Deep Learning for Business: Customer Shopping Preferences

Introduction

Business Problem and Significance

The business problem addressed in this project is understanding and predicting customer shopping preferences to enhance business decision-making. This is significant as it can help businesses tailor their products, marketing strategies, and overall customer experience to better meet customer needs.

Project Goals and Objectives

The goal of this project is to develop predictive models using deep learning to forecast customer purchase behaviors and trends. The specific objectives include:

1. Preprocessing the data to handle missing values, encode categorical variables, and scale numerical features.
2. Experimenting with various deep learning architectures, including Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), Bidirectional RNNs, and pretrained-inspired models.
3. Fine-tuning the models to optimize their performance.
4. Evaluating the models using appropriate metrics to determine their effectiveness in predicting customer shopping trends.

Benefits to the Business

The outcomes of this project will enable businesses to make informed decisions based on data-driven insights. By accurately predicting customer purchase behaviors, businesses can:

- Optimize their inventory management to reduce overstock and stockouts.
- Improve personalized marketing strategies, resulting in higher customer engagement and conversion rates.
- Enhance customer relationship management by understanding customer preferences and providing tailored experiences.

Implications on Decision-Making Processes

The predictive models developed in this project will provide valuable insights that can influence various business decisions, such as:

- **Inventory Management:** By predicting which products are likely to be in high demand, businesses can adjust their inventory levels accordingly.
- **Marketing Strategies:** Insights into customer preferences can help design more effective marketing campaigns, targeting the right customers with the right products.
- **Customer Relationship Management:** Understanding customer behavior patterns allows businesses to personalize their interactions with customers, leading to increased customer loyalty and satisfaction.

Dataset Source

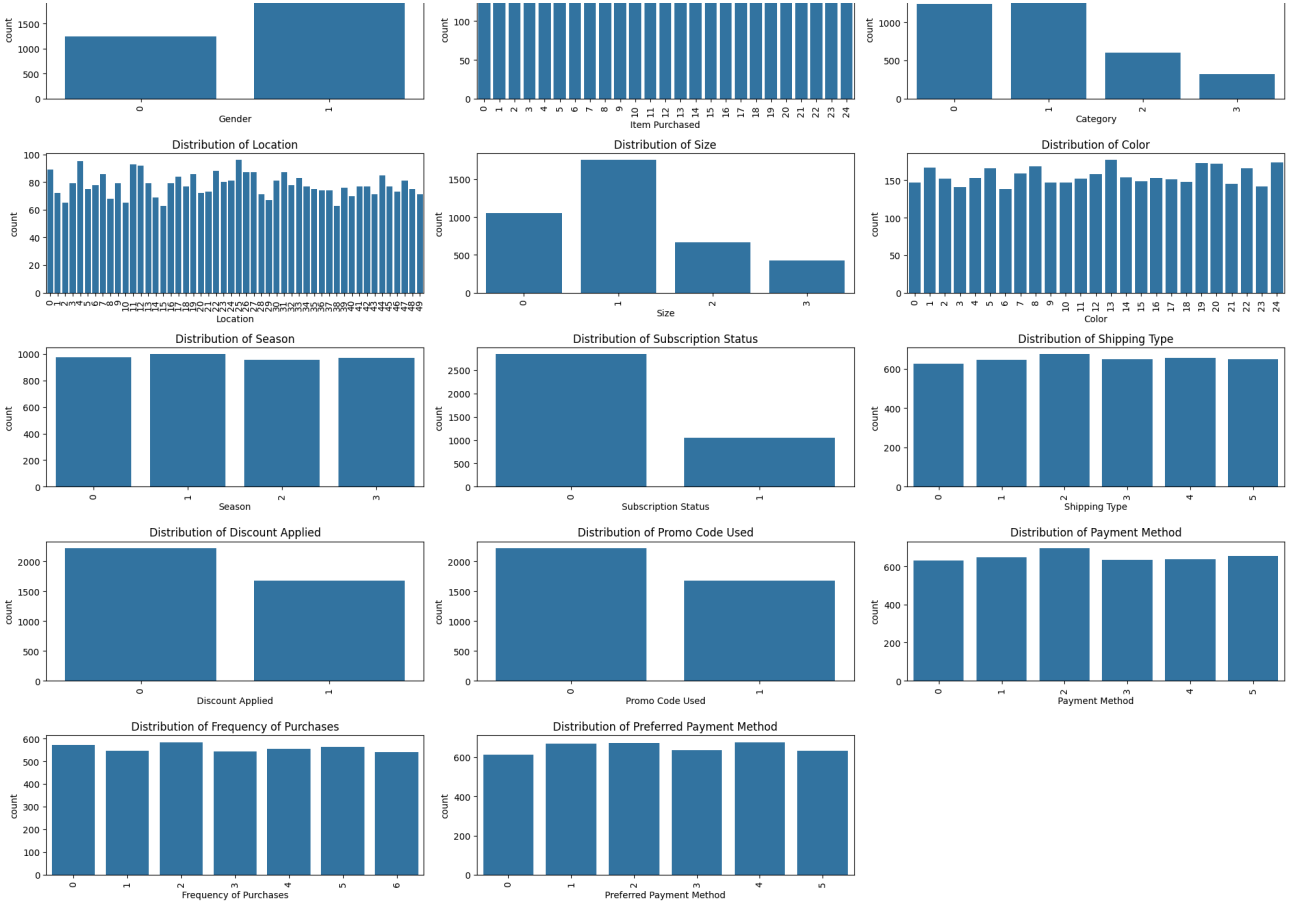
The dataset used in this project is sourced from Kaggle, titled "Customer Shopping Trends Dataset" provided by Sourav Banerjee. This dataset contains updated information on customer shopping trends, including various attributes related to customer demographics, purchase

