Project– 02

Title :- Sales Forecasting and Interactive Dashboard

Objective:

To forecast future sales for a retail company using time series analysis techniques and visualize sales trends through an interactive dashboard.

Workflow:

1. **Data Collection:**
   - Import historical retail sales data (CSV, SQL, or API).
   - Include features such as date, sales amount, store, product category, etc.
2. **Data Preprocessing:**
   - Handle missing values, outliers, and incorrect timestamps.
   - Aggregate data to required time intervals (daily/weekly/monthly).
3. **Time Series Decomposition:**
   - Decompose data into trend, seasonality, and residual components to understand underlying patterns.
4. **Model Training:**
   - **ARIMA Model:** Traditional statistical forecasting based on autocorrelation and seasonality.
   - **Prophet Model:** Facebook's additive model for easy handling of holidays and trends.
   - **LSTM Model:** Deep learning approach to capture complex temporal dependencies in sales data.
5. **Model Evaluation:**
   - Compare performance using RMSE, MAE, and MAPE metrics.
   - Select the best-performing model for final forecasting.
6. **Forecasting:**
   - Generate future sales predictions for upcoming weeks or months.
   - Store predicted values for visualization.
7. **Dashboard Visualization:**
   - **Sales Trend Plot:** Actual vs Predicted sales over time.
   - **Seasonality Charts:** Monthly and weekly sales trends.
   - **Model Comparison Graphs:** Evaluate ARIMA, Prophet, and LSTM forecasts visually.
   - **Interactive Dashboard:** Built using **Plotly Dash / Streamlit / Power BI** for dynamic user interaction.

