Name : Atif Ullah Khan

Reg : B22F0059AI120

class : AI Green

```
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
/kaggle/input/healthcare-diabetes/Healthcare-Diabetes.csv
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

data=pd.read_csv('/kaggle/input/healthcare-diabetes/Healthcare-
Diabetes.csv')

data
```

```
        Id  Pregnancies  Glucose  BloodPressure  SkinThickness
Insulin   BMI  \
0        1            6      148             72             35
0  33.6
1        2            1       85             66             29
0  26.6
2        3            8      183             64              0
0  23.3
3        4            1       89             66             23
94  28.1
4        5            0      137             40             35
168  43.1
...    ...          ...      ...            ...            ...        ..
.   ...
2763  2764            2       75             64             24
55  29.7
2764  2765            8      179             72             42
130  32.7
2765  2766            6       85             78              0
0  31.2
2766  2767            0      129            110             46
130  67.1
2767  2768            2       81             72             15
76  30.1

      DiabetesPedigreeFunction  Age  Outcome
0                        0.627   50        1
1                        0.351   31        0
2                        0.672   32        1
3                        0.167   21        0
4                        2.288   33        1
...                        ...  ...      ...
2763                     0.370   33        0
2764                     0.719   36        1
2765                     0.382   42        0
2766                     0.319   26        1
2767                     0.547   25        0

[2768 rows x 10 columns]
```

```
data.isnull().sum()
```

```
Id                          0
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
```

```
Age                         0
Outcome                     0
dtype: int64
```

```
data.corr()
```

