Name: Kunal Singhal University Roll No. : 2315510107 Section: CB

Assignment Title: End-to-End ANN Model Implementation and Analysis Instructions:

1. Follow the tasks step by step in this notebook.
2. Write your code in the provided cells only.
3. Submit the following:

   Completed .ipynb file

A report (.pdf or .docx) with answers to the analysis questions provided at the end.

✅ Task 1: Import Required Libraries

In [ ]:

In [ ]:
```
import numpy as np
import pandas as pd
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 mean_absolute_error,mean_squared_error,r2_score
```

In [44]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

✅ Task 2: Load Dataset (Use from sklearn.datasets or any open dataset)

In [86]:
```
df = pd.read_csv("Dataset.csv")
```

In [87]:
```
df.head()
```

Out[87]:

	Surname	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMe
0	2023	668.0	33.0	3.0	0.00	2.0	1	
1	2024	627.0	33.0	1.0	0.00	2.0	1	
2	1236	678.0	40.0	10.0	0.00	2.0	1	
3	1362	581.0	34.0	2.0	148882.54	1.0	1	
4	491	716.0	33.0	5.0	0.00	2.0	1	

5 rows × 25 columns

In [88]:
```
df.tail()
```

Out[88]:

	Surname	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsAct
175023	2226	644.0	28.0	7.0	155060.41	1.0	1	
175024	2846	800.0	29.0	2.0	0.00	2.0	0	
175025	1999	771.0	39.0	5.0	0.00	2.0	1	
175026	1336	516.0	35.0	10.0	57369.61	1.0	1	
175027	1570	709.0	36.0	7.0	0.00	1.0	0	

5 rows × 25 columns

✅ Task 3: Perform Exploratory Data Analysis (EDA)

1. Show basic info (shape, columns, datatypes)
2. Describe data
3. Use visualizations (histograms, boxplots, heatmaps, pairplots etc.)

