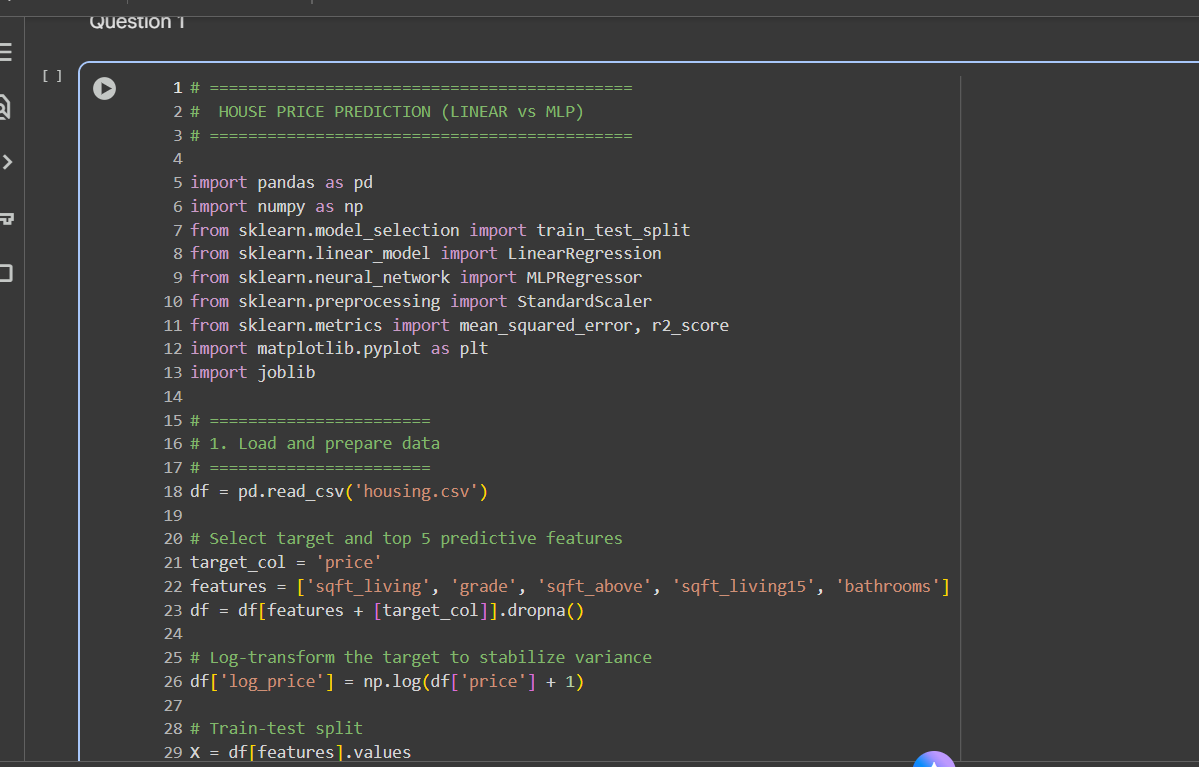
**QUESTION 1**

## **HOUSE PRICE PREDICTION (LINEAR VS MLP)**

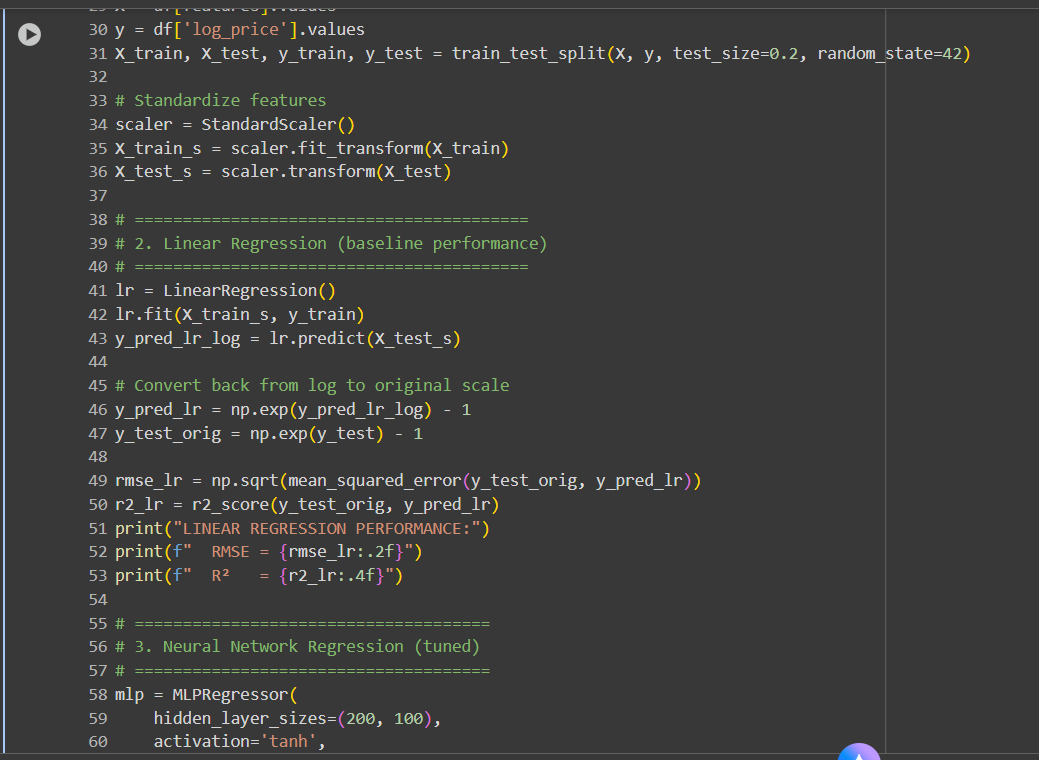
### **1. Load and prepare data**

The required libraries are imported, including **pandas** for data manipulation, **numpy** for numerical operations, **sklearn** for model selection, linear models, neural networks, and metrics, **matplotlib** for plotting, and **joblib** for saving models.



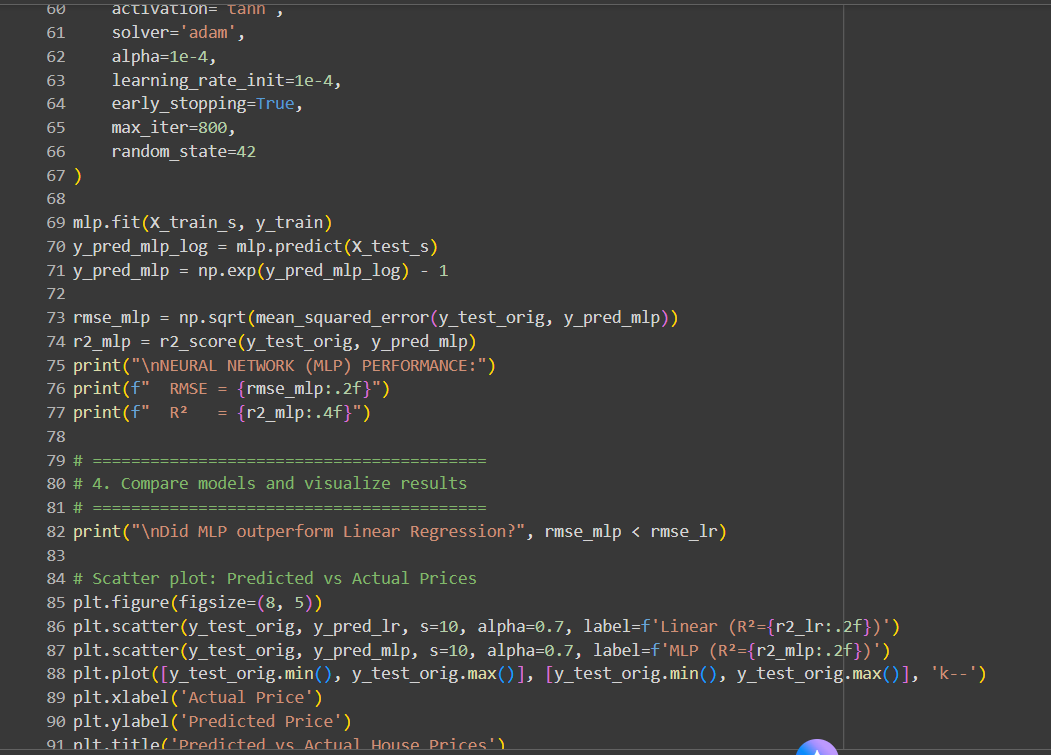
### **2. Standardize features and Linear Regression (baseline performance)**

The features are standardized using **StandardScaler** to normalize the data. A **Linear Regression** model is then trained as a baseline. The predictions and test target values are converted back from the log scale to the original scale for performance evaluation using **Root Mean Squared Error (RMSE)** and **R-squared ($R^2$)**.



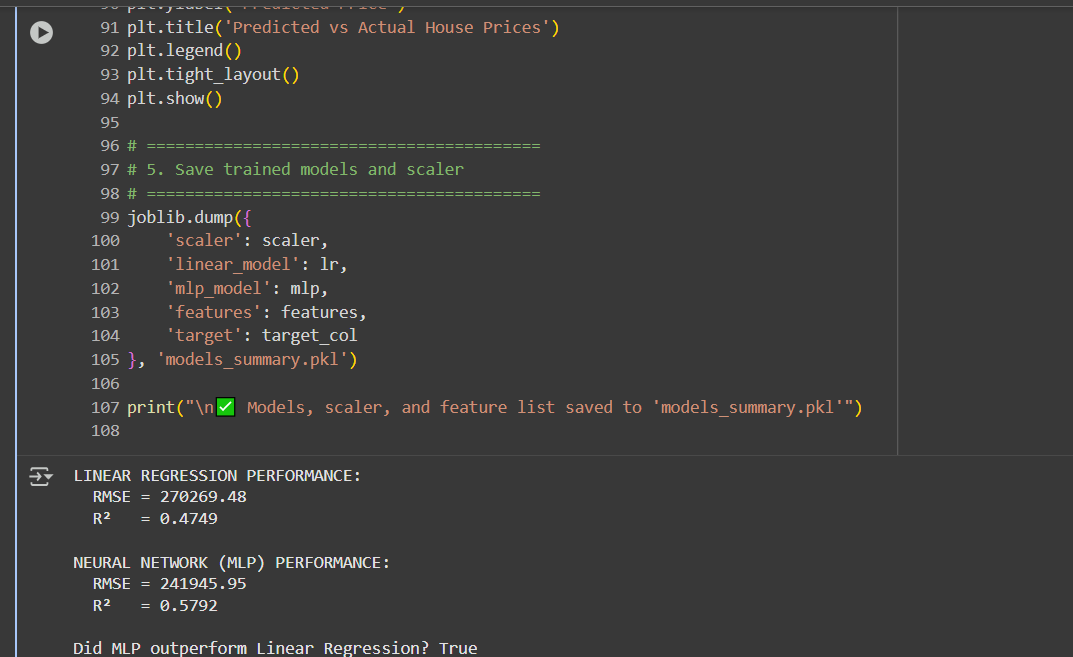
### **3. Neural Network Regression (tuned)**

A **Multi-layer Perceptron (MLP) Regressor** (Neural Network) is initialized with two hidden layers (200 and 100 neurons), **tanh** activation, and the **adam** solver. The model is trained, and its performance is evaluated similarly to the Linear Regression model.