~By HARSH KUMAR SAINI

Code :-

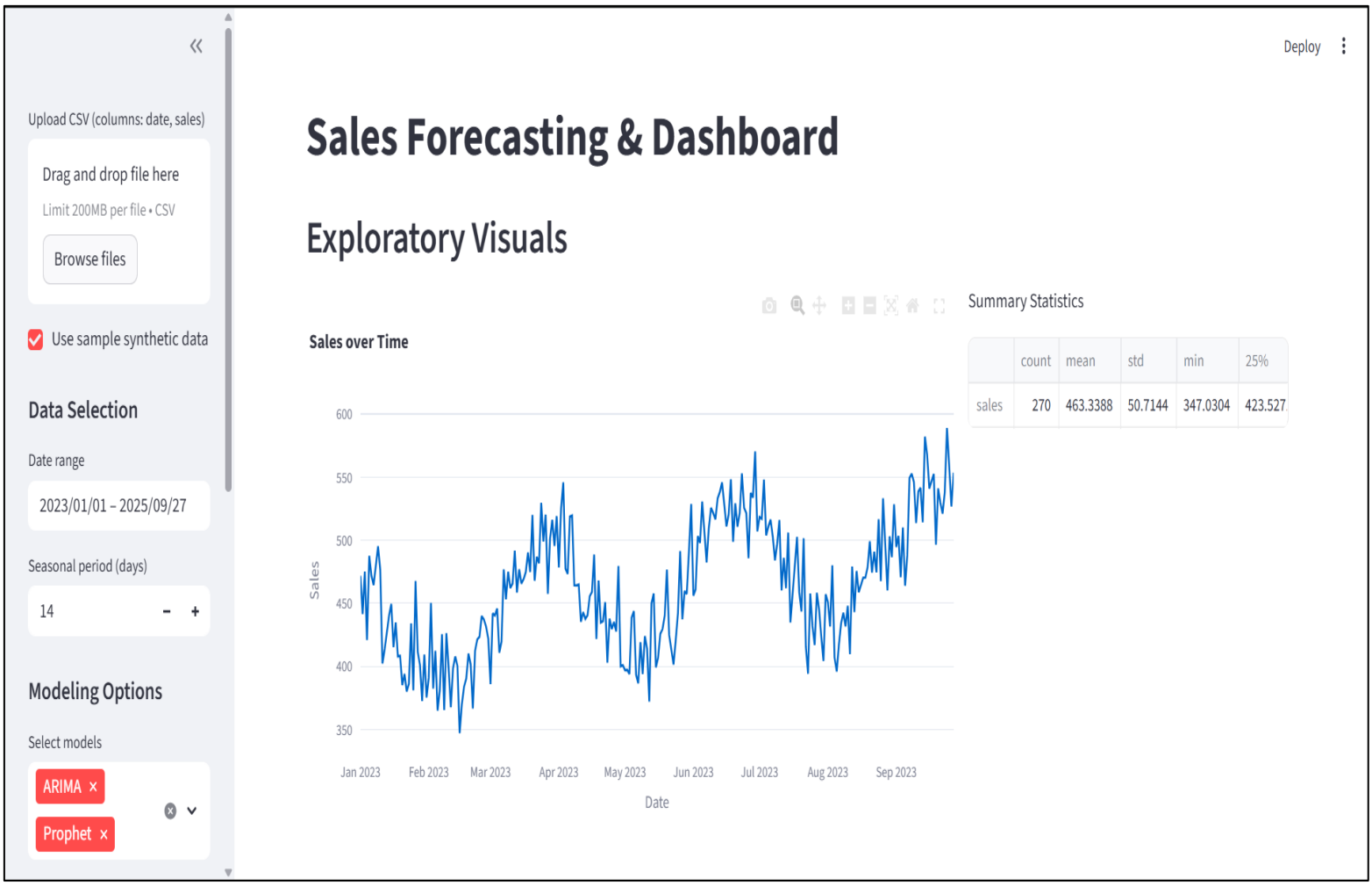
```
import streamlit as st
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
from prophet import Prophet
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_absolute_error, mean_squared_error
st.set_page_config(layout="wide", page_title="Sales Forecast Dashboard")
st.title("Sales Forecasting & Dashboard")
data_file = st.sidebar.file_uploader("Upload CSV (columns: date, sales)", type=["csv"])
use_sample = st.sidebar.checkbox("Use sample synthetic data", value=True)
if data_file is not None:
    df = pd.read_csv(data_file)
elif use_sample:
    np.random.seed(42)
    periods = 1000
    dates = pd.date_range("2021-01-01", periods=periods, freq="D")
    trend = np.linspace(200, 500, periods)
    seasonal = 50 * np.sin(np.arange(periods) / 14)
    noise = np.random.normal(0, 25, periods)
    sales = trend + seasonal + noise
    df = pd.DataFrame({"date": dates, "sales": sales})
else:
    st.stop()
df["date"] = pd.to_datetime(df["date"])
df = df.sort_values("date").reset_index(drop=True)
st.sidebar.header("Data Selection")
min_date = df["date"].min()
max_date = df["date"].max()
date_range = st.sidebar.date_input("Date range", [min_date, max_date])
df_plot = df[(df["date"] >= pd.to_datetime(date_range[0])) & (df["date"] <=
pd.to_datetime(date_range[1]))].copy()
st.header("Exploratory Visuals")
col1, col2 = st.columns([2,1])
with col1:
    fig = px.line(df_plot, x="date", y="sales", title="Sales over Time", labels={"sales":"Sales","date":"Date"})
    st.plotly_chart(fig, use_container_width=True)
with col2:
    st.write("Summary Statistics")
    st.dataframe(df_plot["sales"].describe().to_frame().T)
st.header("Time Series Decomposition")
```

```
period = st.sidebar.number_input("Seasonal period (days)", min_value=2, max_value=365, value=14)
decomp = seasonal_decompose(df.set_index("date")["sales"].asfreq("D").fillna(method="ffill"), model="additive",
period=period)
decomp_df = pd.DataFrame({
   "observed": decomp.observed,
   "trend": decomp.trend,
   "seasonal": decomp.seasonal,
   "resid": decomp.resid
}).reset_index()
fig_decomp = make = go.Figure()
make.add_trace(go.Scatter(x=decomp_df["date"], y=decomp_df["observed"], name="Observed"))
make.add_trace(go.Scatter(x=decomp_df["date"], y=decomp_df["trend"], name="Trend"))
make.add_trace(go.Scatter(x=decomp_df["date"], y=decomp_df["seasonal"], name="Seasonal"))
