Python Automation Goldmine

25 Automation Scripts That Solve Real Business Problems

> "Automation is good, so long as you know exactly where to put the machine." - Eliyahu Goldratt

---

Table of Contents

Chapter 1: Business Process Automation

Script #1: Invoice Generation and Management System

Business Impact: $15,000 annual savings in administrative costs **Time Savings:** 20 hours per week for accounting teams **Error Reduction:** 95% decrease in manual calculation errors **Implementation Time:** 4-6 hours setup

```
#!/usr/bin/env python3
"""
Automated Invoice Generation System
Generates professional PDF invoices from CSV data
"""

import pandas as pd
from reportlab.lib.pagesizes import letter, A4
from reportlab.pdfgen import canvas
from reportlab.lib.utils import ImageReader
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph, Spacer
from reportlab.lib import colors
from reportlab.lib.units import inch
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders
import os
from datetime import datetime, timedelta
import json

class InvoiceGenerator:
    def __init__(self, config_file='invoice_config.json'):
        """Initialize invoice generator with configuration"""
        with open(config_file, 'r') as f:
            self.config = json.load(f)

        self.company_info = self.config['company']
        self.email_config = self.config['email']
        self.invoice_settings = self.config['invoice_settings']

    def load_client_data(self, csv_file):
        """Load client and invoice data from CSV file"""
        try:
            df = pd.read_csv(csv_file)
            required_columns = ['client_name', 'client_email', 'service_description',
                              'quantity', 'rate', 'tax_rate']

            missing_columns = [col for col in required_columns if col not in df.columr
            if missing_columns:
                raise ValueError(f"Missing required columns: {missing_columns}")

            # Calculate totals
            df['subtotal'] = df['quantity'] * df['rate']
            df['tax_amount'] = df['subtotal'] * (df['tax_rate'] / 100)
            df['total'] = df['subtotal'] + df['tax_amount']

            return df

        except Exception as e:
            print(f"Error loading client data: {e}")
            return None

    def generate_invoice_number(self):
        """Generate unique invoice number"""
        current_date = datetime.now()
        return f"INV-{current_date.strftime('%Y%m%d')}-{current_date.strftime('%H%M%S'

    def create_pdf_invoice(self, client_data, invoice_number, output_dir='invoices/'):
        """Generate PDF invoice for client"""

        if not os.path.exists(output_dir):
            os.makedirs(output_dir)

        filename = f"{output_dir}invoice_{invoice_number}_{client_data['client_name'].

        # Create PDF document
        doc = SimpleDocTemplate(filename, pagesize=A4)
        story = []
        styles = getSampleStyleSheet()

        # Custom styles
        company_style = ParagraphStyle(
            'CompanyHeader',
            parent=styles['Heading1'],
            fontSize=24,
            textColor=colors.HexColor('#2c3e50'),
            spaceAfter=30,
            alignment=1  # Center alignment
        )

        # Company header
        company_name = Paragraph(self.company_info['name'], company_style)
        story.append(company_name)

        # Company details
        company_details = f"""
        {self.company_info['address']}<br/>
        {self.company_info['city']}, {self.company_info['state']} {self.company_info['
        Phone: {self.company_info['phone']}<br/>
        Email: {self.company_info['email']}
        """
        story.append(Paragraph(company_details, styles['Normal']))
        story.append(Spacer(1, 0.5*inch))

        # Invoice header
        invoice_date = datetime.now().strftime('%B %d, %Y')
        due_date = (datetime.now() + timedelta(days=self.invoice_settings['payment_ter

        invoice_header_data = [
            ['Invoice Number:', invoice_number],
            ['Invoice Date:', invoice_date],
            ['Due Date:', due_date],