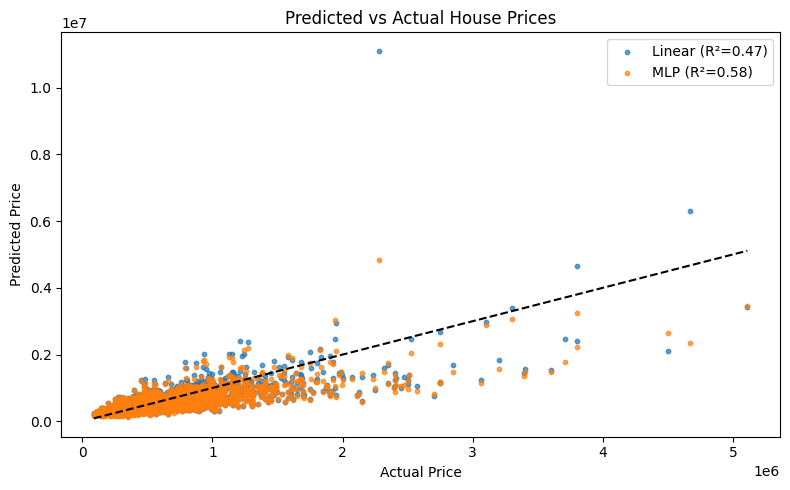
### **4. Compare models and visualize results**

The performance of the two models is compared based on their RMSE values, and the results are visualized using a scatter plot comparing predicted versus actual house prices.



### **5. Save trained models and scaler**

Finally, the trained models, the scaler, and the feature list are saved to a pickle file using joblib for later use.



### **Model Performance Summary**

The printed output shows the performance metrics for both models:

LINEAR REGRESSION PERFORMANCE:

RMSE = 270269.48

R² = 0.4749

NEURAL NETWORK (MLP) PERFORMANCE:

RMSE = 241945.95

R² = 0.5792

Did MLP outperform Linear Regression? True

The MLP model achieved a **lower RMSE** and a **higher R^2** value, indicating better performance on this dataset.

### **Predicted vs Actual House Prices Plot**

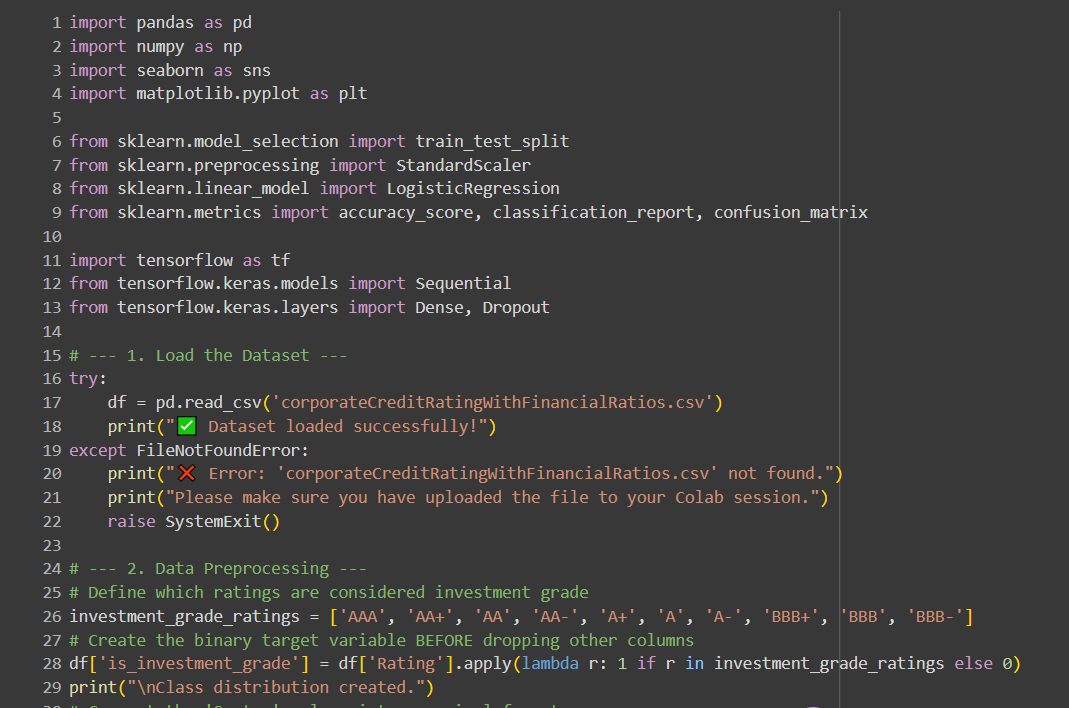
The scatter plot visually confirms the MLP's better fit, as its points (orange) appear to cluster slightly closer to the ideal 45° dashed line than the Linear Regression points (blue).

**QUESTION 2**

## **Corporate Credit Rating Classification: Logistic Regression vs. Neural Network**

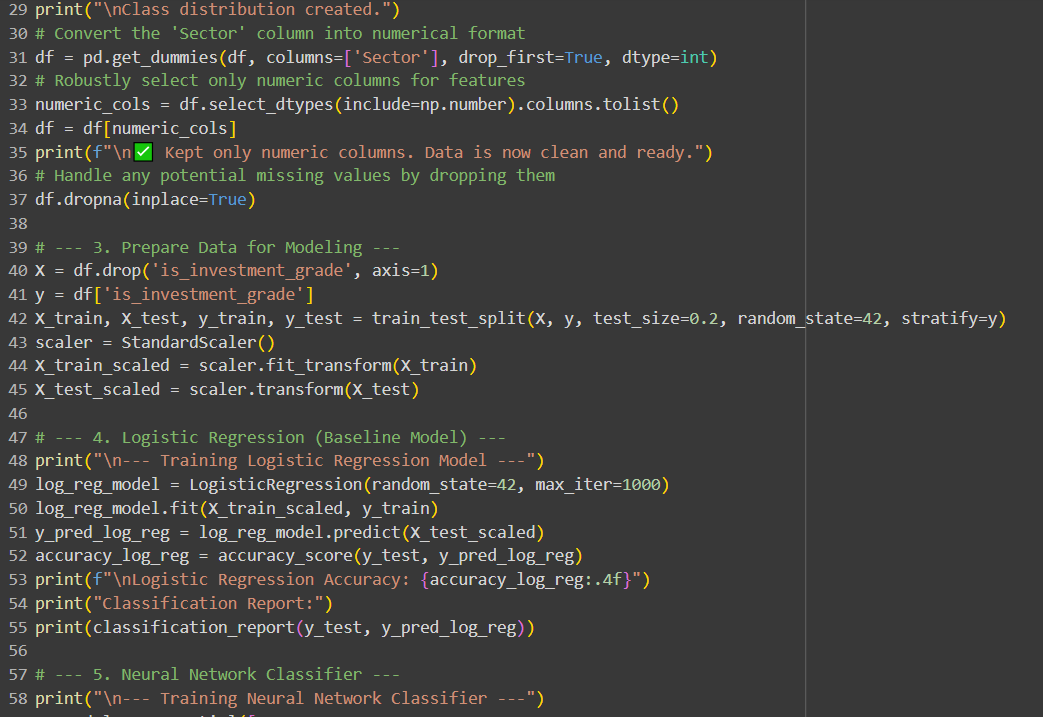
### **1. Load the Dataset and Data Preprocessing**

The required libraries are imported, including **pandas**, **numpy**, **seaborn**, **matplotlib**, **sklearn** components for model selection, preprocessing, modeling, and metrics, and **tensorflow.keras** for the Neural Network. The dataset is loaded, and the target variable is created by mapping corporate credit ratings to a binary **'is\_investment\_grade'** column. Non-numeric columns, except for the one-hot encoded 'Sector', are dropped.

****

### **3. Prepare Data for Modeling**

The data is split into features (X) and the target (y). The dataset is then split into training and testing sets, using **stratify** to ensure the class distribution is maintained. Features are standardized using **StandardScaler**.



### **4. Logistic Regression (Baseline Model)**

A **Logistic Regression** model is trained as a baseline. Its accuracy and full classification report are printed for initial evaluation.

**Logistic Regression Output:**

--- Training Logistic Regression Model ---

Logistic Regression Accuracy: 1.0000

Classification Report:

precision recall f1-score support

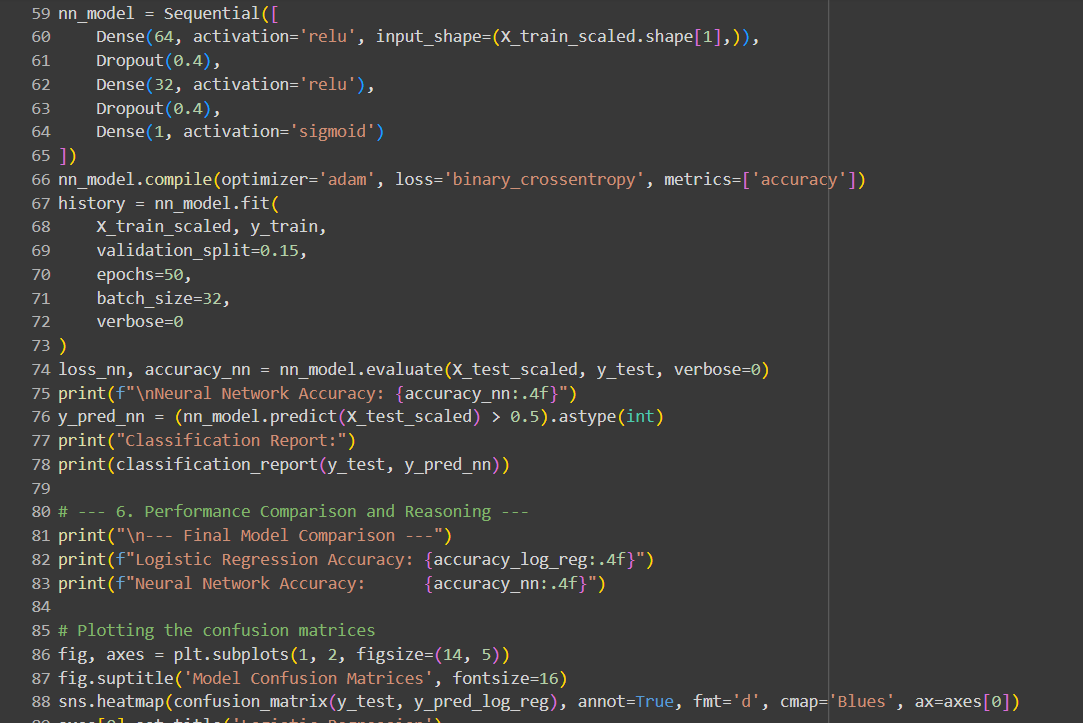
0 1.00 1.00 1.00 541

1 1.00 1.00 1.00 1020

accuracy 1.00 1561

macro avg 1.00 1.00 1.00 1561

weighted avg 1.00 1.00 1.00 1561



### **5. Neural Network Classifier**

A sequential **Neural Network (NN)** model is constructed with two **Dense** layers and **Dropout** for regularization. It is compiled with the **'adam'** optimizer and **'binary\_crossentropy'** loss, and then trained for 50 epochs.