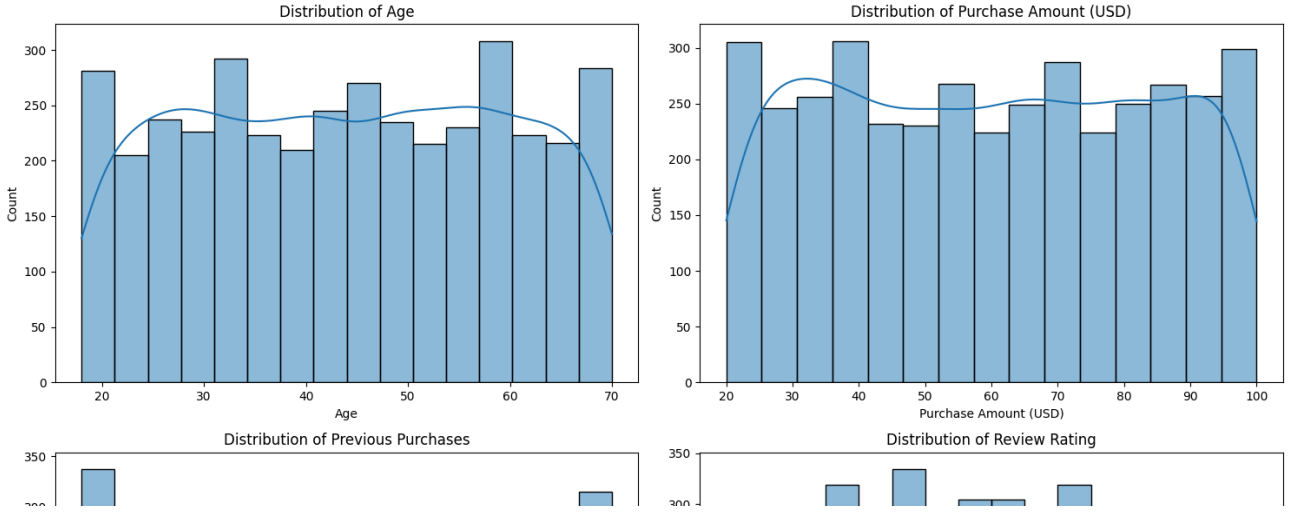
behaviors, and product details. The dataset can be accessed here. This dataset is crucial for developing and training the predictive models to achieve the project's objectives.

