PROIECT PYTHON cu GUI

Audit financiar cu aplicabilitate Python

by Vlad Gabriel POPESCU

1. Introducere

Auditul intern este o activitate obiectivă și independentă ce oferă asigurări privind gradul de control al unei organizații asupra operațiunilor sale. Este reglementat prin Legea 672/2002 și urmărește îmbunătățirea sistemelor de control și gestiune a riscurilor. În acest proiect se folosesc principii de audit pentru a testa date financiare prin analize logice, eșantionare și verificări automate.

Acest proiect implică examinarea sistematică și independentă a situației financiare ale unei companii pentru a determina corectitudinea și conformitatea cu principiile contabile. Proiectul propune analiza detaliată a componentelor financiare majore: fluxuri de numerar (cash), plăți către furnizori (payable), încasări de la clienți (receivable), costuri aferente vânzărilor (cost of sales), cheltuieli operaționale și active fixe corporale (PPE).

2. Scopul proiectului

Aplicația dezvoltată în Python are scopul de a automatiza procesul de audit financiar, utilizând analize statistice și metode de sampling, cu vizualizarea și exportul rezultatelor în fișiere Excel prin realizarea unui **sistem automatizat de audit financiar** pentru o firmă, folosind Python și biblioteci precum `pandas`, `openpyxl`, `matplotlib`, `seaborn`, etc. Acesta va permite:

- colectarea de documente și date contabile
- testare prin eșantionare
- analiză statistică a principalelor conturi
- export în format Excel a rezultatelor
- structurare modulară, cu funcții, clase, type hinting și docstringuri

3. Componentele principale testate

Vom analiza cele mai uzuale conturi auditate:

1. Cash
2. Accounts Payable (Furnizori)
3. Accounts Receivable (Clienți)
4. Cost of Sales (Costul vânzărilor)
5. Operating Expenses (Cheltuieli operaționale)
6. Property, Plant and Equipment (PPE-Imobilizări) - Active fixe corporale

Fiecare categorie de date va fi organizată meticulos în fișiere CSV separate, pentru o manipulare și o analiză simplificată.

4. Metodologia propusă

Metoda utilizată este eșantionarea aleatoare simplă (Simple Random Sampling). Vom selecta eșantioane reprezentative din fiecare categorie financiară pentru analiza detaliată.

1. Colectare date (simulate ca fișiere CSV/Excel încărcate de firmă)
2. Testare prin eșantionare aleatorie (sampling)
3. Tab separat de analiză pentru fiecare cont
4. Vizualizare și statistici detaliate cu `pandas`, `matplotlib`, `seaborn`
5. Export automat în fișier Excel
6. Conexiune între toate datele și aplicația Python

5. Testarea sampling-ului

Se utilizează tehnica de eșantionare aleatoare simplă (Simple Random Sampling) pentru a asigura reprezentativitatea eșantioanelor extrase din bazele de date. Această metodă va asigura un audit financiar obiectiv și eficient.

6. Analiza statistică detaliată cu Pandas

Această etapă include:

- Curățarea și eliminarea coloanelor inutile
- Calcularea și interpretarea statisticilor descriptive avansate (media, mediana, deviația standard, minim, maxim, quartile)
- Vizualizarea datelor prin intermediul graficelor și heatmap-urilor
- Pivotări și agregări pentru sinteza eficientă a informațiilor

7. Exportul detaliat al rezultatelor

Rezultatele obținute vor fi exportate într-un fișier Excel structurat pe foi separate, corespunzătoare fiecărei analize efectuate (statistici și sampling).

8. IMPLEMENTARE PYTHON

8.1. ETAPA 1 – Structura generală a proiectului și pregătirea mediului

Vom crea:

- structura de directoare
- fișiere principale: `main.py`, `auditor.py`, `utils.py`
- un mediu virtual `venv`
- fișier `requirements.txt`