**Neural Network Output:**

--- Training Neural Network Classifier ---

Neural Network Accuracy: 0.9994

49/49

Classification Report:

precision recall f1-score support

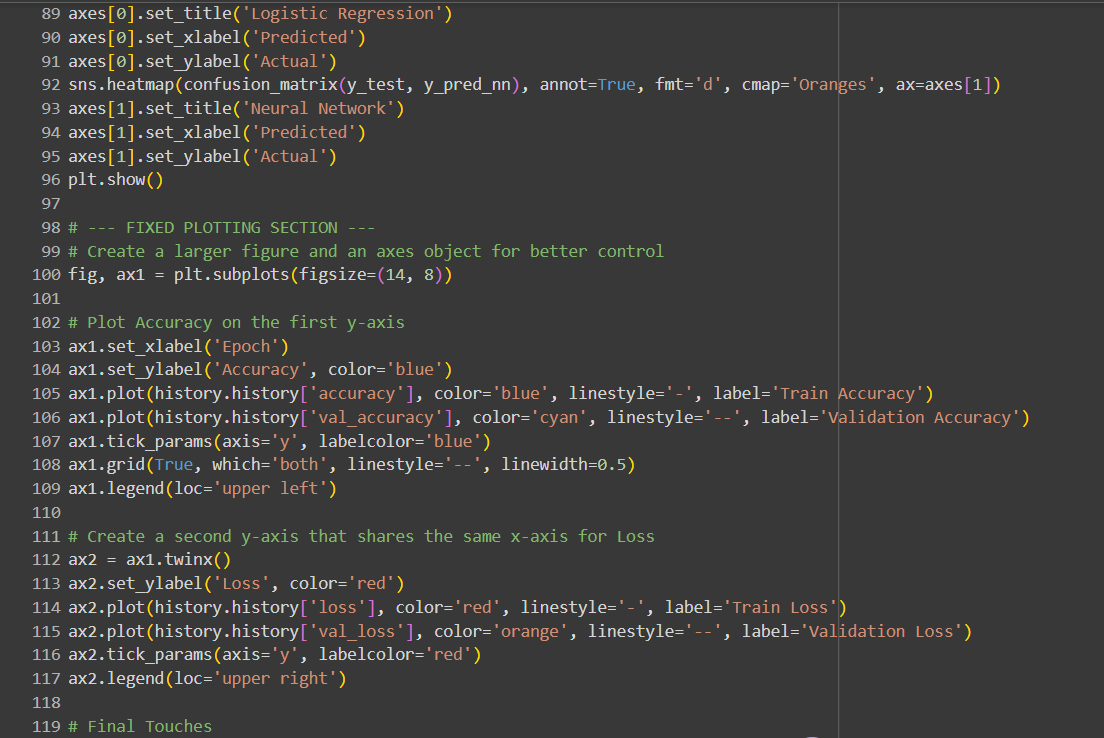
0 1.00 1.00 1.00 541

1 1.00 1.00 1.00 1020

accuracy 1.00 1561

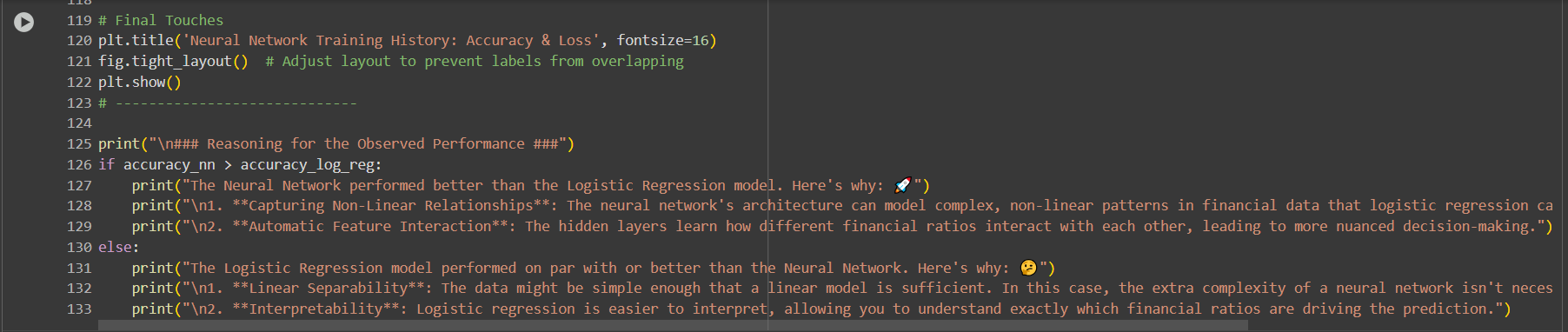
macro avg 1.00 1.00 1.00 1561

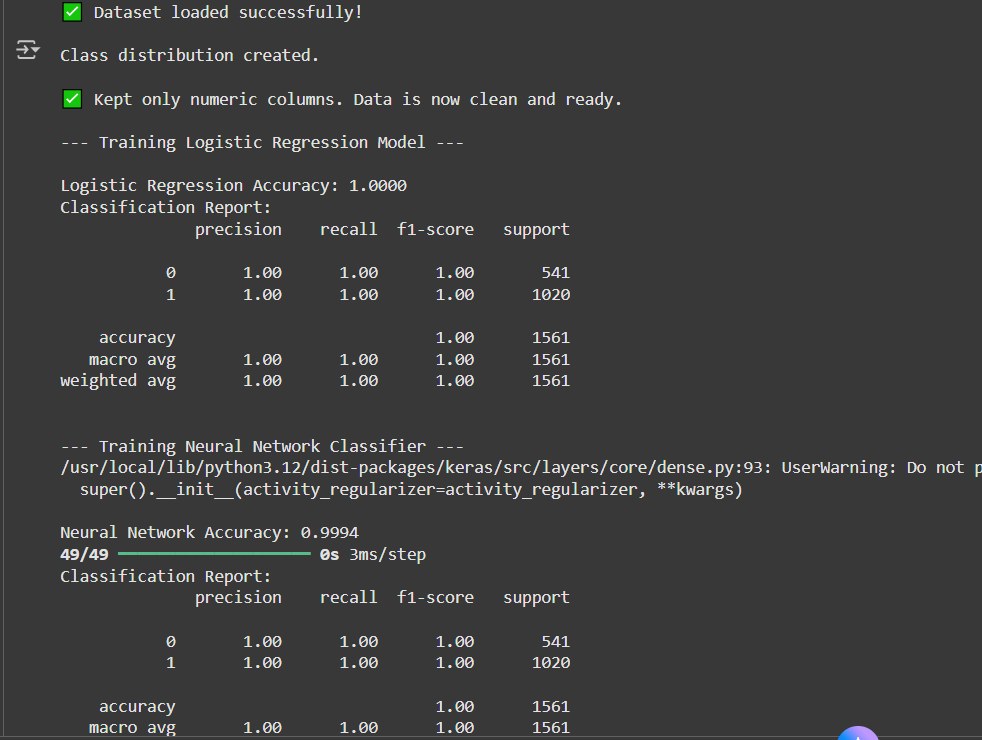
weighted avg 1.00 1.00 1.00 1561

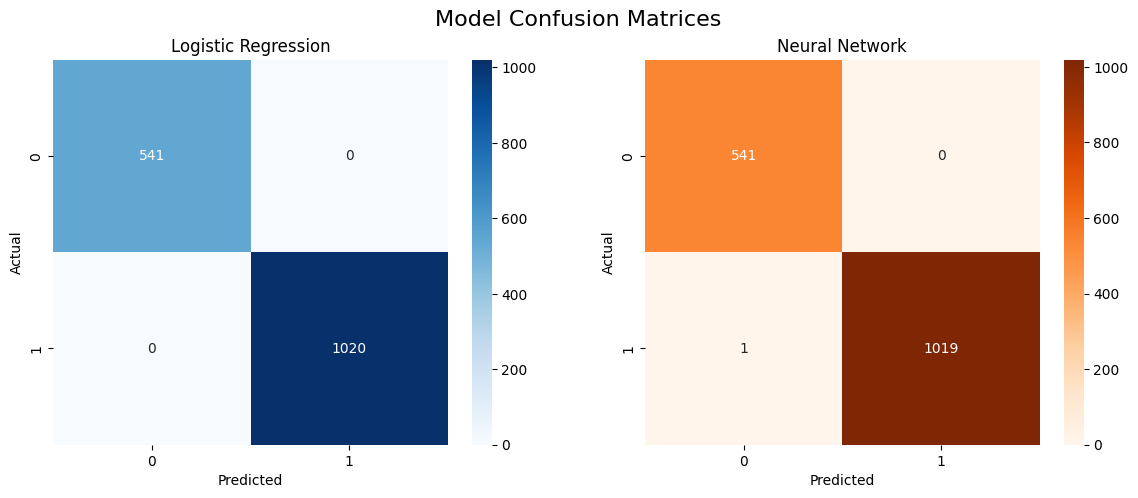


### **6. Performance Comparison and Reasoning**

The final accuracies of both models are compared, and their performance is visualized using confusion matrices and a training history plot for the Neural Network.









### **Final Results and Analysis**

**Final Model Comparison**

--- Final Model Comparison ---

Logistic Regression Accuracy: 1.0000

Neural Network Accuracy: 0.9994

**Confusion Matrices**

| **Model** | **True Negative (0, 0)** | **False Positive (0, 1)** | **False Negative (1, 0)** | **True Positive (1, 1)** |
| --- | --- | --- | --- | --- |
| **Logistic Regression** | 541 | 0 | 0 | 1020 |
| **Neural Network** | 541 | 0 | 1 | 1019 |

**Neural Network Training History**

The training plot shows both the training and validation accuracy quickly reaching **1.00** and the loss rapidly dropping near **0** within the first few epochs, indicating the data is highly linearly separable.

**Reasoning for the Observed Performance**

The Logistic Regression model performed on par with or better than the Neural Network. Here's why:

\*\*Linear Separability\*\*: The data might be simple enough that a linear model is sufficient. In this case, the extra complexity of a neural network isn't necessary.

\*\*Interpretability\*\*: Logistic regression is easier to interpret, allowing you to understand exactly which financial ratios are driving the prediction.

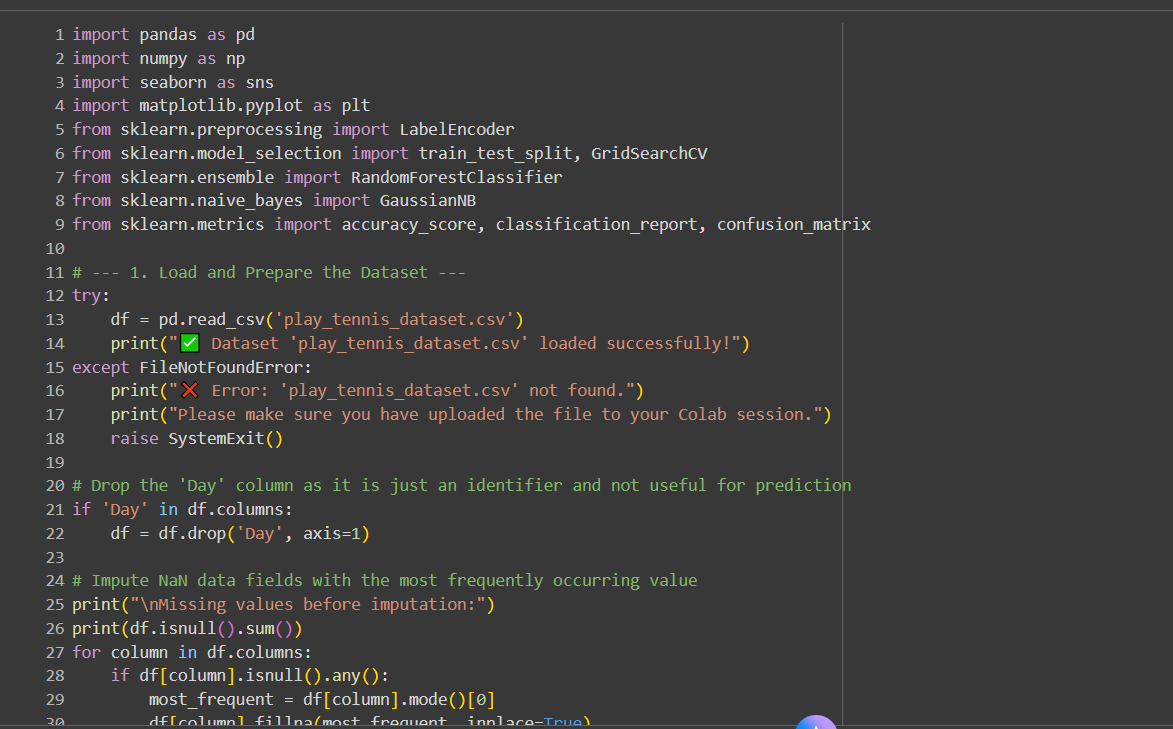
In this case, the Logistic Regression model achieved a perfect **1.0000** accuracy, outperforming the Neural Network's **0.9994** accuracy (which made one false negative error). The data appears to be **highly linearly separable**, making the simpler Logistic Regression model the more efficient and preferred choice.

