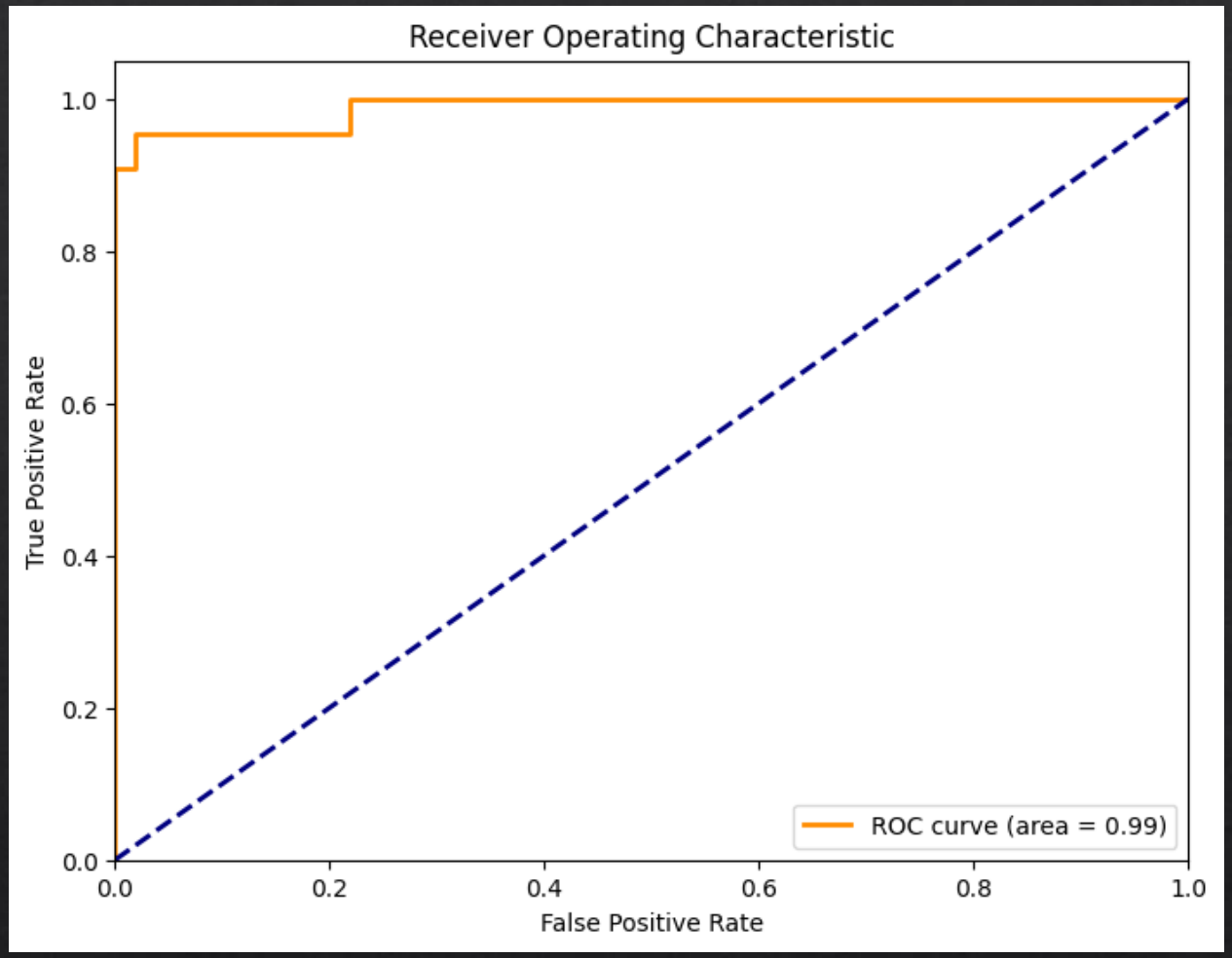
```
Epoch 64/150
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9945 - loss: 0.0145 - val_accuracy: 1.0000 - val_loss: 7.6472e-04
Epoch 65/150
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9938 - loss: 0.0184 - val_accuracy: 1.0000 - val_loss: 6.1679e-04
```