Identify class imbalance if any

In [90]:
```
df.shape
```

Out[90]:
```
(175028, 25)
```

In [91]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175028 entries, 0 to 175027
Data columns (total 25 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   Surname             175028 non-null  int64  
 1   CreditScore         175028 non-null  float64
 2   Age                 175028 non-null  float64
 3   Tenure              175028 non-null  float64
 4   Balance             175028 non-null  float64
 5   NumOfProducts       175028 non-null  float64
 6   HasCrCard           175028 non-null  int64  
 7   IsActiveMember      175028 non-null  int64  
 8   EstimatedSalary     175028 non-null  float64
 9   Exited              175028 non-null  int64  
 10  Surname_tfidf_0     175028 non-null  float64
 11  Surname_tfidf_1     175028 non-null  float64
 12  Surname_tfidf_2     175028 non-null  float64
 13  Surname_tfidf_3     175028 non-null  float64
 14  Surname_tfidf_4     175028 non-null  float64
 15  France              175028 non-null  int64  
 16  Germany             175028 non-null  int64  
 17  Spain               175028 non-null  int64  
 18  Female              175028 non-null  int64  
 19  Male                175028 non-null  int64  
 20  Mem__no__Products   175028 non-null  float64
 21  Cred_Bal_Sal        175028 non-null  float64
 22  Bal_sal             175028 non-null  float64
 23  Tenure_Age          175028 non-null  float64
 24  Age_Tenure_product  175028 non-null  float64
dtypes: float64(16), int64(9)
memory usage: 33.4 MB