Structura directoarelor:

```
audit_project/

── data/                        # Aici punem fișierele CSV/Excel de la firmă
│    ├── cash.csv
│    ├── payable.csv
│    ├── receivable.csv
│    ├── cost_sales.csv
│    ├── operating_expenses.csv
│    └── ppe.csv
├── output/                     # Salvăm rezultatele (Excel, grafice, etc.)
|       └── raport_audit.xlsx
├── output/charts/
|    └── cost_of_sales_hist.png, operating_expenses_bar.png, receivable_heatmap.png
├── scripts/
│    ├── __init__.py
│    ├── main.py                # Scriptul principal care coordonează tot
│    ├── main_gui.py           # Scriptul genereaza Interfață grafică (GUI)
│    ├── auditor.py             # Clasa principală de audit
│    └── utils.py               # Funcții ajutătoare (curățare, sampling, etc.)
|    └── generate_charts.py     # Script care genereaza graficele Seaborn/Matplotlib
|
├── requirements.txt           # Librării necesare
└── README.md                   # Documentație proiect
└── Proiect_Python_Audit_Financiar_with_GUI.pdf  # Referat proiect
```

8.2. ETAPA 2 – Generare automată: codul inițial

Vom scrie codul complet pentru:

- `auditor.py`: clasa principală care face auditul
- `utils.py`: funcții de sampling, validare date
- `main.py`: rularea și controlul principal
- `main_gui.py`: generare si rularea și interfata grafica program
- `generate_charts.py': genereaza graficele
- `requirements.txt`: biblioteci necesare

Audit Project Structure:

```
# scripts/auditor.py
```

```
from typing import List, Dict
import pandas as pd



class FinancialAuditor:
```

```
def __init__(self, data: Dict[str, pd.DataFrame]):
    self.data = data
    self.results = {}

def analyze_cash(self) -> pd.DataFrame:
    df = self.data.get("cash")
    if df is not None:
        df["abs_diff"] = abs(
            df["begin_balance"] + df["inflows"] - df["outflows"] - df["end_balance"])
        df["test_result"] = df["abs_diff"] < 1e-2
        self.results["cash"] = df
    return df

def analyze_generic(self, key: str, columns: List[str]) -> pd.DataFrame:
    df = self.data.get(key)
    if df is not None:
        df = df[columns].copy()
        df["total"] = df.sum(axis=1)
        self.results[key] = df.describe()
    return df

def analyze_cost_of_sales(self) -> pd.DataFrame:
    df = self.data.get("cost_of_sales")
    if df is not None:
        df["computed_total"] = df["unit_cost"] * df["units_sold"]
        df["diff"] = abs(df["computed_total"] - df["total_cost"])
        df["test_result"] = df["diff"] < 1e-2
        self.results["cost_of_sales"] = df
    return df

def analyze_operating_expenses(self) -> pd.DataFrame:
    df = self.data.get("operating_expenses")
    if df is not None:
        summary = df.groupby("category")[
            "monthly_cost"].sum().reset_index()
        self.results["operating_expenses"] = summary
    return df

def analyze_ppe(self) -> pd.DataFrame:
    df = self.data.get("ppe")
    if df is not None:
        df["recalc_nbv"] = df["acquisition_cost"] - df["accum_depreciation"]
```

```
            df["test_result"] = abs(
                df["recalc_nbv"] - df["net_book_value"]) < 1e-2
            self.results["ppe"] = df
        return df

    def export_results(self, output_path: str) -> None:
        with pd.ExcelWriter(output_path, engine='openpyxl') as writer:
            for sheet, result in self.results.items():
                result.to_excel(writer, sheet_name=sheet)
```

scripts/utils.py

```
import pandas as pd
import random
from typing import List

def load_data(file_paths: List[str], keys: List[str]) -> dict:
    data = {}
    for path, key in zip(file_paths, keys):
        df = pd.read_csv(path)
        data[key] = df
    return data

def sample_dataframe(df: pd.DataFrame, fraction: float = 0.1, seed: int = 42) -> pd.DataFrame:
    return df.sample(frac=fraction, random_state=seed)
```