Accuracy – 99%

Evaluate the Model







Save the Model

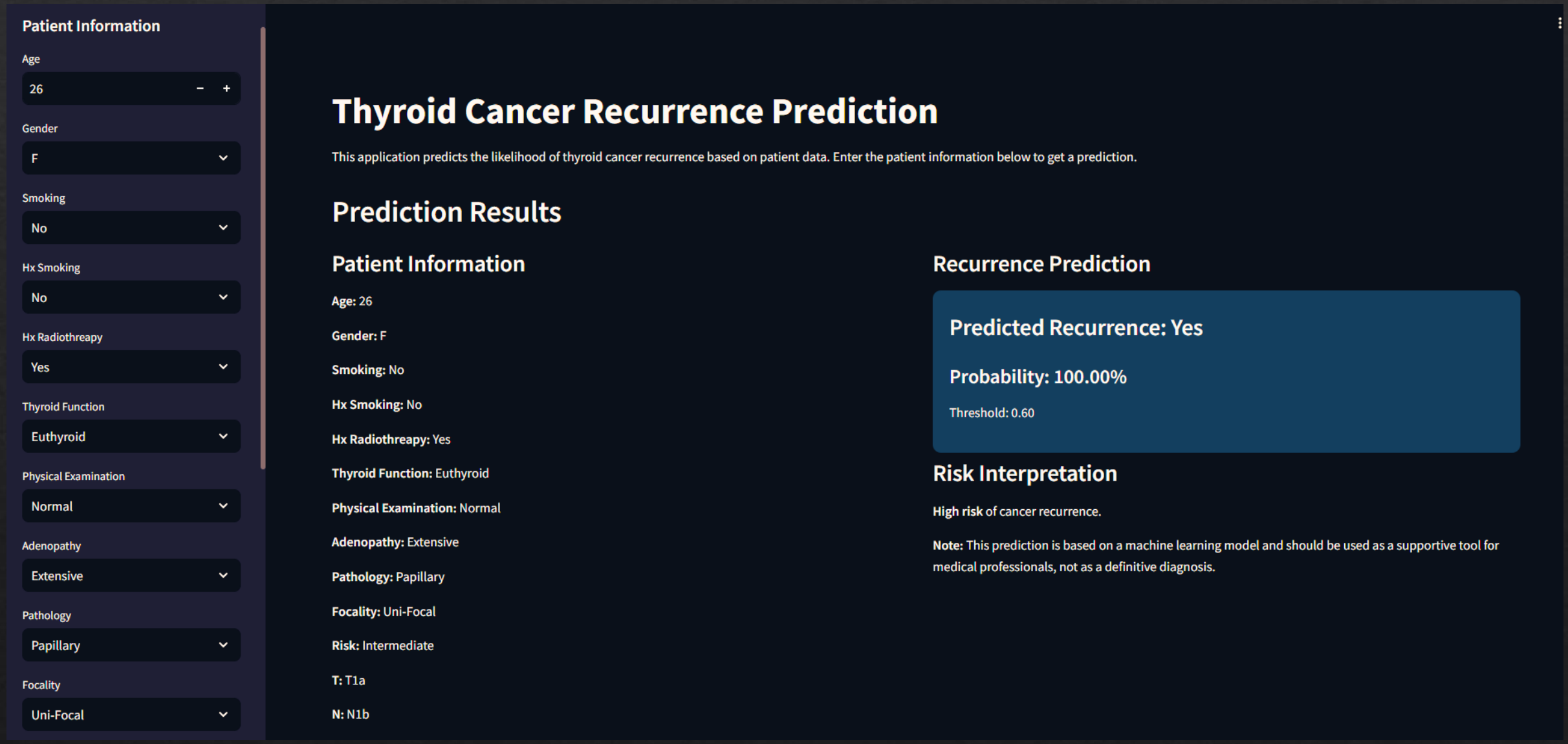
```
Model, preprocessor, and threshold saved successfully.
```