```

In [92]:
```
df.describe()
```

Out[92]:

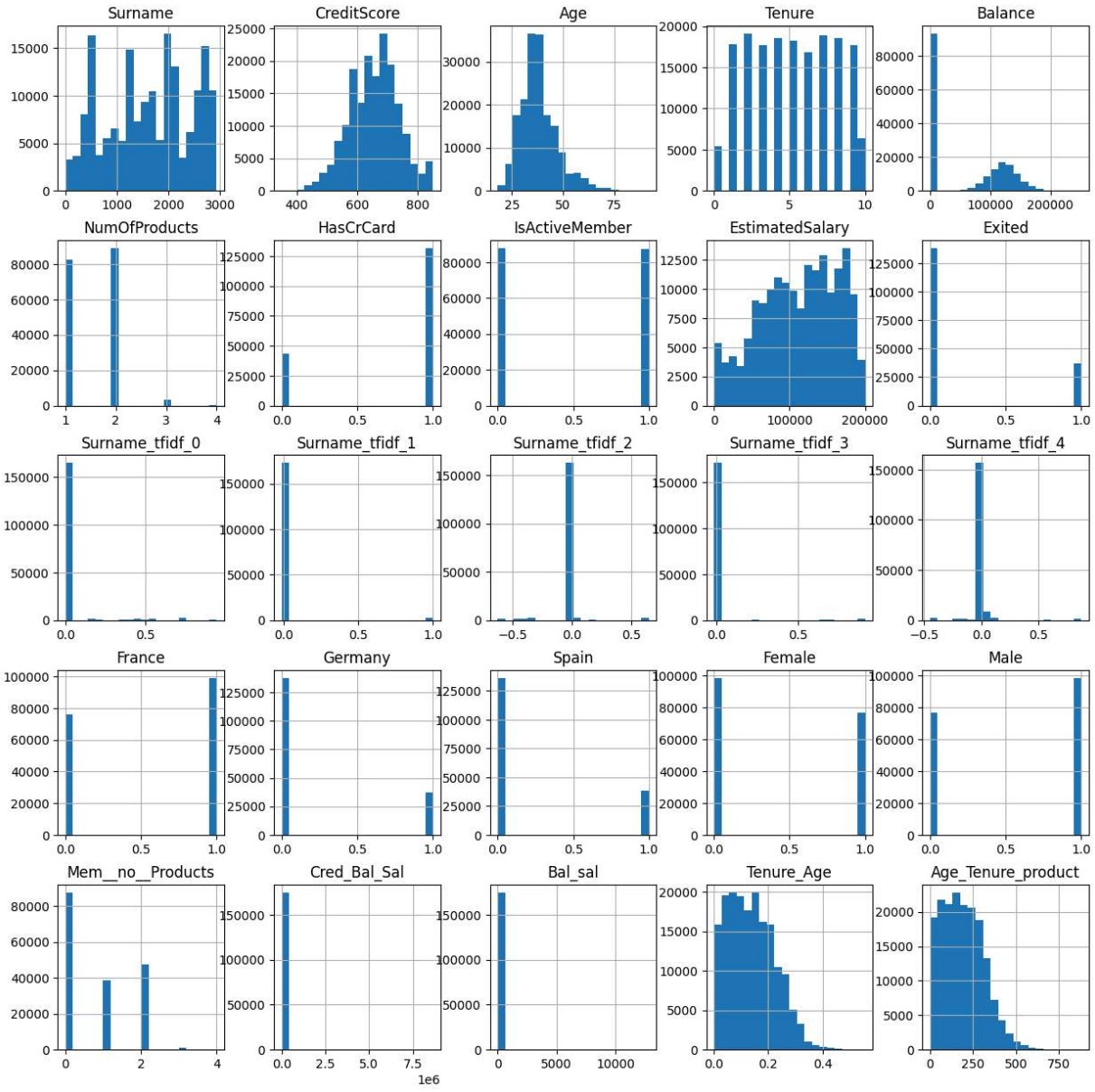
	Surname	CreditScore	Age	Tenure	Balance	NumOf
count	175028.000000	175028.000000	175028.000000	175028.000000	175028.000000	17502
mean	1617.462812	656.113068	38.171337	5.019968	56676.772103	
std	813.852174	81.148273	8.969565	2.811155	62982.418525	
min	0.000000	350.000000	18.000000	0.000000	0.000000	
25%	949.000000	597.000000	32.000000	3.000000	0.000000	
50%	1666.000000	659.000000	37.000000	5.000000	0.000000	
75%	2292.000000	710.000000	42.000000	7.000000	120727.970000	
max	2931.000000	850.000000	92.000000	10.000000	250898.090000	

8 rows × 25 columns

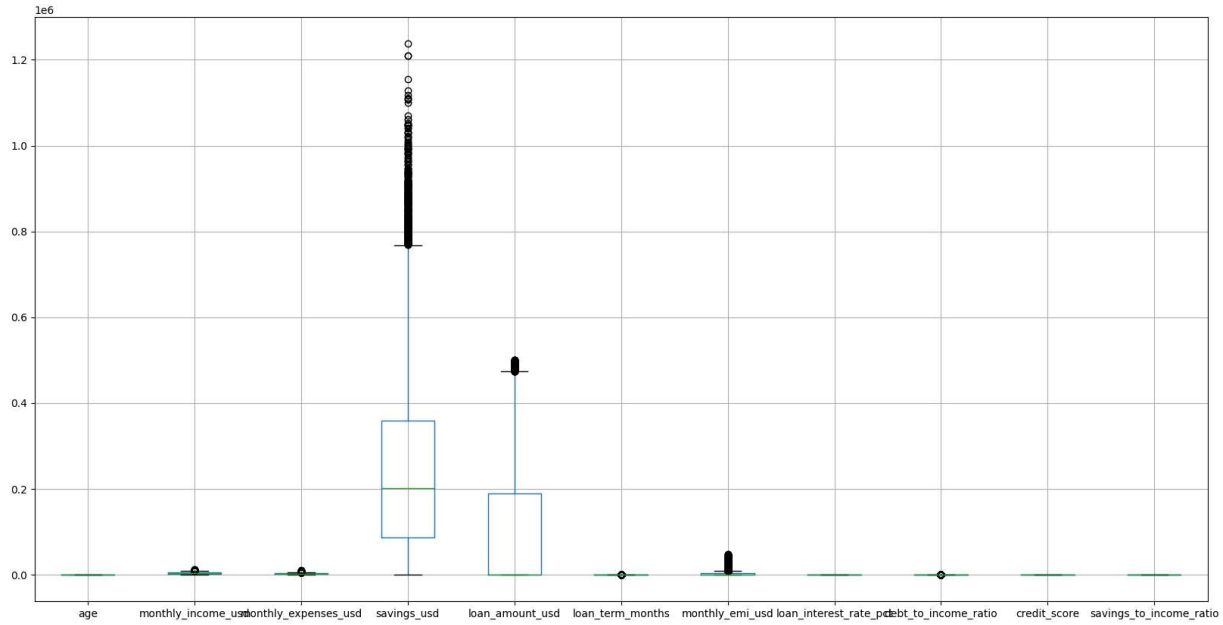
In [93]:
```
df.columns
```

Out[93]:
```
Index(['Surname', 'CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
       'HasCrCard', 'IsActiveMember', 'EstimatedSalary', 'Exited',
       'Surname_tfidf_0', 'Surname_tfidf_1', 'Surname_tfidf_2',
       'Surname_tfidf_3', 'Surname_tfidf_4', 'France', 'Germany', 'Spain',
       'Female', 'Male', 'Mem__no__Products', 'Cred_Bal_Sal', 'Bal_sal',
       'Tenure_Age', 'Age_Tenure_product'],
      dtype='object')
```

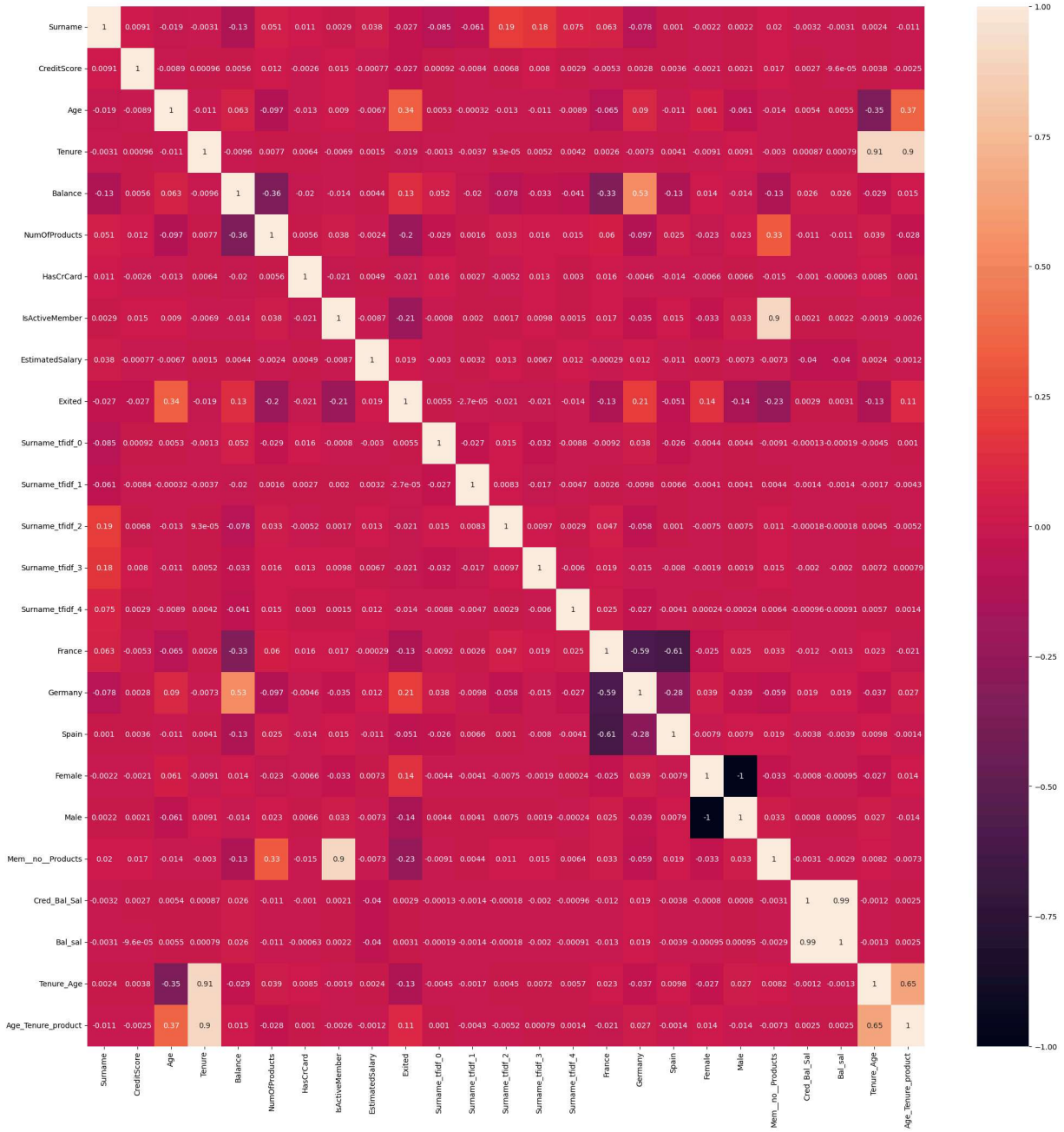
In [95]:
```
df.hist(bins=20,figsize=(15,15))
plt.show()
```



In [54]:
```
df.boxplot(figsize=(20,10))
plt.show()
```



In [98]:
```
corr_matrix = df.corr(numeric_only=True)
plt.figure(figsize=(25, 25))
sns.heatmap(corr_matrix,annot=True,)
plt.show()
```



✅ Task 4: Feature Engineering

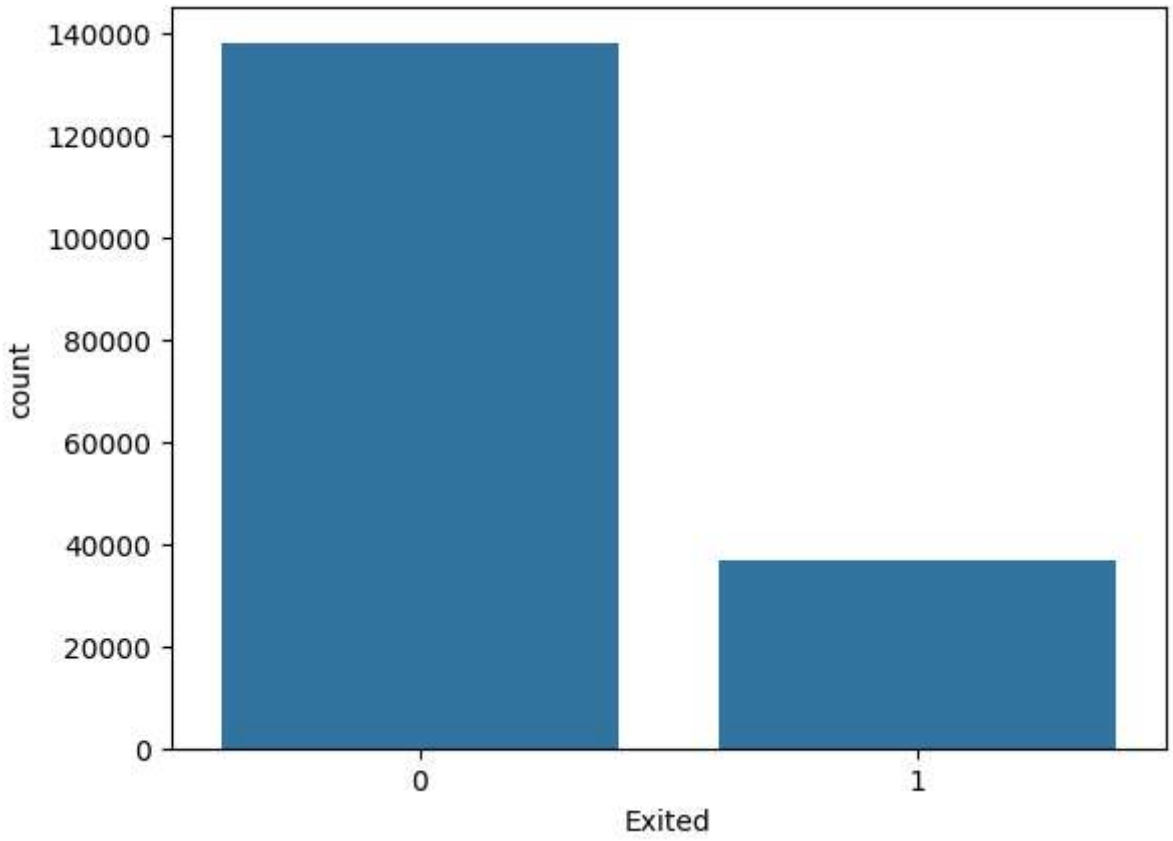
