Assignment-02

Name: Sai Kiran Mohanty

Registration No.: 2341013236

Section: 2c3

In [3]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, con
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)
tf.random.set_seed(42)
```

1. Data Loading

In [5]:

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequalit
df = pd.read_csv(url, sep=';')
df.head()
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quali
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	

In [6]:

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

Out[6]:

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
```

```
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

In [7]:

```python
df.describe()
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	15
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	

2. Exploratory Data Analysis (EDA)

In [9]:

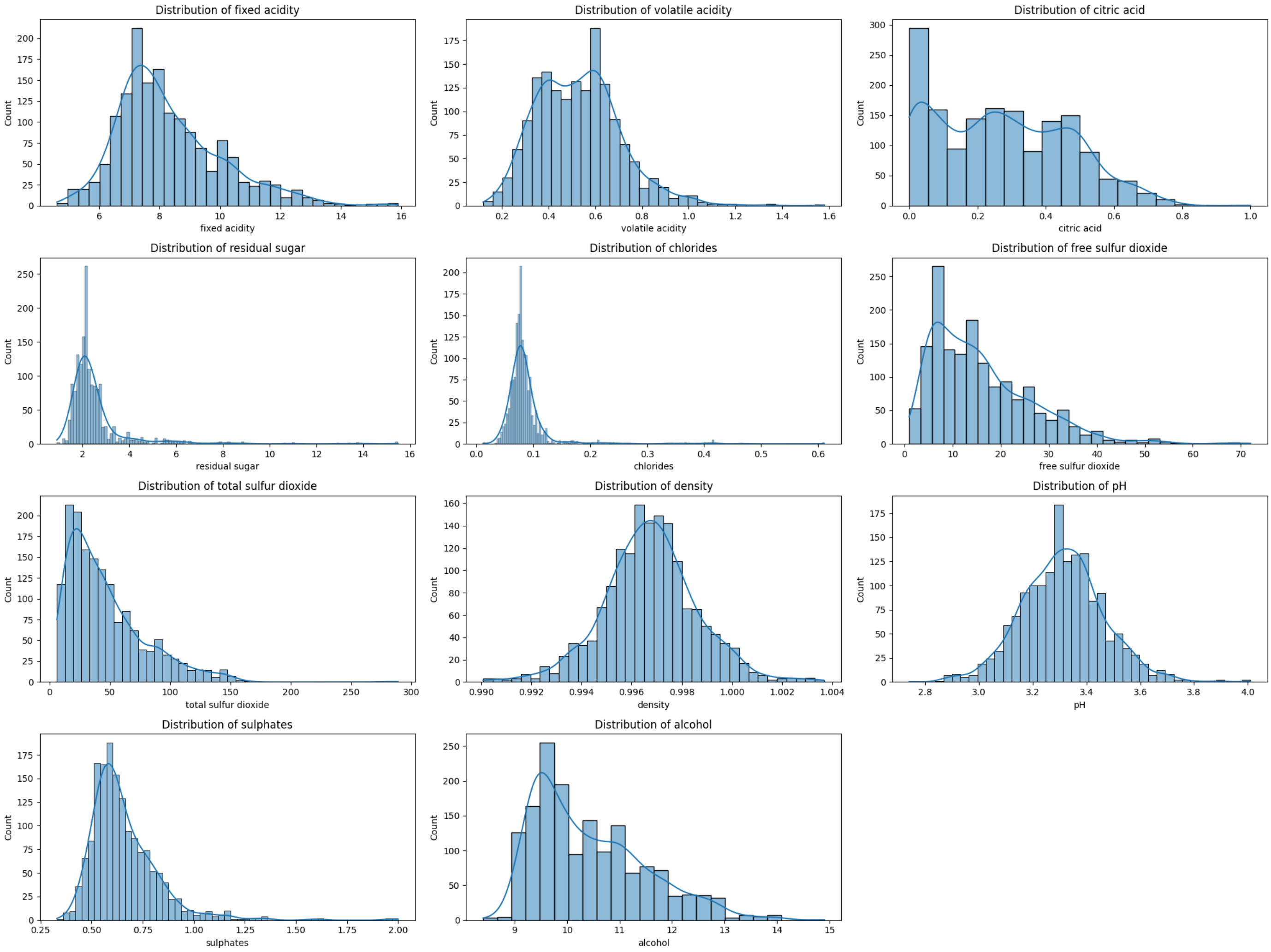
```python
df['quality_binary'] = (df['quality'] >= 6).astype(int)
df['quality_binary'].value_counts()
```