Step 2: Describing Dataset

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv('/content/shopping_trends.csv')

# Determine the dimensions of the dataset
print(f"Dataset Dimensions: {df.shape}")

# Check for missing values
missing_values = df.isnull().sum()
print("Missing Values:\n", missing_values)

# Descriptive statistics
descriptive_stats = df.describe(include='all')
print("Descriptive Statistics:\n", descriptive_stats)

# Visualize the data
# Scatter plot for age vs purchase amount
sns.scatterplot(x='Age', y='Purchase Amount (USD)', data=df)
plt.title('Age vs Purchase Amount')
plt.show()

# Encode categorical variables before calculating correlation matrix
label_encoders = {}
categorical_columns = ['Gender', 'Item Purchased', 'Category', 'Location', 'Size', 'Color', 'Season',
                       'Subscription Status', 'Shipping Type', 'Discount Applied', 'Promo Code Used',
                       'Payment Method', 'Frequency of Purchases', 'Preferred Payment Method']
for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Correlation plot
correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Bar plots for categorical features
plt.figure(figsize=(20, 15))
for i, col in enumerate(categorical_columns, 1):
    plt.subplot(5, 3, i)
    sns.countplot(data=df, x=col)
```

```
    plt.title(f'Distribution of {col}')
    plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# Distribution plots for numerical features
numerical_columns = ['Age', 'Purchase Amount (USD)', 'Previous Purchases', 'Review Rating']

plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(2, 2, i)
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()

# Box plots for numerical features to identify outliers
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(data=df, y=col)
    plt.title(f'Box plot of {col}')
plt.tight_layout()
plt.show()
```


Gender
Item Purchased
Category
Distribution of Location
Location
Distribution of Size
Size
Distribution of Color
Color
Distribution of Season
Season
Distribution of Subscription Status
Subscription Status
Distribution of Shipping Type
Shipping Type
Distribution of Discount Applied
Discount Applied
Distribution of Promo Code Used
Promo Code Used
Distribution of Payment Method
Payment Method
Distribution of Frequency of Purchases
Frequency of Purchases
Distribution of Preferred Payment Method
Preferred Payment Method


Distribution of Age
Age
Distribution of Purchase Amount (USD)
Purchase Amount (USD)

Distribution of Previous Purchases
Distribution of Review Rating

Step 3: Data Preprocessing

```
import numpy as np
from sklearn.preprocessing import StandardScaler

# Handle missing values by filling with forward fill method
df.fillna(method='ffill', inplace=True)

# Encoding categorical variables using LabelEncoder
label_encoders = {}
for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Feature scaling for numerical columns
scaler = StandardScaler()
df[['Age', 'Purchase Amount (USD)', 'Previous Purchases']] = scaler.fit_transform(df[['Age', 'Purchase Amount (USD)', 'Previous Purchases']])

# Prepare numerical features
X_numerical = df[['Age', 'Purchase Amount (USD)', 'Previous Purchases']].values
y = df['Review Rating'].values

# Display preprocessed data
print(df.head())
print(X_numerical.shape, y.shape)
```

```
print(X_train.shape, y_train.shape)
```

```
   Customer ID       Age  Gender  Item Purchased  Category  \
0            1  0.718913       1               2         1
1            2 -1.648629       1              23         1
2            3  0.390088       1              11         1
3            4 -1.517099       1              14         2
4            5  0.061263       1               2         1

   Purchase Amount (USD)  Location  Size  Color  Season  Review Rating  \
0              -0.285629        16     0      7       3            3.1
1               0.178852        18     0     12       3            3.1
2               0.558882        20     2     12       1            3.1
3               1.276716        38     1     12       1            3.5
4              -0.454531        36     1     21       1            2.7