1. Handle missing values if any
2. Encode categorical variables if any
3. Create or remove irrelevant features

In [99]:
```
df.isnull().sum()
```

Out[99]:
```
Surname              0
CreditScore          0
Age                  0
Tenure               0
Balance              0
NumOfProducts        0
HasCrCard            0
IsActiveMember       0
EstimatedSalary      0
Exited               0
Surname_tfidf_0      0
Surname_tfidf_1      0
Surname_tfidf_2      0
Surname_tfidf_3      0
Surname_tfidf_4      0
France               0
Germany              0
Spain                0
Female               0
Male                 0
Mem__no__Products    0
Cred_Bal_Sal         0
Bal_sal              0
Tenure_Age           0
Age_Tenure_product   0
dtype: int64
```

In [101…
```
sns.countplot(x='Exited',data=df)
df['Exited'].value_counts()
```

Out[101…
```
Exited
0    138071
1     36957
Name: count, dtype: int64
```



✅ Task 5: Feature Scaling / Normalization

1. StandardScaler / MinMaxScaler

✅ Task 6: Split Data into Training and Testing Sets (80/20 or 70/30)

In [103…
```
X = df.drop(['Exited'],axis=1)
y = df['Exited']
```

In [105…
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

In [62]:
```
X_test.shape
```

Out[62]: (9728, 32)

In [107…
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

✅ Task 7: Define the ANN Model (Sequential API - Keras)

1. Design architecture (input, hidden, output layers)
2. Use appropriate activation functions

In [109…
```
model = Sequential([
    Dense(128,activation='relu'),
    Dense(64,activation='relu'),
```