            ['Payment Terms:', f"{self.invoice_settings['payment_terms']} days"]
        ]

        invoice_header_table = Table(invoice_header_data, colWidths=[2*inch, 2*inch])
        invoice_header_table.setStyle(TableStyle([
            ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
            ('FONTNAME', (0, 0), (0, -1), 'Helvetica-Bold'),
            ('FONTSIZE', (0, 0), (-1, -1), 10),
            ('BOTTOMPADDING', (0, 0), (-1, -1), 6),
        ]))

        story.append(invoice_header_table)
        story.append(Spacer(1, 0.3*inch))

        # Bill to section
        bill_to = f"""
        <b>Bill To:</b><br/>
        {client_data['client_name']}<br/>
        {client_data.get('client_address', 'Address not provided')}<br/>
        Email: {client_data['client_email']}
        """
        story.append(Paragraph(bill_to, styles['Normal']))
        story.append(Spacer(1, 0.5*inch))

        # Services table
        services_data = [
            ['Description', 'Quantity', 'Rate', 'Amount']
        ]

        services_data.append([
            client_data['service_description'],
            str(client_data['quantity']),
            f"${client_data['rate']:.2f}",
            f"${client_data['subtotal']:.2f}"
        ])

        services_table = Table(services_data, colWidths=[3*inch, 1*inch, 1*inch, 1*inc
        services_table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#3498db')),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
            ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
            ('FONTSIZE', (0, 0), (-1, -1), 10),
            ('GRID', (0, 0), (-1, -1), 1, colors.black),
            ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ]))

        story.append(services_table)
        story.append(Spacer(1, 0.3*inch))

        # Totals section
        totals_data = [
            ['Subtotal:', f"${client_data['subtotal']:.2f}"],
            [f'Tax ({client_data["tax_rate"]}%):', f"${client_data['tax_amount']:.2f}"
            ['Total:', f"${client_data['total']:.2f}"]
        ]

        totals_table = Table(totals_data, colWidths=[4*inch, 1.5*inch])
        totals_table.setStyle(TableStyle([
            ('ALIGN', (0, 0), (-1, -1), 'RIGHT'),
            ('FONTNAME', (0, -1), (-1, -1), 'Helvetica-Bold'),
            ('FONTSIZE', (0, 0), (-1, -1), 12),
            ('LINEBELOW', (0, -1), (-1, -1), 2, colors.black),
            ('BACKGROUND', (0, -1), (-1, -1), colors.HexColor('#ecf0f1')),
        ]))

        story.append(totals_table)
        story.append(Spacer(1, 0.5*inch))

        # Payment instructions
        payment_instructions = f"""
        <b>Payment Instructions:</b><br/>
        Please remit payment within {self.invoice_settings['payment_terms']} days.<br/
        {self.invoice_settings['payment_instructions']}<br/><br/>
        Thank you for your business!
        """
        story.append(Paragraph(payment_instructions, styles['Normal']))

        # Build PDF
        doc.build(story)
        return filename

    def send_invoice_email(self, client_email, client_name, invoice_file, invoice_numb
        """Send invoice via email"""

        try:
            # Create message
            msg = MIMEMultipart()
            msg['From'] = self.email_config['smtp_user']
            msg['To'] = client_email
            msg['Subject'] = f"Invoice {invoice_number} from {self.company_info['name'
            # Email body
            body = f"""
            Dear {client_name},

            Please find attached your invoice #{invoice_number}.

            Payment is due within {self.invoice_settings['payment_terms']} days of the

            If you have any questions, please don't hesitate to contact us.

            Best regards,
            {self.company_info['name']}
            {self.company_info['phone']}
            {self.company_info['email']}
            """

            msg.attach(MIMEText(body, 'plain'))

            # Attach PDF
            with open(invoice_file, "rb") as attachment:
                part = MIMEBase('application', 'octet-stream')
                part.set_payload(attachment.read())