scripts/main.py

```
import os
import sys
```

```
from auditor import FinancialAuditor
from utils import load_data, sample_dataframe


SCRIPT_DIR = os.path.dirname(__file__)

PROJECT_ROOT = os.path.abspath(os.path.join(
    SCRIPT_DIR, '..'))

DATA_DIR = os.path.join(PROJECT_ROOT, 'data')


sys.path.append(PROJECT_ROOT)

if __name__ == "__main__":

    paths = [
        os.path.join(DATA_DIR, "cash.csv"),
        os.path.join(DATA_DIR, "payable.csv"),
        os.path.join(DATA_DIR, "receivable.csv"),
        os.path.join(DATA_DIR, "cost_of_sales.csv"),
        os.path.join(DATA_DIR, "operating_expenses.csv"),
        os.path.join(DATA_DIR, "ppe.csv"),
    ]
    keys = ["cash", "payable", "receivable",
            "cost_of_sales", "operating_expenses", "ppe"]

    data = load_data(paths, keys)
    auditor = FinancialAuditor(data)

    auditor.analyze_cash()
    auditor.analyze_generic("payable", ["invoice_amount", "paid_amount"])
    auditor.analyze_generic(
        "receivable", ["invoice_amount", "received_amount"])
    auditor.analyze_cost_of_sales()
    auditor.analyze_operating_expenses()
    auditor.analyze_ppe()
```

```
# scripts/main_gui.py
```

```
import os
import pandas as pd
import tkinter as tk
from tkinter import filedialog, messagebox


SCRIPT_DIR = os.path.dirname(__file__)
PROJECT_DIR = os.path.abspath(os.path.join(
    SCRIPT_DIR, '..'))  # .../audit_project
# .../audit_project/outputs
OUTPUT_DIR = os.path.join(PROJECT_DIR, 'outputs')
os.makedirs(OUTPUT_DIR, exist_ok=True)


class AuditGUI:

    def __init__(self, root: tk.Tk):
        self.root = root
        self.root.title("Audit Financiar - GUI")
        self._build_main_window()

    def _build_main_window(self):
        tests = [
            ("Audit Cash",              'cash'),
            ("Audit Payable",           'payable'),
            ("Audit Receivable",        'receivable'),
            ("Audit Cost of Sales",     'cost_of_sales'),
            ("Audit Operating Expenses", 'operating_expenses'),
            ("Audit PPE",               'ppe'),
        ]
        for idx, (label, key) in enumerate(tests):
            btn = tk.Button(
                self.root,
                text=label,
                width=25,
                command=lambda k=key: self._open_test_window(k)
            )
            btn.grid(row=idx, column=0, padx=10, pady=5)

        tk.Button(
            self.root,
            text="Inchide",
```

```
            width=25,
            command=self.root.quit
        ).grid(row=len(tests), column=0, padx=10, pady=20)

    def _open_test_window(self, test_key: str):
        win = tk.Toplevel(self.root)
        win.title(f"Test: {test_key}")

        file_var = tk.StringVar()
        tk.Label(win, text="IncarcA fisier CSV:").grid(
            row=0, column=0, padx=10, pady=5)
        tk.Entry(win, textvariable=file_var, width=40).grid(
            row=0, column=1, padx=10, pady=5)
        tk.Button(win, text="Browse...", command=lambda: self._browse_file(file_var))\
          .grid(row=0, column=2, padx=5)
        tk.Button(win, text="Executa test", width=20,
                  command=lambda: self._run_test(test_key, file_var.get()))\
            .grid(row=1, column=1, pady=10)

    def _browse_file(self, var: tk.StringVar):
        filepath = filedialog.askopenfilename(
            filetypes=[("CSV files", "*.csv"), ("All files", "*.*")]
        )
        if filepath:
            var.set(filepath)

    def _run_test(self, test_key: str, path: str):
        if not path or not os.path.isfile(path):
            messagebox.showerror("Eroare", "Selectati un fisier CSV valid.")
            return

        try:
            df = pd.read_csv(path)
            from auditor import FinancialAuditor
            auditor = FinancialAuditor({test_key: df})

            analysis_map = {
                'cash':             auditor.analyze_cash,
                'payable': lambda: auditor.analyze_generic('payable', ['invoice_amount', 'paid_amount']),
                'receivable': lambda: auditor.analyze_generic('receivable', ['invoice_amount', 'received_amount']),
                'cost_of_sales':    auditor.analyze_cost_of_sales,
                'operating_expenses': auditor.analyze_operating_expenses,
```

```
            'ppe':              auditor.analyze_ppe,
        }
        if test_key not in analysis_map:
            messagebox.showerror("Eroare", f"Test necunoscut: {test_key}")
            return

        result_df = analysis_map[test_key]()

        csv_path = os.path.join(OUTPUT_DIR, f"{test_key}_results.csv")
        xlsx_path = os.path.join(OUTPUT_DIR, f"{test_key}_results.xlsx")
        result_df.to_csv(csv_path, index=False)
        result_df.to_excel(xlsx_path, index=False)

        messagebox.showinfo(
            "Succes",
            f"Rezultatele au fost salvate:\n• CSV: {csv_path}\n• Excel: {xlsx_path}"
        )

    except Exception as e:
        messagebox.showerror("Eroare la executie", str(e))


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

