```
import streamlit as st
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import re
from datetime import datetime, timedelta
import warnings
warnings.filterwarnings('ignore')

# Configure the page
st.set_page_config(
    page_title="Smart Budget Tracker",
    page_icon="💰",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Custom CSS for better styling
st.markdown("""
<style>
    .main-header {
        font-size: 3rem;
        color: #1f77b4;
        text-align: center;
        margin-bottom: 2rem;
```

```
    }
    .metric-card {
        background-color: #f0f2f6;
        padding: 1rem;
        border-radius: 10px;
        border-left: 4px solid #1f77b4;
    }
    .section-header {
        color: #1f77b4;
        border-bottom: 2px solid #1f77b4;
        padding-bottom: 0.5rem;
        margin-bottom: 1rem;
    }
</style>
""", unsafe_allow_html=True)


class SmartBudgetTracker:
    def _init_(self):
        self.categories = {
            'Food & Dining': ['zomato', 'swiggy', 'restaurant', 'cafe', 'food', 'dining', 'pizza',
'burger'],
            'Transportation': ['uber', 'ola', 'taxi', 'bus', 'metro', 'fuel', 'petrol', 'transport'],
            'Shopping': ['amazon', 'flipkart', 'myntra', 'shopping', 'mall', 'store'],
            'Entertainment': ['netflix', 'movie', 'cinema', 'concert', 'game', 'entertainment'],
            'Education': ['bookstore', 'books', 'course', 'education', 'university', 'college'],
            'Healthcare': ['hospital', 'clinic', 'pharmacy', 'medicine', 'doctor'],
            'Bills & Utilities': ['electricity', 'water', 'internet', 'phone', 'bill', 'utility'],
```

```
        'Salary & Income': ['salary', 'income', 'payment received'],
        'Other': ['other', 'miscellaneous']
    }

def categorize_transaction(self, description, amount):
    """Categorize transaction using NLP-based keyword matching"""
    description_lower = str(description).lower()

    # Check for income (positive amounts)
    if amount > 0:
        return 'Salary & Income'

    # Check each category
    for category, keywords in self.categories.items():
        if any(keyword in description_lower for keyword in keywords):
            return category

    return 'Other'

def generate_sample_data(self):
    """Generate sample transaction data for demonstration"""
    np.random.seed(42)

    # Sample transaction descriptions
    descriptions = [
        "Zomato Food Delivery", "Swiggy Dinner Order", "Uber Ride to College",
        "Amazon Shopping", "Bookstore Purchase", "Netflix Subscription",
```

```
    "College Fee Payment", "Mobile Recharge", "Electricity Bill",
    "Medical Checkup", "Movie Tickets", "Coffee Cafe",
    "Fuel Station", "Salary Credit", "Freelance Payment"
]

dates = pd.date_range('2024-01-01', '2024-12-31', freq='D')
n_transactions = 500

data = []
for _ in range(n_transactions):
    date = np.random.choice(dates)
    description = np.random.choice(descriptions)
    amount = np.random.choice([
        np.random.uniform(-500, -50),  # Expenses
        np.random.uniform(1000, 5000)  # Income
    ])

    category = self.categorize_transaction(description, amount)

    data.append({
        'Date': date,
        'Description': description,
        'Amount': amount,
        'Category': category,
        'Type': 'Income' if amount > 0 else 'Expense'
    })
```

```
        return pd.DataFrame(data)

    def predict_future_expenses(self, df, months=3):
        """Simple linear regression to predict future expenses"""
        df_expenses = df[df['Type'] == 'Expense'].copy()
        monthly_expenses = df_expenses.groupby(pd.Grouper(key='Date',
freq='M'))['Amount'].sum().abs()

        if len(monthly_expenses) < 2:
            return None

        # Simple linear regression
        x = np.arange(len(monthly_expenses))
        y = monthly_expenses.values

        # Fit linear model
        coefficients = np.polyfit(x, y, 1)
        future_trend = np.polyval(coefficients, np.arange(len(monthly_expenses),
len(monthly_expenses) + months))

        future_dates = pd.date_range(monthly_expenses.index[-1] +
pd.DateOffset(months=1),
                                     periods=months, freq='M')

        return pd.Series(future_trend, index=future_dates)

def main():
    # Initialize tracker
```

```
    tracker = SmartBudgetTracker()

    # Header
    st.markdown('<h1 class="main-header">💰 Smart Budget Tracker</h1>', unsafe_allow_html=True)

    # Sidebar
    st.sidebar.title("Navigation")
    menu_options = ["Dashboard", "Add Transaction", "Budget Analysis", "Future Predictions"]
    selected_menu = st.sidebar.selectbox("Select Section", menu_options)

    # Load or generate data