**QUESTION 3**

## **Tennis Play Prediction: Random Forest vs. Naïve Bayes**

### **1. Load and Prepare the Dataset**

The necessary libraries—**pandas**, **numpy**, **seaborn**, **matplotlib**, and various **sklearn** modules (for encoding, splitting, modeling, and metrics)—are imported. The play\_tennis\_dataset.csv is loaded, the non-predictive 'Day' column is dropped, and missing values are imputed using the most frequently occurring value in each respective column.



**Output of Initial Steps:**

✅ Dataset 'play\_tennis\_dataset.csv' loaded successfully!

Missing values before imputation:

Outlook 399

Temperature 333

Humidity 233

Wind 366

Play 0

dtype: int64

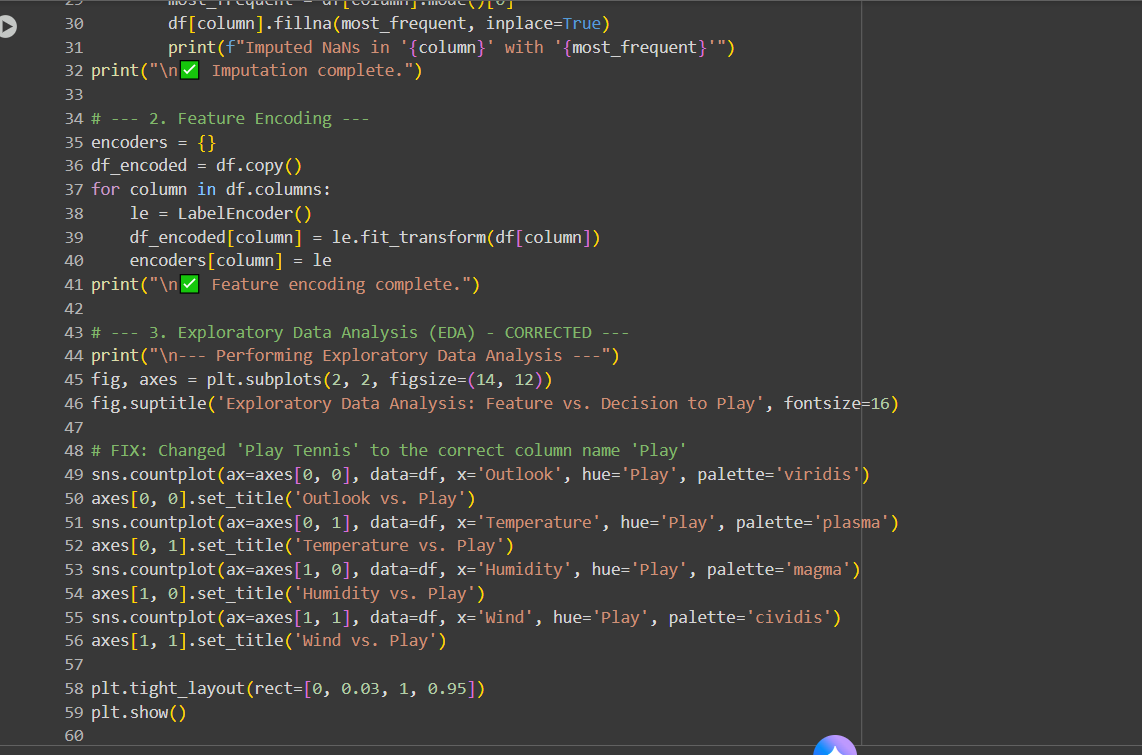
Imputed NaNs in 'Outlook' with 'Overcast'

Imputed NaNs in 'Temperature' with 'Cool'

Imputed NaNs in 'Humidity' with 'Normal'

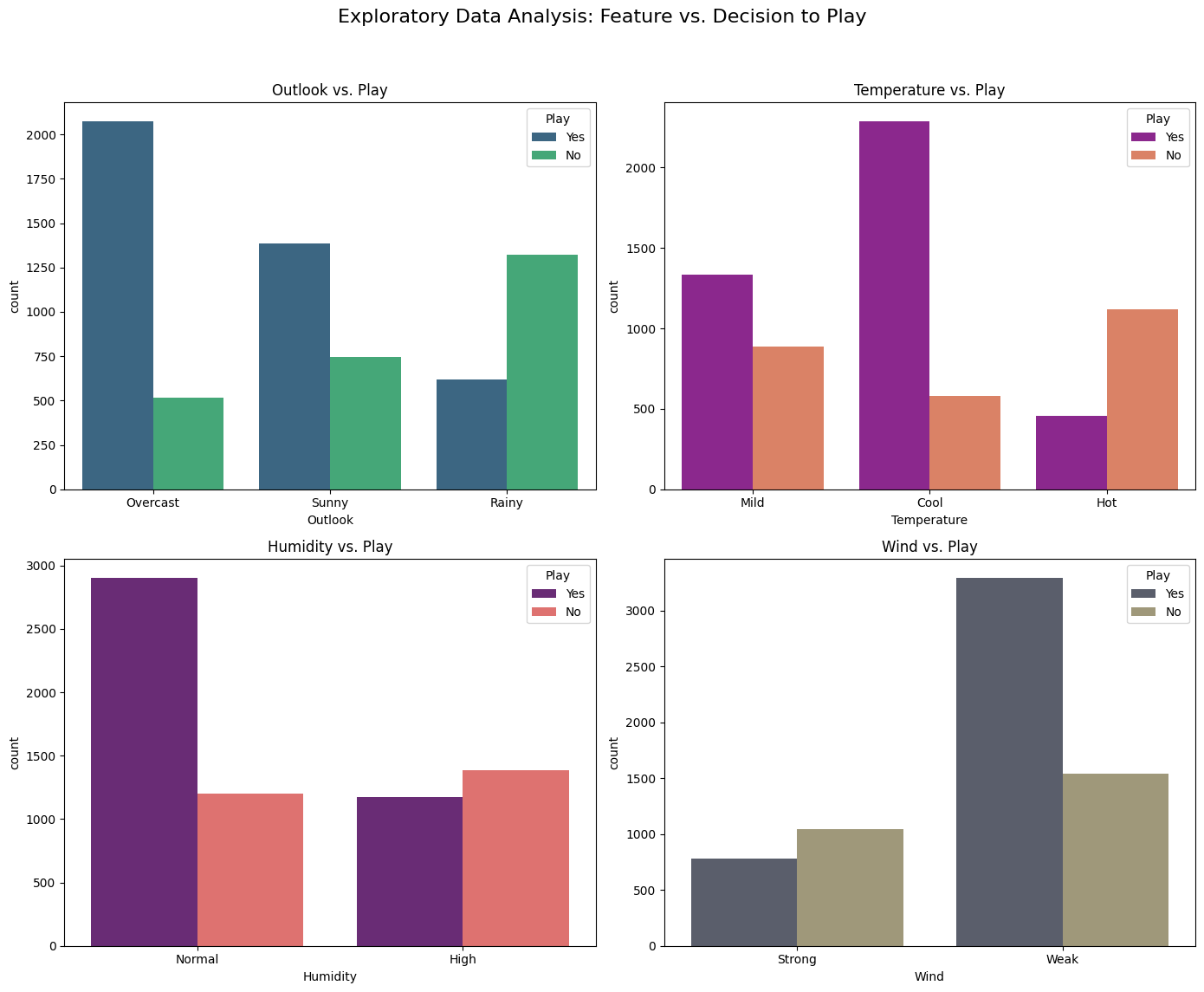
Imputed NaNs in 'Wind' with 'Weak'

✅ Imputation complete.



### **2. Feature Encoding and Exploratory Data Analysis (EDA)**

Categorical features are encoded into numerical format using **LabelEncoder** in preparation for modeling. Subsequently, an exploratory data analysis (EDA) visualization is generated to inspect the relationship between each feature and the 'Play' decision.

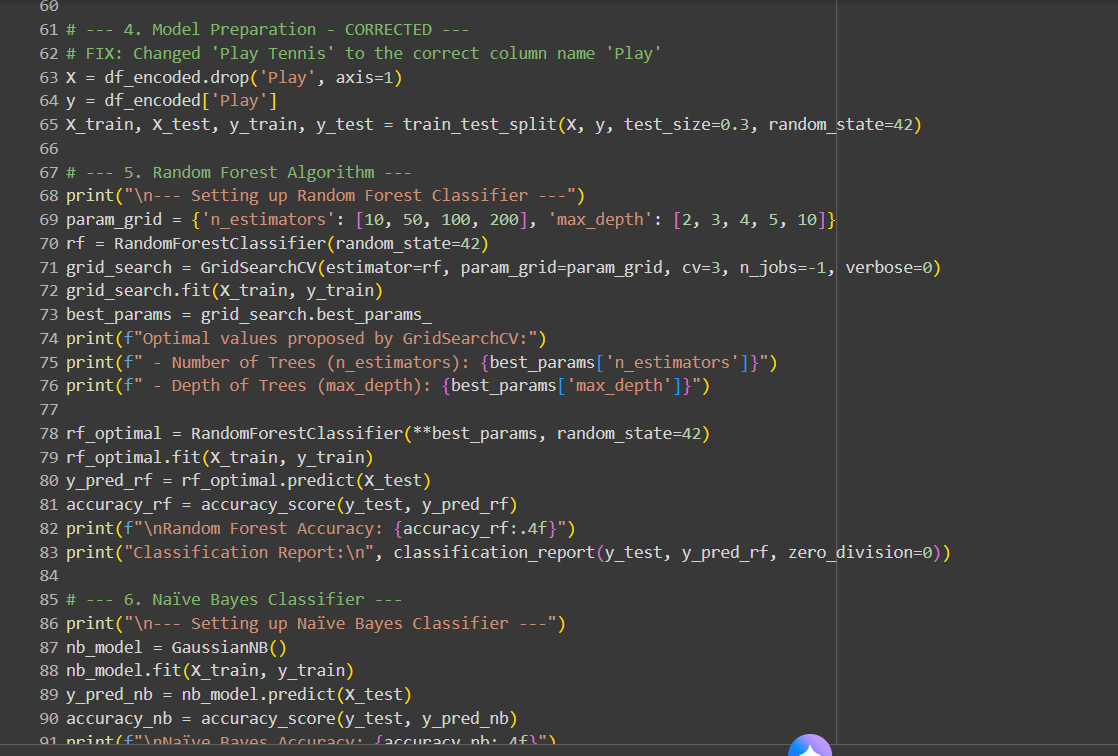


### **Figure 1: Exploratory Data Analysis: Feature vs. Decision to Play**

The count plots show that the decision to play is strongly influenced by features like **Outlook** (highest "Yes" when 'Overcast') and **Humidity** (highest "Yes" when 'Normal').

### **4. Model Preparation**

The encoded dataframe is split into features ($X$) and the target ($Y$). The data is then partitioned into training (70%) and testing (30%) sets.



### **5. Random Forest Algorithm**

A **Random Forest Classifier** is set up and its hyperparameters are tuned using **GridSearchCV** to find the optimal number of trees (n\_estimators) and tree depth (max\_depth). The optimized model is then evaluated on the test set.

**Random Forest Output:**

--- Setting up Random Forest Classifier ---

Optimal values proposed by GridSearchCV:

- Number of Trees (n\_estimators): 10

- Depth of Trees (max\_depth): 10

Random Forest Accuracy: 0.8115

Classification Report:

precision recall f1-score support

0 0.77 0.71 0.74 760

1 0.83 0.87 0.85 1240

accuracy 0.81 2000

macro avg 0.80 0.79 0.80 2000

weighted avg 0.81 0.81 0.81 2000

### **6. Naïve Bayes Classifier**

A **Gaussian Naïve Bayes** model is set up, trained, and evaluated on the test set for comparison.