make.update_layout(title="Decomposition (observed, trend, seasonal)")
st.plotly_chart(make, use_container_width=True)
st.sidebar.header("Modeling Options")
models = st.sidebar.multiselect("Select models", ["ARIMA", "Prophet", "LSTM"], default=["ARIMA","Prophet"])
forecast_days = st.sidebar.number_input("Forecast horizon (days)", min_value=7, max_value=365, value=30)
train_frac = st.sidebar.slider("Train fraction", 0.5, 0.95, 0.8)
if "ARIMA" in models:
   p = st.sidebar.number_input("ARIMA p", 0, 5, 2)
   d = st.sidebar.number_input("ARIMA d", 0, 2, 1)
   q = st.sidebar.number_input("ARIMA q", 0, 5, 2)
if "LSTM" in models:
   seq_len = st.sidebar.number_input("LSTM sequence length", 3, 60, 30)
   epochs = st.sidebar.number_input("LSTM epochs", 1, 200, 50)
   batch = st.sidebar.number_input("LSTM batch size", 1, 256, 16)
if st.sidebar.button("Run Models"):
   df_model = df[["date","sales"]].rename(columns={"date":"ds","sales":"y"})
   split_idx = int(len(df_model) * train_frac)
   train = df_model.iloc[:split_idx].copy()
   test = df_model.iloc[split_idx:].copy()
   results = []
   pred_df = pd.DataFrame({"ds": test["ds"].values})
   if "ARIMA" in models:
      ar_series = train.set_index("ds")["y"].asfreq("D").fillna(method="ffill")
      ar_model = ARIMA(ar_series, order=(int(p),int(d),int(q))).fit()
      ar_fore = ar_model.forecast(steps=len(test))
      pred_df["ARIMA"] = ar_fore.values
      mae = mean_absolute_error(test["y"], pred_df["ARIMA"])
      rmse = mean_squared_error(test["y"], pred_df["ARIMA"], squared=False)
      results.append(["ARIMA", mae, rmse])
   if "Prophet" in models:
      m = Prophet(daily_seasonality=True, weekly_seasonality=True, yearly_seasonality=True)
      m.fit(train)
      future = m.make_future_dataframe(periods=forecast_days)
      forecast = m.predict(future)
      fcst_test = forecast[forecast["ds"].isin(test["ds"])]
      if not fcst_test.empty:
         pred_df["Prophet"] = fcst_test["yhat"].values
         mae = mean_absolute_error(test["y"], pred_df["Prophet"])
         rmse = mean_squared_error(test["y"], pred_df["Prophet"], squared=False)
         results.append(["Prophet", mae, rmse])
      fcst_future = forecast[["ds","yhat"]].tail(forecast_days)
   if "LSTM" in models:
      values = df_model["y"].values.reshape(-1,1)
      scaler = MinMaxScaler()
```