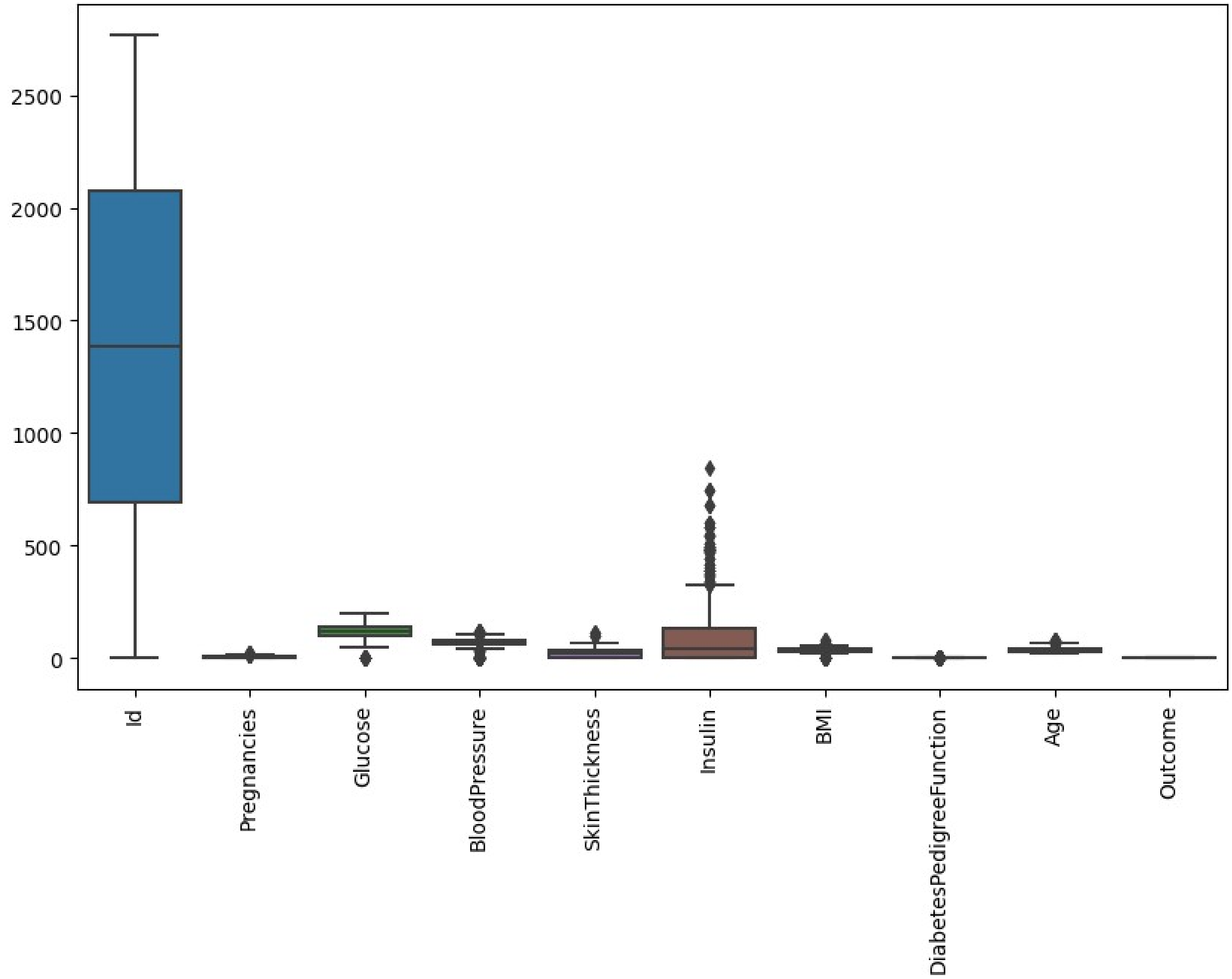
```
                                Id  Pregnancies   Glucose
BloodPressure  \
Id                        1.000000    -0.024222  0.015010
0.009717
Pregnancies              -0.024222     1.000000  0.122839
0.147253
Glucose                   0.015010     0.122839  1.000000
0.142095
BloodPressure             0.009717     0.147253  0.142095
1.000000
SkinThickness             0.017702    -0.068673  0.061023
0.201167
Insulin                   0.007359    -0.075734  0.323445
0.087823
BMI                       0.024007     0.018761  0.225308
0.281560
DiabetesPedigreeFunction -0.009695    -0.027731  0.127195
0.048471
Age                      -0.007404     0.540805  0.256958
0.238684
Outcome                  -0.006298     0.223796  0.460644
0.072900

                          SkinThickness   Insulin       BMI  \
Id                             0.017702  0.007359  0.024007
Pregnancies                   -0.068673 -0.075734  0.018761
Glucose                        0.061023  0.323445  0.225308
BloodPressure                  0.201167  0.087823  0.281560
SkinThickness                  1.000000  0.445345  0.393494
Insulin                        0.445345  1.000000  0.215926
BMI                            0.393494  0.215926  1.000000
DiabetesPedigreeFunction       0.179830  0.190500  0.129766
Age                           -0.111895 -0.073458  0.038175
Outcome                        0.075603  0.123646  0.280928

                          DiabetesPedigreeFunction       Age   Outcome

Id                                       -0.009695 -0.007404 -0.006298

Pregnancies                              -0.027731  0.540805  0.223796

Glucose                                   0.127195  0.256958  0.460644

BloodPressure                             0.048471  0.238684  0.072900
```

SkinThickness	0.179830	-0.111895	0.075603
Insulin	0.190500	-0.073458	0.123646
BMI	0.129766	0.038175	0.280928
DiabetesPedigreeFunction	1.000000	0.028544	0.160664
Age	0.028544	1.000000	0.237050
Outcome	0.160664	0.237050	1.000000

Code Explanation:

The code reads the **Healthcare-Diabetes** dataset using pandas, then creates a boxplot to visualize the distribution of the data for each feature. The plot is displayed with a figure size of 10x6 inches, and the x-axis labels are rotated by 90 degrees for better readability.

```
data=pd.read_csv('/kaggle/input/healthcare-diabetes/Healthcare-Diabetes.csv')
plt.figure(figsize=(10,6))
sns.boxplot(data)
plt.xticks(rotation=90)
plt.show()
```



Explanation of Outlier Removal Process:

1. **Quantiles Calculation**: The code first calculates the first (Q1) and third (Q3) quartiles of the dataset using the `quantile()` function.
2. **IQR Calculation**: The Interquartile Range (IQR) is calculated by subtracting Q1 from Q3 (IQR = Q3 - Q1), which measures the statistical spread of the middle 50% of the data.
3. **Outlier Detection**: Using the IQR, any data points that fall below (Q1 - 1.5 * IQR) or above (Q3 + 1.5 * IQR) are considered outliers.
4. **Data Cleaning**: The dataset is filtered to remove rows with any outliers in any of the columns. The result is stored in `data_cleaned`.
5. **Size Comparison**: Finally, the code prints the original and cleaned dataset sizes to show how many rows were removed due to outliers.

```
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1

data_cleaned = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 *
IQR))).any(axis=1)]
```

```
print(f"Original size: {data.shape}, Cleaned size:
{data_cleaned.shape}")

Original size: (2768, 10), Cleaned size: (2299, 10)
```

Explanation of Data Scaling:

1. **StandardScaler Initialization**: The `StandardScaler` is initialized to standardize the features in the dataset.
2. **Feature Scaling**: The `fit_transform()` method is applied to all features except the target variable (`Outcome`). This scales the data to have a mean of 0 and a standard deviation of 1.
3. **Target Variable Restoration**: The target variable (`Outcome`) is added back to the scaled data to maintain the structure of the dataset.
4. **Displaying the Scaled Data**: The first few rows of the scaled data are displayed using `head()`, showing the standardized features along with the target variable.

```
# Initialize StandardScaler
scaler = StandardScaler()

# Apply scaling (excluding the target 'Outcome')
data_scaled =
pd.DataFrame(scaler.fit_transform(data.drop(columns=['Outcome'])),
columns=data.columns[:-1])

# Add the target variable back
data_scaled['Outcome'] = data['Outcome'].values
print(data_scaled.head())

         Id  Pregnancies   Glucose  BloodPressure  SkinThickness
Insulin  \
0 -1.731425     0.679232  0.839738       0.149033       0.882845 -
0.713633
1 -1.730174    -0.825341 -1.127124      -0.163012       0.509169 -
0.713633
2 -1.728922     1.281062  1.932439      -0.267027      -1.296931 -
0.713633
3 -1.727671    -0.825341 -1.002244      -0.163012       0.135494
0.123547
4 -1.726419    -1.126256  0.496317      -1.515209       0.882845
0.782604

        BMI  DiabetesPedigreeFunction       Age  Outcome
0  0.181135                  0.478509  1.432495        1
1 -0.685773                 -0.369130 -0.181079        0
2 -1.094459                  0.616712 -0.096154        1
3 -0.500007                 -0.934224 -1.030329        0
4  1.357654                  5.579704 -0.011229        1
```

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import callbacks
```

Explanation of Feature and Target Separation:

- **X_scaled**: The feature matrix is created by dropping the 'Outcome' column from the `data_scaled` dataframe. This matrix contains all the input features.
- **y**: The target variable (`Outcome`) is extracted from the `data_scaled` dataframe, representing the outcome we aim to predict.

```python
X_scaled = data_scaled.drop(columns=['Outcome'])
y = data_scaled['Outcome']
```

Explanation of Data Splitting:

The dataset is split into training and testing sets using the `train_test_split` function from Scikit-learn:

- **X_scaled**: The feature matrix after scaling.
- **y**: The target variable (Outcome).
- **test_size=0.2**: 20% of the data is reserved for testing, while the remaining 80% is used for training.
- **random_state=42**: A fixed random seed to ensure reproducibility of the split.

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

Explanation of Model Architecture and Training Process:

1. **Early Stopping Callback**:
    - **min_delta=0.001**: The model will stop training if the improvement in validation loss is smaller than this value.
    - **patience=20**: If the validation loss does not improve for 20 consecutive epochs, training will stop.
    - **restore_best_weights=True**: The model will restore the best weights (i.e., the weights from the epoch with the lowest validation loss) after stopping.
2. **Model Architecture**:
    - **Dense Layer**: Each layer is fully connected (Dense) with a specified number of units and activation functions.
    - **ReLU Activation**: For hidden layers, ReLU is used as the activation function to introduce non-linearity.
    - **Dropout**: Dropout layers are added to reduce overfitting by randomly setting a fraction of input units to 0 during training.

- **Sigmoid Activation**: The output layer uses a sigmoid activation function to output values between 0 and 1, suitable for binary classification.

3. **Optimizer**:
   - **Adam Optimizer**: A widely-used adaptive optimizer with a learning rate of 0.00009 to minimize the binary cross-entropy loss.
4. **Model Compilation**: The model is compiled using binary cross-entropy loss, which is appropriate for binary classification tasks, and accuracy is used as the evaluation metric.
5. **Model Training**:
   - The model is trained on `X_train` and `y_train` with a batch size of 32 for up to 150 epochs.
   - A validation split of 0.2 is used to reserve 20% of the data for validation during training.
   - The `EarlyStopping` callback is used to halt training early if validation loss does not improve.