UI CODE

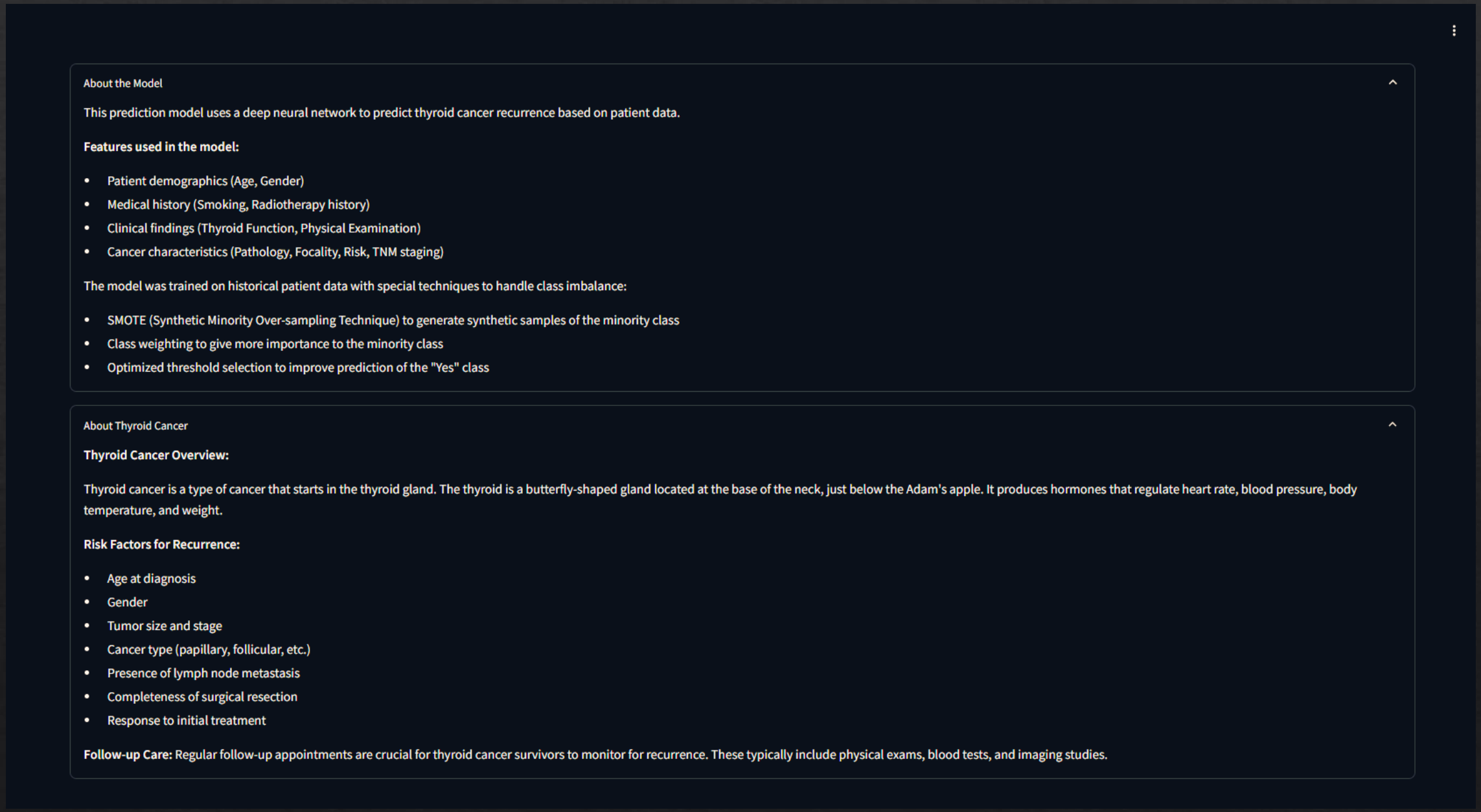
```python
import streamlit as st
import pandas as pd
import numpy as np
import pickle
from tensorflow.keras.models import load_model
import os

# Set page title and layout
st.set_page_config(page_title="Thyroid Cancer Recurrence Prediction", layout="wide")

# Title and description
st.title("Thyroid Cancer Recurrence Prediction")
st.markdown("""
This application predicts the likelihood of thyroid cancer recurrence based on patient data.
Enter the patient information below to get a prediction.
""")
```

```python
with st.expander("About Thyroid Cancer"):
    st.write("""
    **Thyroid Cancer Overview:**

    Thyroid cancer is a type of cancer that starts in the thyroid gland. The thyroid is a butterfly-shaped gland located at the base of the neck, just below the Adam's ap

    **Risk Factors for Recurrence:**
    - Age at diagnosis
    - Gender
    - Tumor size and stage
    - Cancer type (papillary, follicular, etc.)
    - Presence of lymph node metastasis
    - Completeness of surgical resection
    - Response to initial treatment

    **Follow-up Care:**
    Regular follow-up appointments are crucial for thyroid cancer survivors to monitor for recurrence. These typically include physical exams, blood tests, and imaging st
    """)
```

Prediction



Prediction



GitHub Repo- Thyroid-Cancer-Detect-APP

Conclusion

i. Successfully built an AI model for thyroid cancer recurrence.
ii. Handled imbalanced data using SMOTE and class weighting.
iii. Deployed as a Streamlit web app for real-world usability.
iv. Can serve as a supportive tool for medical professionals.

Project - 6

Vehicle Price Prediction System Using Deep Learning

Presented by – Aritra Mukherjee

ID: UMIP270247

Machine Learning Intern
At - Unified Mentor Pvt. Limited

Introduction To Project – Vehicle Price Prediction

Rapid increase in online vehicle marketplaces has made price estimation crucial.

Manual pricing can be inaccurate and biased.

Our solution uses machine learning to predict accurate vehicle prices based on features like make, model, fuel type, etc.

GOAL

1. Build a regression model using Deep Learning to accurately predict vehicle prices.
2. Analyze patterns from various vehicle specifications.
3. Assist buyers/sellers in determining fair market prices.

