```python
import tkinter as tk
from tkinter import ttk, messagebox
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from datetime import datetime, timedelta
import calendar

class ExpenseTracker:
    def __init__(self, root):
        self.root = root
        self.root.title("Expense Tracker")
        self.root.geometry("1000x700")
        self.root.configure(bg='#f0f0f0')

        # Initialize database
        self.init_db()

        # Create GUI
        self.create_gui()

        # Load initial data
        self.load_expenses()
        self.update_report()
```

```
  def init_db(self):
    """Initialize the SQLite database and create tables if they don't exist"""
    self.conn = sqlite3.connect('expenses.db')
    self.c = self.conn.cursor()

    # Create expenses table
    self.c.execute('''CREATE TABLE IF NOT EXISTS expenses
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  date TEXT NOT NULL,
                  amount REAL NOT NULL,
                  category TEXT NOT NULL,
                  description TEXT)''')

    # Create categories table with some default categories
    self.c.execute('''CREATE TABLE IF NOT EXISTS categories
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  name TEXT NOT NULL UNIQUE)''')

    # Insert default categories if they don't exist
    default_categories = ['Food', 'Transport', 'Entertainment', 'Utilities', 'Rent', 'Shopping',
'Healthcare', 'Other']
    for category in default_categories:
      try:
        self.c.execute("INSERT INTO categories (name) VALUES (?)", (category,))
      except sqlite3.IntegrityError:
```

```
            pass  # Category already exists

        self.conn.commit()

    def create_gui(self):
        # Create main frames
        self.input_frame = tk.LabelFrame(self.root, text="Add New Expense", padx=10, pady=10, bg='#f0f0f0')
        self.input_frame.pack(pady=10, padx=10, fill="x")

        self.list_frame = tk.LabelFrame(self.root, text="Recent Expenses", padx=10, pady=10, bg='#f0f0f0')
        self.list_frame.pack(pady=10, padx=10, fill="both", expand=True)

        self.report_frame = tk.LabelFrame(self.root, text="Reports", padx=10, pady=10, bg='#f0f0f0')
        self.report_frame.pack(pady=10, padx=10, fill="both", expand=True)

        # Input form
        tk.Label(self.input_frame, text="Date:", bg='#f0f0f0').grid(row=0, column=0, padx=5, pady=5, sticky='e')
        self.date_entry = tk.Entry(self.input_frame, width=15)
        self.date_entry.grid(row=0, column=1, padx=5, pady=5)
        self.date_entry.insert(0, datetime.now().strftime("%Y-%m-%d"))

        tk.Label(self.input_frame, text="Amount:", bg='#f0f0f0').grid(row=0, column=2, padx=5, pady=5, sticky='e')
```

```
        self.amount_entry = tk.Entry(self.input_frame, width=15)
        self.amount_entry.grid(row=0, column=3, padx=5, pady=5)

        tk.Label(self.input_frame, text="Category:", bg='#f0f0f0').grid(row=0, column=4, padx=5, pady=5, sticky='e')
        self.category_var = tk.StringVar()
        self.category_dropdown = ttk.Combobox(self.input_frame, textvariable=self.category_var, width=13)
        self.category_dropdown.grid(row=0, column=5, padx=5, pady=5)
        self.load_categories()

        tk.Label(self.input_frame, text="Description:", bg='#f0f0f0').grid(row=1, column=0, padx=5, pady=5, sticky='e')
        self.desc_entry = tk.Entry(self.input_frame, width=50)
        self.desc_entry.grid(row=1, column=1, columnspan=3, padx=5, pady=5, sticky='we')

        add_btn = tk.Button(self.input_frame, text="Add Expense", command=self.add_expense, bg='#4CAF50', fg='white')
        add_btn.grid(row=1, column=4, columnspan=2, padx=5, pady=5, sticky='we')

        # Expenses list
        columns = ("id", "date", "amount", "category", "description")
        self.tree = ttk.Treeview(self.list_frame, columns=columns, show="headings")

        self.tree.heading("id", text="ID")
        self.tree.heading("date", text="Date")
        self.tree.heading("amount", text="Amount")
```

```
self.tree.heading("category", text="Category")
self.tree.heading("description", text="Description")

self.tree.column("id", width=40)
self.tree.column("date", width=100)