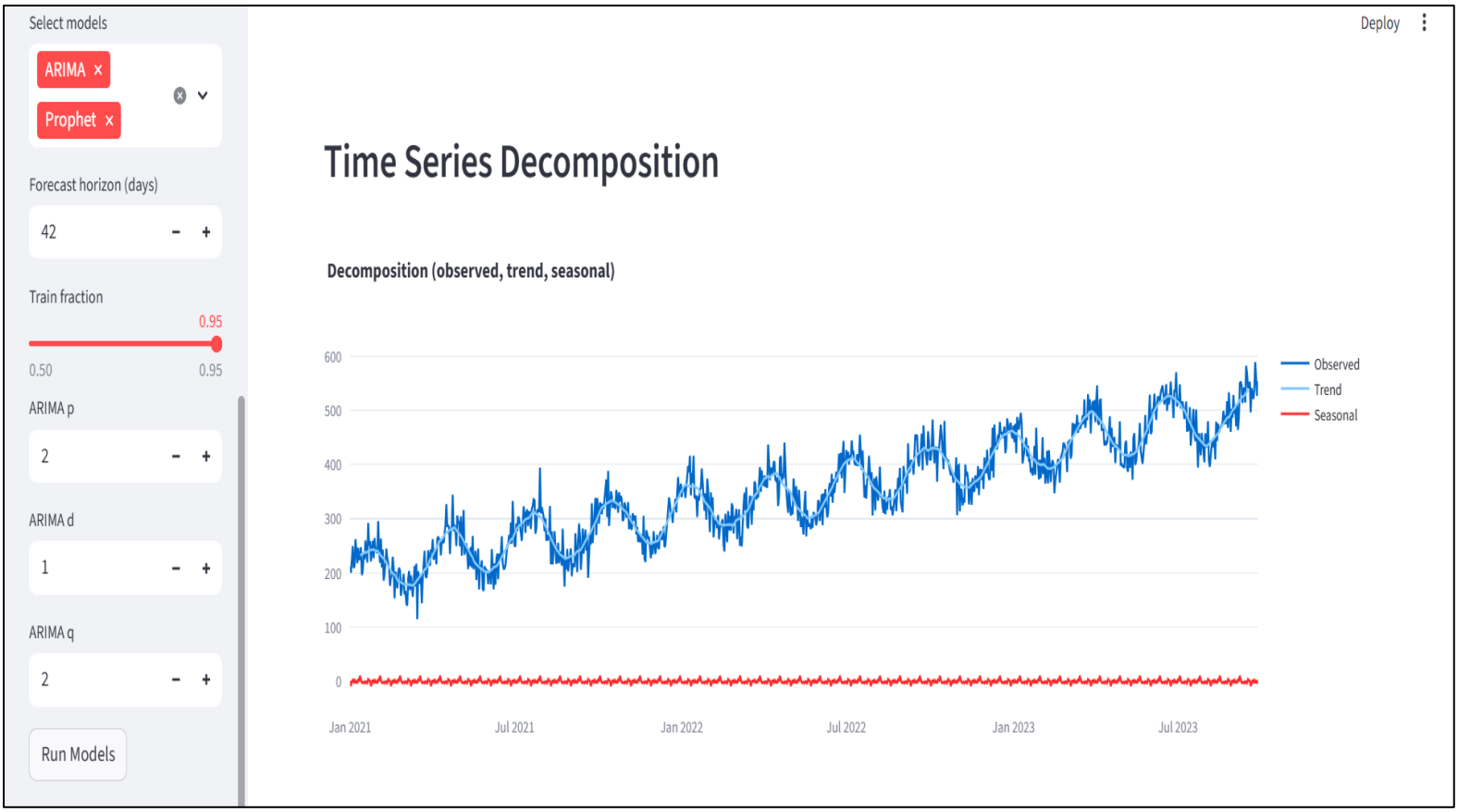
```
scaled = scaler.fit_transform(values)
    def create_seq(arr, L):
      X=[]; Y=[]
      for i in range(len(arr)-L):
        X.append(arr[i:i+L])
        Y.append(arr[i+L])
      return np.array(X), np.array(Y)
    X_all, Y_all = create_seq(scaled, int(seq_len))
    split_seq = int(len(X_all) * train_frac)
    X_tr, X_te = X_all[:split_seq], X_all[split_seq:]
    y_tr, y_te = Y_all[:split_seq], Y_all[split_seq:]
    X_tr = X_tr.reshape((X_tr.shape[0], X_tr.shape[1], 1))
    X_te = X_te.reshape((X_te.shape[0], X_te.shape[1], 1))
    model = Sequential()
    model.add(LSTM(64, input_shape=(int(seq_len),1)))
    model.add(Dense(1))
    model.compile(optimizer="adam", loss="mse")
    es = EarlyStopping(monitor="val_loss", patience=5, restore_best_weights=True)
    model.fit(X_tr, y_tr, validation_data=(X_te, y_te), epochs=int(epochs), batch_size=int(batch), callbacks=[es],
verbose=0)
    preds = model.predict(X_te)
    preds_inv = scaler.inverse_transform(preds).flatten()
    true_inv = scaler.inverse_transform(y_te).flatten()
    test_dates_for_lstm = df_model["ds"].iloc[len(df_model)-len(preds_inv):].values
    pred_lstm_series = pd.Series(preds_inv, index=test_dates_for_lstm)
    pred_for_test = []
    for d in test["ds"].iloc[:len(preds_inv)]:
      if d in pred_lstm_series.index:
        pred_for_test.append(pred_lstm_series.loc[d])
      else:
        pred_for_test.append(np.nan)
    pred_df["LSTM"] = pred_for_test
    mae = mean_absolute_error(true_inv[:len(preds_inv)], preds_inv)
    rmse = mean_squared_error(true_inv[:len(preds_inv)], preds_inv, squared=False)
    results.append(["LSTM", mae, rmse])
  st.subheader("Model Performance")
  if results:
    res_df = pd.DataFrame(results, columns=["Model","MAE","RMSE"])
    st.dataframe(res_df)
  st.subheader("Combined Forecast Plot")
  figc = go.Figure()