```
early_stopping = callbacks.EarlyStopping(
    min_delta=0.001,
    patience=20,
    restore_best_weights=True,
)


model = Sequential()

# Add layers to the model
model.add(Dense(units=32, kernel_initializer='uniform',
activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(units=32, kernel_initializer='uniform',
activation='relu'))
model.add(Dense(units=16, kernel_initializer='uniform',
activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(units=8, kernel_initializer='uniform',
activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=1, kernel_initializer='uniform',
activation='sigmoid'))


opt = Adam(learning_rate=0.00009)
model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])


history = model.fit(X_train, y_train, batch_size=32, epochs=150,
callbacks=[early_stopping], validation_split=0.2)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/
dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Epoch 1/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.6358 - loss:
0.6930 - val_accuracy: 0.6591 - val_loss: 0.6924
Epoch 2/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6451 - loss:
0.6923 - val_accuracy: 0.6591 - val_loss: 0.6916
Epoch 3/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6667 - loss:
0.6913 - val_accuracy: 0.6591 - val_loss: 0.6905
Epoch 4/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6495 - loss:
0.6903 - val_accuracy: 0.6591 - val_loss: 0.6891
Epoch 5/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6572 - loss:
0.6887 - val_accuracy: 0.6591 - val_loss: 0.6867
Epoch 6/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6492 - loss:
0.6860 - val_accuracy: 0.6591 - val_loss: 0.6817
Epoch 7/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6580 - loss:
0.6793 - val_accuracy: 0.6591 - val_loss: 0.6718
Epoch 8/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6484 - loss:
0.6681 - val_accuracy: 0.6591 - val_loss: 0.6544
Epoch 9/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6483 - loss:
0.6483 - val_accuracy: 0.6591 - val_loss: 0.6282
Epoch 10/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6426 - loss:
0.6211 - val_accuracy: 0.6591 - val_loss: 0.5972
Epoch 11/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6600 - loss:
0.5948 - val_accuracy: 0.6591 - val_loss: 0.5689
Epoch 12/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6551 - loss:
0.5682 - val_accuracy: 0.6591 - val_loss: 0.5475
Epoch 13/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6501 - loss:
0.5665 - val_accuracy: 0.6591 - val_loss: 0.5362
Epoch 14/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6587 - loss:
0.5373 - val_accuracy: 0.6591 - val_loss: 0.5274
Epoch 15/150
```