Dataset Overview

File:

dataset.csv – Given By Unified Mentor.

Cleaned using forward fill and null removal.

Key Features:

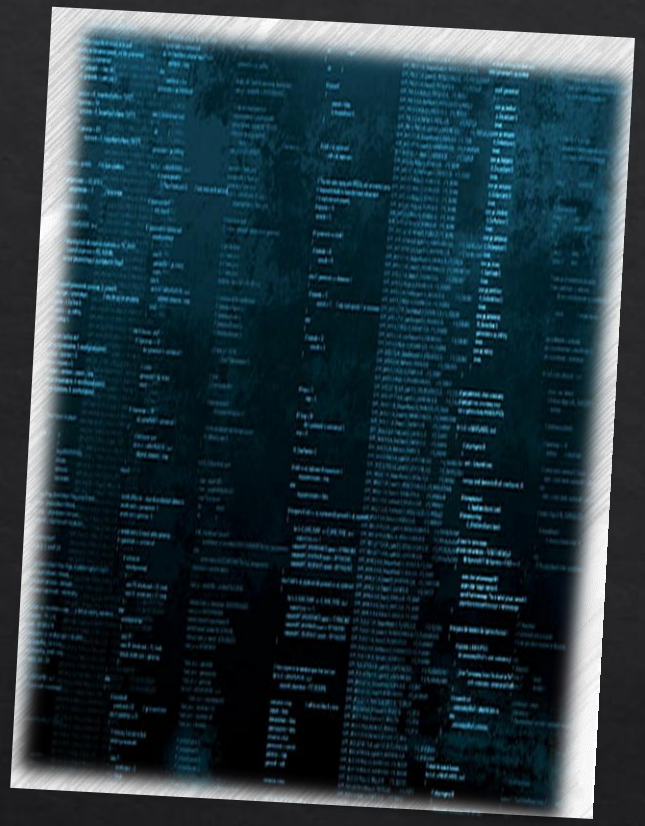
- Name,
- model,
- year,
- mileage,
- fuel type,
- transmission,
- engine size,
- price (target)

🖌️ Preprocessing:

- Label Encoding for categorical
- Standard Scaling for numerical

Tech Stack

Technologies & Tools Used



Component	Tools/Libraries	Purpose
Programming Language	Python	Core language for development
Data Manipulation	pandas, NumPy	Handling, cleaning, and transforming data
Visualization	matplotlib, seaborn	Plotting distributions and trends
Preprocessing	scikit-learn (LabelEncoder, StandardScaler)	Encoding categorical features and scaling numerical ones
Modeling	TensorFlow, Keras	Building and training deep learning models
Model Saving	joblib	Saving pre-trained models