```
    Dense(32,activation='relu'),
    Dense(1,activation='sigmoid')
])
```

In [110…
```
model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
dense_20 (Dense)	?	0 (unbuilt)
dense_21 (Dense)	?	0 (unbuilt)
dense_22 (Dense)	?	0 (unbuilt)
dense_23 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

✅ Task 8: Compile the Model

- Select optimizer, loss function, and evaluation metrics

In [115…
```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

✅ Task 9: Train the Model

- Use model.fit()
- Track performance using training history

In [112…
```
m = model.fit(X_train, y_train, epochs=50, validation_split=0.2)
```

```
Epoch 1/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - accuracy: 0.8551 - loss: 0.3436 - val_
accuracy: 0.8617 - val_loss: 0.3324
Epoch 2/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 14s 4ms/step - accuracy: 0.8624 - loss: 0.3282 - val_
accuracy: 0.8619 - val_loss: 0.3333
Epoch 3/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 14s 4ms/step - accuracy: 0.8637 - loss: 0.3264 - val_
accuracy: 0.8605 - val_loss: 0.3307
Epoch 4/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 14s 4ms/step - accuracy: 0.8644 - loss: 0.3250 - val_
accuracy: 0.8620 - val_loss: 0.3309
Epoch 5/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 14s 4ms/step - accuracy: 0.8647 - loss: 0.3238 - val_
accuracy: 0.8621 - val_loss: 0.3304
Epoch 6/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8646 - loss: 0.3232 - val_a
ccuracy: 0.8633 - val_loss: 0.3279
Epoch 7/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8650 - loss: 0.3222 - val_a
ccuracy: 0.8638 - val_loss: 0.3276
Epoch 8/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 13s 4ms/step - accuracy: 0.8645 - loss: 0.3216 - val_
accuracy: 0.8619 - val_loss: 0.3304
Epoch 9/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 14s 4ms/step - accuracy: 0.8652 - loss: 0.3207 - val_
accuracy: 0.8629 - val_loss: 0.3280
Epoch 10/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 14s 4ms/step - accuracy: 0.8658 - loss: 0.3201 - val_
accuracy: 0.8630 - val_loss: 0.3287
Epoch 11/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 14s 4ms/step - accuracy: 0.8657 - loss: 0.3195 - val_
accuracy: 0.8625 - val_loss: 0.3277
Epoch 12/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8661 - loss: 0.3191 - val_a
ccuracy: 0.8634 - val_loss: 0.3286
Epoch 13/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 13s 4ms/step - accuracy: 0.8664 - loss: 0.3183 - val_
accuracy: 0.8621 - val_loss: 0.3284
Epoch 14/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 16s 5ms/step - accuracy: 0.8666 - loss: 0.3176 - val_
accuracy: 0.8636 - val_loss: 0.3296
Epoch 15/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8668 - loss: 0.3169 - val_a
ccuracy: 0.8641 - val_loss: 0.3279
Epoch 16/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8675 - loss: 0.3164 - val_a
ccuracy: 0.8640 - val_loss: 0.3281
Epoch 17/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8671 - loss: 0.3162 - val_a
ccuracy: 0.8601 - val_loss: 0.3317
Epoch 18/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8674 - loss: 0.3149 - val_a
ccuracy: 0.8622 - val_loss: 0.3326
Epoch 19/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8678 - loss: 0.3147 - val_a
```