self.tree.column("amount", width=80)
self.tree.column("category", width=100)
self.tree.column("description", width=200)

tree_scroll = ttk.Scrollbar(self.list_frame, orient="vertical", command=self.tree.yview)
self.tree.configure(yscrollcommand=tree_scroll.set)

self.tree.pack(side="left", fill="both", expand=True)
tree_scroll.pack(side="right", fill="y")

# Bind double click to delete
self.tree.bind("<Double-1>", self.delete_expense)

# Report section
report_top_frame = tk.Frame(self.report_frame, bg='#f0f0f0')
report_top_frame.pack(fill="x")

tk.Label(report_top_frame, text="Report Type:", bg='#f0f0f0').pack(side="left", padx=5)
self.report_type = tk.StringVar(value="Category")
report_combo = ttk.Combobox(report_top_frame, textvariable=self.report_type,
                            values=["Category", "Daily", "Monthly"], width=10)
```

```
        report_combo.pack(side="left", padx=5)
        report_combo.bind("<<ComboboxSelected>>", self.update_report)

        tk.Label(report_top_frame, text="Time Period:", bg='#f0f0f0').pack(side="left", padx=5)
        self.period_var = tk.StringVar(value="This Month")
        period_combo = ttk.Combobox(report_top_frame, textvariable=self.period_var,
                                    values=["This Week", "This Month", "Last Month", "This Year", "All Time"],
width=10)
        period_combo.pack(side="left", padx=5)
        period_combo.bind("<<ComboboxSelected>>", self.update_report)

        refresh_btn = tk.Button(report_top_frame, text="Refresh Report",
command=self.update_report, bg='#2196F3', fg='white')
        refresh_btn.pack(side="right", padx=5)

        # Matplotlib figure
        self.fig, self.ax = plt.subplots(figsize=(8, 4))
        self.canvas = FigureCanvasTkAgg(self.fig, master=self.report_frame)
        self.canvas.get_tk_widget().pack(fill="both", expand=True)

    def load_categories(self):
        """Load categories from database into the dropdown"""
        self.c.execute("SELECT name FROM categories ORDER BY name")
        categories = [row[0] for row in self.c.fetchall()]
        self.category_dropdown['values'] = categories
        if categories:
            self.category_var.set(categories[0])
```

```
    def add_expense(self):
        """Add a new expense to the database"""
        try:
            date = self.date_entry.get()
            amount = float(self.amount_entry.get())
            category = self.category_var.get()
            description = self.desc_entry.get()

            if not date or not category:
                messagebox.showerror("Error", "Please fill in all required fields")
                return

            self.c.execute("INSERT INTO expenses (date, amount, category, description) VALUES (?, ?, ?, ?)",
                           (date, amount, category, description))
            self.conn.commit()

            self.load_expenses()
            self.update_report()

            # Clear input fields
            self.amount_entry.delete(0, tk.END)
            self.desc_entry.delete(0, tk.END)

        except ValueError:
```

```
            messagebox.showerror("Error", "Please enter a valid amount")

    def load_expenses(self):
        """Load expenses from database into the treeview"""
        # Clear existing data
        for item in self.tree.get_children():
            self.tree.delete(item)

        self.c.execute("SELECT id, date, amount, category, description FROM expenses ORDER BY date DESC LIMIT 100")
        for row in self.c.fetchall():
            self.tree.insert("", "end", values=row)

    def delete_expense(self, event):
        """Delete the selected expense"""
        item = self.tree.selection()[0]
        expense_id = self.tree.item(item, "values")[0]

        if messagebox.askyesno("Confirm Delete", "Are you sure you want to delete this expense?"):
            self.c.execute("DELETE FROM expenses WHERE id=?", (expense_id,))
            self.conn.commit()
            self.load_expenses()
            self.update_report()

    def update_report(self, event=None):
        """Update the report based on selected filters"""
```

```
report_type = self.report_type.get()
period = self.period_var.get()