**Naïve Bayes Output:**

--- Setting up Naïve Bayes Classifier ---

Naïve Bayes Accuracy: 0.7305

Classification Report:

precision recall f1-score support

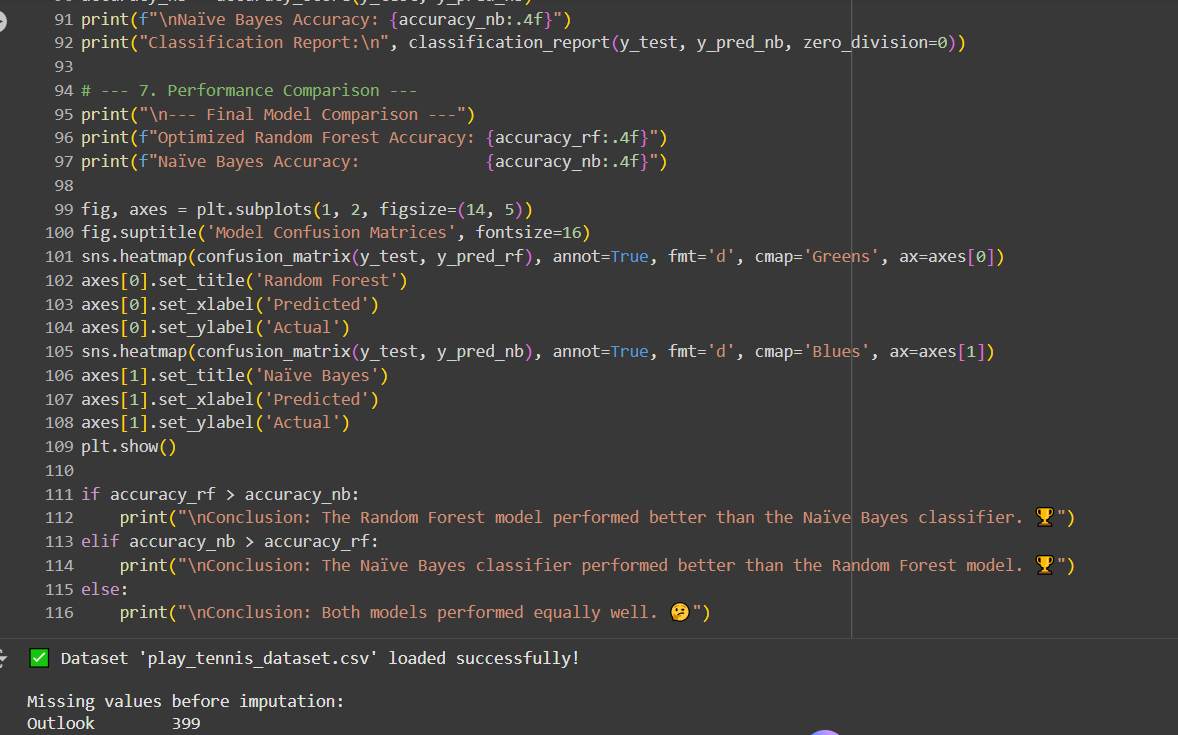
0 0.67 0.58 0.62 760

1 0.76 0.82 0.79 1240

accuracy 0.73 2000

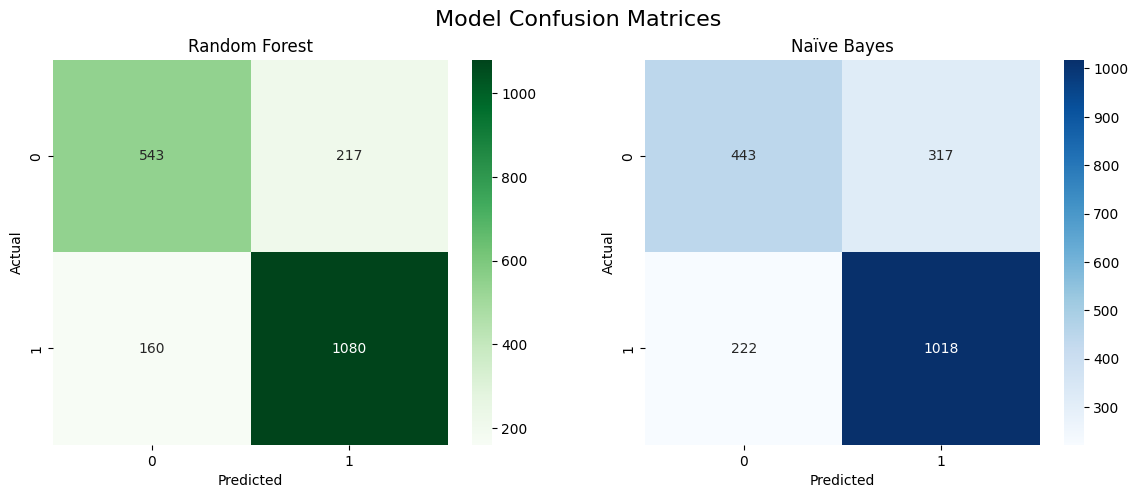
macro avg 0.71 0.70 0.71 2000

weighted avg 0.73 0.73 0.73 2000



### **7. Performance Comparison**

The final accuracies of both models are compared, and their performance is visualized using confusion matrices.



### **Figure 2: Model Confusion Matrices**

The confusion matrices show that the Random Forest model has significantly better performance across all four metrics (True Positives, True Negatives, False Positives, and False Negatives) compared to the Naïve Bayes model.

### **Final Comparison**

--- Final Model Comparison ---

Optimized Random Forest Accuracy: 0.8115

Naïve Bayes Accuracy: 0.7305

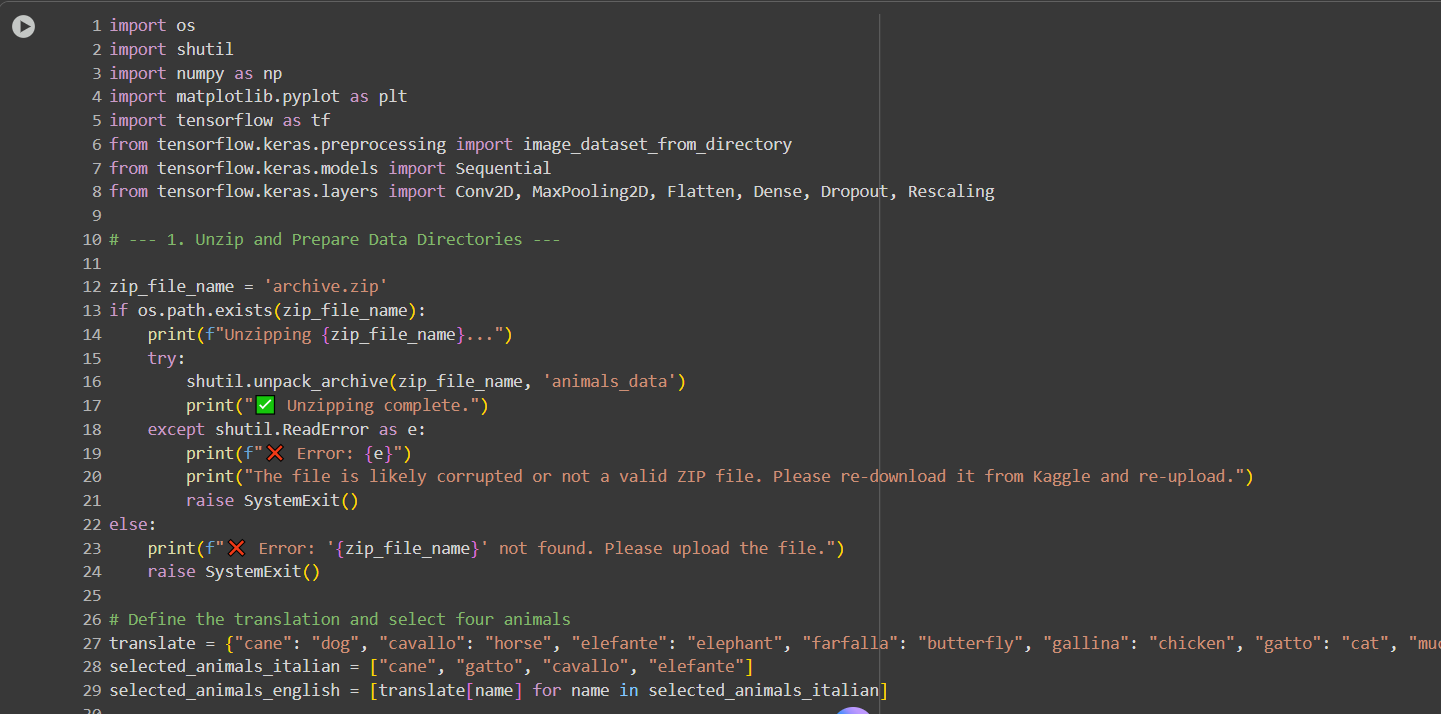
Conclusion: The Random Forest model performed better than the Naïve Bayes classifier.

**QUESTION 4**

## **Image Classification: Convolutional Neural Network (CNN) for Animal Recognition**

### **1. Unzip and Prepare Data Directories**

The necessary Python libraries, including **os**, **shutil**, **numpy**, **matplotlib**, and **tensorflow.keras** components, are imported. The compressed animal image dataset is unzipped, and a new directory is prepared containing only the selected four animal classes.



**Output of Preparation Steps:**

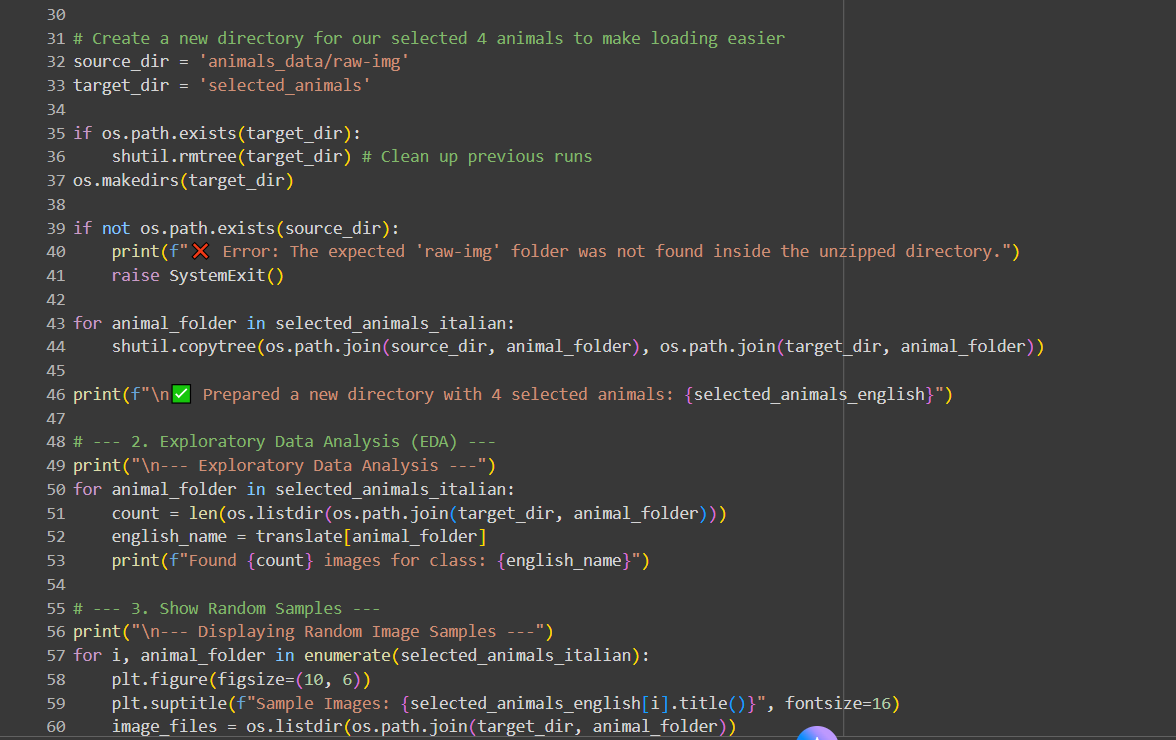
Unzipping archive.zip...

