**Project Title:**

**“Enterprise Data Governance Simulation: Classification, Quality, Access & Compliance Audit”**

**Project Scope (Matches LinkedIn Points)**

| **Feature** | **Included** | **How We'll Do It** |
| --- | --- | --- |
| Data Cataloging | Yes | Classify columns (PII, Financial, ID, General) using Python |
| Access Control Audit | Yes | Map user roles to sensitive datasets |
| Data Quality Assessment | Yes | Check for nulls, duplicates, and invalid formats |
| Python Automation | Yes | All logic in reusable .py scripts |
| Data Lineage Visualization | Yes | Use Mermaid.js or Graphviz to show flow |
| Compliance Simulation (PIPEDA) | Yes | Flag PII access by non-authorized roles |
| Cloud Migration Simulation | Yes | Compare metadata pre/post move (simulate S3 or GCS target) |
| Reporting & Dashboards | Yes | CSV exports + optional Streamlit dashboard |

**Datasets (each with 100+ records)**

| **Dataset** | **Description** |
| --- | --- |
| customers.csv | Includes customer\_id, name, email, dob, etc. |
| transactions.csv | Includes transaction\_id, amount, date, customer\_id |
| employees.csv | Includes employee\_id, name, role, dataset access |
| reports.csv | Summary-level KPIs (revenue, trends, etc.) |

CODE : ACCESS AUDIT

import pandas as pd

# step 1: Load the needed files and merge info

# load tagged catalog (with data classification)

tagged\_catalog = pd.read\_csv('tagged\_catalog.csv')

#load employee access info

employees = pd.read\_csv('employees.csv')

#show columns as samples to verify

print("Tagged catalog columns:", tagged\_catalog.columns.tolist())

print(tagged\_catalog.head())

print("employees columns:",employees.columns.tolist())

print(employees.head())

A computer screen with white text

AI-generated content may be incorrect.

#Step 2: Define sensitive data categories to check

#Senistive categories that need restricted access

sensitive\_categories= ['PII','Financial']

#Step 3: Identify which datasets contain sensitive data

#find dataset that have any sensitive columns

sensitive\_datasets = tagged\_catalog[tagged\_catalog['data\_classification'].isin(sensitive\_categories)]['dataset\_name'].unique()

print('Sensitive datasets:',sensitive\_datasets)

#Step 4: Define access rules (example)

#Example access rules byb role for sensitive datasets

#Roles allowes to access sentsitive data fully

allowed\_roles = ['Manager','data\_engineer','compliance\_officer']

#Roles not allowed to access sensitive datasets (or limited)

restricted\_roles = ['intern','analyst']

#Step 5: Audit employee access

#Function to flag risky access

def audit\_access(row):

dataset = row['dataset\_name']

role = row['role'].lower()

access = row['access\_level'].lower()

if dataset in sensitive\_datasets:

if role in restricted\_roles:

return 'RISK - Unauthorized access to sensitive data'

elif role in allowed\_roles and access =='read':

return 'Warning -Limited access to sensitive data'

else:

return 'OK'

else:

return 'OK'

#apply the audit function

employees['access\_audit']= employees.apply(audit\_access, axis=1)

#view eisky entries only

risk\_report = employees[employees['access\_audit'] != 'OK']

print(risk\_report)

#Step 6: Save the audit report

risk\_report.to\_csv('access\_risk\_report.csv', index=False)

print("Access risk report saved as access\_risk\_report.csv")

A computer screen shot of a computer

AI-generated content may be incorrect.

CODE: Cloud migration data

import pandas as pd

# Define simulated cloud migration report

data = [

{"dataset\_name": "customers", "data\_classification": "PII", "encrypted": "No", "eligible\_for\_cloud": "No", "migration\_status": "Blocked", "reason": "Needs encryption"},

{"dataset\_name": "transactions", "data\_classification": "Financial", "encrypted": "Yes", "eligible\_for\_cloud": "Yes", "migration\_status": "Migrated", "reason": "-"},

{"dataset\_name": "reports", "data\_classification": "General", "encrypted": "-", "eligible\_for\_cloud": "Yes", "migration\_status": "Migrated", "reason": "-"},

{"dataset\_name": "employees", "data\_classification": "PII", "encrypted": "Yes", "eligible\_for\_cloud": "Yes", "migration\_status": "Migrated", "reason": "-"}

]

# Convert to DataFrame

cloud\_migration = pd.DataFrame(data)

# Save to CSV

cloud\_migration.to\_csv('cloud\_migration\_report.csv', index=False)

