

PROIECT PYTHON

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 eşantionarea aleatoare simplă (Simple Random Sampling). Vom selecta eşantioane reprezentative din fiecare categorie financiară pentru analiza detaliată.

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

5. Testarea sampling-ului

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

6. Analiza statistică detaliată cu Pandas

Această etapă include:

- Curăţarea şi eliminarea coloanelor inutile
- Calcularea şi interpretarea statisticilor descriptive avansate (media, mediana, deviaţia standard, minim, maxim, quartile)
- Vizualizarea datelor prin intermediul graficelor şi heatmap-urilor
- Pivotări şi agregări pentru sinteza eficientă a informaţiilor

7. Exportul detaliat al rezultatelor

Rezultatele obţinute vor fi exportate într-un fişier Excel structurat pe foi separate, corespunzătoare fiecărei analize efectuate (statistici şi sampling).

8. IMPLEMENTARE PYTHON

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

Vom crea:

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

Structura directoarelor:

audit_project/

```
├── data/                                # Aici punem fişierele 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
│   ├── auditor.py                    # Clasa principală de audit
│   ├── utils.py                      # Funcţii ajutătoare (curăţare, sampling, etc.)
│   └── generate_charts.py            # Script care generează graficele
└── requirements.txt                  # Librării necesare
```

8.2. ETAPA 2 – Generare automată: codul iniţial

Vom scrie codul complet pentru:

- auditor.py: clasa principală care face auditul
- utils.py: funcţii de sampling, validare date
- main.py: rularea şi controlul principal
- generate_charts.py: generează graficele
- requirements.txt: biblioteci necesare

Audit Project Structure:

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

```
from typing
```

```
import List, Dict
```

```
import pandas as pd
```

```
class FinancialAuditor:
```

```
    """
```

```
    Clasa principală pentru audit financiar.
```

```
    Permite încărcarea datelor, analiza conturilor şi salvarea rezultatelor.
```

```
    """
```

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

```
    """Inițializează cu un dicționar de DataFrame-uri."""
```

```
    self.data = data
```

```
    self.results = {}
```

```
def analyze_cash(self) -> pd.DataFrame:
```

```
    """Analizează datele de cash."""
```

```
    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:
```

```
    """
```

```
    Analiză generică: sumar, statistici și test de logică simplu pe coloane cheie.
```

```
    """
```

```
    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:

```

```

    """Exportă rezultatele în Excel."""

```

```

    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:
```

```
    """Încarcă fișierele CSV în DataFrame-uri și le mapează la cheile date."""
```

```
    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:
```

```
    """Returnează un eșantion din DataFrame."""
```

```
    return df.sample(frac=fraction, random_state=seed)
```

```
# scripts/main.py
```

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

```
# Adaugă folderul părinte
```

```
(root-ul proiectului) în sys.path înainte de orice import
```

```
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
```

```
if __name__ == "__main__":
```

```
    # Calea către fișierele CSV de test
```

```
paths = ["data/cash.csv", "data/payable.csv", "data/receivable.csv",  
        ", "data/cost_of_sales.csv", "data/operating_expenses.csv", "data/ppe.csv"]
```

```
keys = ["cash", "payable", "receivable", "cost_of_sales.csv",  
        "operating_expenses.csv", "ppe.csv"]
```

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

```
# Audit cash și payable
```

```
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()
```

```
# Salvăm rezultatele în Excel
```

```
output_file = os.path.join("outputs", "audit_results.xlsx")  
auditor.export_results(output_file)
```

```
print(f"Rezultatele au fost salvate în {output_file}")
```

```
# requirements.txt
```

```
pandas  
openpyxl  
matplotlib  
seaborn
```

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**: generează graficele
- **main.py**: Scriptul principal care:
 - încarcă datele din `data/*.csv`
 - rulează analiza financiară
 - salvează rezultatele în `outputs/audit_results.xlsx`
- **requirements.txt**: cu librăriile necesare (pandas, openpyxl, matplotlib, seaborn)

Mai sus au fost prezentate următoarele 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

Având aceste fișiere codul a fost extinsa clasa `FinancialAuditor` cu metode de analiză pentru:

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

Codul din `auditor.py` acum include următoarele 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 următoarele:

- 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.) și pentru a completa analiza vizuală a datelor vom folosi stilul implicit **Seaborn** (care este stabil și suportat) pentru generarea scriptului graficelor în locația: `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

Script degerare grafice:

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`

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:

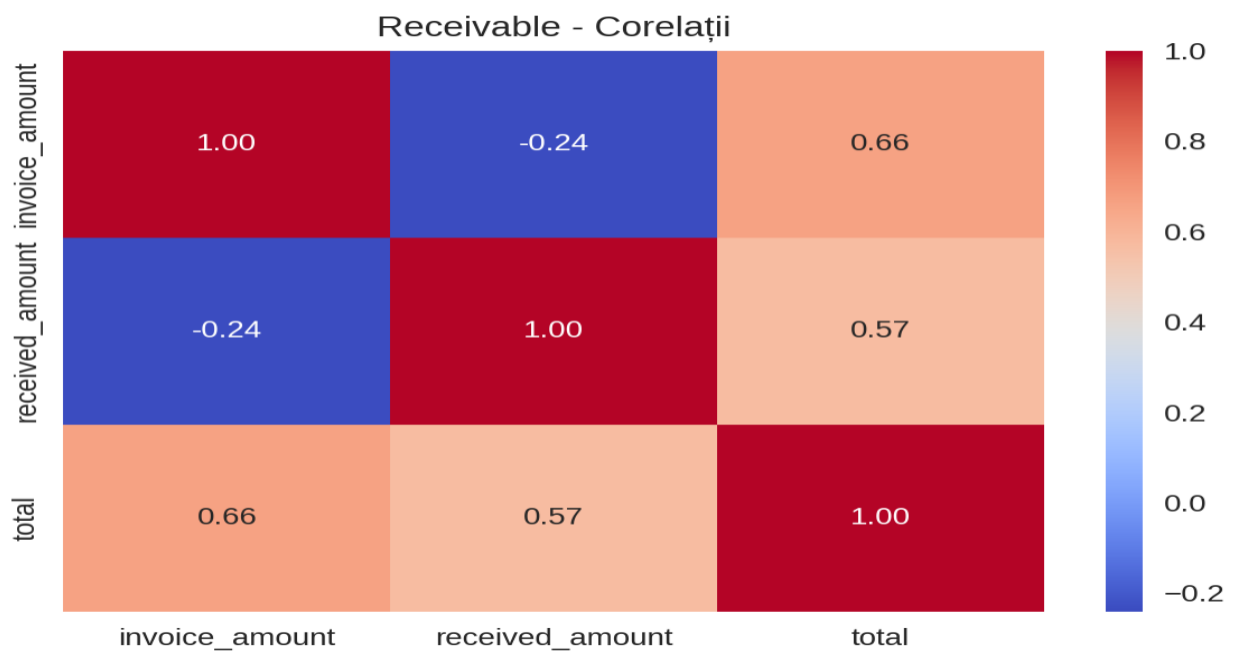


Figura 1: Heatmap - Corelații între câmpuri în contul de clienți

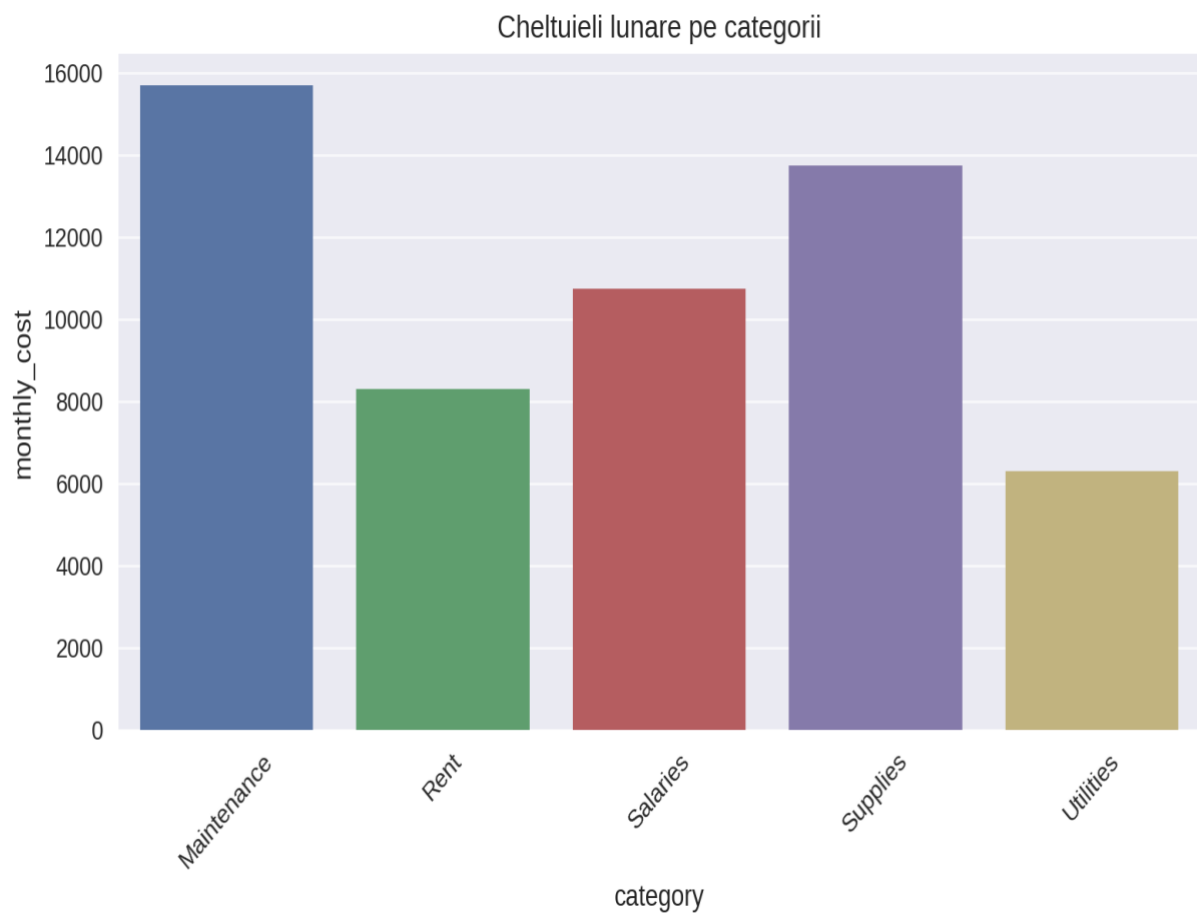


Figura 2: Bar chart - Cheltuieli lunare pe categorii

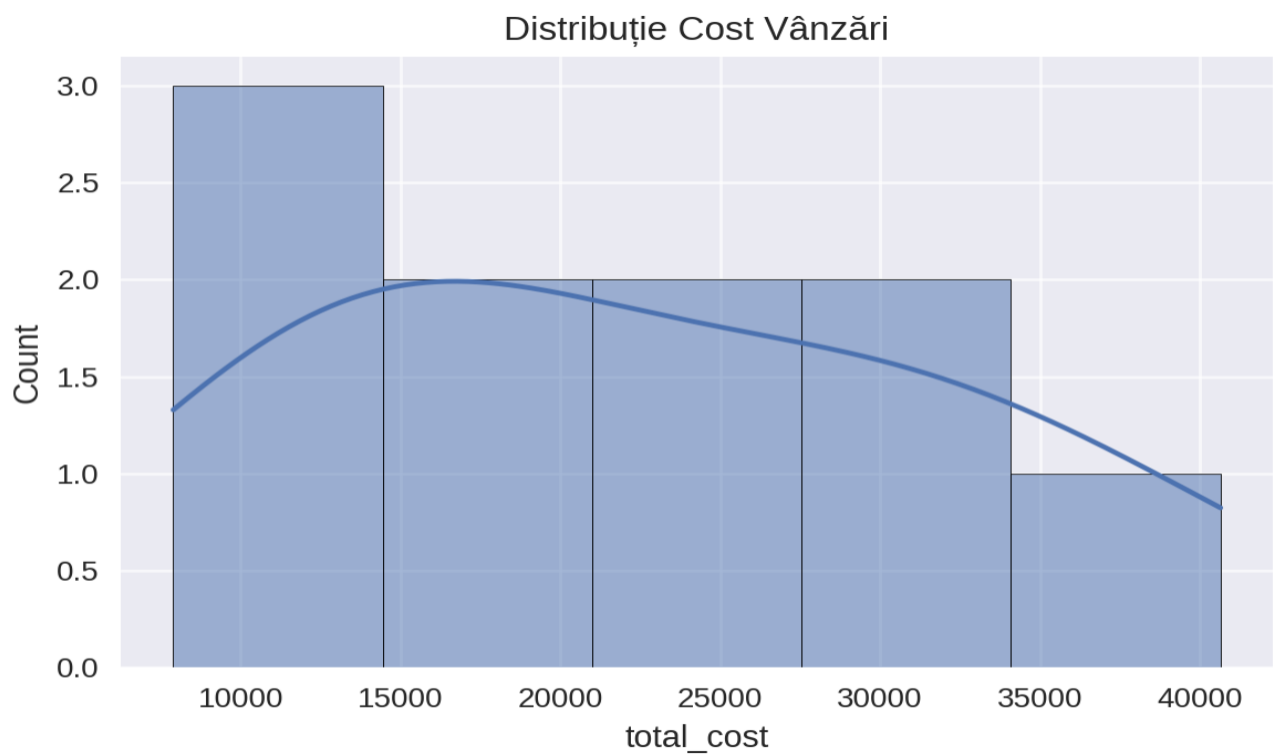


Figura 3: Histogramă - Distribuția costurilor de vânzări

Concluzii și recomandări finale

Proiectul demonstrează cum principiile clasice de audit pot fi transpuse integral într-un flux de lucru automatizat, scalabil și reproductibil.

- **Teoretic**, 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

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 găsi pe site-ul GitHub:

https://github.com/wladone/Popescu_Vlad_Financiar_Audit_Project.git