   Subscription Status  Payment Method  Shipping Type  Discount Applied  \
0                    1               2              1                 1
1                    1               0              1                 1
2                    1               1              2                 1
3                    1               4              3                 1
4                    1               1              2                 1

   Promo Code Used  Previous Purchases  Preferred Payment Method  \
0                1           -0.785831                         5
1                1           -1.616552                         1
2                1           -0.162789                         2
3                1            1.637107                         4
4                1            0.391025                         4

   Frequency of Purchases
0                       3
1                       3
2                       6
3                       6
4                       0
(3900, 3) (3900,)
```

Step 4: Fitting Different Deep Learning Architectures

Imports and Data Preparation

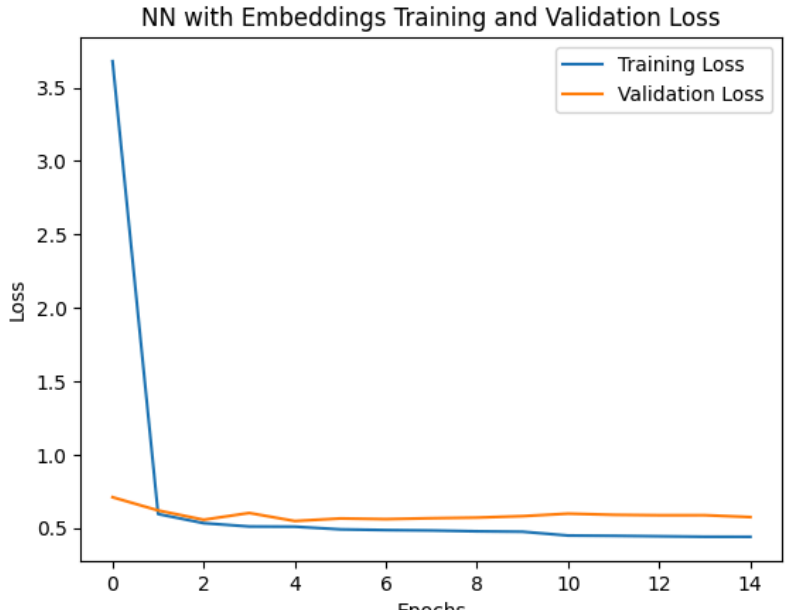
```
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Embedding, Flatten, Concatenate, LSTM, SimpleRNN, Conv1D, MaxPooling1D, Bidirectional
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('shopping_trends.csv')

# Handle missing values by filling with forward fill method
df.fillna(method='ffill', inplace=True)

# Encoding categorical variables using LabelEncoder
label_encoders = {}
categorical_columns = ['Gender', 'Item Purchased', 'Category', 'Location', 'Size', 'Color', 'Season',
                       'Subscription Status', 'Payment Method', 'Shipping Type', 'Discount Applied',
                       'Promo Code Used', 'Preferred Payment Method', 'Frequency of Purchases']

for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Feature scaling for numerical columns
scaler = StandardScaler()
df[['Age', 'Purchase Amount (USD)', 'Previous Purchases']] = scaler.fit_transform(df[['Age', 'Purchase Amount (USD)', 'Previous Purchases']])

# Prepare numerical features and target variable
X_numerical = df[['Age', 'Purchase Amount (USD)', 'Previous Purchases']].values
y = df['Review Rating'].values

# Split data into training and testing sets
X_train_num, X_test_num, y_train, y_test = train_test_split(X_numerical, y, test_size=0.2, random_state=42)

# Split categorical data
X_train_cat = [df[col].values[:X_train_num.shape[0]] for col in categorical_columns]
X_test_cat = [df[col].values[X_train_num.shape[0]:] for col in categorical_columns]

X_train = [X_train_num] + [np.array(cat) for cat in X_train_cat]
X_test = [X_test_num] + [np.array(cat) for cat in X_test_cat]

# Define the MAPE metric
def mape(y_true, y_pred):
    y_true = tf.convert_to_tensor(y_true)
    y_pred = tf.convert_to_tensor(y_pred)
    return tf.reduce_mean(tf.abs((y_true - y_pred) / y_true)) * 100

