Guarantee Assurance Initiative for Project Portfolio Execution

This Jupiter notebook constitutes part of a research project. The project's full details are available in the webpage at https://evgenii-sokolov.github.io/Portfolio_website/Projects_execution_study.html or in the PowerPoint presentation at https://github.com/Evgenii-Sokolov/Projects_execution_study/blob/main/PES_Report.pptx.

This Jupyter notebook is designed to randomly generate the dataset required for the project's purposes. The notebook utilizes randomization techniques to create a dataset that accurately represents the desired parameters and variables needed for the analysis.

Import the required libraries.

In [1]:

```
import pandas as pd
import numpy as np
```

We are creating a dataframe with the required columns.

In [2]:

```
df = pd.DataFrame(columns=['Contract_ID', 'Project', 'Contractor', 'Start_date', 'End_da
```

We are populating the columns with values of project names, contractors, and contracts.

In [3]:

```
Contract_ID_list = []
Project_list = []
Contractor_list = []

for i in range(1, 5001):
    Contract_ID_list.append(f'Contract_ID_{i}')
    Project_list.append(f'Project_{np.random.randint(1, 5)}')
    Contractor_list.append(f'Contractor_{np.random.randint(1, 100)}')

df['Contract_ID'] = Contract_ID_list
df["Project"] = Project_list
df["Contractor"] = Contractor_list
```

We are populating the "Start Date" column with random dates.

In [4]:

```
date_range = pd.date_range(start='01-01-2018', end='01-01-2022')
df['Start_date'] = pd.to_datetime(date_range).to_series().sample(n=5000, replace=True).r
```

We are calculating the contract end date taking into account the desired duration.

In [5]:

```
def generate_random_end_date(start_date):
    random_offset = pd.DateOffset(months=np.random.randint(2, 12))
    return start_date + random_offset

df['End_date'] = df['Start_date'].apply(generate_random_end_date)
```

We are generating the dates of guarantee provision based on the project requirements.

In [6]:

```
df['AP_G_date_plan'] = df['Start_date'] + pd.DateOffset(months=1)
df['CE_G_date_plan'] = df['Start_date'] + pd.DateOffset(days=20)
```

```
df['WP_G_date_plan'] = df['End_date'] + pd.DateOffset(months=-1)
```

Next, we proceed to fill in the columns that contain the responsibility for non-provision of guarantees. We determine the type and magnitude of the responsibility.

In [7]:

```
df['G_penalty_type'] = np.random.choice(['fine', 'forfeit'], size=len(df))
```

In [8]:

```
mask_fine = df['G_penalty_type'] == 'fine'
df.loc[mask_fine, 'G_penalty_value'] = np.random.choice([0.1, 0.15, 0.2], size=mask_fine
```

In [9]:

```
mask_forfeit = df['G_penalty_type'] == 'forfeit'
df.loc[mask_forfeit, 'G_penalty_value'] = np.random.choice([0.01, 0.02, 0.05, 0.06], siz
```

We proceed to fill in the actual dates of guarantee provision. Initially, we assign a deadline that meets the project conditions for all types of guarantees. Then, selectively, we assign certain groups of guarantees values that deviate from the deadline for guarantee provision.

In [10]:

```
df['AP_G_date_fact'] = df['AP_G_date_plan']
df['CE_G_date_fact'] = df['CE_G_date_plan']
df['WP_G_date_fact'] = df['WP_G_date_plan']
```

In [11]:

```
def generate_random_AP_G_date_fact(AP_G_date_plan):
    random_offset = pd.DateOffset(days=np.random.randint(1, 30))
    return AP_G_date_plan + random_offset

selected_rows_AP = df.sample(frac=0.11).index
df.loc[selected_rows_AP, 'AP_G_date_fact'] = df.loc[selected_rows_AP, 'AP_G_date_plan'].
```

In [12]:

```
def generate_random_WP_G_date_fact(WP_G_date_plan):
    random_offset = pd.DateOffset(days=np.random.randint(10, 50))
    return WP_G_date_plan + random_offset

selected_rows_WP = df.sample(frac=0.13).index
df.loc[selected_rows_WP, 'WP_G_date_fact'] = df.loc[selected_rows_WP, 'WP_G_date_plan'].
```

In order to accurately fill in the values for the actual date of guarantee provision for contract execution, a different approach is required. It is crucial for us to establish a specific distribution of this parameter based on the type and size of responsibility. The following code block is designed to achieve these objectives. We will generate samples of varying sizes and ranges for each type and size of responsibility to populate the data accordingly.