            encoders.encode_base64(part)
            part.add_header(
                'Content-Disposition',
                f'attachment; filename= {os.path.basename(invoice_file)}',
            )
            msg.attach(part)

            # Send email
            server = smtplib.SMTP(self.email_config['smtp_server'], self.email_config[
            server.starttls()
            server.login(self.email_config['smtp_user'], self.email_config['smtp_passw
            text = msg.as_string()
            server.sendmail(self.email_config['smtp_user'], client_email, text)
            server.quit()

            return True

        except Exception as e:
            print(f"Error sending email: {e}")
            return False

    def process_bulk_invoices(self, csv_file, send_emails=False):
        """Process multiple invoices from CSV file"""

        client_data = self.load_client_data(csv_file)
        if client_data is None:
            return

        results = []
        for index, row in client_data.iterrows():
            invoice_number = self.generate_invoice_number()

            try:
                # Generate PDF
                pdf_file = self.create_pdf_invoice(row, invoice_number)

                result = {
                    'client_name': row['client_name'],
                    'invoice_number': invoice_number,
                    'total_amount': row['total'],
                    'pdf_file': pdf_file,
                    'status': 'Generated'
                }

                # Send email if requested
                if send_emails:
                    email_sent = self.send_invoice_email(
                        row['client_email'],
                        row['client_name'],
                        pdf_file,
                        invoice_number
                    )
                    result['email_sent'] = email_sent
                    result['status'] = 'Sent' if email_sent else 'Generated (Email Fai

                results.append(result)
                print(f"  Processed invoice for {row['client_name']}: {invoice_number}

            except Exception as e:
                print(f"  Error processing invoice for {row['client_name']}: {e}")
                results.append({
                    'client_name': row['client_name'],
                    'invoice_number': 'Failed',
                    'error': str(e),
                    'status': 'Error'
                })

        # Generate summary report
        self.generate_summary_report(results)
        return results

    def generate_summary_report(self, results):
        """Generate summary report of processed invoices"""

        total_invoices = len(results)
        successful = len([r for r in results if r['status'] != 'Error'])
        total_amount = sum([r.get('total_amount', 0) for r in results if r['status'] !

        print(f"\n  INVOICE PROCESSING SUMMARY")
        print(f"{'='*50}")
        print(f"Total invoices processed: {total_invoices}")
        print(f"Successful: {successful}")
        print(f"Failed: {total_invoices - successful}")
        print(f"Total invoice amount: ${total_amount:.2f}")
        print(f"{'='*50}")

# Configuration file example (invoice_config.json)
config_example = {
    "company": {
        "name": "Your Business Name",
        "address": "123 Business Street",
        "city": "Business City",
        "state": "ST",
        "zip": "12345",
        "phone": "(555) 123-4567",
        "email": "billing@yourbusiness.com"
    },
    "email": {
        "smtp_server": "smtp.gmail.com",
        "smtp_port": 587,
        "smtp_user": "your-email@gmail.com",
        "smtp_password": "your-app-password"
    },
    "invoice_settings": {
        "payment_terms": 30,
        "payment_instructions": "Please pay by check or bank transfer. Contact us for
    }
}

# Usage example
if __name__ == "__main__":
    # Create invoice generator
    generator = InvoiceGenerator()

    # Process invoices from CSV file
    results = generator.process_bulk_invoices('client_data.csv', send_emails=True)

    print("Invoice processing complete!")
```