```
ccuracy: 0.8622 - val_loss: 0.3305
Epoch 20/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 10s 1ms/step - accuracy: 0.8684 - loss: 0.3136 - val_
accuracy: 0.8621 - val_loss: 0.3308
Epoch 21/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8677 - loss: 0.3133 - val_a
ccuracy: 0.8626 - val_loss: 0.3298
Epoch 22/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8680 - loss: 0.3122 - val_a
ccuracy: 0.8611 - val_loss: 0.3328
Epoch 23/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8690 - loss: 0.3118 - val_a
ccuracy: 0.8618 - val_loss: 0.3320
Epoch 24/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8690 - loss: 0.3115 - val_a
ccuracy: 0.8617 - val_loss: 0.3338
Epoch 25/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8690 - loss: 0.3103 - val_a
ccuracy: 0.8610 - val_loss: 0.3336
Epoch 26/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8696 - loss: 0.3098 - val_a
ccuracy: 0.8613 - val_loss: 0.3346
Epoch 27/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8701 - loss: 0.3089 - val_a
ccuracy: 0.8631 - val_loss: 0.3357
Epoch 28/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - accuracy: 0.8695 - loss: 0.3083 - val_
accuracy: 0.8617 - val_loss: 0.3376
Epoch 29/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8700 - loss: 0.3077 - val_a
ccuracy: 0.8611 - val_loss: 0.3353
Epoch 30/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8707 - loss: 0.3069 - val_a
ccuracy: 0.8612 - val_loss: 0.3357
Epoch 31/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8714 - loss: 0.3063 - val_a
ccuracy: 0.8591 - val_loss: 0.3434
Epoch 32/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8706 - loss: 0.3056 - val_a
ccuracy: 0.8592 - val_loss: 0.3379
Epoch 33/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 14s 4ms/step - accuracy: 0.8720 - loss: 0.3051 - val_
accuracy: 0.8592 - val_loss: 0.3393
Epoch 34/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 14s 4ms/step - accuracy: 0.8719 - loss: 0.3042 - val_
accuracy: 0.8593 - val_loss: 0.3393
Epoch 35/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 14s 4ms/step - accuracy: 0.8717 - loss: 0.3035 - val_
accuracy: 0.8610 - val_loss: 0.3395
Epoch 36/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 15s 4ms/step - accuracy: 0.8721 - loss: 0.3029 - val_
accuracy: 0.8607 - val_loss: 0.3397
Epoch 37/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 15s 4ms/step - accuracy: 0.8722 - loss: 0.3024 - val_
accuracy: 0.8578 - val_loss: 0.3432
Epoch 38/50
```

```
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 12s 4ms/step - accuracy: 0.8725 - loss: 0.3015 - val_
accuracy: 0.8600 - val_loss: 0.3441
Epoch 39/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 14s 2ms/step - accuracy: 0.8727 - loss: 0.3011 - val_
accuracy: 0.8577 - val_loss: 0.3434
Epoch 40/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8734 - loss: 0.3006 - val_a
ccuracy: 0.8604 - val_loss: 0.3443
Epoch 41/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8733 - loss: 0.2996 - val_a
ccuracy: 0.8598 - val_loss: 0.3449
Epoch 42/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8737 - loss: 0.2988 - val_a
ccuracy: 0.8577 - val_loss: 0.3465
Epoch 43/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8744 - loss: 0.2983 - val_a
ccuracy: 0.8577 - val_loss: 0.3492
Epoch 44/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8738 - loss: 0.2978 - val_a
ccuracy: 0.8573 - val_loss: 0.3451
Epoch 45/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8737 - loss: 0.2975 - val_a
ccuracy: 0.8570 - val_loss: 0.3514
Epoch 46/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8745 - loss: 0.2968 - val_a
ccuracy: 0.8590 - val_loss: 0.3482
Epoch 47/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8747 - loss: 0.2960 - val_a
ccuracy: 0.8590 - val_loss: 0.3517
Epoch 48/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8750 - loss: 0.2968 - val_a
ccuracy: 0.8583 - val_loss: 0.3498
Epoch 49/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8758 - loss: 0.2948 - val_a
ccuracy: 0.8562 - val_loss: 0.3538
Epoch 50/50
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8760 - loss: 0.2944 - val_a
ccuracy: 0.8570 - val_loss: 0.3539
```

In [114…

```
model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
dense_20 (Dense)	(None, 128)	3,200
dense_21 (Dense)	(None, 64)	8,256
dense_22 (Dense)	(None, 32)	2,080
dense_23 (Dense)	(None, 1)	33

Total params: 40,709 (159.02 KB)

Trainable params: 13,569 (53.00 KB)

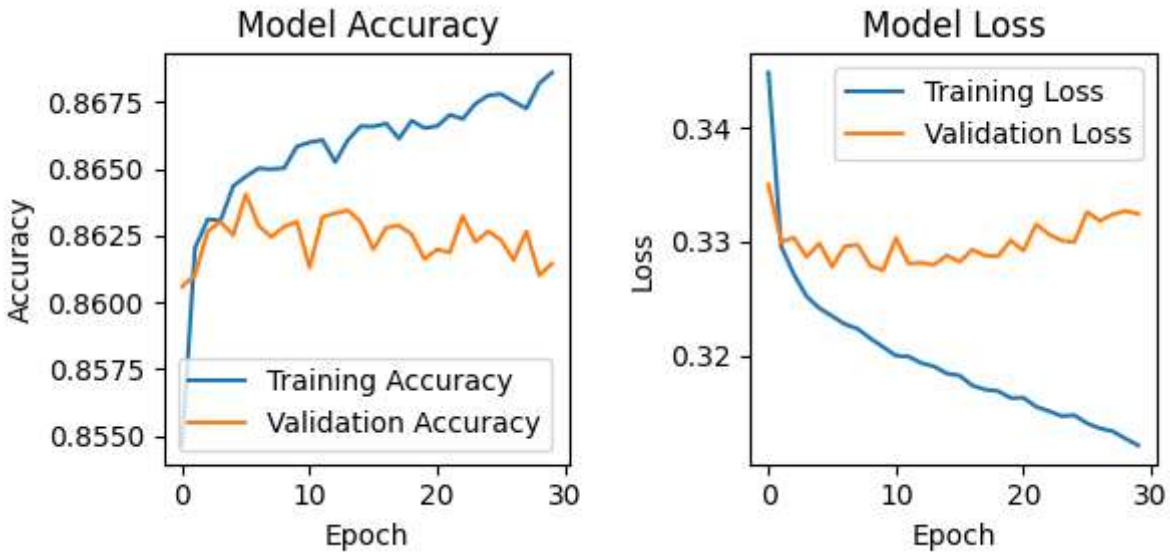
```
Non-trainable params: 0 (0.00 B)
Optimizer params: 27,140 (106.02 KB)
```