In [ ]:

```
selected_rows_001 = df[df['G_penalty_value'] == 0.01].sample(frac=0.23).index
selected_rows_002 = df[df['G_penalty_value'] == 0.02].sample(frac=0.20).index
selected_rows_005 = df[df['G_penalty_value'] == 0.05].sample(frac=0.10).index
selected_rows_006 = df[df['G_penalty_value'] == 0.06].sample(frac=0.10).index
selected_rows_01 = df[df['G_penalty_value'] == 0.1].sample(frac=0.40).index
selected_rows_015 = df[df['G_penalty_value'] == 0.15].sample(frac=0.30).index
selected_rows_02 = df[df['G_penalty_value'] == 0.2].sample(frac=0.30).index
```

In [14]:

```
def generate_random_CE_G_001_date_fact(CE_G_001_date_fact):
    random_offset = pd.DateOffset(days=np.random.randint(20, 100))
    return CE_G_001_date_fact + random_offset

def generate_random_CE_G_002_date_fact(CE_G_002_date_fact):
    random_offset = pd.DateOffset(days=np.random.randint(20, 80))
    return CE_G_002_date_fact + random_offset

def generate_random_CE_G_005_date_fact(CE_G_005_date_fact):
    random_offset = pd.DateOffset(days=np.random.randint(20, 50))
    return CE_G_005_date_fact + random_offset

def generate_random_CE_G_006_date_fact(G_006_date_fact):
    random_offset = pd.DateOffset(days=np.random.randint(20, 50))
    return G_006_date_fact + random_offset

def generate_random_CE_G_01_date_fact(G_01_date_fact):
    random_offset = pd.DateOffset(days=np.random.randint(20, 150))
    return G_01_date_fact + random_offset

def generate_random_CE_G_015_date_fact(G_015_date_fact):
    random_offset = pd.DateOffset(days=np.random.randint(20, 100))
    return G_015_date_fact + random_offset

def generate_random_CE_G_02_date_fact(G_02_date_fact):
    random_offset = pd.DateOffset(days=np.random.randint(20, 100))
    return G_02_date_fact + random_offset


df.loc[selected_rows_001, 'CE_G_date_fact'] = df.loc[selected_rows_001, 'CE_G_date_fact'
df.loc[selected_rows_002, 'CE_G_date_fact'] = df.loc[selected_rows_002, 'CE_G_date_fact'
df.loc[selected_rows_005, 'CE_G_date_fact'] = df.loc[selected_rows_005, 'CE_G_date_fact'
df.loc[selected_rows_006, 'CE_G_date_fact'] = df.loc[selected_rows_006, 'CE_G_date_fact'
df.loc[selected_rows_01, 'CE_G_date_fact'] = df.loc[selected_rows_01, 'CE_G_date_fact'].
df.loc[selected_rows_015, 'CE_G_date_fact'] = df.loc[selected_rows_015, 'CE_G_date_fact'
df.loc[selected_rows_02, 'CE_G_date_fact'] = df.loc[selected_rows_02, 'CE_G_date_fact'].
```

Now, let's validate the resulting dataset by displaying it on the screen.

In [15]:
```
df
```

Out[15]:

	Contract_ID	Project	Contractor	Start_date	End_date	AP_G_date_plan	AP_G_date_fact	CE_G_date_
0	Contract_ID_1	Project_2	Contractor_57	2020-01-12	2020-10-12	2020-02-12	2020-02-12	2020-0
1	Contract_ID_2	Project_2	Contractor_52	2018-12-16	2019-11-16	2019-01-16	2019-01-16	2019-0
2	Contract_ID_3	Project_2	Contractor_34	2019-06-05	2019-12-05	2019-07-05	2019-07-05	2019-0
3	Contract_ID_4	Project_1	Contractor_82	2019-01-13	2019-11-13	2019-02-13	2019-02-13	2019-0
4	Contract_ID_5	Project_1	Contractor_50	2020-03-01	2020-05-01	2020-04-01	2020-04-01	2020-0
...	...	...	...	...	...	...	...	
4995	Contract_ID_4996	Project_4	Contractor_86	2019-08-22	2020-01-22	2019-09-22	2019-09-22	2019-0
4996	Contract_ID_4997	Project_1	Contractor_64	2021-02-11	2021-12-11	2021-03-11	2021-03-11	2021-0
4997	Contract_ID_4998	Project_2	Contractor_15	2018-02-16	2018-10-16	2018-03-16	2018-03-16	2018-0
4998	Contract_ID_4999	Project_1	Contractor_38	2019-01-15	2019-04-15	2019-02-15	2019-02-15	2019-0
4999	Contract_ID_5000	Project_1	Contractor_14	2019-01-24	2019-05-24	2019-02-24	2019-02-24	2019-0

5000 rows × 13 columns

We will save the dataset to an Excel file for further use in the project.

In [19]:

```
#df.to_excel("data.xlsx")
```

Thank you for your time and attention!