Script #2: Automated Expense Tracking and Reporting

Business Impact: $8,000 annual savings in bookkeeping costs **Time Savings:** 15 hours per month for finance teams **Accuracy Improvement:** 90% reduction in categorization errors **Implementation Time:** 3-4 hours setup

```
 #!/usr/bin/env python3
"""
Automated Expense Tracking System
Processes receipts and categorizes expenses automatically
```

```
"""

import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import cv2
import pytesseract
from PIL import Image
import re
import os
import json
from pathlib import Path
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
import pickle

class ExpenseTracker:
    def __init__(self, config_file='expense_config.json'):
        """Initialize expense tracker with configuration"""
        with open(config_file, 'r') as f:
            self.config = json.load(f)

        self.categories = self.config['expense_categories']
        self.keywords = self.config['category_keywords']
        self.email_config = self.config['email']

        # Load or create ML model for categorization
        self.model_file = 'expense_model.pkl'
        self.load_or_train_model()

    def load_or_train_model(self):
        """Load existing model or train new one"""
        if os.path.exists(self.model_file):
            with open(self.model_file, 'rb') as f:
                self.classifier = pickle.load(f)
        else:
            self.train_categorization_model()

    def train_categorization_model(self):
        """Train ML model for expense categorization"""
        training_data = []
        labels = []

        # Create training data from keywords
        for category, keywords in self.keywords.items():
            for keyword in keywords:
                training_data.append(keyword)
                labels.append(category)

        # Create and train pipeline
        self.classifier = Pipeline([
            ('tfidf', TfidfVectorizer(stop_words='english', lowercase=True)),
            ('classifier', MultinomialNB())
        ])

        self.classifier.fit(training_data, labels)

        # Save model
        with open(self.model_file, 'wb') as f:
            pickle.dump(self.classifier, f)

    def extract_text_from_receipt(self, image_path):
        """Extract text from receipt image using OCR"""
        try:
            # Load and preprocess image
            image = cv2.imread(image_path)
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

            # Apply preprocessing
            _, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTS

            # Extract text using OCR
            text = pytesseract.image_to_string(thresh, config='--psm 6')
            return text

        except Exception as e:
            print(f"Error extracting text from {image_path}: {e}")
            return ""

    def parse_receipt_data(self, receipt_text, filename):
        """Parse receipt text to extract expense information"""

        expense_data = {
            'filename': filename,
            'date': None,
            'amount': None,
            'vendor': None,
            'description': receipt_text[:200],  # First 200 chars
            'category': 'Uncategorized',
            'raw_text': receipt_text
        }

        # Extract date
        date_patterns = [
            r'\d{1,2}/\d{1,2}/\d{4}',
            r'\d{1,2}-\d{1,2}-\d{4}',
            r'\d{4}-\d{1,2}-\d{1,2}',
            r'[A-Za-z]{3}\s+\d{1,2},?\s+\d{4}'
        ]
        for pattern in date_patterns:
            date_match = re.search(pattern, receipt_text)
            if date_match:
                try:
                    date_str = date_match.group()
                    expense_data['date'] = pd.to_datetime(date_str).strftime('%Y-%m-%c
                    break
                except:
                    continue

        # Extract amount (look for dollar amounts)
        amount_patterns = [
            r'\$\s*\d+\.\d{2}',
            r'\d+\.\d{2}\s*\$',
            r'Total\s*:?\s*\$?\s*\d+\.\d{2}',
            r'Amount\s*:?\s*\$?\s*\d+\.\d{2}'
        ]

        amounts = []
        for pattern in amount_patterns:
            amount_matches = re.findall(pattern, receipt_text, re.IGNORECASE)
            for match in amount_matches:
                # Extract numeric value
                numeric_value = re.findall(r'\d+\.\d{2}', match)
                if numeric_value:
                    amounts.append(float(numeric_value[0]))

        if amounts:
            # Take the largest amount (likely the total)
            expense_data['amount'] = max(amounts)

        # Extract vendor name (usually near the top)
        lines = receipt_text.split('\n')[:10]  # Check first 10 lines
        for line in lines:
            line = line.strip()
            if len(line) > 3 and len(line) < 50 and not re.search(r'\d+\.\d{2}', line)
                if not any(word in line.lower() for word in ['receipt', 'invoice', 'da
                    expense_data['vendor'] = line
                    break

        # Categorize expense using ML model
        if hasattr(self, 'classifier'):
            try:
                category = self.classifier.predict([receipt_text])[0]
                expense_data['category'] = category
            except:
                expense_data['category'] = self.categorize_by_keywords(receipt_text)