```
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6525 - loss: 0.5292 - val_accuracy: 0.6591 - val_loss: 0.5230
Epoch 16/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6958 - loss: 0.5300 - val_accuracy: 0.6546 - val_loss: 0.5201
Epoch 17/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6958 - loss: 0.5249 - val_accuracy: 0.7269 - val_loss: 0.5170
Epoch 18/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7198 - loss: 0.5327 - val_accuracy: 0.7698 - val_loss: 0.5147
Epoch 19/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7107 - loss: 0.5279 - val_accuracy: 0.7698 - val_loss: 0.5132
Epoch 20/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7430 - loss: 0.5267 - val_accuracy: 0.7698 - val_loss: 0.5120
Epoch 21/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7637 - loss: 0.5112 - val_accuracy: 0.7652 - val_loss: 0.5106
Epoch 22/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7695 - loss: 0.5095 - val_accuracy: 0.7607 - val_loss: 0.5092
Epoch 23/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7571 - loss: 0.5341 - val_accuracy: 0.7652 - val_loss: 0.5075
Epoch 24/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7808 - loss: 0.5253 - val_accuracy: 0.7630 - val_loss: 0.5056
Epoch 25/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7691 - loss: 0.5169 - val_accuracy: 0.7630 - val_loss: 0.5042
Epoch 26/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7819 - loss: 0.5060 - val_accuracy: 0.7652 - val_loss: 0.5028
Epoch 27/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7844 - loss: 0.4874 - val_accuracy: 0.7652 - val_loss: 0.5016
Epoch 28/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7780 - loss: 0.4952 - val_accuracy: 0.7630 - val_loss: 0.5001
Epoch 29/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7876 - loss: 0.4979 - val_accuracy: 0.7585 - val_loss: 0.4985
Epoch 30/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7729 - loss: 0.5028 - val_accuracy: 0.7540 - val_loss: 0.4968
Epoch 31/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7681 - loss:
```

```
0.5147 - val_accuracy: 0.7517 - val_loss: 0.4955
Epoch 32/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7714 - loss:
0.4784 - val_accuracy: 0.7517 - val_loss: 0.4940
Epoch 33/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7732 - loss:
0.4997 - val_accuracy: 0.7494 - val_loss: 0.4928
Epoch 34/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7547 - loss:
0.4991 - val_accuracy: 0.7494 - val_loss: 0.4920
Epoch 35/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7848 - loss:
0.4741 - val_accuracy: 0.7472 - val_loss: 0.4913
Epoch 36/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7503 - loss:
0.5139 - val_accuracy: 0.7494 - val_loss: 0.4904
Epoch 37/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7671 - loss:
0.4969 - val_accuracy: 0.7494 - val_loss: 0.4895
Epoch 38/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7564 - loss:
0.4971 - val_accuracy: 0.7494 - val_loss: 0.4889
Epoch 39/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7632 - loss:
0.4975 - val_accuracy: 0.7472 - val_loss: 0.4878
Epoch 40/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7725 - loss:
0.4910 - val_accuracy: 0.7472 - val_loss: 0.4871
Epoch 41/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7770 - loss:
0.4703 - val_accuracy: 0.7494 - val_loss: 0.4869
Epoch 42/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7542 - loss:
0.4955 - val_accuracy: 0.7494 - val_loss: 0.4867
Epoch 43/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7633 - loss:
0.5183 - val_accuracy: 0.7494 - val_loss: 0.4862
Epoch 44/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7725 - loss:
0.4878 - val_accuracy: 0.7472 - val_loss: 0.4857
Epoch 45/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7688 - loss:
0.4997 - val_accuracy: 0.7472 - val_loss: 0.4852
Epoch 46/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7727 - loss:
0.4816 - val_accuracy: 0.7472 - val_loss: 0.4852
Epoch 47/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7651 - loss:
0.4938 - val_accuracy: 0.7517 - val_loss: 0.4853
```

```
Epoch 48/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7738 - loss: 
0.4855 - val_accuracy: 0.7494 - val_loss: 0.4847
Epoch 49/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7665 - loss: 
0.4809 - val_accuracy: 0.7472 - val_loss: 0.4849
Epoch 50/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7685 - loss: 
0.4860 - val_accuracy: 0.7472 - val_loss: 0.4844
Epoch 51/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7810 - loss: 
0.4857 - val_accuracy: 0.7494 - val_loss: 0.4844
Epoch 52/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7839 - loss: 
0.4624 - val_accuracy: 0.7494 - val_loss: 0.4842
Epoch 53/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7725 - loss: 
0.4923 - val_accuracy: 0.7494 - val_loss: 0.4842
Epoch 54/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7887 - loss: 
0.4794 - val_accuracy: 0.7494 - val_loss: 0.4842
Epoch 55/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7737 - loss: 
0.4750 - val_accuracy: 0.7494 - val_loss: 0.4844
Epoch 56/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7917 - loss: 
0.4599 - val_accuracy: 0.7494 - val_loss: 0.4839
Epoch 57/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7751 - loss: 
0.4649 - val_accuracy: 0.7494 - val_loss: 0.4841
Epoch 58/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7795 - loss: 
0.4665 - val_accuracy: 0.7494 - val_loss: 0.4840
Epoch 59/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7660 - loss: 
0.5006 - val_accuracy: 0.7494 - val_loss: 0.4835
Epoch 60/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7827 - loss: 
0.4856 - val_accuracy: 0.7517 - val_loss: 0.4832
Epoch 61/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7527 - loss: 
0.4890 - val_accuracy: 0.7494 - val_loss: 0.4828
Epoch 62/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7850 - loss: 
0.4687 - val_accuracy: 0.7472 - val_loss: 0.4827
Epoch 63/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7938 - loss: 
0.4600 - val_accuracy: 0.7540 - val_loss: 0.4824
Epoch 64/150
```