A screenshot of a computer

AI-generated content may be incorrect.

print("cloud\_migration\_report.csv has been created!")

cloud\_migration

CODE:Compliance checker

#step 1 load employee access and tagged data

import pandas as pd

#Load files

employees = pd.read\_csv('employees.csv')

tagged = pd.read\_csv('tagged\_catalog.csv')

#Define roles Not allowed to access PII

restricted\_roles = ['intern','analyst']

#Get datasets that contain PII

pii\_datasets=tagged[tagged['data\_classification']=='PII']['dataset\_name'].unique()

print("PII datasets:", pii\_datasets)

#Step 2: Flag PIPEDA violations

#function to checkfor PII access by restricted roles

def check\_pipeda\_violation(row):

if row['dataset\_name'] in pii\_datasets and row['role'].lower() in restricted\_roles:

return 'violation - Unauthorized PII Access'

return 'Compliant'

# Apply to employees datasets

employees['pipeda\_flag'] = employees.apply(check\_pipeda\_violation,axis =1)

#view violation

violations = employees[employees['pipeda\_flag'] != 'Compliant']

violations

violations.to\_csv('pipeda\_violations.csv', index=False)

print("Violations report saved as pipeda\_violations.csv")

CODE: Data governance dashnoard

import pandas as pd

import plotly.express as px

from dash import Dash, dcc, html, dash\_table

#load data

tagged = pd.read\_csv("tagged\_catalog.csv")

quality = pd.read\_csv("data\_quality\_report.csv")

migration = pd.read\_csv("cloud\_migration\_report.csv")

violations = pd.read\_csv("pipeda\_violations.csv")

#Start Dash app

app = Dash(\_\_name\_\_)

app.title = "data Governance Dashboard"

#Data Classification chart

classification\_fig = px.histogram(tagged,x="data\_classification",

color="data\_classification",

title="Data classification count")

#data quality issues chart

quality\_fig = px.bar(

quality,

x="dataset",

y=["null\_values",

"duplicate\_rows",

"invalid\_amounts",

"invalid\_revenue"],

title="Data Quality Issues by Dataset",

barmode = "stack"

)

#cloud migration status

migration\_fig = px.histogram(migration,

x="migration\_status",

color = "migration\_status",

title="Cloud Migration Status")

#Layout

app.layout=html.Div([

html.H1("Data Governance Dashboard",

style={'textAlign': 'center'}),

dcc.Graph(figure=classification\_fig),

dcc.Graph(figure=quality\_fig),

dcc.Graph(figure=migration\_fig),

html.H2("PIPEDA violations",style = {'marginTop':'30px'}),

dash\_table.DataTable(

data=violations.to\_dict('records'),

columns=[{"name":i,"id":i} for i in violations.columns],

style\_table={'overflowX' : 'auto'},

style\_cell={'textAlign':'left','padding' : '5px'},

page\_size=10

)

])

#export to html file

app.run(debug=False, port=8050)

A graph of data classification

AI-generated content may be incorrect.

CODE: Data Lineage

#Step 2: Import libraries and define lineage

import networkx as nx

import matplotlib.pyplot as plt

#Sub-step1 create a directed path

G= nx.DiGraph()

#Sub-Step2 dEFINE dataset nodes

datasets = ["customers","transactions","reports","employees"]

#sub-step 3 Define lineage connections

edges = [

("customers","transactions"), # customer\_id linked customers -> transaction

("transactions","reports"), # used for revenue reporting

("employees","customers"), #employee access to customers

("employees","transactions"),

("employees","reports")

]

#sub step 4 add nodes and edge to the graph

G.add\_nodes\_from(datasets)

G.add\_edges\_from(edges)

#Step 3: Draw the graph

#substep 1 sraw graph

plt.figure(figsize=(10,6))

pos = nx.spring\_layout(G, seed=42)

nx.draw\_networkx\_nodes(G,pos,node\_color = 'skyblue', node\_size =2000)

nx.draw\_networkx\_edges(G,pos, arrowstyle='->', arrowsize=20, edge\_color = 'gray',width=2)

nx.draw\_networkx\_labels(G,pos,font\_size=12,font\_weight='bold')

plt.title("Data Lineage map")

plt.axis('off')

plt.show()

A diagram of a diagram

AI-generated content may be incorrect.

CODE: Data Quality check

This script will: Load each dataset (customers.csv, transactions.csv, etc.)