✅ Task 10: Visualize Model Performance

- Plot loss and accuracy curves for training and validation

In [138…

```
plt.figure(figsize=(6,3))
plt.subplot(1, 2, 1)
plt.plot(m.history['accuracy'], label='Training Accuracy')
plt.plot(m.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(m.history['loss'], label='Training Loss')
plt.plot(m.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()
```



In [ ]:

✅ Task 11: Evaluate Model on Test Data

- Accuracy, loss
- Predict test set

In [119…

```
loss, mae = model.evaluate(X_test,y_test)
```

```
1094/1094 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8591 - loss: 0.3481
```

In [120…
```
y_pred = model.predict(X_test)
```

```
1094/1094 ━━━━━━━━━━━━━━━━━━━━ 1s 917us/step
```

✅ Task 12: Confusion Matrix and Classification Report

- Use sklearn metrics to analyze performance

In [121…
```
mse = mean_squared_error(y_test, y_pred)
mse
```

Out[121…
```
0.10393407940864563
```

In [123…
```
mae = mean_absolute_error(y_test, y_pred)
mae
```

Out[123…
```
0.1914944052696228
```

In [125…
```
rmse = np.sqrt(mse)
rmse
```

Out[125…
```
np.float64(0.32238808819285747)
```

In [126…
```
r2 = r2_score(y_test, y_pred)
r2
```

Out[126…
```
0.3765411376953125
```

✅ Task 13: Hyperparameter Tuning (Try any of the following)

- Vary batch size, epochs, optimizers
- Use Keras Tuner or manual tuning

In [131…
```
model = Sequential([
    Dense(128,activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32,activation='relu'),
    Dense(1,activation='sigmoid')
])
```

```
c:\Users\This PC\Desktop\DeepLearning\.venv\Lib\site-packages\keras\src\layers\core
\dense.py:92: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a la
yer. When using Sequential models, prefer using an `Input(shape)` object as the firs
t layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [132…
```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_7"
```

Layer (type)	Output Shape	Param #
dense_28 (Dense)	(None, 128)	3,200
dense_29 (Dense)	(None, 32)	4,128
dense_30 (Dense)	(None, 1)	33

Total params: 7,361 (28.75 KB)

Trainable params: 7,361 (28.75 KB)

Non-trainable params: 0 (0.00 B)

In [133…