# Define embedding input layers for categorical features
def create_embedding_layers(input_data, categorical_columns, embedding_dim=8):
    inputs = []
    embeddings = []
    for col in categorical_columns:
```

```
        vocab_size = input_data[col].nunique()
        input_layer = Input(shape=(1,), name=f'{col}_input')
        embedding_layer = Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=1)(input_layer)
        flatten_layer = Flatten()(embedding_layer)
        inputs.append(input_layer)
        embeddings.append(flatten_layer)
    return inputs, embeddings

# Plot training and validation loss
def plot_history(history, title):
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
# Prepare data inputs for the model
numerical_input_data = X_numerical
categorical_input_data = [df[col].values for col in categorical_columns]
```

Simple Neural Network with Embeddings

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Simple Neural Network with Embeddings and Improvements
def create_nn_model_with_embeddings(input_data, embedding_dim=8):
    numerical_input = Input(shape=(X_numerical.shape[1],))
    inputs, embeddings = create_embedding_layers(input_data, categorical_columns, embedding_dim)
    merged_embeddings = Concatenate()(embeddings)
    merged_inputs = Concatenate()([numerical_input, merged_embeddings])
    x = Dense(128, activation='relu')(merged_inputs)
    x = Dense(64, activation='relu')(x)
    x = Dense(32, activation='relu')(x)
    output = Dense(1)(x)
    model = Model(inputs=[numerical_input] + inputs, outputs=output)
    model.compile(optimizer='adam', loss='mse', metrics=[mape])
    return model

# Create and train the model with early stopping and learning rate reduction
nn_model_with_embeddings = create_nn_model_with_embeddings(df)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)
nn_history = nn_model_with_embeddings.fit(
    [X_train_num] + X_train_cat, y_train, epochs=50, batch_size=32, validation_split=0.2,
    callbacks=[early_stopping, reduce_lr]
)
nn_loss, nn_mape = nn_model_with_embeddings.evaluate([X_test_num] + X_test_cat, y_test)
print(f"NN with Embeddings Mean Absolute Percentage Error: {nn_mape}")

# Plot training and validation loss for NN with Embeddings
plot_history(nn_history, 'NN with Embeddings Training and Validation Loss')
```