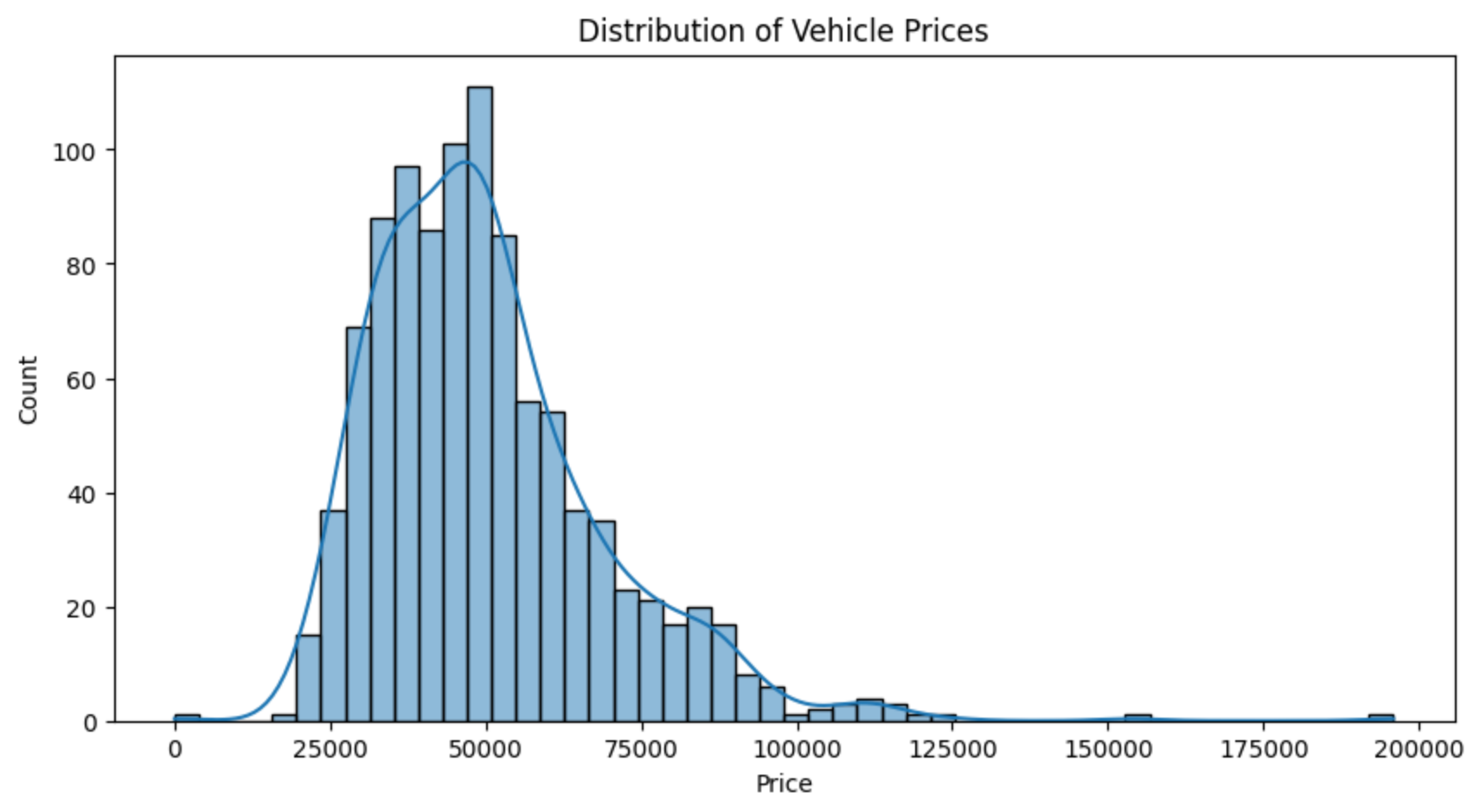
Data Preprocessing

```
df = pd.read_csv('dataset.csv')
df.fillna(method='ffill', inplace=True)
df.dropna(inplace=True)
df.reset_index(drop=True, inplace=True)
df.head()
```