Out[9]:

```
quality_binary
1    855
0    744
Name: count, dtype: int64
```

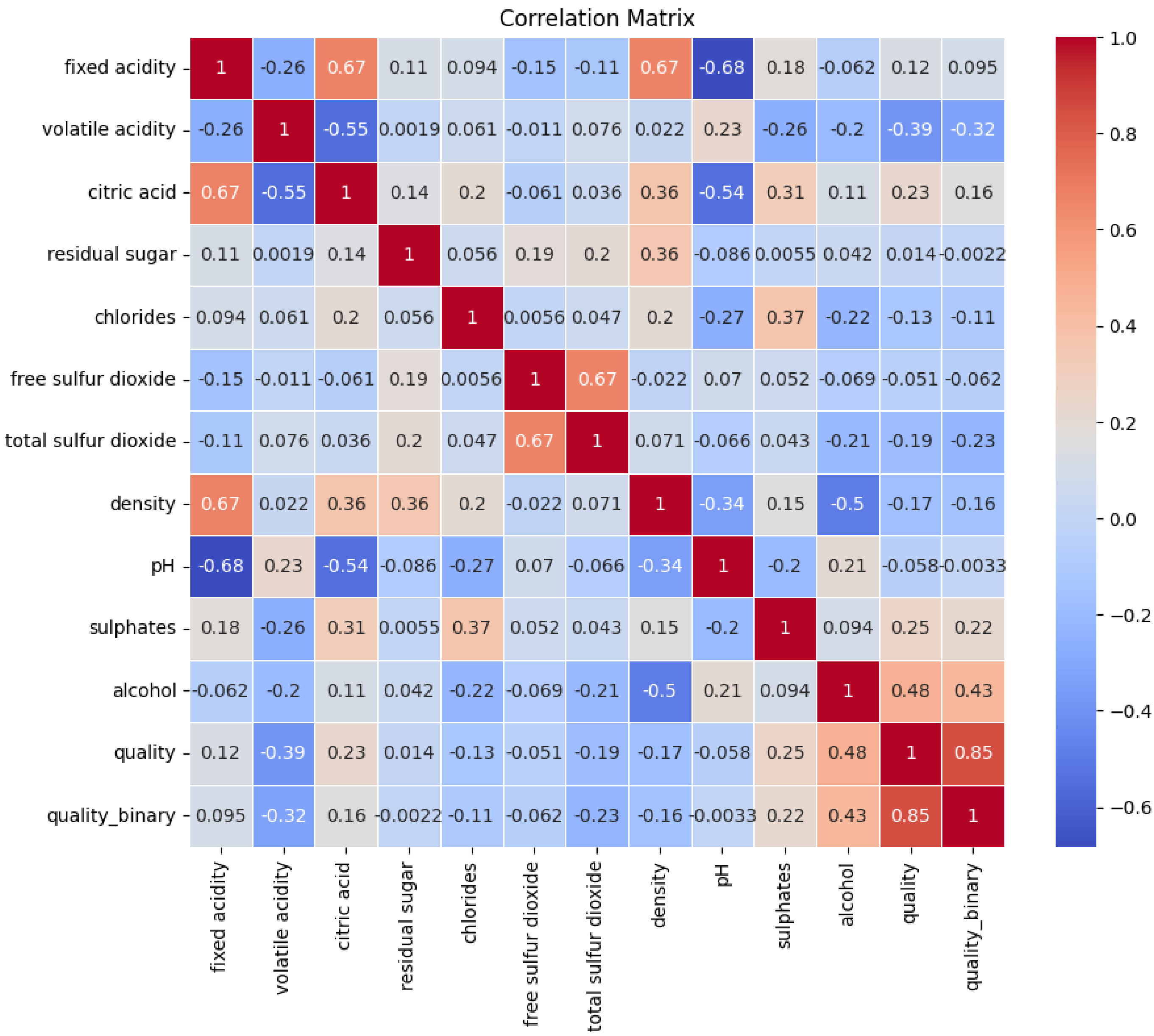
In [10]:

```python
plt.figure(figsize=(20, 15))
for i, col in enumerate(df.columns[:-2]):
    plt.subplot(4, 3, i+1)
    sns.histplot(data=df, x=col, kde=True)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```



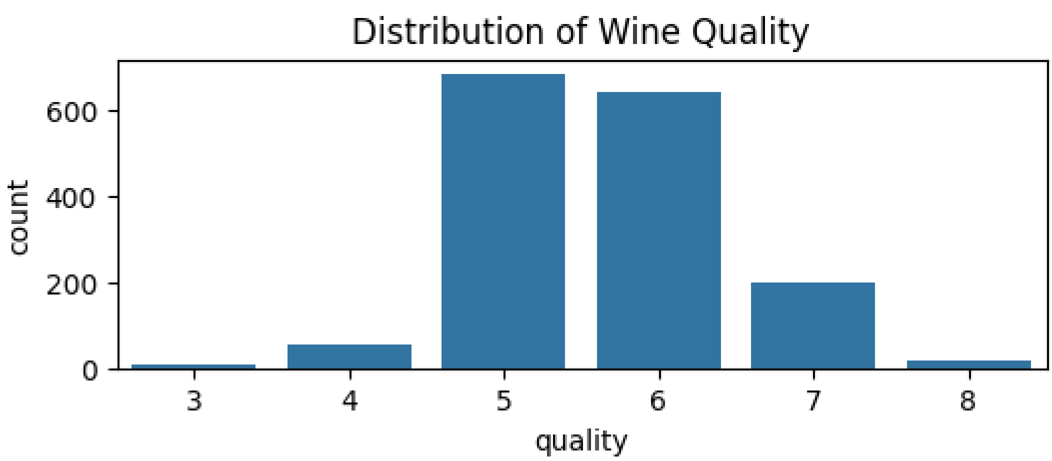
In [11]:

```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



In [12]:

```python
plt.figure(figsize=(6, 2))
sns.countplot(data=df, x='quality')
plt.title('Distribution of Wine Quality')
plt.show()
```



3. Data Preprocessing

In [14]:

```
X = df.drop(['quality', 'quality_binary'], axis=1)
y = df['quality_binary']
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: (1279, 11)
Testing set shape: (320, 11)
```

In [15]:

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

4. Model Building and Training

In [17]:

```
def train_evaluate_model(architecture, name, epochs=20, batch_size=32):
    model = Sequential()
    model.add(Dense(architecture[0], activation='relu', input_shape=(X_train_scaled.shap
    for units in architecture[1:]:
        model.add(Dense(units, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

    history = model.fit(X_train_scaled, y_train,epochs=epochs,batch_size=batch_size,vali

    y_pred_proba = model.predict(X_test_scaled)
    y_pred = (y_pred_proba > 0.5).astype(int)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f"\n{name} Performance:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4, 2))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    plt.figure(figsize=(6, 3))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss')
```

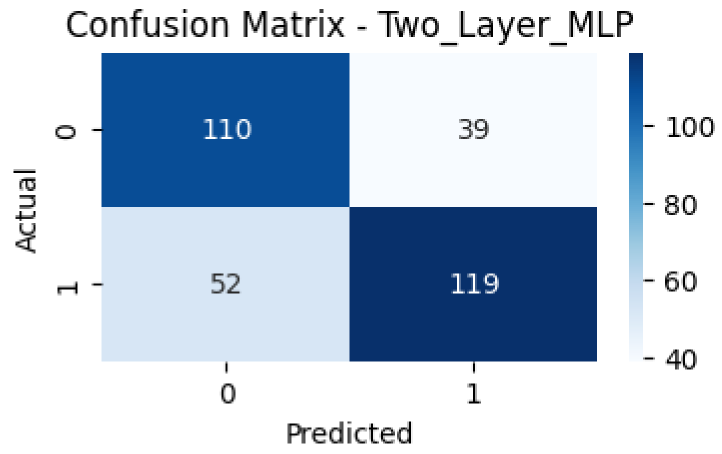
```
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'Loss Curves - {name}')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'Accuracy Curves - {name}')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.tight_layout()
    plt.show()

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

    return model, history, accuracy, precision, recall, f1
```