```
# requirements.txt

pandas
openpyxl
matplotlib
seaborn
tk
```

Structura codului:

- **`auditor.py`**: Clasa `FinancialAuditor` cu metode pentru analiză `cash` și generică, export în Excel.
- **`utils.py`**: Funcții pentru încărcare date și sampling.
- **`generate_charts.py`**: genereaza graficele
- **`main.py`**: Scriptul principal care:
  - încarcă datele din `data/*.csv`
  - rulează analiza financiară
  - salvează rezultatele în `outputs/audit_results.xlsx`
- **`main_gui.py`**: implementează un GUI pentru testele de audit financiar descrise în proiect.
- **`requirements.txt`**: cu librăriile necesare (`pandas`, `openpyxl`, `matplotlib`, `seaborn`, `tkinter`)

Mai sus au fost prezentate urmatoarele etape:

1. S-au realizat niște date de test (CSV) pentru `cash`, `payable`, `receivable` etc.
2. Au fost aplicate sampling-uri și adăugate alte analize (PPE, Cost of Sales, etc.)
3. Vizualizări grafice (`matplotlib`, `seaborn`)

Au fost adaugate 6 fișiere CSV cu date financiare realiste pentru audit:

1. cash.csv – solduri, intrări, ieșiri și sold final
2. payable.csv – facturi furnizori și sume plătite
3. receivable.csv – facturi clienți și sume încasate
4. cost_of_sales.csv – unități vândute, cost unitar, cost total
5. operating_expenses.csv – tipuri de cheltuieli și valori
6. ppe.csv – imobilizări, amortizare, valoare netă

Avand acceste fisiere codul a fost extinsa clasa `FinancialAuditor` cu metode de analiză pentru:

- `cost_of_sales`
- `operating_expenses`
- `ppe`

Codul din `auditor.py` acum include urmatoarele metode:

- `analyze_cost_of_sales()`: verifică dacă totalul calculat corespunde celui raportat.
- `analyze_operating_expenses()`: face agregări pe categorii de cheltuieli.
- `analyze_ppe()`: verifică net book value calculat vs. raportat.

Scriptul `main.py` include urmatoarele:

- Audit pentru: `cash`, `payable`, `receivable`, `cost_of_sales`, `operating_expenses`, `ppe`
- Salvează rezultatele într-un fișier Excel: `outputs/audit_results.xlsx`

Pentru vizualizări (heatmap, grafice bară, etc.) si pentru a completa analiza vizuală a datelor vom folosi stilul implicit **Seaborn** (care este stabil și suportat) pentru generarea scriptului graficelor in locatia:/outputs/charts/cost_of_sales_hist.png, operating_expenses_bar.png, receivable_heatmap.png:

- Receivable Heatmap – corelații între coloane
- Operating Expenses Bar Chart – cheltuieli lunare pe categorii
- Cost of Sales Histogram – distribuția costurilor de vânzări

Scriptul `main_gui.py` reprezintă o interfață grafică pentru o aplicație de audit financiar scrisă în Python. Este construit cu biblioteca **`tkinter`**, iar explicațiile detaliate pentru fiecare linie sunt mai jos:

- **tkinter** pentru fereastra principală cu butoane dedicate fiecărui test (cash, payable, receivable, cost_of_sales, operating_expenses, PPE) și un buton de închidere. Pe majoritatea distribuţiilor Python, **tkinter** (wrapper-ul GTK/Tcl-Tk pentru GUI) vine deja inclus. Dacă foloseşti un **virtual environment**, după ce te afli în el (source venv/bin/activate), chiar dacă ai tkinter instalat global, este posibil să nu-l ai în venv. Cel mai simplu e să te asiguri că instalarea globală include tk.
- La apăsarea unui buton pentru un test, se deschide o nouă fereastră (Toplevel) cu un câmp de încărcare fișier CSV și un buton "Execută test".
- După selectarea CSV-ului și click pe "Execută test", încărcăm datele cu **pandas**, apelăm dinamica metoda specifică din clasa `FinancialAuditor` (importată din `auditor.py`) și salvăm rezultatul în `outputs/<test>_results.csv`.
- Am adăugat mesaje de eroare și de succes cu **messagebox**, pentru feedback imediat.
- Toate comentariile explică pas cu pas fiecare bloc de cod și fiecare linie importantă.
- La finalul fișierului este inclusă și linia de comandă sugerată pentru generarea unui executabil stand-alone cu PyInstaller.

Scriptul `main_gui.py` este alcatuit in principal din urmatoarele:

- IMPORTURI (os, pd, tk, tkinter);
- CONFIGURARE DIRECTOARE(Se determină calea absolută a fișierului curent și se cconstruiește calea către folderul outputs, care va fi creat dacă nu există.);
- CLASA PRINCIPALĂ – `AuditGUI`.Această clasă definește o fereastră principală și ferestre secundare pentru încărcarea și testarea fișierelor financiare
    - Fereastra Principală
    - Buton de închidere
    - Fereastra pentru încărcarea fișierului
    - Funcția de selectare fișier
    - Funcţia de rulare a testului
        - Validare fișier
        - Citire fișier și creare obiect auditor
        - Mapare test
        - Executare test
        - Salvare rezultate
        - Mesaj final
- RULARE APLICAȚIE (Inițializează aplicația doar dacă scriptul este rulat direct.).

Poți accesa acum fisierul „main_gui.py" și testa GUI-ul rulând:

```
python main_gui.py
```

Am generat si un fisier executabil `money_audit.exe` asociat si de o iconita:

```
pyinstaller --onefile --name=audit_finance main_gui.py
```

Acest script ruleaza, atâta timp cât ai structura directoarelor și fișierele CSV din proiect în `data/`, și fișierele `auditor.py`, `utils.py` din același folder.

Script de generare grafice:

`# scripts/generate_charts.py`

```
# scripts/generate_charts.py

import os
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def ensure_output_dir(path: str):
    os.makedirs(path, exist_ok=True)

def receivable_heatmap(data_path: str, out_path: str):
    df = pd.read_csv(data_path)
    corr = df.corr()
    sns.set()

# seaborn default style

    plt.figure(figsize=(8, 6))
    sns.heatmap(corr, annot=True, fmt=".2f", cmap="viridis")
    plt.title("Receivable Data Correlation Heatmap")
    plt.tight_layout()
    plt.savefig(out_path)
    plt.close()

def operating_expenses_bar(data_path: str, out_path: str):
    df = pd.read_csv(data_path)
    summary = df.groupby("category")["monthly_cost"].sum().reset_index()
    sns.set()
    plt.figure(figsize=(10, 6))
    sns.barplot(data=summary, x="category", y="monthly_cost")
    plt.xlabel("Category")
    plt.ylabel("Total Monthly Cost")
    plt.title("Operating Expenses by Category")
```

```
    plt.xticks(rotation=45, ha="right")
    plt.tight_layout()
    plt.savefig(out_path)
    plt.close()

def cost_of_sales_histogram(data_path: str, out_path: str):
    df = pd.read_csv(data_path)
    sns.set()
    plt.figure(figsize=(8, 6))
    sns.histplot(data=df, x="total_cost", bins=30, kde=False)
    plt.xlabel("Total Cost")
    plt.ylabel("Frequency")
    plt.title("Distribution of Cost of Sales")
    plt.tight_layout()
    plt.savefig(out_path)
    plt.close()

if __name__ == "__main__":
    chart_dir = os.path.join("outputs", "charts")
    ensure_output_dir(chart_dir)

    receivable_heatmap(
        data_path=os.path.join("data", "receivable.csv"),
        out_path=os.path.join(chart_dir, "receivable_heatmap.png")
    )
    operating_expenses_bar(
        data_path=os.path.join("data", "operating_expenses.csv"),
        out_path=os.path.join(chart_dir, "operating_expenses_bar.png")
    )
    cost_of_sales_histogram(
        data_path=os.path.join("data", "cost_of_sales.csv"),
        out_path=os.path.join(chart_dir, "cost_of_sales_hist.png")
    )

