**LSTM EXAMPLE OF PERSONAL FINANCE**

**SETPS 1, 2 , 6 AND 8 EXAMPLE SOLUTION**

**1. Data Visualization**

We can use libraries like matplotlib and seaborn to create visualizations that provide insights into financial data trends and predictions.

**Add these features to the code:**

* Visualize correlations in the data using a heatmap.
* Plot the distribution of the target variable (Savings for Property (£)).
* Compare predicted values vs. true values for better insight into the model's performance.

**2. Decision-Making Support**

To add decision-making support:

* Allow the user to input financial goals (e.g., target savings).
* Provide insights based on predictions, like expected savings per interval (e.g., daily, weekly, monthly).
* Recommend steps to achieve defined financial goals based on predictions.

pip install streamlit, matplotlib, tensorflow,

pip install streamlit

streamlit run finance\_prediction\_app.py

import streamlit as st  
import pandas as pd  
import numpy as np  
from sklearn.preprocessing import MinMaxScaler  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import LSTM, Dense  
import plotly.graph\_objects as go  
  
# Step 0: File Upload and Basic Validation  
st.title("Personal Finance Management System with LSTM and Interactive Features")  
  
uploaded\_file = st.file\_uploader("Upload your CSV file:", type=["csv"])  
if uploaded\_file is not None:  
 try:  
 # Read uploaded CSV file  
 data = pd.read\_csv(uploaded\_file)  
 st.write("Data Preview:", data.head())  
 except Exception as e:  
 st.error(f"Error reading file: {e}")  
 st.stop()  
else:  
 st.warning("Please upload a CSV file to proceed.")  
 st.stop()  
  
# Step 1: Data Cleaning and Preprocessing  
st.subheader("Step 1: Data Cleaning and Preprocessing")  
  
# Define target column  
target\_col = 'Savings for Property (£)'  
  
try:  
 # Drop non-numeric columns  
 non\_numeric\_columns = data.select\_dtypes(exclude=["number"]).columns  
 if non\_numeric\_columns.size > 0:  
 st.write("Dropping non-numeric columns:", non\_numeric\_columns.tolist())  
 data = data.drop(columns=non\_numeric\_columns)  
  
 # Check if the target column exists  
 if target\_col not in data.columns:  
 st.error(f"Required column '{target\_col}' is missing in the dataset.")  
 st.stop()  
  
 # Drop rows with missing target values and fill remaining NaNs with column means  
 data\_cleaned = data.dropna(subset=[target\_col])  
 data\_cleaned.fillna(data\_cleaned.mean(), inplace=True)  
  
 st.write("Cleaned Data Preview:", data\_cleaned.head())  
except Exception as e:  
 st.error(f"Data cleaning error: {e}")  
 st.stop()  
  
# Step 2: Data Visualization  
st.subheader("Step 2: Data Visualization")  
try:  
 fig = go.Figure()  
 fig.add\_trace(go.Scatter(x=np.arange(len(data\_cleaned)), y=data\_cleaned[target\_col], mode='lines', name=target\_col))  
 fig.update\_layout(title="Savings for Property Over Time", xaxis\_title="Index", yaxis\_title="Savings (£)")  
 st.plotly\_chart(fig)  
except Exception as e:  
 st.error(f"Data visualization error: {e}")  
 st.stop()  
  
# Step 3: LSTM Model for Forecasting  
st.subheader("Step 3: LSTM Model for Forecasting")  
try:  
 # Scale the data  
 scaler = MinMaxScaler()  
 data\_cleaned\_scaled = scaler.fit\_transform(data\_cleaned[[target\_col]])  
  
 # Create sequences for LSTM  
 sequence\_length = 10  
 X, y = [], []  
 for i in range(len(data\_cleaned\_scaled) - sequence\_length):  
 X.append(data\_cleaned\_scaled[i:i+sequence\_length])  
 y.append(data\_cleaned\_scaled[i+sequence\_length])  
  
 X, y = np.array(X), np.array(y)  
  
 # Split into training and testing sets  
 split\_idx = int(0.8 \* len(X))  
 X\_train, X\_test = X[:split\_idx], X[split\_idx:]  
 y\_train, y\_test = y[:split\_idx], y[split\_idx:]  
  
 # Define LSTM model  
 model = Sequential([  
 LSTM(50, activation='relu', input\_shape=(X\_train.shape[1], X\_train.shape[2])),  
 Dense(1)  
 ])  
 model.compile(optimizer='adam', loss='mse')  
 st.write("Training LSTM model...")  
 model.fit(X\_train, y\_train, epochs=5, batch\_size=16, verbose=0)  
  
 # Predictions  
 y\_pred = model.predict(X\_test)  
 y\_pred\_rescaled = scaler.inverse\_transform(y\_pred)  
 y\_test\_rescaled = scaler.inverse\_transform(y\_test)  
  
 # Plot predictions vs actual  
 fig = go.Figure()  
 fig.add\_trace(go.Scatter(y=y\_test\_rescaled.flatten(), mode='lines', name='Actual'))  
 fig.add\_trace(go.Scatter(y=y\_pred\_rescaled.flatten(), mode='lines', name='Predicted'))  
 fig.update\_layout(title="LSTM Predictions vs Actual", xaxis\_title="Index", yaxis\_title="Savings (£)")  
 st.plotly\_chart(fig)  
except Exception as e:  
 st.error(f"Model training/prediction error: {e}")  
 st.stop()  
  
# Step 4: Decision-Making Support  
st.subheader("Step 4: Decision-Making Support")  
try:  
 current\_savings = data\_cleaned[target\_col].iloc[-1] # Ensure data\_cleaned is defined before this step  
 interval = st.selectbox("Select Interval:", ["Daily", "Weekly", "Monthly"])  
 savings\_goal = st.number\_input("Set Your Savings Target (£):", min\_value=0.0, step=100.0)  
  
 if current\_savings < savings\_goal:  
 st.warning(f"You need to save an additional £{savings\_goal - current\_savings:.2f} to meet your target.")  
 else:  
 st.success("Congratulations! You have met your savings goal.")  
except Exception as e:  
 st.error(f"Decision-making support error: {e}")  
  
# Step 5: Interactivity and Real-Time Updates  
st.subheader("Step 5: Interactivity and Real-Time Updates")  
try:  
 real\_time\_savings = st.slider("Adjust Current Savings (£):", min\_value=0, max\_value=int(current\_savings + 5000), value=int(current\_savings))  
 updated\_goal\_status = "Met" if real\_time\_savings >= savings\_goal else "Not Met"  
 st.write(f"Updated Goal Status: {updated\_goal\_status}")  
except Exception as e:  
 st.error(f"Real-time updates error: {e}")  
  
# Step 6: Scenario Planning and Forecasting  
st.subheader("Step 6: Scenario Planning and Forecasting")  
try:  
 scenario\_increase = st.number\_input("Increase Savings by (%):", min\_value=0, max\_value=100, step=5)  
 forecasted\_savings = real\_time\_savings \* (1 + scenario\_increase / 100)  
 st.write(f"If you increase savings by {scenario\_increase}%, your forecasted savings will be £{forecasted\_savings:.2f}.")  
except Exception as e:  
 st.error(f"Scenario planning error: {e}")