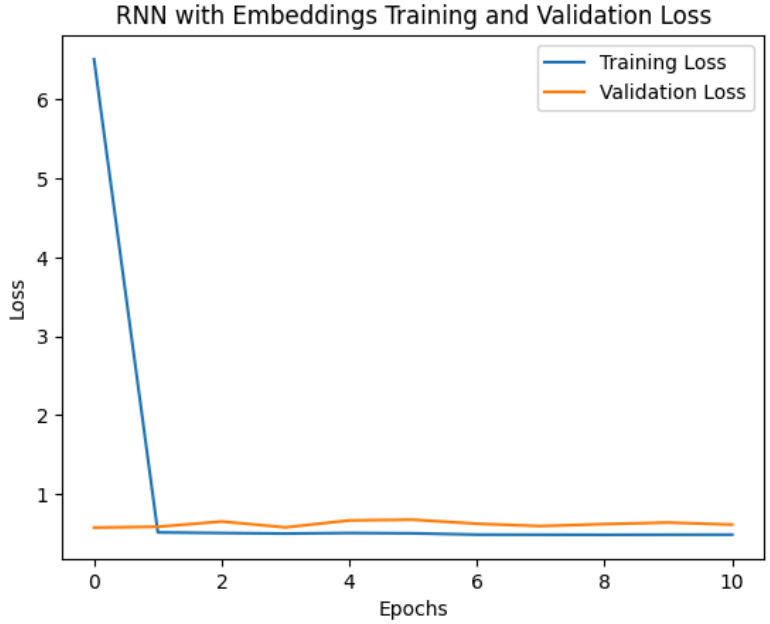
```
Epoch 1/50
78/78 [==============================] - 3s 9ms/step - loss: 3.6805 - mape: 39.4027 - val_loss: 0.7098 - val_
Epoch 2/50
78/78 [==============================] - 1s 8ms/step - loss: 0.5952 - mape: 18.1513 - val_loss: 0.6197 - val_
Epoch 3/50
78/78 [==============================] - 1s 7ms/step - loss: 0.5330 - mape: 17.3712 - val_loss: 0.5562 - val_
Epoch 4/50
78/78 [==============================] - 1s 8ms/step - loss: 0.5104 - mape: 17.0898 - val_loss: 0.6024 - val_
Epoch 5/50
78/78 [==============================] - 1s 10ms/step - loss: 0.5091 - mape: 17.0075 - val_loss: 0.5476 - val
Epoch 6/50
78/78 [==============================] - 1s 8ms/step - loss: 0.4908 - mape: 16.7431 - val_loss: 0.5650 - val_
Epoch 7/50
78/78 [==============================] - 1s 11ms/step - loss: 0.4859 - mape: 16.7069 - val_loss: 0.5605 - val
Epoch 8/50
78/78 [==============================] - 1s 6ms/step - loss: 0.4832 - mape: 16.6248 - val_loss: 0.5666 - val_
Epoch 9/50
78/78 [==============================] - 0s 6ms/step - loss: 0.4774 - mape: 16.5203 - val_loss: 0.5709 - val_
Epoch 10/50
78/78 [==============================] - 0s 6ms/step - loss: 0.4746 - mape: 16.4738 - val_loss: 0.5808 - val_
Epoch 11/50
78/78 [==============================] - 0s 6ms/step - loss: 0.4487 - mape: 16.0698 - val_loss: 0.5979 - val_
Epoch 12/50
78/78 [==============================] - 0s 6ms/step - loss: 0.4468 - mape: 16.0167 - val_loss: 0.5899 - val_
Epoch 13/50
78/78 [==============================] - 0s 5ms/step - loss: 0.4436 - mape: 15.9788 - val_loss: 0.5870 - val_
Epoch 14/50
78/78 [==============================] - 0s 5ms/step - loss: 0.4404 - mape: 15.9415 - val_loss: 0.5872 - val_
Epoch 15/50
78/78 [==============================] - 0s 6ms/step - loss: 0.4400 - mape: 15.8561 - val_loss: 0.5744 - val_
25/25 [==============================] - 0s 2ms/step - loss: 0.5857 - mape: 17.9845
NN with Embeddings Mean Absolute Percentage Error: 17.984468460083008
```