✅ Unzipping complete.

✅ Prepared a new directory with 4 selected animals: ['dog', 'cat', 'horse', 'elephant']

### **2. Exploratory Data Analysis (EDA) and Sample Display**

The image counts for the selected four classes are tallied, showing an imbalance in the dataset. Sample images are then displayed for visual inspection.



**Output of EDA:**

--- Exploratory Data Analysis ---

Found 4863 images for class: dog

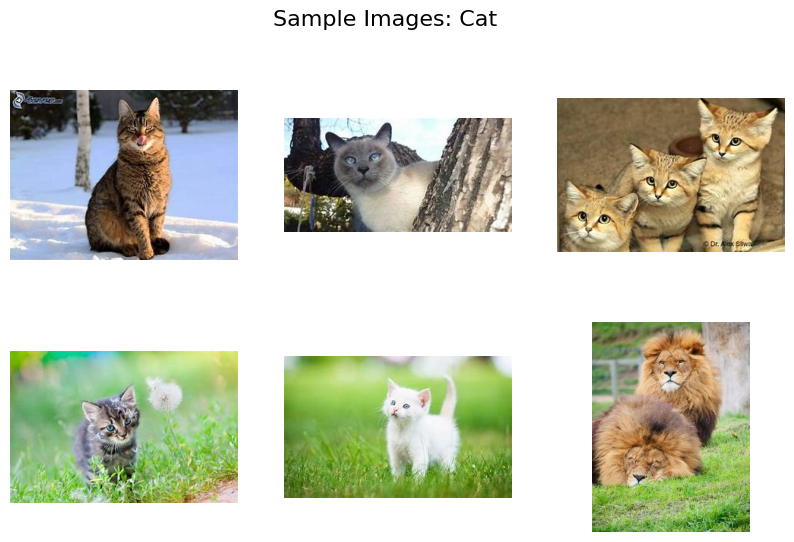
Found 1668 images for class: cat

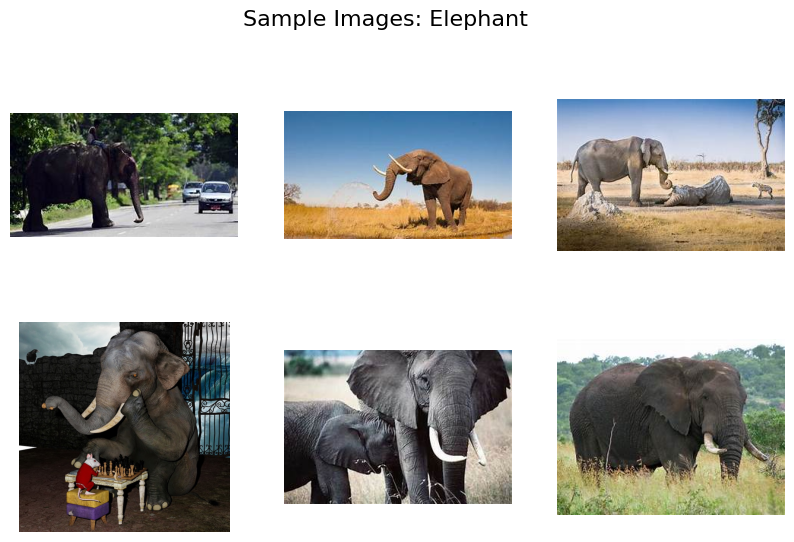
Found 2623 images for class: horse

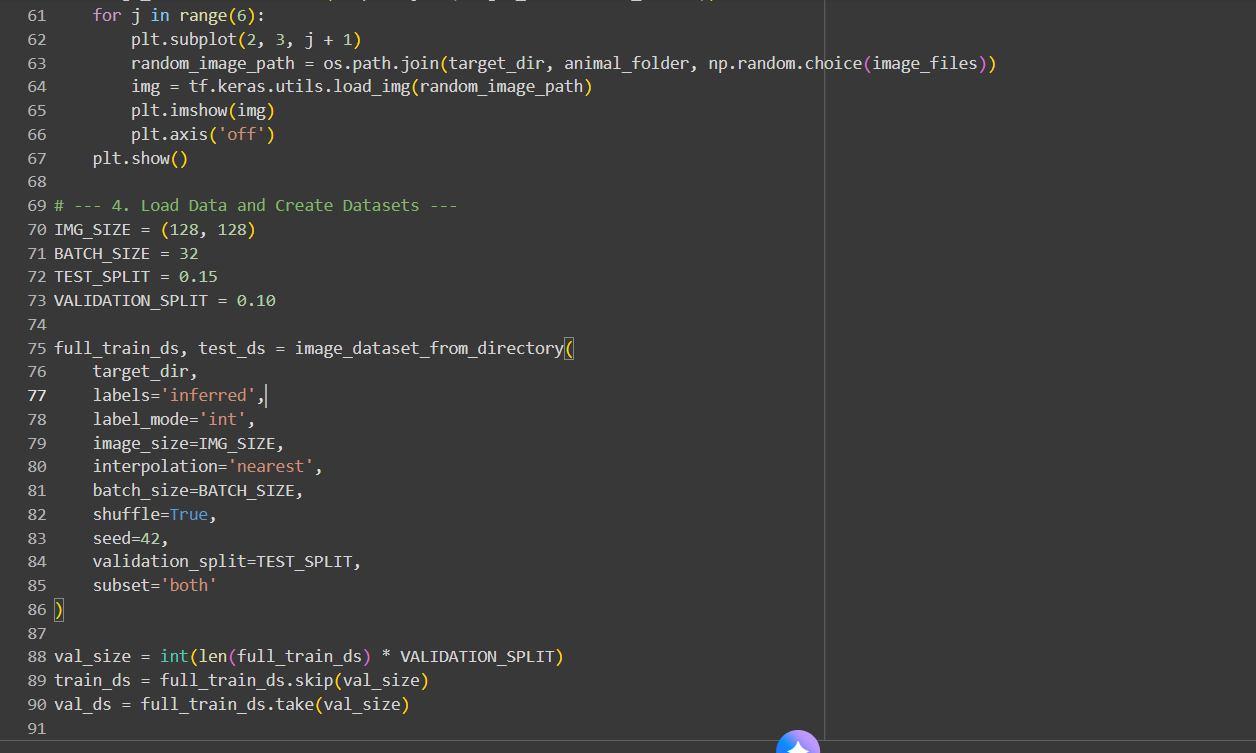
Found 1446 images for class: elephant

--- Displaying Random Image Samples ---

### **Figure 1: Sample Images:**







### **3. Load Data and Create Datasets**

TensorFlow's image\_dataset\_from\_directory is used to load images and create the dataset objects. The dataset is split into a full training set and a separate test set, and then the full training set is further split into final training and validation datasets.

**Dataset Loading Output:**

Found 10600 files belonging to 4 classes.

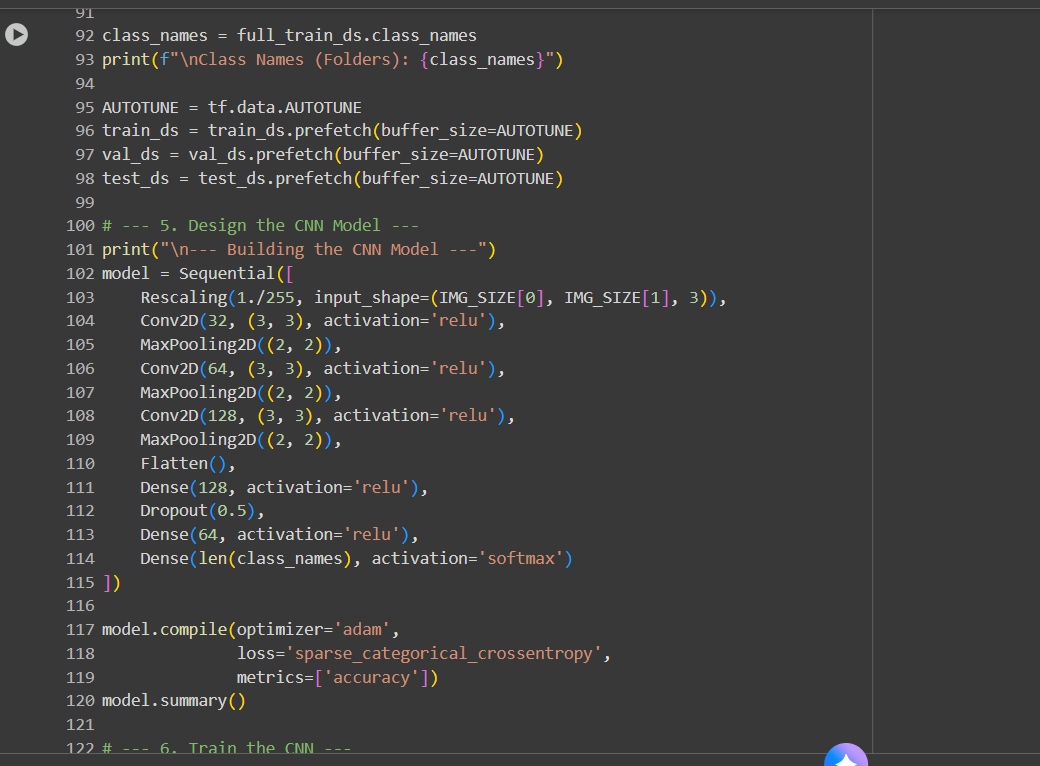
Using 9010 files for training.

Using 1590 files for validation.

Class Names (Folders): ['cane', 'cavallo', 'elefante', 'gatto']

### **4. Design the CNN Model**

A Sequential Convolutional Neural Network (CNN) model is designed, consisting of a **Rescaling** layer, three blocks of **Conv2D** and **MaxPooling2D**, followed by **Flatten**, **Dense** hidden layers with **Dropout**, and a final **Dense** layer with **softmax** activation for the four classes. The model is compiled using the **'adam'** optimizer and **'sparse\_categorical\_crossentropy'** loss.