```
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7576 - loss:
0.4949 - val_accuracy: 0.7472 - val_loss: 0.4823
Epoch 65/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8097 - loss:
0.4608 - val_accuracy: 0.7494 - val_loss: 0.4824
Epoch 66/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7526 - loss:
0.4959 - val_accuracy: 0.7449 - val_loss: 0.4824
Epoch 67/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7610 - loss:
0.4960 - val_accuracy: 0.7494 - val_loss: 0.4822
Epoch 68/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7835 - loss:
0.4741 - val_accuracy: 0.7494 - val_loss: 0.4817
Epoch 69/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7823 - loss:
0.4668 - val_accuracy: 0.7517 - val_loss: 0.4815
Epoch 70/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7904 - loss:
0.4717 - val_accuracy: 0.7517 - val_loss: 0.4817
Epoch 71/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7740 - loss:
0.4789 - val_accuracy: 0.7540 - val_loss: 0.4817
Epoch 72/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7890 - loss:
0.4598 - val_accuracy: 0.7494 - val_loss: 0.4818
Epoch 73/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7712 - loss:
0.4938 - val_accuracy: 0.7472 - val_loss: 0.4814
Epoch 74/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7902 - loss:
0.4797 - val_accuracy: 0.7472 - val_loss: 0.4814
Epoch 75/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8036 - loss:
0.4377 - val_accuracy: 0.7472 - val_loss: 0.4810
Epoch 76/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7836 - loss:
0.4778 - val_accuracy: 0.7517 - val_loss: 0.4809
Epoch 77/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7699 - loss:
0.4819 - val_accuracy: 0.7517 - val_loss: 0.4810
Epoch 78/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7792 - loss:
0.4808 - val_accuracy: 0.7517 - val_loss: 0.4806
Epoch 79/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7780 - loss:
0.4711 - val_accuracy: 0.7517 - val_loss: 0.4803
Epoch 80/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7659 - loss:
```

```
0.4863 - val_accuracy: 0.7517 - val_loss: 0.4805
Epoch 81/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7850 - loss:
0.4800 - val_accuracy: 0.7517 - val_loss: 0.4806
Epoch 82/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7763 - loss:
0.4706 - val_accuracy: 0.7517 - val_loss: 0.4802
Epoch 83/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7664 - loss:
0.4839 - val_accuracy: 0.7540 - val_loss: 0.4801
Epoch 84/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7808 - loss:
0.4840 - val_accuracy: 0.7517 - val_loss: 0.4800
Epoch 85/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7716 - loss:
0.4667 - val_accuracy: 0.7540 - val_loss: 0.4799
Epoch 86/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7870 - loss:
0.4605 - val_accuracy: 0.7562 - val_loss: 0.4798
Epoch 87/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8012 - loss:
0.4565 - val_accuracy: 0.7585 - val_loss: 0.4800
Epoch 88/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7976 - loss:
0.4796 - val_accuracy: 0.7540 - val_loss: 0.4801
Epoch 89/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7775 - loss:
0.4634 - val_accuracy: 0.7494 - val_loss: 0.4798
Epoch 90/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7766 - loss:
0.4803 - val_accuracy: 0.7494 - val_loss: 0.4796
Epoch 91/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7706 - loss:
0.4834 - val_accuracy: 0.7517 - val_loss: 0.4794
Epoch 92/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7712 - loss:
0.4695 - val_accuracy: 0.7540 - val_loss: 0.4796
Epoch 93/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7763 - loss:
0.4797 - val_accuracy: 0.7517 - val_loss: 0.4796
Epoch 94/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7815 - loss:
0.4820 - val_accuracy: 0.7540 - val_loss: 0.4794
Epoch 95/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7809 - loss:
0.4719 - val_accuracy: 0.7517 - val_loss: 0.4791
Epoch 96/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7866 - loss:
0.4708 - val_accuracy: 0.7517 - val_loss: 0.4790
```