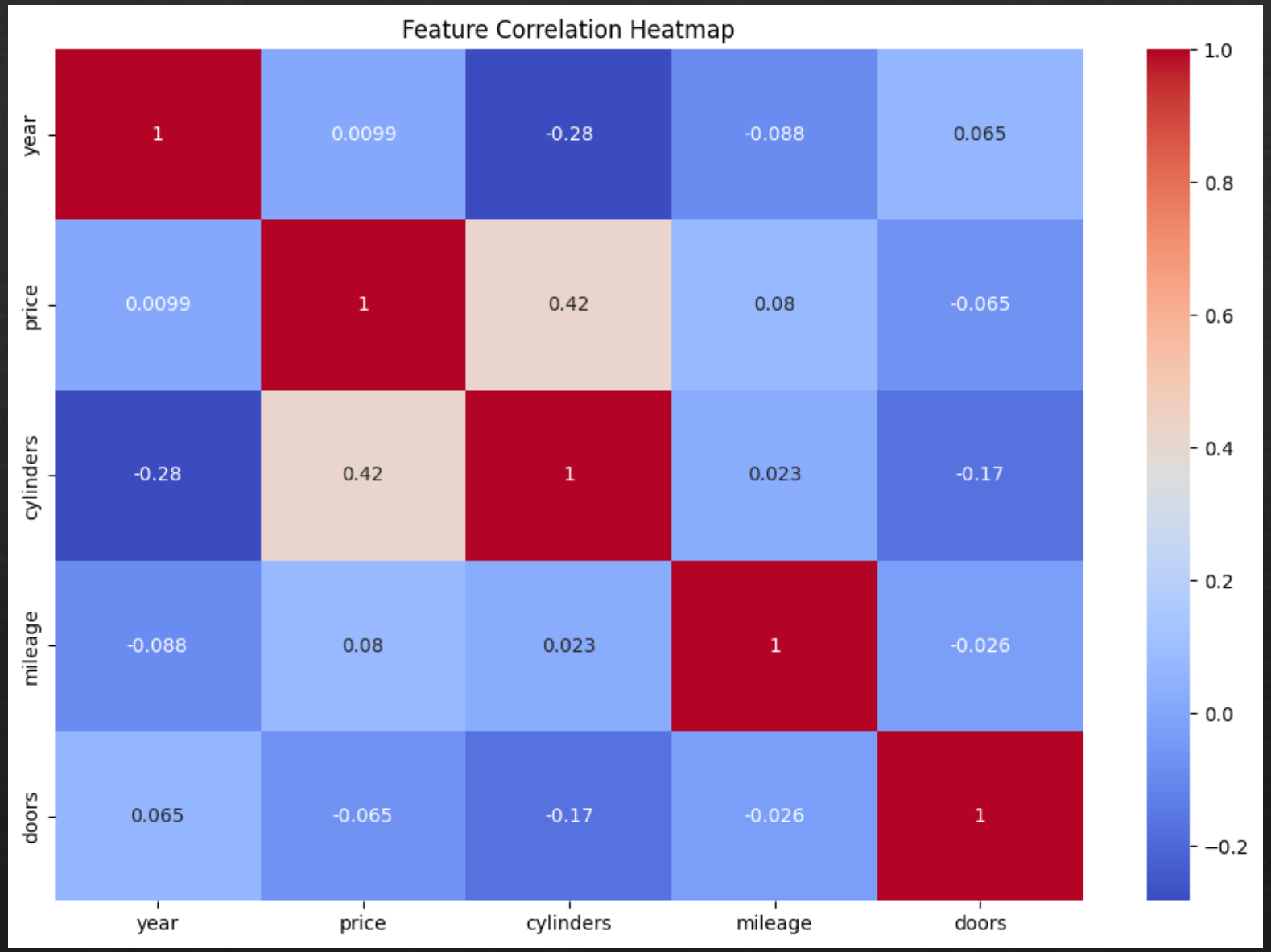
✓ 0.0s

	name	description	make	model	year	price	engine	cylinders	fuel	mileage	transmission	trim	body	doors	exterior_color	interior_color	drivetrain
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	Four-wheel Drive
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	Four-wheel Drive
2	2024 GMC Yukon XL Denali	Al West is committed to offering every custome...	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	Four-wheel Drive
3	2023 Dodge Durango Pursuit	White Knuckle Clearcoat 2023 Dodge Durango Pur...	Dodge	Durango	2023	46835.0	16V MPFI OHV	8.0	Gasoline	32.0	8-Speed Automatic	Pursuit	SUV	4.0	White Knuckle Clearcoat	Black	All-wheel Drive
4	2024 RAM 3500 Laramie	\n \n 2024 Ram 3500 Laramie Billet...	RAM	3500	2024	81663.0	24V DDI OHV Turbo Diesel	6.0	Diesel	10.0	6-Speed Automatic	Laramie	Pickup Truck	4.0	Silver	Black	Four-wheel Drive

Visualize price distribution



Correlation heatmap for numeric features



Encode categorical features

```
▷Initialize Reactive Jupyter | Sync all Stale code
X = df.drop(['price', 'description', 'name'], axis=1)
y = df['price']
non_numeric_cols = X.select_dtypes(include='object').columns
le = LabelEncoder()
for col in non_numeric_cols:
    X[col] = le.fit_transform(X[col].astype(str))
✓ 0.0s
```

Model Training & Evaluation

```
▷Initialize Reactive Jupyter | Sync all Stale code
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
joblib.dump(scaler, 'scaler.pkl')
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
✓ 0.0s
```

Model Architecture

▷Initialize Reactive Jupyter | Sync all Stale code

```
model = Sequential([
    Dense(256, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dense(1)
])

# model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.compile(optimizer='adam', loss=MeanSquaredError(), metrics=['mae'])
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=60, batch_size=32)
```

✓ 13.1s

Train the Model

```
Epoch 1/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 2992164096.0000 - mae: 50769.0430 - val_loss: 2564723712.0000 - val_mae: 48044.1289
Epoch 2/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 2797558528.0000 - mae: 49527.8516 - val_loss: 2554057216.0000 - val_mae: 47936.5820
Epoch 3/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 2952999680.0000 - mae: 50784.8750 - val_loss: 2509486592.0000 - val_mae: 47487.3477
Epoch 4/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 2816957440.0000 - mae: 49750.2734 - val_loss: 2382494464.0000 - val_mae: 46189.1602
Epoch 5/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 2696035328.0000 - mae: 48464.5508 - val_loss: 2100228096.0000 - val_mae: 43163.4922
Epoch 6/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 2209021184.0000 - mae: 43440.3320 - val_loss: 1610236416.0000 - val_mae: 37291.6133
Epoch 7/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 1638714880.0000 - mae: 36677.9023 - val_loss: 969609088.0000 - val_mae: 27613.0605
Epoch 8/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 994236608.0000 - mae: 26144.8926 - val_loss: 444174880.0000 - val_mae: 15890.3594
Epoch 9/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 400846048.0000 - mae: 14784.4629 - val_loss: 258497280.0000 - val_mae: 11556.2012
Epoch 10/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 319617312.0000 - mae: 12955.9746 - val_loss: 229022128.0000 - val_mae: 11116.0742
```

```
Epoch 57/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 176172592.0000 - mae: 9548.7031 - val_loss: 241109792.0000 - val_mae: 10551.8418
Epoch 58/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 173828928.0000 - mae: 9879.8750 - val_loss: 243169216.0000 - val_mae: 10608.9580
Epoch 59/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 204306816.0000 - mae: 10508.3291 - val_loss: 242121104.0000 - val_mae: 10523.8086
Epoch 60/60
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 190206384.0000 - mae: 10283.1367 - val_loss: 248958992.0000 - val_mae: 10698.8008
```