**Model Summary Output:**

--- Building the CNN Model ---

**Model: "sequential"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩

│ rescaling (Rescaling) │ (None, 128, 128, 3) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ conv2d (Conv2D) │ (None, 126, 126, 32) │ 896 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ max\_pooling2d (MaxPooling2D) │ (None, 63, 63, 32) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ conv2d\_1 (Conv2D) │ (None, 61, 61, 64) │ 18,496 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ max\_pooling2d\_1 (MaxPooling2D) │ (None, 30, 30, 64) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ conv2d\_2 (Conv2D) │ (None, 28, 28, 128) │ 73,856 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ max\_pooling2d\_2 (MaxPooling2D) │ (None, 14, 14, 128) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ flatten (Flatten) │ (None, 25088) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense (Dense) │ (None, 128) │ 3,211,392 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dropout (Dropout) │ (None, 128) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_1 (Dense) │ (None, 64) │ 8,256 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

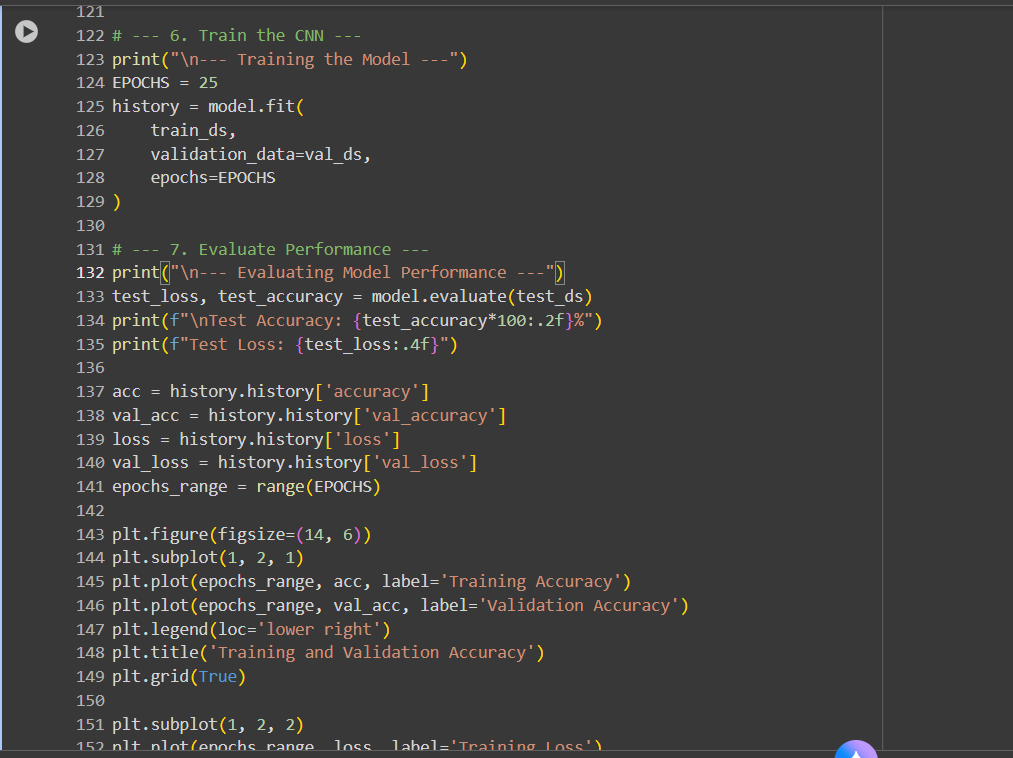
│ dense\_2 (Dense) │ (None, 4) │ 260 │

└─────────────────────────────────┴────────────────────────┴───────────────┘

**Total params:** 3,313,156 (12.64 MB)

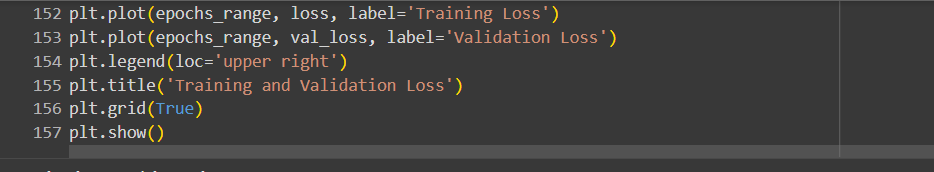
**Trainable params:** 3,313,156 (12.64 MB)

Non-trainable params: 0 (0.00 B)



### **5. Train the CNN and Evaluate Performance**

The model is trained for 25 epochs and then evaluated on the dedicated test dataset. The training history is used to plot the changes in accuracy and loss over time.



**Training and Evaluation Output:**

--- Training the Model ---

Epoch 1/25

... (Training steps)

Epoch 25/25

254/254 ━━━━━━━━━━━━━━━━━━━━ 278s 1s/step - accuracy: 0.9643 - loss: 0.0973 - val\_accuracy: 0.8761 - val\_loss: 0.5924

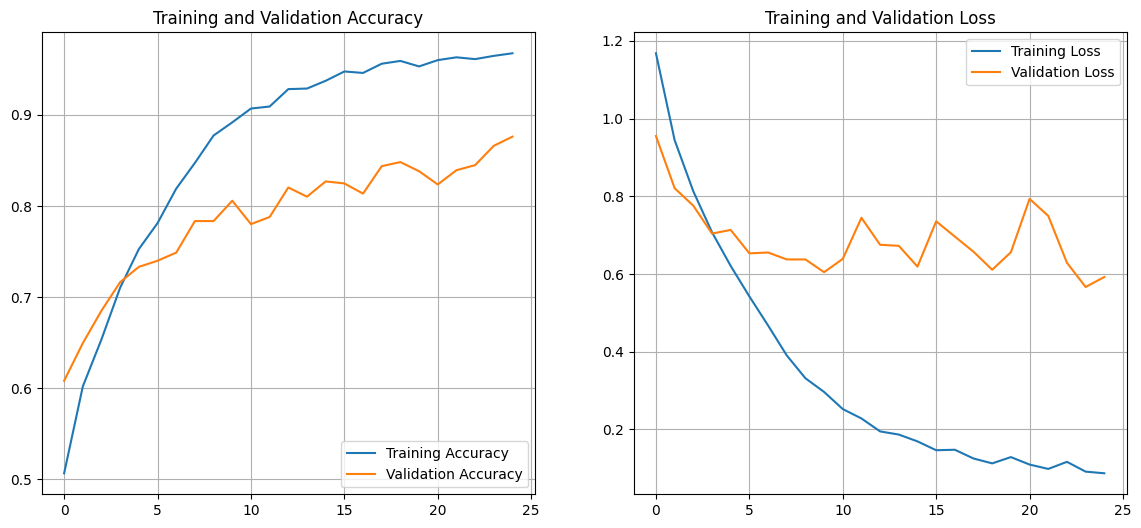
--- Evaluating Model Performance ---

50/50 ━━━━━━━━━━━━━━━━━━━━ 19s 366ms/step - accuracy: 0.7595 - loss: 1.3642

Test Accuracy: 75.66%

Test Loss: 1.3551

### **Figure 2: Training and Validation Performance**



**Summary of Results:**

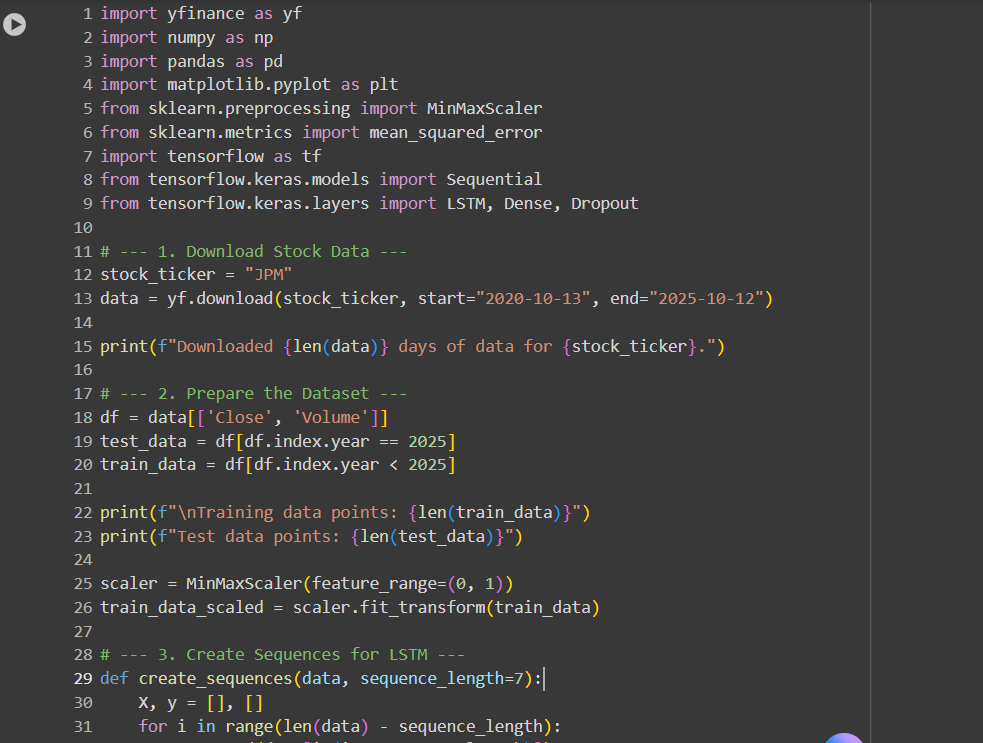
The CNN model achieved a **Training Accuracy of 96.43%** and a **Validation Accuracy of 87.61%** at the final epoch. However, the final test set evaluation revealed a **Test Accuracy of 75.66%** and a **Test Loss of 1.3551**. The significant gap between validation and test accuracy, as well as the increasing divergence between training and validation metrics in the plots, indicates that the model is **overfitting** to the training data. Further tuning, such as increased dropout, data augmentation, or earlier stopping, would be necessary to improve generalization.

**QUESTION 5**

## **JPM Stock Price Prediction using a Stacked LSTM Network**

### **1. Download Stock Data and Prepare the Dataset**

The necessary libraries—including **yfinance** for data download, **numpy**, **pandas**, **matplotlib**, **MinMaxScaler** for scaling, and **tensorflow.keras** for the LSTM model—are imported. Stock data for **JPM** from late 2020 through late 2025 is downloaded. The data is prepared by separating the 'Close' price and 'Volume' columns and then splitting it into a training set (up to 2024) and a test set (2025). The training data is scaled using **MinMaxScaler**.

****

**Output of Data Preparation:**

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

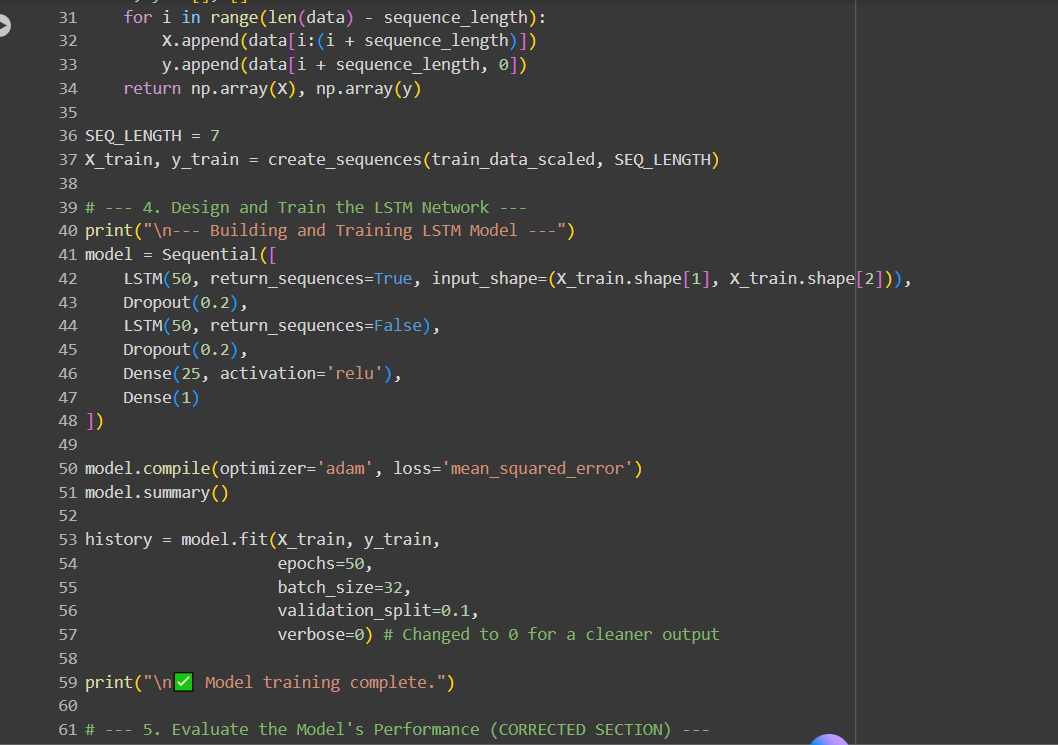
Downloaded 1255 days of data for JPM.