  figc.add_trace(go.Scatter(x=df_model["ds"], y=df_model["y"], name="Actual", line=dict(color="black")))
  if "ARIMA" in models:
    figc.add_trace(go.Scatter(x=pred_df["ds"], y=pred_df["ARIMA"], name="ARIMA"))
  if "Prophet" in models:
    figc.add_trace(go.Scatter(x=fcst_future["ds"], y=fcst_future["yhat"], name="Prophet Future"))
  if "LSTM" in models:
    if "LSTM" in pred_df.columns:
      figc.add_trace(go.Scatter(x=pred_df["ds"], y=pred_df["LSTM"], name="LSTM"))
  figc.update_layout(title="Actual vs Forecasts", xaxis_title="Date", yaxis_title="Sales")
  st.plotly_chart(figc, use_container_width=True)
  st.subheader("Forecast Table")
  if "Prophet" in models:
    out = fcst_future.rename(columns={"ds":"date","yhat":"forecast"})[["date","forecast"]].reset_index(drop=True)
    st.dataframe(out)
  else:
    last_date = df_model["ds"].iloc[-1]
    future_dates = pd.date_range(last_date + pd.Timedelta(days=1), periods=forecast_days)
```

```
if "ARIMA" in models:
        ar_full = ARIMA(df_model.set_index("ds")["y"].asfreq("D").fillna(method="ffill"), order=(int(p),int(d),int(q))).fit()
        ar_future_full = ar_full.forecast(steps=forecast_days)
        out = pd.DataFrame({"date": future_dates, "forecast": ar_future_full.values})
        st.dataframe(out)
    elif "LSTM" in models:
        seq = scaled[-int(seq_len):]
        preds_f = []
        seq_copy = seq.copy()
        for _ in range(forecast_days):
            x_input = seq_copy.reshape((1, int(seq_len), 1))
            p = model.predict(x_input)
            preds_f.append(p.flatten()[0])
            seq_copy = np.vstack([seq_copy[1:], p])
        preds_inv_f = scaler.inverse_transform(np.array(preds_f).reshape(-1,1)).flatten()
        out = pd.DataFrame({"date": future_dates, "forecast": preds_inv_f})
        st.dataframe(out)
st.success("Forecasting complete")
```

Output: -



~By HARSH KUMAR SAINI



~By HARSH KUMAR SAINI