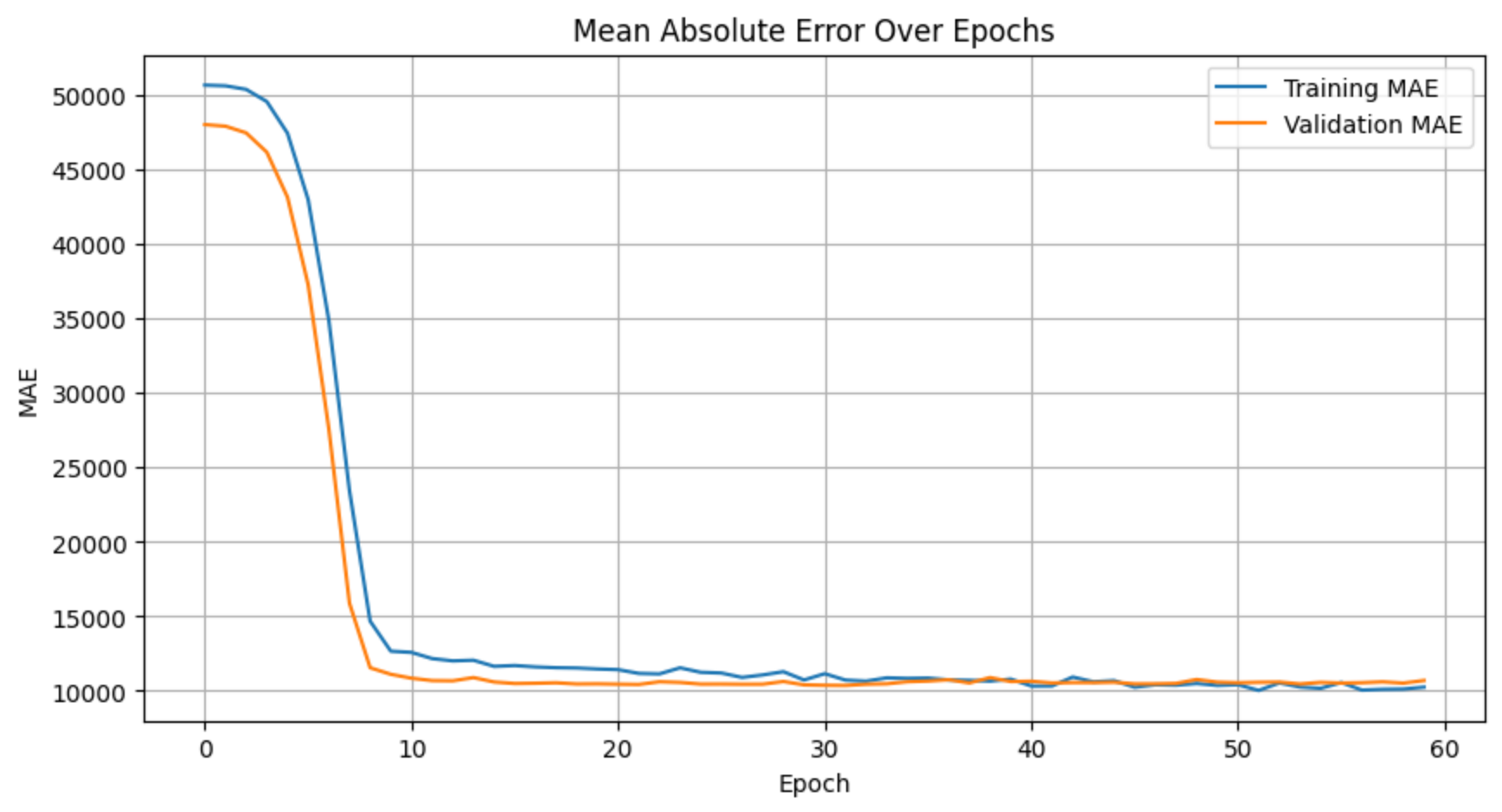
Save the Trained Model

```
model.save('vehicle_price_model.h5')
print("Model Saved Successfully.")
```