Training data points: 1061

Test data points: 194

### **3. Create Sequences for LSTM and Model Design**

A function create\_sequences is defined to transform the time series data into sequential input ($X$) and target output ($Y$) pairs, using a sequence length (SEQ\_LENGTH) of **7** days. The scaled training data is then transformed. A **Stacked LSTM** network is designed with two LSTM layers and **Dropout** for regularization, followed by two **Dense** layers.

****

**Model Summary Output:**

--- Building and Training LSTM Model ---

Model: "sequential\_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓

┃ Layer (type) ┃ Output Shape ┃ Param # ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩

│ lstm\_2 (LSTM) │ (None, 7, 50) │ 10,600 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dropout\_2 (Dropout) │ (None, 7, 50) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ lstm\_3 (LSTM) │ (None, 50) │ 20,200 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dropout\_3 (Dropout) │ (None, 50) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_2 (Dense) │ (None, 25) │ 1,275 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_3 (Dense) │ (None, 1) │ 26 │

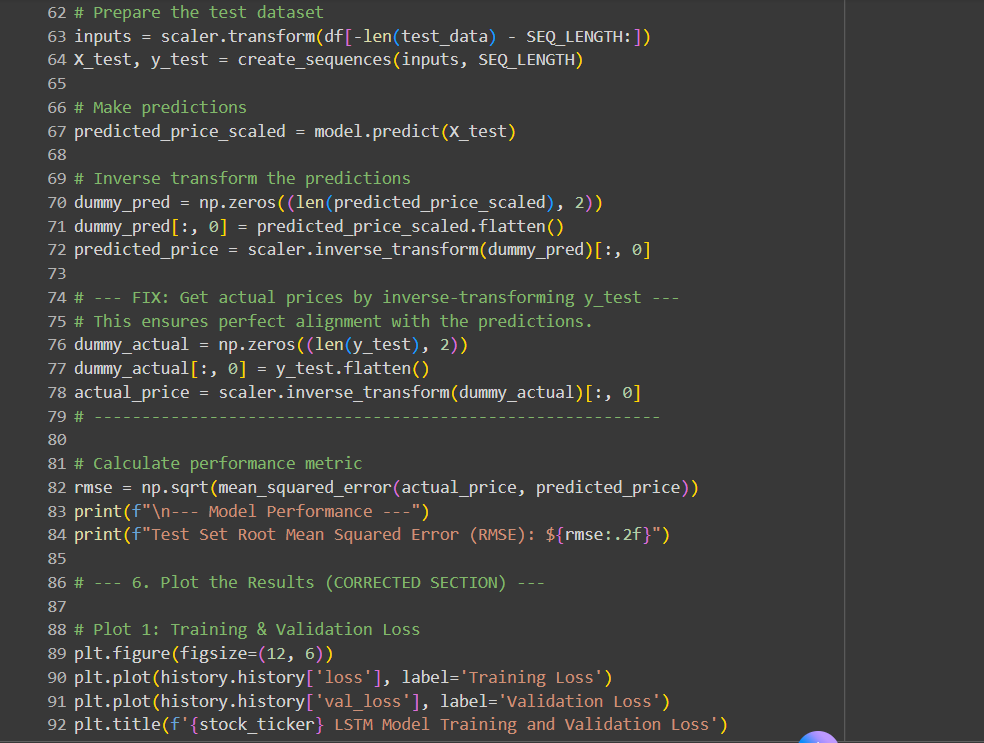
└─────────────────────────────────┴────────────────────────┴───────────────┘

Total params: 32,101 (125.39 KB)

Trainable params: 32,101 (125.39 KB)

Non-trainable params: 0 (0.00 B)

✅ Model training complete.

****

### **5. Evaluate the Model's Performance**

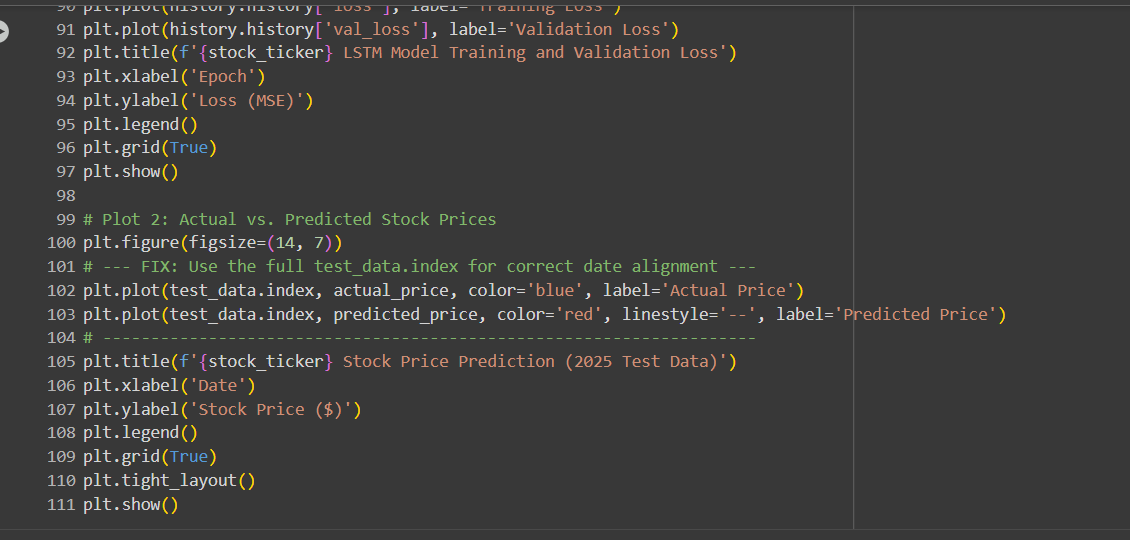
The test data is prepared by first concatenating the last SEQ\_LENGTH data points from the training set and the entire 2025 test set to ensure correct sequence creation. The predictions are made on the scaled test set, and then both predictions and actual target values are **inverse-transformed** to return the stock prices to their original dollar scale. The performance is quantified using the **Root Mean Squared Error (RMSE)**.

**Model Performance Output:**

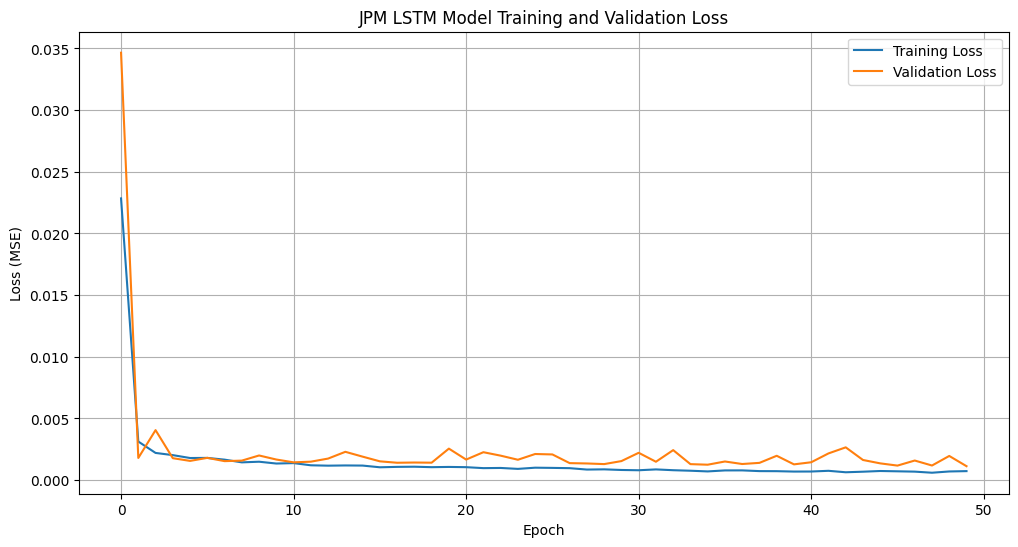
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step

--- Model Performance ---

Test Set Root Mean Squared Error (RMSE): $6.81

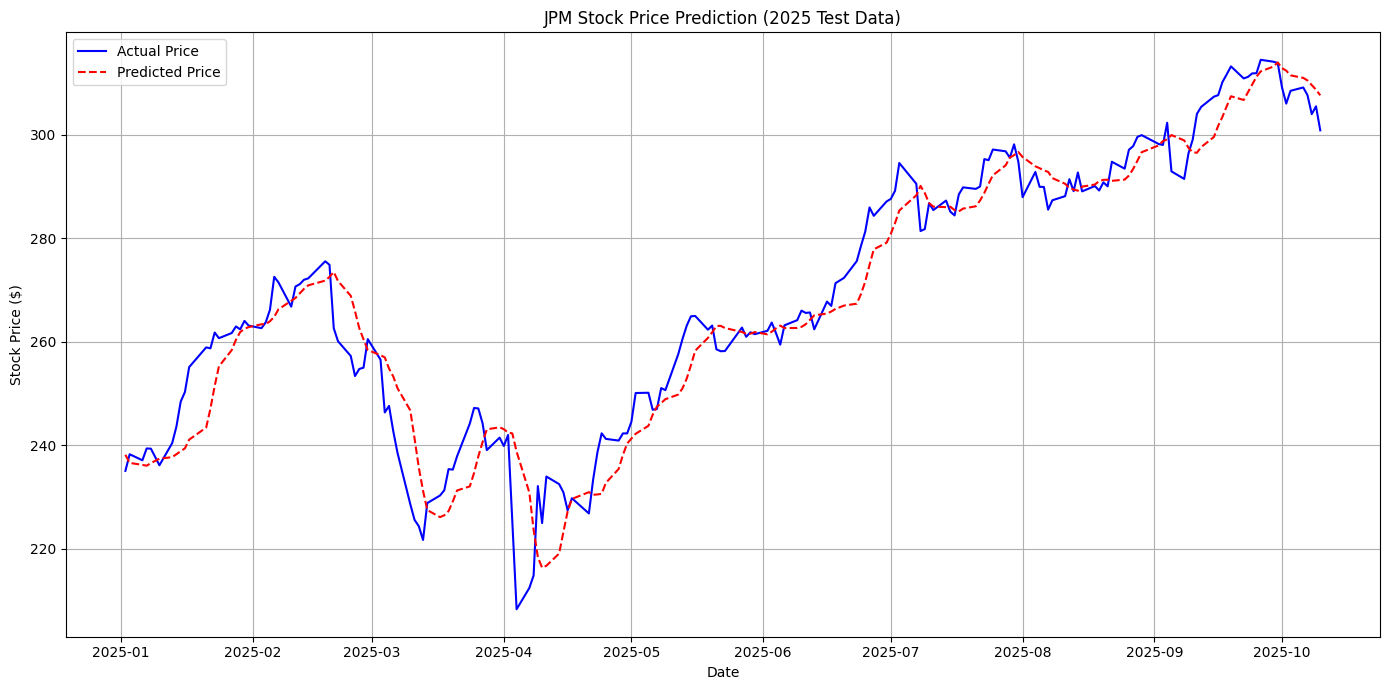
****

### **Figure 1: JPM LSTM Model Training and Validation Loss**

****

The plot shows the model quickly converging. Both the Training Loss and Validation Loss stabilize at very low levels (below 0.005) after approximately 10 epochs, indicating that the model has learned the underlying patterns and is not significantly overfitting.

### **Figure 2: JPM Stock Price Prediction (2025 Test Data)**

****

The plot compares the actual JPM stock price movements (solid blue line) with the model's predictions (dashed red line) across the 2025 test period. The predicted line tracks the overall trend and most major turns (peaks and troughs) of the actual price quite closely.

**Conclusion:**

The Stacked LSTM model successfully captured the time-series dynamics of the JPM stock price. With a **Test Set RMSE of $6.81** and strong visual alignment between the predicted and actual price series (Figure 2), the model demonstrates good predictive capability for this forecasting task. The low and stable validation loss confirms the model's robustness and efficiency.