    if 'transactions' not in st.session_state:
        st.session_state.transactions = tracker.generate_sample_data()

    df = st.session_state.transactions

    if selected_menu == "Dashboard":
        show_dashboard(df, tracker)
    elif selected_menu == "Add Transaction":
        add_transaction(df, tracker)
    elif selected_menu == "Budget Analysis":
        budget_analysis(df, tracker)
    elif selected_menu == "Future Predictions":
        future_predictions(df, tracker)

def show_dashboard(df, tracker):
```

```
    """Main dashboard view"""

    # Key metrics
    col1, col2, col3, col4 = st.columns(4)

    total_income = df[df['Type'] == 'Income']['Amount'].sum()
    total_expenses = df[df['Type'] == 'Expense']['Amount'].sum()
    net_savings = total_income + total_expenses  # Expenses are negative
    avg_monthly_expense = df[df['Type'] == 'Expense'].groupby(
        pd.Grouper(key='Date', freq='M')
    )['Amount'].sum().abs().mean()

    with col1:
        st.metric("Total Income", f"₹{total_income:,.2f}")
    with col2:
        st.metric("Total Expenses", f"₹{abs(total_expenses):,.2f}")
    with col3:
        st.metric("Net Savings", f"₹{net_savings:,.2f}")
    with col4:
        st.metric("Avg Monthly Expense", f"₹{avg_monthly_expense:,.0f}")

    # Charts
    col1, col2 = st.columns(2)

    with col1:
        st.markdown('<h3 class="section-header">Monthly Income vs Expenses</h3>',
unsafe_allow_html=True)
```

```
        monthly_data = df.groupby([pd.Grouper(key='Date', freq='M'),
'Type'])['Amount'].sum().unstack()
        monthly_data['Expense'] = monthly_data['Expense'].abs()

        fig = go.Figure()
        fig.add_trace(go.Bar(x=monthly_data.index, y=monthly_data['Income'],
name='Income', marker_color='green'))
        fig.add_trace(go.Bar(x=monthly_data.index, y=monthly_data['Expense'],
name='Expenses', marker_color='red'))
        fig.update_layout(barmode='group', height=400)
        st.plotly_chart(fig, use_container_width=True)

    with col2:
        st.markdown('<h3 class="section-header">Expense Distribution by
Category</h3>', unsafe_allow_html=True)
        expense_by_category = df[df['Type'] ==
'Expense'].groupby('Category')['Amount'].sum().abs()

        fig = px.pie(values=expense_by_category.values,
names=expense_by_category.index,
                     title="Expense Categories")
        fig.update_traces(textposition='inside', textinfo='percent+label')
        fig.update_layout(height=400)
        st.plotly_chart(fig, use_container_width=True)

    # Recent transactions
    st.markdown('<h3 class="section-header">Recent Transactions</h3>',
unsafe_allow_html=True)
    recent_transactions = df.sort_values('Date', ascending=False).head(10)[['Date',
'Description', 'Amount', 'Category']]
```

```
    recent_transactions['Amount'] = recent_transactions['Amount'].apply(lambda x: f"₹{x:,.2f}")
    st.dataframe(recent_transactions, use_container_width=True)


def add_transaction(df, tracker):
    """Add new transaction manually"""
    st.markdown('<h2 class="section-header">Add New Transaction</h2>', unsafe_allow_html=True)

    col1, col2 = st.columns(2)

    with col1:
        transaction_date = st.date_input("Date", datetime.now())
        description = st.text_input("Description", placeholder="e.g., Zomato Food Delivery")
        amount = st.number_input("Amount", value=0.0, step=10.0)

    with col2:
        transaction_type = st.selectbox("Type", ["Expense", "Income"])
        if transaction_type == "Expense" and amount > 0:
            amount = -amount

        # Auto-categorize or manual override
        auto_category = tracker.categorize_transaction(description, amount)
        category = st.selectbox("Category",
                                list(tracker.categories.keys()) + ['Other'],
                                index=list(tracker.categories.keys()).index(auto_category)
```

```
                        if auto_category in tracker.categories.keys() else
len(tracker.categories))

    if st.button("Add Transaction"):
        if description and amount != 0:
            new_transaction = {
                'Date': transaction_date,
                'Description': description,
                'Amount': amount,
                'Category': category,
                'Type': 'Income' if amount > 0 else 'Expense'
            }

            # Add to dataframe
            new_df = pd.DataFrame([new_transaction])
            st.session_state.transactions = pd.concat([df, new_df], ignore_index=True)

            st.success("Transaction added successfully!")
            st.rerun()
        else:
            st.error("Please fill in all fields correctly.")