        else:
            expense_data['category'] = self.categorize_by_keywords(receipt_text)

        return expense_data

    def categorize_by_keywords(self, text):
        """Categorize expense based on keywords"""
        text_lower = text.lower()

        category_scores = {}
        for category, keywords in self.keywords.items():
            score = sum(1 for keyword in keywords if keyword.lower() in text_lower)
            if score > 0:
                category_scores[category] = score

        if category_scores:
            return max(category_scores, key=category_scores.get)

        return 'Uncategorized'

    def process_receipt_folder(self, folder_path):
        """Process all receipt images in a folder"""
        expenses = []

        image_extensions = ['.jpg', '.jpeg', '.png', '.bmp', '.tiff']
        folder = Path(folder_path)

        for image_file in folder.iterdir():
            if image_file.suffix.lower() in image_extensions:
                print(f"Processing {image_file.name}...")

                # Extract text from receipt
                receipt_text = self.extract_text_from_receipt(str(image_file))

                if receipt_text:
                    # Parse expense data
                    expense_data = self.parse_receipt_data(receipt_text, image_file.na
                    expenses.append(expense_data)
                    print(f"  Processed: ${expense_data.get('amount', 'N/A')} - {exper
                else:
                    print(f"  Could not extract text from {image_file.name}")

        return expenses

    def process_csv_expenses(self, csv_file):
        """Process expenses from CSV file"""
        try:
            df = pd.read_csv(csv_file)

            # Ensure required columns exist
            required_columns = ['date', 'amount', 'description']
            missing_columns = [col for col in required_columns if col not in df.columr

            if missing_columns:
                print(f"Warning: Missing columns {missing_columns}. Adding with defaul
                for col in missing_columns:
                    df[col] = None

            # Add category if not present
            if 'category' not in df.columns:
                df['category'] = df['description'].apply(
                    lambda x: self.categorize_by_keywords(str(x)) if pd.notna(x) else
                )

            return df.to_dict('records')

        except Exception as e:
            print(f"Error processing CSV file: {e}")
            return []

    def generate_expense_report(self, expenses, output_file='expense_report.xlsx'):
        """Generate comprehensive expense report"""

        if not expenses:
            print("No expenses to report")
            return None

        # Convert to DataFrame
        df = pd.DataFrame(expenses)

        # Clean and convert data types
        df['date'] = pd.to_datetime(df['date'], errors='coerce')
        df['amount'] = pd.to_numeric(df['amount'], errors='coerce')

        # Remove rows with invalid data
        df = df.dropna(subset=['amount'])

        # Add additional calculated fields
        df['month'] = df['date'].dt.to_period('M')
        df['year'] = df['date'].dt.year
        df['quarter'] = df['date'].dt.quarter

        # Create Excel writer object
        with pd.ExcelWriter(output_file, engine='xlsxwriter') as writer:

            # Raw data sheet
            df.to_excel(writer, sheet_name='Raw Data', index=False)

            # Summary by category
            category_summary = df.groupby('category')['amount'].agg(['sum', 'count', '
            category_summary.columns = ['Total Amount', 'Count', 'Average Amount']
            category_summary.to_excel(writer, sheet_name='Category Summary')

            # Monthly summary
            monthly_summary = df.groupby('month')['amount'].sum().round(2)
            monthly_summary.to_excel(writer, sheet_name='Monthly Summary')

            # Quarterly summary
            quarterly_summary = df.groupby(['year', 'quarter'])['amount'].sum().round(
            quarterly_summary.to_excel(writer, sheet_name='Quarterly Summary')

            # Top expenses
            top_expenses = df.nlargest(20, 'amount')[['date', 'vendor', 'amount', 'cat
            top_expenses.to_excel(writer, sheet_name='Top Expenses', index=False)
        print(f"  Expense report generated: {output_file}")

        # Generate visualizations
        self.create_expense_charts(df)

        return output_file

    def create_expense_charts(self, df):
        """Create expense visualization charts"""

        plt.style.use('seaborn-v0_8')
        fig, axes = plt.subplots(2, 2, figsize=(15, 12))
        fig.suptitle('Expense Analysis Dashboard', fontsize=16, fontweight='bold')

        # 1. Expenses by Category (Pie Chart)
        category_totals = df.groupby('category')['amount'].sum()
        axes[0, 0].pie(category_totals.values, labels=category_totals.index, autopct='
        axes[0, 0].set_title('Expenses by Category')

        # 2. Monthly Trend (Line Chart)
        monthly_totals = df.groupby(df['date'].dt.to_period('M'))['amount'].sum()
        axes[0, 1].plot(monthly_totals.index.astype(str), monthly_totals.values, marke
        axes[0, 1].set_title('Monthly Expense Trend')
        axes[0, 1].tick_params(axis='x', rotation=45)