```
m = model.fit(X_train, y_train, epochs=30, validation_split=0.2)
```

```
Epoch 1/30
```

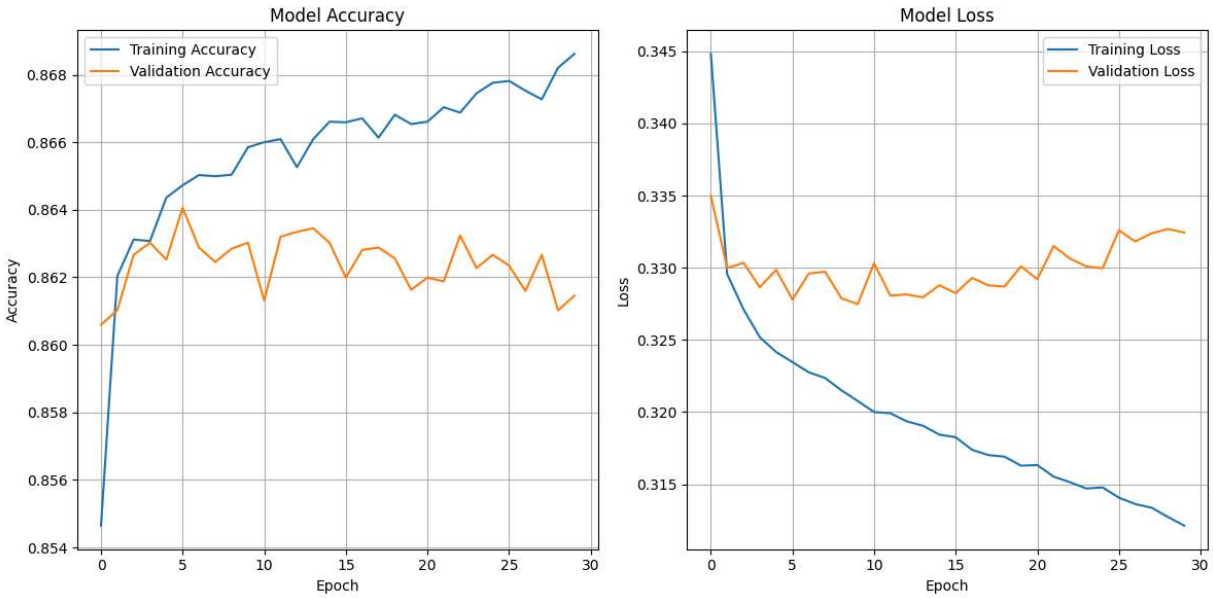
```
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8546 - loss: 0.3448 - val_a
ccuracy: 0.8606 - val_loss: 0.3350
Epoch 2/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8620 - loss: 0.3296 - val_a
ccuracy: 0.8610 - val_loss: 0.3300
Epoch 3/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8631 - loss: 0.3271 - val_a
ccuracy: 0.8627 - val_loss: 0.3303
Epoch 4/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8631 - loss: 0.3252 - val_a
ccuracy: 0.8630 - val_loss: 0.3287
Epoch 5/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8644 - loss: 0.3242 - val_a
ccuracy: 0.8625 - val_loss: 0.3299
Epoch 6/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8647 - loss: 0.3235 - val_a
ccuracy: 0.8641 - val_loss: 0.3278
Epoch 7/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8650 - loss: 0.3228 - val_a
ccuracy: 0.8629 - val_loss: 0.3296
Epoch 8/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8650 - loss: 0.3224 - val_a
ccuracy: 0.8625 - val_loss: 0.3297
Epoch 9/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8650 - loss: 0.3215 - val_a
ccuracy: 0.8628 - val_loss: 0.3279
Epoch 10/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8659 - loss: 0.3208 - val_a
ccuracy: 0.8630 - val_loss: 0.3275
Epoch 11/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8660 - loss: 0.3200 - val_a
ccuracy: 0.8613 - val_loss: 0.3303
Epoch 12/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8661 - loss: 0.3199 - val_a
ccuracy: 0.8632 - val_loss: 0.3281
Epoch 13/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8653 - loss: 0.3194 - val_a
ccuracy: 0.8633 - val_loss: 0.3282
Epoch 14/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8661 - loss: 0.3191 - val_a
ccuracy: 0.8635 - val_loss: 0.3280
Epoch 15/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8666 - loss: 0.3184 - val_a
ccuracy: 0.8630 - val_loss: 0.3288
Epoch 16/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8666 - loss: 0.3183 - val_a
ccuracy: 0.8620 - val_loss: 0.3282
Epoch 17/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8667 - loss: 0.3174 - val_a
ccuracy: 0.8628 - val_loss: 0.3293
Epoch 18/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8661 - loss: 0.3170 - val_a
ccuracy: 0.8629 - val_loss: 0.3288
Epoch 19/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8668 - loss: 0.3169 - val_a
ccuracy: 0.8626 - val_loss: 0.3287
```

```
Epoch 20/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8665 - loss: 0.3163 - val_a
ccuracy: 0.8616 - val_loss: 0.3301
Epoch 21/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8666 - loss: 0.3163 - val_a
ccuracy: 0.8620 - val_loss: 0.3292
Epoch 22/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8670 - loss: 0.3155 - val_a
ccuracy: 0.8619 - val_loss: 0.3315
Epoch 23/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8669 - loss: 0.3151 - val_a
ccuracy: 0.8632 - val_loss: 0.3306
Epoch 24/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8674 - loss: 0.3147 - val_a
ccuracy: 0.8623 - val_loss: 0.3301
Epoch 25/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8678 - loss: 0.3148 - val_a
ccuracy: 0.8627 - val_loss: 0.3300
Epoch 26/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8678 - loss: 0.3141 - val_a
ccuracy: 0.8623 - val_loss: 0.3326
Epoch 27/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8675 - loss: 0.3136 - val_a
ccuracy: 0.8616 - val_loss: 0.3318
Epoch 28/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8673 - loss: 0.3134 - val_a
ccuracy: 0.8627 - val_loss: 0.3324
Epoch 29/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8682 - loss: 0.3127 - val_a
ccuracy: 0.8610 - val_loss: 0.3327
Epoch 30/30
3501/3501 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8686 - loss: 0.3121 - val_a
ccuracy: 0.8615 - val_loss: 0.3324
```

In [134…

```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(m.history['accuracy'], label='Training Accuracy')
plt.plot(m.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(m.history['loss'], label='Training Loss')
plt.plot(m.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



✅ Task 14: Save and Load the Model (Optional)

📄 Report Questions (To be submitted separately)

- Dataset Selection: What dataset did you choose? Describe its purpose and target.
- EDA Findings: What key insights did you gain during EDA?
- Feature Engineering: What transformations did you apply? Why?
- ANN Architecture: List number of layers, neurons, activation functions.
- Performance Summary: What were the final accuracy and loss?
- Confusion Matrix Analysis: What did it reveal about model behavior?
- Hyperparameter Tuning: What parameters did you change and what was the result?
- Improvement Ideas: Suggest one way to improve performance.
- Overfitting/Underfitting: Did you face it? How did you address it?