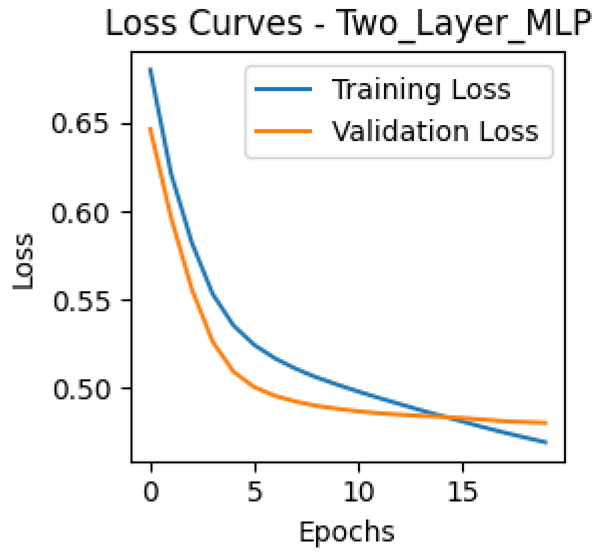
In [18]:

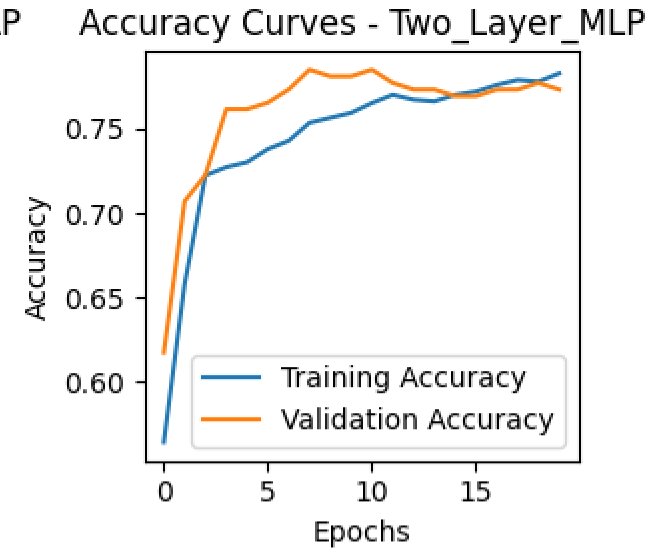
```
architectures = [
    ([32, 16], "Two_Layer_MLP"),
    ([128, 64, 32, 16], "Deep_MLP")
]
results = []
for arch, name in architectures:
    model, history, accuracy, precision, recall, f1 = train_evaluate_model(arch, name)
    results.append({
        'name': name,
        'architecture': str(arch),
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1
    })
```

```
Epoch 1/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.5360 - loss: 0.7032 - val_accuracy:
0.6172 - val_loss: 0.6466
Epoch 2/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6341 - loss: 0.6408 - val_accuracy:
0.7070 - val_loss: 0.5964
Epoch 3/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7017 - loss: 0.6052 - val_accuracy:
0.7227 - val_loss: 0.5558
Epoch 4/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7027 - loss: 0.5790 - val_accuracy:
0.7617 - val_loss: 0.5262
Epoch 5/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7099 - loss: 0.5625 - val_accuracy:
0.7617 - val_loss: 0.5092
Epoch 6/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7190 - loss: 0.5516 - val_accuracy:
0.7656 - val_loss: 0.5005
Epoch 7/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7237 - loss: 0.5436 - val_accuracy:
0.7734 - val_loss: 0.4955
```

```
Epoch 8/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7388 - loss: 0.5372 - val_accuracy:
0.7852 - val_loss: 0.4923
Epoch 9/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7436 - loss: 0.5318 - val_accuracy:
0.7812 - val_loss: 0.4899
Epoch 10/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7492 - loss: 0.5271 - val_accuracy:
0.7812 - val_loss: 0.4882
Epoch 11/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7507 - loss: 0.5226 - val_accuracy:
0.7852 - val_loss: 0.4868
Epoch 12/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7574 - loss: 0.5184 - val_accuracy:
0.7773 - val_loss: 0.4857
Epoch 13/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7522 - loss: 0.5142 - val_accuracy:
0.7734 - val_loss: 0.4849
Epoch 14/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7492 - loss: 0.5101 - val_accuracy:
0.7734 - val_loss: 0.4842
Epoch 15/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7502 - loss: 0.5063 - val_accuracy:
0.7695 - val_loss: 0.4837
Epoch 16/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7534 - loss: 0.5028 - val_accuracy:
0.7695 - val_loss: 0.4830
Epoch 17/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7584 - loss: 0.4991 - val_accuracy:
0.7734 - val_loss: 0.4821
Epoch 18/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7626 - loss: 0.4955 - val_accuracy:
0.7734 - val_loss: 0.4812
Epoch 19/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7614 - loss: 0.4922 - val_accuracy:
0.7773 - val_loss: 0.4807
Epoch 20/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7648 - loss: 0.4891 - val_accuracy:
0.7734 - val_loss: 0.4803
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Two_Layer_MLP Performance:
Accuracy: 0.7156
Precision: 0.7532
Recall: 0.6959
F1 Score: 0.7234
```