def budget_analysis(df, tracker):
    """Budget analysis and insights"""
    st.markdown('<h2 class="section-header">Budget Analysis & Insights</h2>',
unsafe_allow_html=True)

    # Monthly spending trends
```

```
    monthly_expenses = df[df['Type'] == 'Expense'].groupby(
        pd.Grouper(key='Date', freq='M')
    )['Amount'].sum().abs()

    col1, col2 = st.columns(2)

    with col1:
        st.markdown('<h4>Monthly Spending Trend</h4>', unsafe_allow_html=True)
        fig = px.line(x=monthly_expenses.index, y=monthly_expenses.values,
                      labels={'x': 'Month', 'y': 'Amount (₹)'})
        fig.update_traces(line=dict(color='red', width=3))
        st.plotly_chart(fig, use_container_width=True)

    with col2:
        st.markdown('<h4>Category-wise Monthly Analysis</h4>',
unsafe_allow_html=True)
        category_monthly = df[df['Type'] == 'Expense'].groupby(
            [pd.Grouper(key='Date', freq='M'), 'Category']
        )['Amount'].sum().abs().unstack().fillna(0)

        fig = px.area(category_monthly, title="Spending by Category Over Time")
        st.plotly_chart(fig, use_container_width=True)

    # Spending insights
    st.markdown('<h4>Spending Insights</h4>', unsafe_allow_html=True)

    current_month = datetime.now().replace(day=1)
```

```
    last_month = (current_month - timedelta(days=1)).replace(day=1)

    current_month_expenses = df[
        (df['Date'].dt.to_period('M') == current_month.strftime('%Y-%m')) &
        (df['Type'] == 'Expense')
    ]['Amount'].sum()

    last_month_expenses = df[
        (df['Date'].dt.to_period('M') == last_month.strftime('%Y-%m')) &
        (df['Type'] == 'Expense')
    ]['Amount'].sum()

    if last_month_expenses != 0:
        change_percent = ((abs(current_month_expenses) - abs(last_month_expenses)) /
abs(last_month_expenses)) * 100
        change_text = "increase" if change_percent > 0 else "decrease"

        insight_col1, insight_col2, insight_col3 = st.columns(3)

        with insight_col1:
            st.metric("Current Month Spending", f"₹{abs(current_month_expenses):,.2f}",
                      f"{change_percent:+.1f}%")

        with insight_col2:
            # Top spending category
            top_category = df[
                (df['Date'].dt.to_period('M') == current_month.strftime('%Y-%m')) &
```

```
            (df['Type'] == 'Expense')
        ].groupby('Category')['Amount'].sum().abs().idxmax()
        st.metric("Top Spending Category", top_category)

    with insight_col3:
        # Average daily spending
        daily_avg = df[
            (df['Date'].dt.to_period('M') == current_month.strftime('%Y-%m')) &
            (df['Type'] == 'Expense')
        ].groupby('Date')['Amount'].sum().abs().mean()
        st.metric("Avg Daily Spending", f"₹{daily_avg:,.1f}")

def future_predictions(df, tracker):
    """Future expense predictions"""
    st.markdown('<h2 class="section-header">Future Expense Predictions</h2>',
unsafe_allow_html=True)

    # Generate predictions
    future_expenses = tracker.predict_future_expenses(df, months=6)

    if future_expenses is not None:
        # Historical data
        historical_expenses = df[df['Type'] == 'Expense'].groupby(
            pd.Grouper(key='Date', freq='M')
        )['Amount'].sum().abs()

        # Combine historical and predicted data
```

```
    combined_data = pd.concat([historical_expenses, future_expenses])
    combined_data['Type'] = ['Historical'] * len(historical_expenses) + ['Predicted'] *
len(future_expenses)

    # Create plot
    fig = go.Figure()

    # Historical data
    fig.add_trace(go.Scatter(
        x=historical_expenses.index,
        y=historical_expenses.values,
        mode='lines+markers',
        name='Historical Expenses',
        line=dict(color='blue', width=3)
    ))

    # Predicted data
    fig.add_trace(go.Scatter(
        x=future_expenses.index,
        y=future_expenses.values,
        mode='lines+markers',
        name='Predicted Expenses',
        line=dict(color='red', width=3, dash='dash')
    ))

    fig.update_layout(
        title="Expense Prediction for Next 6 Months",
```

```
    xaxis_title="Month",
    yaxis_title="Amount (₹)",
    height=500
)

st.plotly_chart(fig, use_container_width=True)

# Prediction summary
st.markdown('<h4>Prediction Summary</h4>', unsafe_allow_html=True)

pred_col1, pred_col2, pred_col3 = st.columns(3)

with pred_col1:
    avg_predicted = future_expenses.mean()
    st.metric("Avg Predicted Monthly", f"₹{avg_predicted:,.0f}")

with pred_col2:
    max_predicted = future_expenses.max()
    st.metric("Max Predicted Month", f"₹{max_predicted:,.0f}")

with pred_col3:
    total_predicted = future_expenses.sum()
    st.metric("Total Predicted (6 months)", f"₹{total_predicted:,.0f}")

# Recommendations
st.markdown('<h4>💡 Smart Recommendations</h4>', unsafe_allow_html=True)
```

```
        current_avg = df[df['Type'] == 'Expense'].groupby(
            pd.Grouper(key='Date', freq='M')
        )['Amount'].sum().abs().mean()

        predicted_avg = future_expenses.mean()

        if predicted_avg > current_avg * 1.1:
            st.warning("⚠ Your expenses are predicted to increase. Consider reviewing your spending habits.")
            st.info("💡 *Tips:* Try to reduce discretionary spending and set monthly budget limits.")
        else:
            st.success("✅ Your spending pattern looks stable. Keep up the good financial habits!")

    else:
        st.info("Need more historical data (at least 2 months) to generate predictions.")

if _name_ == "_main_":
    main()
```