✓ 0.0s

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
Model Saved Successfully.
```

Mean Absolute Error Over Epochs

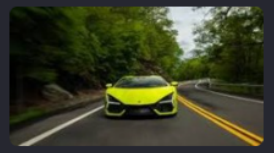


UI CODE

```python
import streamlit as st
import pandas as pd
import numpy as np
import tensorflow as tf
import joblib

# Load model and scaler
model = tf.keras.models.load_model('vehicle_price_model.h5')
scaler = joblib.load('scaler.pkl')

# Set page config and styling
st.set_page_config(page_title="Vehicle Price Predictor", page_icon="🚗", layout="wide")
st.markdown("""
    <style>
    .main { background-color: #f9f9f9; padding: 20px; border-radius: 10px; }
    .stButton>button { background-color: #008CBA; color: white; font-weight: bold; }
    .stTabs [role="tab"] { font-size: 18px; }
    .highlight { font-size: 26px; font-weight: bold; color: green; padding: 10px 0; }
    .inr {
        font-size: 20px;
        font-weight: bold;
        color: #444;
        background-color: #fff3cd;
        padding: 10px;
        border-radius: 8px;
        border: 1px solid #ffeeba;
        margin-top: 10px;
    }
    </style>
""", unsafe_allow_html=True)
```

```python
with tab2:
    st.title("📘 Vehicle Info & Help")
    st.markdown("""
    ### 🔧 How to Use:
    - Fill all fields under "Predict Price" tab.
    - Click on **Predict Price** to get the estimate.
    - The model gives price in both **USD** and approximate **INR** range.

    ### 📱 Field Guide:
    - **Make & Model**: Brand and model name.
    - **Trim**: Variant with extra features.
    - **Engine**: Engine configuration/type (e.g., EcoBoost, V6).
    - **Fuel**: Fuel type like Gasoline or Hybrid.
    - **Transmission**: Auto, Manual, or CVT.
    - **Drivetrain**: Power delivery - FWD, RWD, AWD.
    - **Mileage**: Vehicle distance used (in miles).

    ### 💡 Notes:
    - INR value assumes ₹83/USD with ±5% range.
    - This tool is based on machine learning predictions and may vary from actual dealer prices.

    ### 💌 Contact:
    Email: aritra.work.official@gmail.com
    """)
```

Prediction



📊 Vehicle Options

🔧 Customize your vehicle and predict its price in **USD & INR**.

Made by **Aritra**

📬 aritra.work.official@gmail.com

Deploy ⋮

🚙 Predict Price | 📘 Info & Help

🚙 Vehicle Price Predictor

Fill in the vehicle details to estimate the price in **USD** and **INR**.

Make: Ford	Body Type: SUV
Model: Wagoneer	Cylinders: 4 (2 – 16)
Year: 2024 (1990 – 2025)	Number of Doors: 2
Mileage (in miles): 0	Exterior Color: White
Fuel Type: Gasoline	Interior Color: Black
Transmission: Automatic	Drivetrain: FWD
Trim Level: Base	Engine Type: 16V GDI DOHC Turbo Hybrid

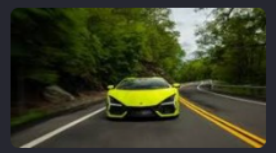
🔍 Predict Price

💰 **Estimated Price: $120,000.00**

IN Estimated INR Range: ₹9,462,000 – ₹10,458,000

Note: This prediction is based on the model's training data and may vary from actual market prices. Factors like vehicle condition, local market, and specific features can affect the actual price.

Prediction



📊 Vehicle Options

🔧 Customize your vehicle and predict its price in **USD & INR**.

Made by **Aritra**

📬 aritra.work.official@gmail.com

Deploy ⋮

🚗 Predict Price | 📘 Info & Help

📘 Vehicle Info & Help

🔧 How to Use:

- Fill all fields under "Predict Price" tab.
- Click on **Predict Price** to get the estimate.
- The model gives price in both **USD** and approximate **INR** range.

📑 Field Guide:

- **Make & Model**: Brand and model name.
- **Trim**: Variant with extra features.
- **Engine**: Engine configuration/type (e.g., EcoBoost, V6).
- **Fuel**: Fuel type like Gasoline or Hybrid.
- **Transmission**: Auto, Manual, or CVT.
- **Drivetrain**: Power delivery - FWD, RWD, AWD.
- **Mileage**: Vehicle distance used (in miles).

💡 Notes:

- INR value assumes ₹83/USD with ±5% range.
- This tool is based on machine learning predictions and may vary from actual dealer prices.

💌 Contact:

Email: aritra.work.official@gmail.com

GitHub Repo- Vehicle-Price-Prediction-APP

Conclusion

A deep learning-based model can effectively predict vehicle prices using structured tabular data.

Practical use in e-commerce platforms and dealership valuation.

Thank You!