        # 3. Category Breakdown (Bar Chart)
        category_totals.plot(kind='bar', ax=axes[1, 0])
        axes[1, 0].set_title('Total Expenses by Category')
        axes[1, 0].tick_params(axis='x', rotation=45)

        # 4. Expense Distribution (Histogram)
        axes[1, 1].hist(df['amount'], bins=20, edgecolor='black', alpha=0.7)
        axes[1, 1].set_title('Expense Amount Distribution')
        axes[1, 1].set_xlabel('Amount ($)')
        axes[1, 1].set_ylabel('Frequency')

        plt.tight_layout()
        plt.savefig('expense_analysis_charts.png', dpi=300, bbox_inches='tight')
        plt.show()

        print("  Charts saved as 'expense_analysis_charts.png'")

    def send_expense_report(self, report_file, recipient_email):
        """Email expense report to specified recipient"""

        try:
            msg = MIMEMultipart()
            msg['From'] = self.email_config['smtp_user']
            msg['To'] = recipient_email
            msg['Subject'] = f"Expense Report - {datetime.now().strftime('%B %Y')}"

            # Email body
            body = """
            Dear Team,
            Please find attached the latest expense report.

            The report includes:
            - Raw expense data
            - Category summaries
            - Monthly and quarterly trends
            - Top expense items

            Best regards,
            Automated Expense Tracker
            """

            msg.attach(MIMEText(body, 'plain'))

            # Attach Excel report
            with open(report_file, "rb") as attachment:
                part = MIMEBase('application', 'octet-stream')
                part.set_payload(attachment.read())

            encoders.encode_base64(part)
            part.add_header(
                'Content-Disposition',
                f'attachment; filename= {os.path.basename(report_file)}',
            )
            msg.attach(part)

            # Attach charts if they exist
            chart_file = 'expense_analysis_charts.png'
            if os.path.exists(chart_file):
                with open(chart_file, "rb") as attachment:
                    part = MIMEBase('application', 'octet-stream')
                    part.set_payload(attachment.read())

                encoders.encode_base64(part)
                part.add_header(
                    'Content-Disposition',
                    f'attachment; filename= {chart_file}',
                )
                msg.attach(part)

            # Send email
            server = smtplib.SMTP(self.email_config['smtp_server'], self.email_config[
            server.starttls()
            server.login(self.email_config['smtp_user'], self.email_config['smtp_passw
            text = msg.as_string()
            server.sendmail(self.email_config['smtp_user'], recipient_email, text)
            server.quit()

            print(f"  Report sent to {recipient_email}")
            return True

        except Exception as e:
            print(f"  Error sending email: {e}")
            return False

# Configuration file example (expense_config.json)
expense_config_example = {
    "expense_categories": [
        "Office Supplies",
        "Travel",
        "Meals & Entertainment",
        "Professional Services",
        "Marketing",
        "Software & Subscriptions",
        "Equipment",
        "Utilities",
        "Insurance",
        "Uncategorized"
    ],
    "category_keywords": {
        "Office Supplies": ["office", "supplies", "paper", "pen", "stapler", "printer"
        "Travel": ["hotel", "flight", "uber", "taxi", "gas", "mileage", "parking"],
        "Meals & Entertainment": ["restaurant", "lunch", "dinner", "coffee", "caterin
        "Professional Services": ["consultant", "lawyer", "accountant", "contractor"],
        "Marketing": ["advertising", "marketing", "promotion", "social media", "websit
        "Software & Subscriptions": ["software", "subscription", "saas", "license", "c
        "Equipment": ["computer", "monitor", "phone", "equipment", "hardware"],
        "Utilities": ["electricity", "water", "internet", "phone", "utility"],
        "Insurance": ["insurance", "premium", "coverage"]
    },
    "email": {
        "smtp_server": "smtp.gmail.com",
        "smtp_port": 587,
        "smtp_user": "your-email@gmail.com",
        "smtp_password": "your-app-password"
    }
}

# Usage example
if __name__ == "__main__":
    # Create expense tracker
    tracker = ExpenseTracker()

    # Process receipts from folder
    receipt_expenses = tracker.process_receipt_folder('receipts/')

    # Process CSV expenses
    csv_expenses = tracker.process_csv_expenses('manual_expenses.csv')

    # Combine all expenses
    all_expenses = receipt_expenses + csv_expenses

    # Generate comprehensive report
    report_file = tracker.generate_expense_report(all_expenses)

    # Send report via email
    if report_file:
        tracker.send_expense_report(report_file, 'finance@company.com')

    print("Expense processing complete!")
```

This automation script provides comprehensive expense tracking with OCR receipt processing, automatic categorization, and detailed reporting. It saves significant time and improves accuracy in expense management processes.

Continue with 23 more automation scripts covering web scraping, file management, marketing automation, and advanced business intelligence systems.