```
Epoch 97/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7857 - loss:
0.4616 - val_accuracy: 0.7517 - val_loss: 0.4789
Epoch 98/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7662 - loss:
0.5030 - val_accuracy: 0.7517 - val_loss: 0.4787
Epoch 99/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7892 - loss:
0.4598 - val_accuracy: 0.7494 - val_loss: 0.4785
Epoch 100/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7831 - loss:
0.4798 - val_accuracy: 0.7494 - val_loss: 0.4782
Epoch 101/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7874 - loss:
0.4583 - val_accuracy: 0.7494 - val_loss: 0.4778
Epoch 102/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7921 - loss:
0.4390 - val_accuracy: 0.7494 - val_loss: 0.4776
Epoch 103/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7959 - loss:
0.4597 - val_accuracy: 0.7517 - val_loss: 0.4772
Epoch 104/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7878 - loss:
0.4716 - val_accuracy: 0.7540 - val_loss: 0.4772
Epoch 105/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7796 - loss:
0.5063 - val_accuracy: 0.7540 - val_loss: 0.4773
Epoch 106/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7914 - loss:
0.4645 - val_accuracy: 0.7540 - val_loss: 0.4775
Epoch 107/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8013 - loss:
0.4689 - val_accuracy: 0.7540 - val_loss: 0.4778
Epoch 108/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7797 - loss:
0.4528 - val_accuracy: 0.7562 - val_loss: 0.4782
Epoch 109/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7883 - loss:
0.4645 - val_accuracy: 0.7562 - val_loss: 0.4775
Epoch 110/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7892 - loss:
0.4676 - val_accuracy: 0.7517 - val_loss: 0.4776
Epoch 111/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7880 - loss:
0.4678 - val_accuracy: 0.7540 - val_loss: 0.4772
Epoch 112/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7724 - loss:
0.4790 - val_accuracy: 0.7540 - val_loss: 0.4770
Epoch 113/150
```

```
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7905 - loss:
0.4467 - val_accuracy: 0.7540 - val_loss: 0.4772
Epoch 114/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7861 - loss:
0.4779 - val_accuracy: 0.7540 - val_loss: 0.4769
Epoch 115/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7867 - loss:
0.4716 - val_accuracy: 0.7562 - val_loss: 0.4766
Epoch 116/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7835 - loss:
0.4841 - val_accuracy: 0.7562 - val_loss: 0.4764
Epoch 117/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7975 - loss:
0.4530 - val_accuracy: 0.7562 - val_loss: 0.4764
Epoch 118/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7838 - loss:
0.4716 - val_accuracy: 0.7540 - val_loss: 0.4764
Epoch 119/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7713 - loss:
0.4754 - val_accuracy: 0.7540 - val_loss: 0.4756
Epoch 120/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7687 - loss:
0.4978 - val_accuracy: 0.7562 - val_loss: 0.4752
Epoch 121/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7954 - loss:
0.4622 - val_accuracy: 0.7562 - val_loss: 0.4754
Epoch 122/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7749 - loss:
0.4803 - val_accuracy: 0.7585 - val_loss: 0.4753
Epoch 123/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7607 - loss:
0.5052 - val_accuracy: 0.7562 - val_loss: 0.4749
Epoch 124/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7883 - loss:
0.4644 - val_accuracy: 0.7540 - val_loss: 0.4749
Epoch 125/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7942 - loss:
0.4715 - val_accuracy: 0.7562 - val_loss: 0.4753
Epoch 126/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7855 - loss:
0.4678 - val_accuracy: 0.7540 - val_loss: 0.4749
Epoch 127/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7826 - loss:
0.4770 - val_accuracy: 0.7562 - val_loss: 0.4746
Epoch 128/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7894 - loss:
0.4835 - val_accuracy: 0.7540 - val_loss: 0.4748
Epoch 129/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7892 - loss:
```

