PYTHON LEVEL 3 – PROJECT

Project Overview

In this project, you will develop a Python application to evaluate the risk associated with loan applicants in a banking context. You should apply your knowledge on Python functions, comprehensions, built-in functions, object-oriented programming, and modularization, and use **Streamlit** to create an interactive user interface.

Objective

Build a modular Python application that:

- Processes loan applicants (individual and business).
- Computes their financial risk score.
- Categorizes their risk level.
- Generates a summary report for the risk analysis team.
- Provides an interactive dashboard.

Required Files and Structure

Your final project should contain at least these files:

loan_risk_analyzer/
```
├── app.py              # Entry point
├── sample_data.py      # Preloaded applicant data
├── risk_tools.py       # Functions for analysis and summary
├── client.py           # Class definitions for applicants
├── requirements.txt    # Client class definition
```

Input Data

Applicants will be represented as a list of dictionaries with fields: ID, income, loan_amount, credit_score, and age. Example:

```
[
{"id": "A001", "income": 50000, "loan_amount": 20000, "credit_score": 650, "age": 40},
{"id": "A002", "income": 30000, "loan_amount": 18000, "credit_score": 590, "age": 22}
]
```

Risk Score Formula

- For **individual applicants**:
  - Risk score = (loan_amount / income) * 100
  - +10 if credit_score < 600
  - +5 if age < 25 or age > 60
- For **business applicants**:
  - risk_score = (loan_amount / (income + 0.3 * revenue)) * 100
  - +8 if credit_score < 620

Risk Categories:

- Low: score < 20
- Medium: 20 <= score <= 40
- High: score > 40

Requirements

1. Programming and OOP

Applicant Module
Create a module called **client.py** containing at least the following classes:

- **ApplicantBase** (Abstract Class)
    a. Common attributes: **id**, **income**, **loan_amount**, **credit_score**, **age**
    b. Abstract method: **calculate_risk_score**()
- **LoanApplicant** (inherits from ApplicantBase): implements **calculate_risk_score**()
- **BusinessApplicant** (inherits from LoanApplicant)
    a. Adds a new attribute: **revenue**
    b. Adjusts risk formula to include business performance.

Portfolio Analyzer Module
Create **portfolio_analyzer.py**, which defines a **PortfolioAnalyzer** class to handle all operations related to a group of applicants.

- **Responsibilities**:
    o Store applicants (loan and business types)
    o Compute portfolio-level statistics
    o Convert data into a table for visualization
    o Filter applicants by risk category
- **Minimum required methods**:

Method	Type	Description
make_applicant(row)	@staticmethod	Returns the correct applicant object (LoanApplicant or BusinessApplicant)
from_dicts(rows)	@classmethod	Builds a portfolio (dict) from a list of dictionaries
summarize()	instance	Calculates total count, average risk, highest risk, and risk distribution
to_dataframe()	Instance	Returns a pandas.DataFrame of all applicants
filter_by_category(category)	instance	Returns only applicants of the selected risk category

- **Example Usage**
```
portfolio = PortfolioAnalyzer.from_dicts(sample_data)
summary = portfolio.summarize()
df = portfolio.to_dataframe()
```

Streamlit Application
Create a Streamlit app in app.py. Your app should include:

1. **Side Bar**
    a. **Data Uploading**
    Allow user to update a CSV file with columns **id**, **income**, **loan_amount**, **credit_score**, **age**, and, optionally, **revenue**. If no file is uploaded, load the default sample_data.

    b. **Adding Applicants Manually**
        ▪ Use a sidebar form to let users add new applicants.

- Use a form (st.sidebar.form()) to group input fields together and handle submission.
- Include a dropdown menu (st.selectbox()) to choose the applicant type ("individual" or "business").
- Use input fields (st.text_input() and st.number_input()) to collect data such as ID, income, loan amount, credit score, and age.
- If the applicant type is "business", include an additional numeric input for revenue.
- When the form is submitted, create a dictionary for the new applicant and append it to the data stored in st.session_state.rows.
  - session_state prevents Streamlit from rerunning the entire script and losing data when interacting with the app. Example:
    ```
    if "rows" not in st.session_state:
        st.session_state.rows = None
    ```
- Display a success message after each addition using st.sidebar.success().

2. **Main Body**
   a. **Display Data and Analysis**
   Include a section or tab showing the input data in a table (st.dataframe()).
   Compute the risk. Show the results table in a new tab or section.
   - Allow filtering by category (All, Low, Medium, High).
   - Include a **chart** (st.bar_chart()) showing the distribution.

   When filtering applicants by risk category, the table and chart should both update to display only the selected category (use filter_by_category() to produce a filtered df)

   b. **Summary**
   In a new tab or section, display overall portfolio statistics such as:
   - Total number of applicants
   - Number of high-risk applicants
   - Average risk score
   - Highest-risk applicant (ID and score)

   You can present these statistics using:
   - st.write() (simple text display),
   - st.table() (compact table display), or
   - st.metric() (optional cleaner layout for visual dashboards).

   (Optional) Display a bar chart for the distribution of risk categories.

   How to use st.metric()
   Obtain the summary dictionary: summary = portfolio.summarize()
   Create four columns side by side (col1, col2, col3, col4 = st.columns(4))
   In the first column: st.metric "Total Applicants" with value summary["total"]
   In the second column: Show metric "High-Risk Applicants" with value summary["high_risk"]
   In the third column: Show metric "Average Risk Score" with value summary["average_score"]
   In the fourth column:
     If summary["highest_risk"] exists:
       Extract applicant_id, score, category
       Show metric "Highest Risk" with applicant_id and score (and category)

Otherwise:

Show metric "Highest Risk" with placeholder "—"

3. Optional UI Enhancement: Tabs

To organize the interface, you can use Streamlit tabs. This step is optional but recommended.

Tab demonstration

```
tab1, tab2, tab3 = st.tabs(["Data Overview", "Risk Analysis", "Summary"])
with tab1:
    # Show input data
    st.dataframe(...)
with tab2:
    # Show filtered risk table and chart
with tab3:
    # Show summary metrics
```

4. Deployment

- Create a GitHub repository named **loan-risk-analyzer-level3** and upload your full project (all modules).
- Add a README.md describing setup and usage.
- Deploy your Streamlit app on Streamlit Cloud.

Example Output