Recurrent Neural Network (RNN) with Embeddings

```
# Recurrent Neural Network with Embeddings and Improvements
def create_rnn_model_with_embeddings(input_data, embedding_dim=8):
    numerical_input = Input(shape=(X_numerical.shape[1],))
    inputs, embeddings = create_embedding_layers(input_data, categorical_columns, embedding_dim)
    merged_embeddings = Concatenate()(embeddings)
    merged_inputs = Concatenate()([numerical_input, merged_embeddings])
    x = tf.expand_dims(merged_inputs, axis=1)
    x = LSTM(128, return_sequences=True)(x)
    x = LSTM(64)(x)
    x = Dense(32, activation='relu')(x)
    output = Dense(1)(x)
    model = Model(inputs=[numerical_input] + inputs, outputs=output)
    model.compile(optimizer='adam', loss='mse', metrics=[mape])
    return model

# Create and train the RNN model with early stopping and learning rate reduction
rnn_model_with_embeddings = create_rnn_model_with_embeddings(df)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)
rnn_history = rnn_model_with_embeddings.fit(
    [X_train_num] + X_train_cat, y_train, epochs=50, batch_size=32, validation_split=0.2,
    callbacks=[early_stopping, reduce_lr]
)
rnn_loss, rnn_mape = rnn_model_with_embeddings.evaluate([X_test_num] + X_test_cat, y_test)
print(f"RNN with Embeddings Mean Absolute Percentage Error: {rnn_mape}")

# Plot training and validation loss for RNN with Embeddings
plot_history(rnn_history, 'RNN with Embeddings Training and Validation Loss')
```

```
Epoch 1/50
78/78 [==============================] - 9s 35ms/step - loss: 6.5109 - mape: 55.5365 - val_loss: 0.5723 - val
Epoch 2/50
78/78 [==============================] - 2s 22ms/step - loss: 0.5112 - mape: 17.3638 - val_loss: 0.5827 - val
Epoch 3/50
78/78 [==============================] - 3s 44ms/step - loss: 0.5030 - mape: 17.1216 - val_loss: 0.6489 - val
Epoch 4/50
78/78 [==============================] - 2s 27ms/step - loss: 0.4963 - mape: 16.9237 - val_loss: 0.5750 - val
Epoch 5/50
78/78 [==============================] - 1s 12ms/step - loss: 0.5033 - mape: 17.0506 - val_loss: 0.6629 - val
Epoch 6/50
78/78 [==============================] - 1s 13ms/step - loss: 0.4991 - mape: 16.9545 - val_loss: 0.6724 - val
Epoch 7/50
78/78 [==============================] - 1s 12ms/step - loss: 0.4828 - mape: 16.7123 - val_loss: 0.6210 - val
Epoch 8/50
78/78 [==============================] - 1s 11ms/step - loss: 0.4817 - mape: 16.6743 - val_loss: 0.5915 - val
Epoch 9/50
78/78 [==============================] - 1s 11ms/step - loss: 0.4804 - mape: 16.7054 - val_loss: 0.6167 - val
Epoch 10/50
78/78 [==============================] - 1s 11ms/step - loss: 0.4820 - mape: 16.7048 - val_loss: 0.6359 - val
Epoch 11/50
78/78 [==============================] - 1s 11ms/step - loss: 0.4824 - mape: 16.7039 - val_loss: 0.6099 - val
25/25 [==============================] - 0s 6ms/step - loss: 0.5927 - mape: 17.6449
RNN with Embeddings Mean Absolute Percentage Error: 17.644874572753906
```



Convolutional Neural Network (CNN) with Embeddings

```
# Convolutional Neural Network with Embeddings and Improvements
def create_cnn_model_with_embeddings(input_data, embedding_dim=8):
    numerical_input = Input(shape=(X_numerical.shape[1],))
    inputs, embeddings = create_embedding_layers(input_data, categorical_columns, embedding_dim)
    merged_embeddings = Concatenate()(embeddings)
    merged_inputs = Concatenate()([numerical_input, merged_embeddings])
    x = tf.expand_dims(merged_inputs, axis=2)
    x = Conv1D(64, kernel_size=3, activation='relu', padding='same')(x)
    x = MaxPooling1D(pool_size=2)(x)
    x = Conv1D(32, kernel_size=3, activation='relu', padding='same')(x)
    x = MaxPooling1D(pool_size=2)(x)
    x = Flatten()(x)
    x = Dense(32, activation='relu')(x)
    output = Dense(1)(x)
    model = Model(inputs=[numerical_input] + inputs, outputs=output)
    model.compile(optimizer='adam', loss='mse', metrics=[mape])
    return model

# Create and train the CNN model with early stopping and learning rate reduction
cnn_model_with_embeddings = create_cnn_model_with_embeddings(df)
cnn_history = cnn_model_with_embeddings.fit(
    [X_train_num] + X_train_cat, y_train, epochs=50, batch_size=32, validation_split=0.2,
    callbacks=[early_stopping, reduce_lr]
)
cnn_loss, cnn_mape = cnn_model_with_embeddings.evaluate([X_test_num] + X_test_cat, y_test)
print(f"CNN with Embeddings Mean Absolute Percentage Error: {cnn_mape}")

# Plot training and validation loss for CNN with Embeddings
plot_history(cnn_history, 'CNN with Embeddings Training and Validation Loss')
```