```
0.4717 - val_accuracy: 0.7540 - val_loss: 0.4746
Epoch 130/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7963 - loss:
0.4497 - val_accuracy: 0.7540 - val_loss: 0.4745
Epoch 131/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7991 - loss:
0.4456 - val_accuracy: 0.7540 - val_loss: 0.4741
Epoch 132/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7856 - loss:
0.4483 - val_accuracy: 0.7562 - val_loss: 0.4742
Epoch 133/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7771 - loss:
0.4715 - val_accuracy: 0.7562 - val_loss: 0.4741
Epoch 134/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7731 - loss:
0.4729 - val_accuracy: 0.7585 - val_loss: 0.4742
Epoch 135/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7749 - loss:
0.4759 - val_accuracy: 0.7585 - val_loss: 0.4745
Epoch 136/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7957 - loss:
0.4505 - val_accuracy: 0.7585 - val_loss: 0.4742
Epoch 137/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7804 - loss:
0.4673 - val_accuracy: 0.7585 - val_loss: 0.4738
Epoch 138/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8024 - loss:
0.4408 - val_accuracy: 0.7607 - val_loss: 0.4735
Epoch 139/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7611 - loss:
0.5283 - val_accuracy: 0.7607 - val_loss: 0.4737
Epoch 140/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8083 - loss:
0.4508 - val_accuracy: 0.7607 - val_loss: 0.4734
Epoch 141/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7997 - loss:
0.4444 - val_accuracy: 0.7540 - val_loss: 0.4735
Epoch 142/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8103 - loss:
0.4481 - val_accuracy: 0.7540 - val_loss: 0.4734
Epoch 143/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7909 - loss:
0.4802 - val_accuracy: 0.7540 - val_loss: 0.4730
Epoch 144/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7748 - loss:
0.4709 - val_accuracy: 0.7562 - val_loss: 0.4728
Epoch 145/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7947 - loss:
0.4519 - val_accuracy: 0.7540 - val_loss: 0.4731
```

```
Epoch 146/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7924 - loss:
0.4609 - val_accuracy: 0.7540 - val_loss: 0.4732
Epoch 147/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7870 - loss:
0.4553 - val_accuracy: 0.7607 - val_loss: 0.4737
Epoch 148/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7938 - loss:
0.4489 - val_accuracy: 0.7585 - val_loss: 0.4733
Epoch 149/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7885 - loss:
0.4704 - val_accuracy: 0.7585 - val_loss: 0.4738
Epoch 150/150
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7811 - loss:
0.4615 - val_accuracy: 0.7585 - val_loss: 0.4733
```

Code Explanation:

The code visualizes the training and validation accuracy and loss over epochs during the training of a machine learning model. It plots two graphs:

1. **Accuracy vs Epochs**: Shows the progression of training and validation accuracy.
2. **Loss vs Epochs**: Shows the progression of training and validation loss.

Both plots are displayed side by side using subplots, with appropriate labels and legends for clarity.

```python
import matplotlib.pyplot as plt

# Extracting the history of accuracy and loss
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Create a figure for the subplots
plt.figure(figsize=(14, 6))

# Plot Accuracy vs. Epochs
plt.subplot(1, 2, 1)
plt.plot(train_accuracy, label='Training Accuracy')
plt.plot(val_accuracy, label='Validation Accuracy')
plt.title('Accuracy vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot Loss vs. Epochs
plt.subplot(1, 2, 2)
plt.plot(train_loss, label='Training Loss')
```

```python
plt.plot(val_loss, label='Validation Loss')
plt.title('Loss vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Display the plots
plt.tight_layout()
plt.show()
```