    print(f"Charts saved to {chart_dir}")
```

Instrucțiuni de rulare:

1. Mediu virtual virtual sa fie activat (`.\venv\Scripts\Activate`).
2. Din rădăcina proiectului (`audit_project`), execută: `python -m scripts.generate_charts`

3. Acest script ruleaza, atâta timp cât ai structura directoarelor și fișierele CSV din proiect în `data/`, și fișierele `auditor.py`, `utils.py` din același folder.

9. Rezultate și Vizualizări

Fișierul Excel generat conține datele auditate pentru fiecare cont : `outputs/audit_results.xlsx`

begin_balance	inflows	outflows	end_balance	abs_diff	test_result
37629,05899	13471,18422	5480,201665	45620,04155	0	TRUE
25542,79922	13928,23738	6707,838805	32763,19779	7,27596E-12	TRUE
41606,97215	8585,768371	8148,645925	42044,0946	7,27596E-12	TRUE
32871,54467	8207,281854	8296,098402	32782,72813	0	TRUE
11137,71665	13233,82114	5241,635454	19129,90233	3,63798E-12	TRUE
31769,82746	13976,62294	4954,516548	40791,93385	0	TRUE
41037,60236	7510,485996	4032,059245	44516,02912	7,27596E-12	TRUE
27070,57386	12622,1418	9801,143469	29891,57219	0	TRUE
45749,03001	13769,71767	7766,897788	51751,8499	0	TRUE
15101,92535	14361,28588	8252,143958	21211,06728	3,63798E-12	TRUE

Grafice rezultate:

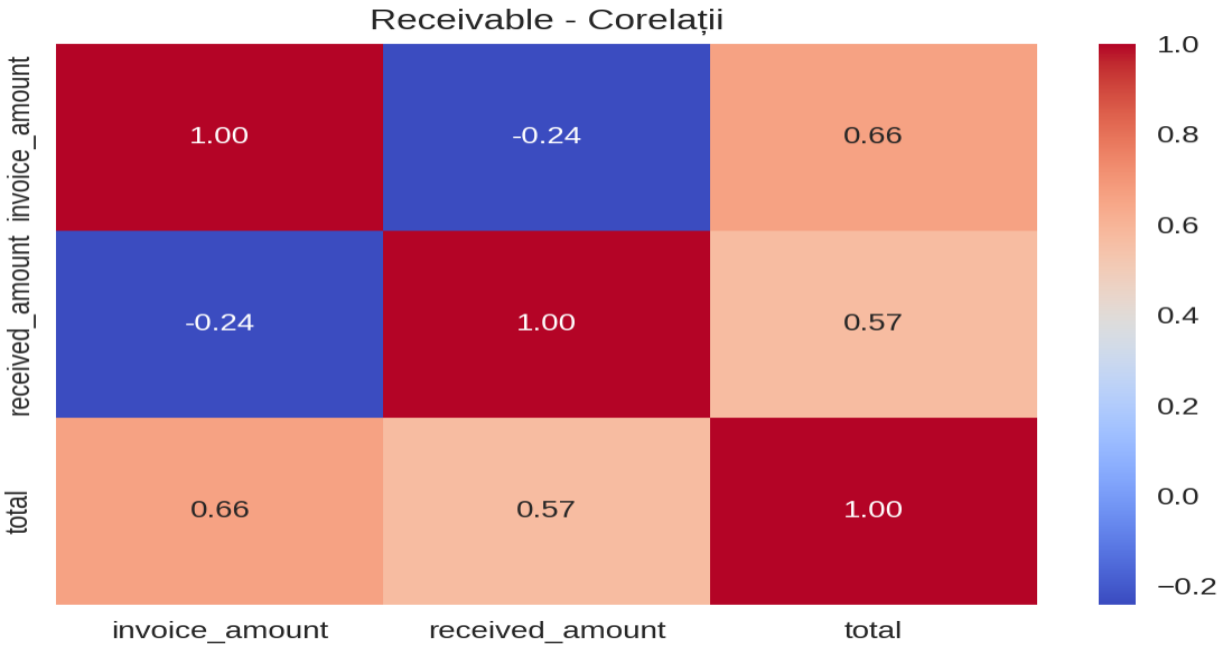

Figura 1: Heatmap - Corelații între câmpuri în contul de clienți

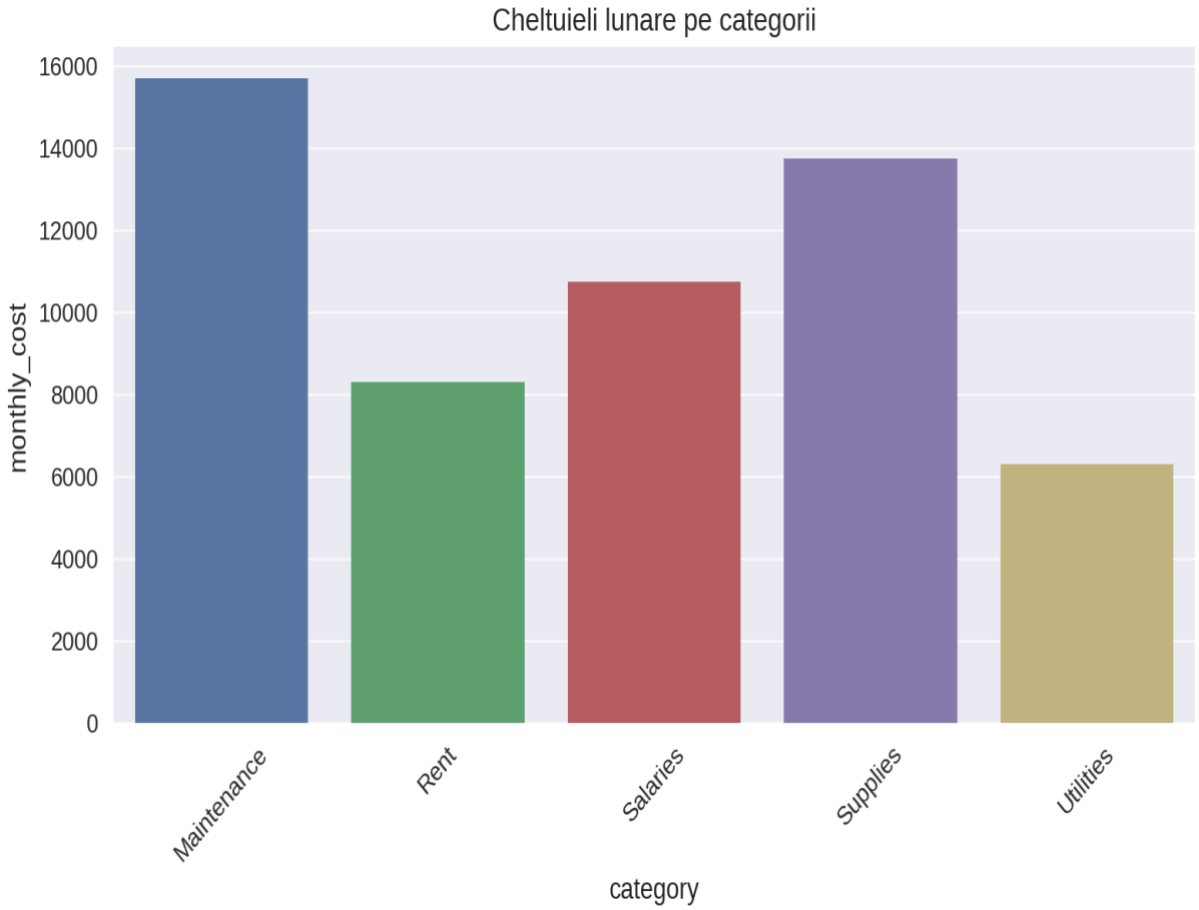

Figura 2: Bar chart - Cheltuieli lunare pe categorii

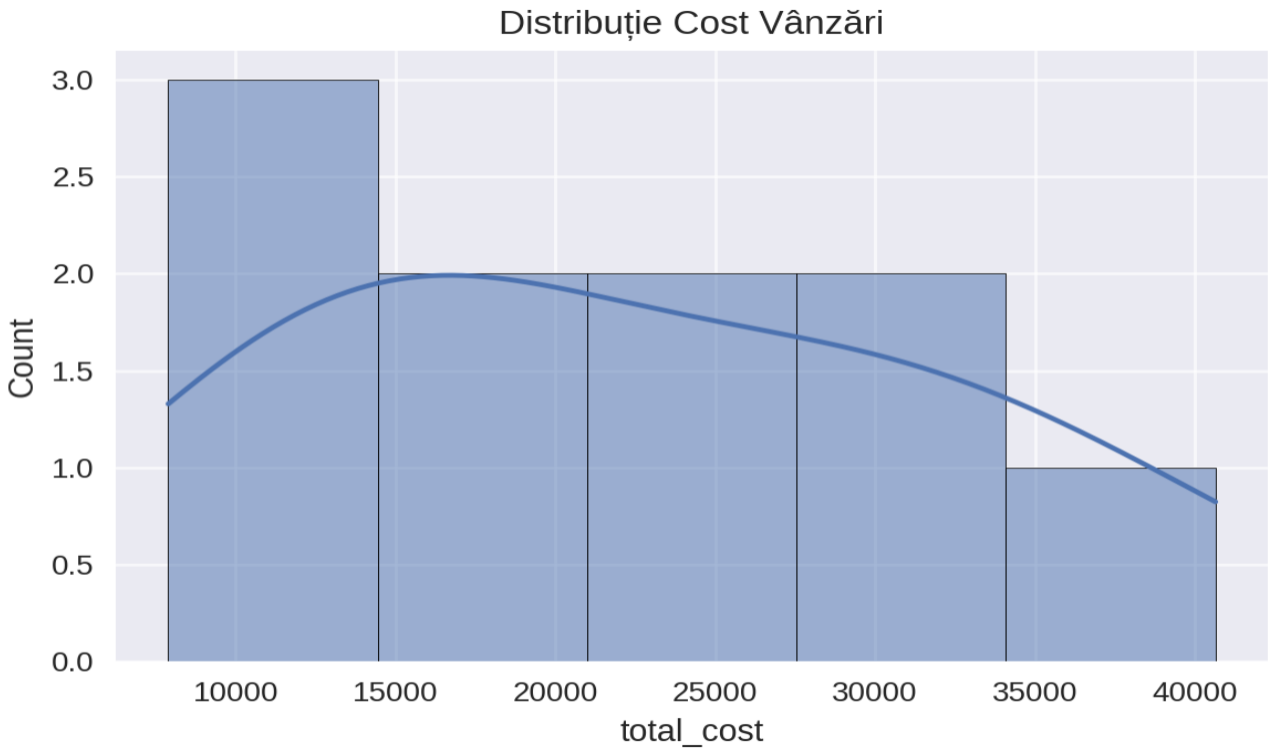

Figura 3: Histogramă - Distribuția costurilor de vânzări

Concluzii și recomandări finale

Proiectul demonstrează cum principiile clasice de audit pot fi transpuse integral într-un flux de lucru automatizat, scalabil și reproductibil si demonstreaza o abordare completa si scalabila pentru automatizarea auditului financiar:

- Structura modulara, reutilizabila
- Separare clara intre date, logica, interfata si output
- Rapoarte si grafice usor de exportat si integrat in documentatii
- Interfata grafica utila pentru contabili sau personal non-tehnic
- Aplică metode de eșantionare și teste de consistență pentru validitatea datelor financiare.
- **Implementare**, codul este modular (separă încărcarea datelor, logica de audit, exportul și vizualizările), folosind cele mai populare biblioteci Python din domeniu (pandas, openpyxl, matplotlib, seaborn).
- **Avantaje**:
    - Reducerea erorii umane prin automatizare
    - Raportare rapidă și vizuală, gata de integrare în rapoarte de management
    - Ușor de extins cu noi conturi sau teste specifice
- **Moduri de utilizare**:
    - Departamente de audit intern în IMM-uri
    - Suport pentru firme de contabilitate la analiza preliminară a datelor
    - Cadrul didactic pentru demonstrarea practică a auditului financiar cu Python

Aceasta aplicatie poate fi extinsa cu:

- Conexiuni la baze de date SQL
- Upload date prin web (Flask, Django)
- Integrare cu sisteme de raportare

În ansamblu, proiectul oferă o bază solidă pentru orice organizație care dorește să își eficientizeze procesele de control financiar și să adopte un cadru modern, bazat pe date, aplicația realizată oferă o soluție completă și automată pentru realizarea auditului financiar, fiind ușor de adaptat și extins pentru cazuri reale. Prin utilizarea Python și a bibliotecilor Pandas și openpyxl, aplicația este robustă, eficientă și ușor de întreținut. Proiectul respectă cele mai bune practici și poate fi extins sau adaptat după necesitățile reale ale firmei auditate. Este extensibil pentru domenii diverse și pregătit pentru integrare GitHub.

Proiectul se poate gasi pe site-ul GitHub:
https://github.com/wladone/Popescu_Vlad_Financiar_Audit_Project.git