Check for:

❌ Missing values (null, NaN)

❌ Duplicate rows

❌ Invalid values (e.g., wrong date format or negative amounts)

Save a quality report

# Step 1: Import libraries and define file list

import pandas as pd

# listof dataset files to check

files= ['customers.csv','transactions.csv','employees.csv','reports.csv']

#create an empty list to collect result

quality\_report=[]

# Step 2: Loop through each file and run checks

for file in files:

try:

df = pd.read\_csv(file)

dataset\_name=file.replace('.csv','')

#basic checks

total\_rows= len(df)

null\_count= df.isnull().sum().sum()

duplicate\_count = df.duplicated().sum()

#sample custom checks:

#for transaction :amount should be >0

invalid\_amounts=0

if 'amount' in df.columns:

invalid\_amounts = df[df['amount'] <=0].shape[0]

#for reports :total revenue should be >0

invalid\_revenue =0

if 'total\_revenue' in df.columns:

invalid\_revenue = df[df['total\_revenue'] <=0].shape[0]

#append results

quality\_report.append({

'dataset': dataset\_name,

'total\_rows': total\_rows,

'null\_values': null\_count,

'duplicate\_rows': duplicate\_count,

'invalid\_amounts': invalid\_amounts,

'invalid\_revenue': invalid\_revenue

})

except Exception as e:

print(f"Failed to process{file}:{e}")

#Step 3: Save and show the report

report\_df = pd.DataFrame(quality\_report)

# Show in notebook

report\_df

report\_df.to\_csv('data\_quality\_report.csv', index=False)

print("Data quality report saved as data\_quality\_report.csv")

CODE: PowerBi style dashboard

What We'll Include in the Report: Section Visual Type Shows 🔐 Data Classification Count Bar chart How many columns are PII, Financial, etc. 👥 Access Violations Table Employees who accessed restricted data 🧪 Data Quality Summary Table / Bar Chart Missing, duplicate, or invalid values per file ☁️ Cloud Migration Readiness Bar chart Eligible vs Blocked datasets

🛠️ Tools: Python

matplotlib and seaborn for charts

Pandas DataFrames for table-like Power BI visuals

#step 1 : Import libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Set chart style

sns.set(style='whitegrid')

#Step 2 : Load required files

#Load datasets

tagged = pd.read\_csv('tagged\_catalog.csv')

violations = pd.read\_csv('pipeda\_violations.csv')

quality = pd.read\_csv('data\_quality\_report.csv')

migration = pd.read\_csv('cloud\_migration\_report.csv')

#step 3 : Data classification count(Bar chart)

plt.figure(figsize=(8,5))

sns.countplot(data=tagged, x='data\_classification', palette='pastel')

plt.title('Column Count by Data Classification')

plt.xlabel('Classification type')

plt.ylabel('Number of Columns')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

#Step 4: Data Quality Summary (Bar Chart)

plt.figure(figsize=(10,5))

quality.set\_index('dataset')[['null\_values','duplicate\_rows','invalid\_amounts','invalid\_revenue']].plot(kind='bar',stacked=True,colormap='coolwarm')

plt.title("Data Quality Issues by Dataset")

plt.ylabel("Issue Count")

plt.xlabel("Dataset")

plt.tight\_layout()

plt.show()

#Step 5: Cloud Migration Summary (Bar Chart)

if 'migration\_status' in migration.columns:

plt.figure(figsize=(6, 4))

sns.countplot(data=migration, x='migration\_status', palette='muted')

plt.title('Cloud Migration Status')

plt.xlabel('Status')

plt.ylabel('Dataset Count')

plt.tight\_layout()

plt.show()

# Step 6: Access Violation Table

# Show key access violation fields

violations[['employee\_id', 'name', 'role', 'access\_level', 'dataset\_name', 'pipeda\_flag']]

CODE : tagging columns

import pandas as pd

# step 1: load the metadata file

metadata=pd.read\_csv('datasets.csv')

# step 2 : Define rules to tag column

def tag\_column(column\_name):

col=column\_name.lower()

if any(keyword in col for keyword in ['name','email','dob','date\_of\_birth','address','phone']):

return 'PII' #Personally identifiable information

elif any(keyword in col for keyword in ['amount', 'revenue','price','cost','total']):

return 'Financial'

elif any(keyword in col for keyword in ['id','transaction\_id','customer\_id','employee\_id','report\_id']):

return 'identifier'

else:

return 'General'

#step 3 : apply tagging function to each column

metadata['data\_classification']=metadata['Column\_name'].apply(tag\_column)

#step 4 : save the tagged catalog to new csv

metadata.head()

metadata.to\_csv('tagged\_catalog.csv', index=False)

print("Tagging complete! Output saved to tagged\_catalog.csv")