```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the data
data = pd.read_excel('Credit Card Defaulter Prediction.xlsx', sheet_name='in')

# Check column names
print("Columns in the DataFrame:", data.columns)

# Strip any leading or trailing spaces from column names
data.columns = data.columns.str.strip()

# Check for unique values in the EDUCATION and MARRIAGE columns
print("Unique values in EDUCATION before encoding:", data['EDUCATION'].unique())
print("Unique values in MARRIAGE before encoding:", data['MARRIAGE'].unique())

# Convert columns to string type to avoid mixed type issues
data['SEX'] = data['SEX'].astype(str)
data['EDUCATION'] = data['EDUCATION'].astype(str)
data['MARRIAGE'] = data['MARRIAGE'].astype(str)
data['default'] = data['default'].astype(str)

# Check for missing values in 'default' column and handle them
if 'default' not in data.columns:
    raise KeyError("The 'default' column is not present in the DataFrame.")

if data['default'].isnull().sum() > 0:
    print("Missing values in 'default' column found. Dropping rows with missing values.'
    data = data.dropna(subset=['default'])

# Encode categorical variables
label_encoder = LabelEncoder()
data['SEX'] = label_encoder.fit_transform(data['SEX'])
data['EDUCATION'] = label_encoder.fit_transform(data['EDUCATION'])
data['MARRIAGE'] = label_encoder.fit_transform(data['MARRIAGE'])
data['default'] = label_encoder.fit_transform(data['default'])

# Separate features and target variable from the dataset
X = data.drop(columns=['ID', 'default'])
y = data['default']

# Standardize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# Step 3: Build the Neural Network
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])
```

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Step 4: Train the Model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

# Step 5: Evaluate the Model
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}')

# Plot training and validation accuracy and loss
plt.figure(figsize=(14, 6))

# Accuracy Line Plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

# Loss Line Plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.tight_layout()
plt.show()

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Default', 'Default']
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Classification Report as DataFrame
report = classification_report(y_test, y_pred, output_dict=True)
report_df = pd.DataFrame(report).iloc[:-1, :-1]  # Exclude 'accuracy' row and 'support'

# Classification Report Heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(report_df, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Classification Report Heatmap')
plt.show()

# Bar Chart for Target Variable Distribution
plt.figure(figsize=(8, 6))
y.value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.xlabel('Default')
plt.ylabel('Count')
plt.title('Distribution of Default Classes')
plt.xticks(ticks=[0, 1], labels=['No Default', 'Default'], rotation=0)
plt.show()

# Pie Chart for Target Variable Distribution
```

```
plt.figure(figsize=(8, 8))
y.value_counts().plot(kind='pie', autopct='%1.1f%%', colors=['skyblue', 'salmon'], labe
plt.title('Proportion of Default Classes')
plt.ylabel('')
plt.show()
```

```
Columns in the DataFrame: Index(['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARR
       'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6', 'default '
      dtype='object')
Unique values in EDUCATION before encoding: ['University' 'Graduate school' '
Unique values in MARRIAGE before encoding: ['Married' 'Single' 'Other' 0]
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: Us
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.7698 - loss: 0.533
Epoch 2/50
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8104 - loss: 0.469
Epoch 3/50
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8122 - loss: 0.460
Epoch 4/50
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8163 - loss: 0.447
Epoch 5/50
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8188 - loss: 0.444
Epoch 6/50
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8228 - loss: 0.438
Epoch 7/50
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8195 - loss: 0.437
Epoch 8/50
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8146 - loss: 0.438
Epoch 9/50
```