# Determine date range based on period selection
today = datetime.now().date()

if period == "This Week":
    start_date = today - timedelta(days=today.weekday())
    end_date = start_date + timedelta(days=6)
    date_condition = f"AND date BETWEEN '{start_date}' AND '{end_date}'"
elif period == "This Month":
    start_date = today.replace(day=1)
    end_date = today.replace(day=calendar.monthrange(today.year, today.month)[1])
    date_condition = f"AND date BETWEEN '{start_date}' AND '{end_date}'"
elif period == "Last Month":
    first_day_this_month = today.replace(day=1)
    last_day_last_month = first_day_this_month - timedelta(days=1)
    start_date = last_day_last_month.replace(day=1)
    end_date = last_day_last_month
    date_condition = f"AND date BETWEEN '{start_date}' AND '{end_date}'"
elif period == "This Year":
    start_date = today.replace(month=1, day=1)
    end_date = today.replace(month=12, day=31)
    date_condition = f"AND date BETWEEN '{start_date}' AND '{end_date}'"
else:  # All Time
    date_condition = ""
```

```
        # Clear previous plot
        self.ax.clear()

        if report_type == "Category":
            # Generate category report
            query = f"SELECT category, SUM(amount) FROM expenses WHERE 1=1 {date_condition} GROUP BY category ORDER BY SUM(amount) DESC"
            self.c.execute(query)
            data = self.c.fetchall()

            if not data:
                self.ax.text(0.5, 0.5, "No data available", ha='center', va='center', transform=self.ax.transAxes)
                self.canvas.draw()
                return

            categories = [row[0] for row in data]
            amounts = [row[1] for row in data]

            self.ax.pie(amounts, labels=categories, autopct='%1.1f%%', startangle=90)
            self.ax.set_title('Expenses by Category')

        elif report_type == "Daily":
            # Generate daily report
            query = f"SELECT date, SUM(amount) FROM expenses WHERE 1=1 {date_condition} GROUP BY date ORDER BY date"
```

```
        self.c.execute(query)
        data = self.c.fetchall()

        if not data:
            self.ax.text(0.5, 0.5, "No data available", ha='center', va='center',
transform=self.ax.transAxes)
            self.canvas.draw()
            return

        dates = [row[0] for row in data]
        amounts = [row[1] for row in data]

        self.ax.bar(dates, amounts)
        self.ax.set_title('Daily Expenses')
        self.ax.set_xlabel('Date')
        self.ax.set_ylabel('Amount')
        plt.setp(self.ax.xaxis.get_majorticklabels(), rotation=45)

    elif report_type == "Monthly":
        # Generate monthly report
        query = f"SELECT strftime('%Y-%m', date) as month, SUM(amount) FROM expenses
WHERE 1=1 {date_condition} GROUP BY month ORDER BY month"
        self.c.execute(query)
        data = self.c.fetchall()

        if not data:
```

```
            self.ax.text(0.5, 0.5, "No data available", ha='center', va='center',
transform=self.ax.transAxes)
            self.canvas.draw()
            return

        months = [row[0] for row in data]
        amounts = [row[1] for row in data]

        self.ax.bar(months, amounts)
        self.ax.set_title('Monthly Expenses')
        self.ax.set_xlabel('Month')
        self.ax.set_ylabel('Amount')
        plt.setp(self.ax.xaxis.get_majorticklabels(), rotation=45)

    self.fig.tight_layout()
    self.canvas.draw()

  def __del__(self):
    """Close database connection when the app is closed"""
    if hasattr(self, 'conn'):
      self.conn.close()

if __name__ == "__main__":
  root = tk.Tk()
  app = ExpenseTracker(root)
  root.mainloop()
```

```

Features of this Expense Tracker:

1. Add Expenses: Input date, amount, category, and description
2. View Expenses: See a list of recent expenses with all details
3. Delete Expenses: Double-click on an expense to delete it
4. Reports: Visualize expenses by category, daily, or monthly
5. Time Filters: Filter reports by different time periods (week, month, year, etc.)
6. SQLite Database: All data is stored in a local SQLite database
7. Data Visualization: Uses Matplotlib to create pie charts and bar graphs
```