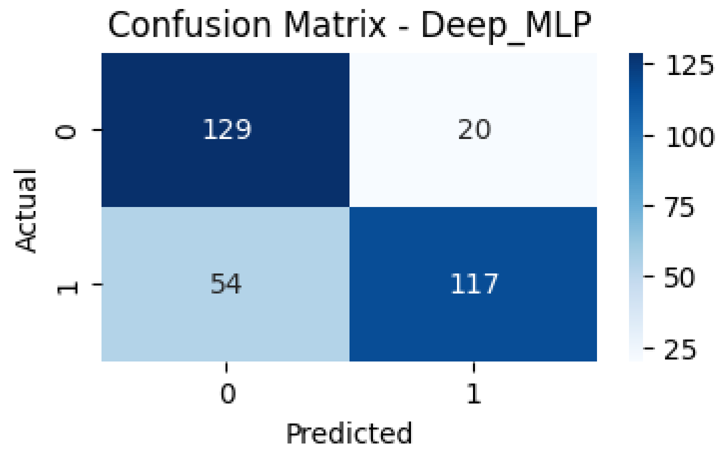


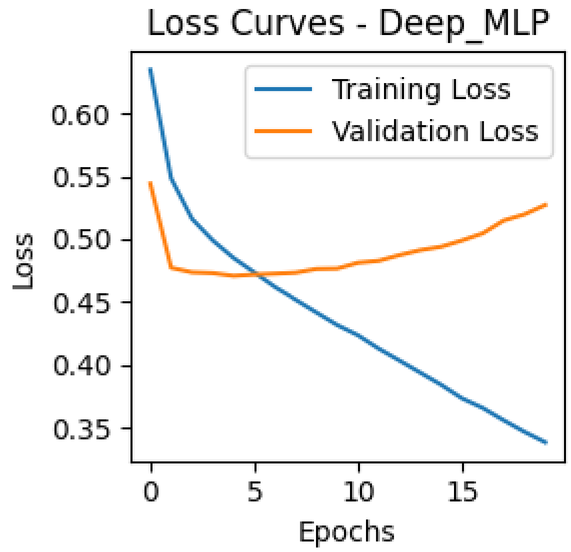


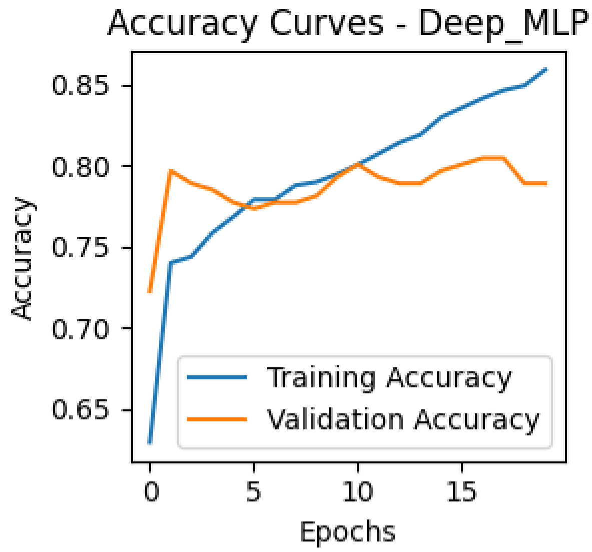

```
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.74      0.71       149
           1       0.75      0.70      0.72       171

    accuracy                           0.72       320
   macro avg       0.72      0.72      0.72       320
weighted avg       0.72      0.72      0.72       320
```

```
Epoch 1/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.5810 - loss: 0.6662 - val_accurac
y: 0.7227 - val_loss: 0.5441
Epoch 2/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7283 - loss: 0.5708 - val_accuracy:
0.7969 - val_loss: 0.4772
Epoch 3/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7243 - loss: 0.5418 - val_accuracy:
0.7891 - val_loss: 0.4735
Epoch 4/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7355 - loss: 0.5239 - val_accuracy:
0.7852 - val_loss: 0.4729
Epoch 5/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7487 - loss: 0.5098 - val_accuracy:
0.7773 - val_loss: 0.4707
Epoch 6/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7579 - loss: 0.4981 - val_accuracy:
```

```
0.7734 - val_loss: 0.4718
Epoch 7/20
32/32 0s 6ms/step - accuracy: 0.7612 - loss: 0.4866 - val_accuracy:
0.7773 - val_loss: 0.4725
Epoch 8/20
32/32 0s 4ms/step - accuracy: 0.7682 - loss: 0.4755 - val_accuracy:
0.7773 - val_loss: 0.4733
Epoch 9/20
32/32 0s 5ms/step - accuracy: 0.7683 - loss: 0.4645 - val_accuracy:
0.7812 - val_loss: 0.4762
Epoch 10/20
32/32 0s 4ms/step - accuracy: 0.7714 - loss: 0.4541 - val_accuracy:
0.7930 - val_loss: 0.4765
Epoch 11/20
32/32 0s 4ms/step - accuracy: 0.7769 - loss: 0.4451 - val_accuracy:
0.8008 - val_loss: 0.4812
Epoch 12/20
32/32 0s 4ms/step - accuracy: 0.7888 - loss: 0.4336 - val_accuracy:
0.7930 - val_loss: 0.4827
Epoch 13/20
32/32 0s 4ms/step - accuracy: 0.7965 - loss: 0.4239 - val_accuracy:
0.7891 - val_loss: 0.4873
Epoch 14/20
32/32 0s 4ms/step - accuracy: 0.8074 - loss: 0.4135 - val_accuracy:
0.7891 - val_loss: 0.4912
Epoch 15/20
32/32 0s 4ms/step - accuracy: 0.8191 - loss: 0.4040 - val_accuracy:
0.7969 - val_loss: 0.4939
Epoch 16/20
32/32 0s 4ms/step - accuracy: 0.8215 - loss: 0.3935 - val_accuracy:
0.8008 - val_loss: 0.4989
Epoch 17/20
32/32 0s 4ms/step - accuracy: 0.8285 - loss: 0.3853 - val_accuracy:
0.8047 - val_loss: 0.5046
Epoch 18/20
32/32 0s 5ms/step - accuracy: 0.8368 - loss: 0.3758 - val_accuracy:
0.8047 - val_loss: 0.5148
Epoch 19/20
32/32 0s 4ms/step - accuracy: 0.8475 - loss: 0.3656 - val_accuracy:
0.7891 - val_loss: 0.5197
Epoch 20/20
32/32 0s 5ms/step - accuracy: 0.8560 - loss: 0.3567 - val_accuracy:
0.7891 - val_loss: 0.5271
10/10 0s 3ms/step

Deep_MLP Performance:
Accuracy: 0.7688
Precision: 0.8540
Recall: 0.6842
F1 Score: 0.7597
```








```
Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.87      0.78       149
           1       0.85      0.68      0.76       171

    accuracy                           0.77       320
   macro avg       0.78      0.77      0.77       320
weighted avg       0.78      0.77      0.77       320
```

In [19]:

```
results_df = pd.DataFrame(results)
print("\nModel Comparison:")
results_df
```

```
Model Comparison:
```

Out[19]:

	name	architecture	accuracy	precision	recall	f1
0	Two_Layer_MLP	[32, 16]	0.715625	0.753165	0.695906	0.723404
1	Deep_MLP	[128, 64, 32, 16]	0.768750	0.854015	0.684211	0.759740