Customer and Marketing Analytics

Raisa Nurlatifah

Import libraries

In [3]:
```
import pandas as pd
import numpy as np
import itertools
import random
from collections import Counter
```

Task 1: Conjoint Design Survey

The task is to find the optimum design survey for conjoint analysis for a SaaS company to understand users' most preferences on support options, with given attribute as below: | Support Options | Price | |:-------------------------|-----:| | Standard Email Support | $40||Standard Phone Support|45 | | On-Site Support | $50||24/7 Phone Support|65 | | Livechat Support | $85||Dedicated Account Manager|90 | | Account Manager | $95||Custom Support Plan|99 |

Create survey

The survey form will consist of 25 questions, each question consists of A, B, C for the choices and D for none.

Load dataframe

In [10]:
```
# Create and display dataframe

data_1 = {
    'Support Options': ['Standard Email Support', 'Standard Phone Support', 'On-Site Support', '24/7 Phon
    'Price': ['$40', '$45',' $50', '$65','$85', '$90', '$95', '$99']
}

df_1 = pd.DataFrame(data_1)


df_1
```

Out[10]:

	Support Options	Price
0	Standard Email Support	$40
1	Standard Phone Support	$45
2	On-Site Support	$50
3	24/7 Phone Support	$65
4	Livechat Support	$85
5	Dedicated Account Manager	$90
6	Account Manager	$95
7	Custom Support Plan	$99

Generate combinations

- The data to analyze has 2 attribute consist of 8 levels.
- The survey determined to have 25 questions
- Each of question will have 3 options with 1 additional None option
- Combination to generate will be based on either `Total Required Profiles` or `Total Unique Profiles`.

$$\text{Total Unique Profiles} = \text{Levels of A} \times \text{Levels of B} \times \text{Levels of C}$$

$$\text{Total Required Profiles} = \text{Number of Options} \times \text{Number of Questions}$$

If `Required Profiles` > `Unique Profiles` then same profiles must be reuse to fill the required profiles while ensuring each question still have distinct choices by avoiding same repetition in a single question.

Based on formula above and calculation below:

Total Unique Profile $= 8 \times 8 = 64$
Total Required Profiles $= 3 \times 25 = 75$

The `Total Required Profile` > `Total Unique Profile` so there must be profile being reused.

In [13]:
```python
def generate_combinations(df_attributes, num_combinations, seed=52):
    """
    Generates a balanced selection of attribute combinations.

    Parameters
    ----------
    df_attributes : pandas DataFrame
        DataFrame containing attribute values in each column.
    num_combinations : int
        Number of combinations to generate.
    seed : int, optional
        Random seed for reproducibility, by default 52.

    Returns
    -------
    pandas DataFrame
        DataFrame with a balanced selection of combinations.
    """
    if df_attributes.empty:
        return pd.DataFrame()

    # Step 1: Generate all possible combinations (ignoring NaN)
    attribute_lists = [df_attributes[col].dropna().tolist() for col in df_attributes.columns]
    all_combinations = list(itertools.product(*attribute_lists))
    max_possible_combinations = len(all_combinations)

    # Step 2: Shuffle all combinations
    random.seed(seed)
    random.shuffle(all_combinations)

    # Step 3: Ensure balanced selection
    selected_combinations = []
    count_per_level = {col: Counter() for col in df_attributes.columns}

    for combination in all_combinations:
        # Ensure balanced attribute distribution
        if len(selected_combinations) >= num_combinations:
            break

        fair = all(
            count_per_level[col][combination[i]] < (num_combinations / len(attribute_lists[i]))
            for i, col in enumerate(df_attributes.columns)
        )

        if fair:
            selected_combinations.append(combination)
```

```
            for i, col in enumerate(df_attributes.columns):
                count_per_level[col][combination[i]] += 1

    # Step 4: If we need more profiles than unique combinations, duplicate while keeping balance
    if len(selected_combinations) < num_combinations:
        extra_needed = num_combinations - len(selected_combinations)

        additional_combinations = []
        while len(additional_combinations) < extra_needed:
            sample = random.choice(selected_combinations)

            # Ensure reuse does not cause over-sampling
            fair_duplicate = all(
                count_per_level[col][val] <= min(count_per_level[col].values()) + 1
                for col, val in zip(df_attributes.columns, sample)
            )

            if fair_duplicate:
                additional_combinations.append(sample)
                for col, val in zip(df_attributes.columns, sample):
                    count_per_level[col][val] += 1

        selected_combinations.extend(additional_combinations)

    # Step 5: Convert to DataFrame
    df_combinations = pd.DataFrame(selected_combinations, columns=df_attributes.columns)

    return df_combinations
```

In [14]:
```
# Generate required combination

df_attr_1 = df_1
num_combo_1  = 75

survey_1 = generate_combinations(df_attributes = df_attr_1,
                                 num_combinations = num_combo_1)

survey_1
```

Out[14]:

	Support Options	Price
0	On-Site Support	$85
1	On-Site Support	$40
2	Dedicated Account Manager	$65
3	24/7 Phone Support	$90
4	Account Manager	$90
...	...	...
70	Standard Phone Support	$99
71	Standard Phone Support	$85
72	Livechat Support	$45
73	Custom Support Plan	$95
74	Account Manager	$95

75 rows × 2 columns

Uniform randomization

The combination preferred in conjoint survey has to be uniformly distributed in the sense of each frequency of occurrence is almost similar. While maintaining attribute level balance, some profiles are duplicated while some are

not. Hence, the `distribution threshold` is increased to 2 since the attributes exceeding the number of maximum possible profile.

In [17]:
```
# check frequency of occurrence
print(survey_1['Support Options'].value_counts(),
      '\n',
      survey_1['Price'].value_counts())
```

```
Support Options
On-Site Support              10
Account Manager              10
Standard Email Support       10
Standard Phone Support       10
Dedicated Account Manager     9
Livechat Support              9
Custom Support Plan           9
24/7 Phone Support            8
Name: count, dtype: int64
 Price
$90     10
$45     10
$95     10
 $50    10
$85      9
$40      9
$99      9
$65      8
Name: count, dtype: int64
```

The combination is perfectly distributed due to its maximum possibilities.

Validate combinations

- A combination of attribute must appears in each question equally
- An evenly distributed combination means the frequency of occurrence is equal (or at least similar. threshold = 2)

In [21]:
```
def check_attribute(df_pairs, df_attribute):
    """
    Check if the columns of a DataFrame 'df_pairs' have the same unique values
    as the corresponding columns in another DataFrame 'df_attribute'.

    Parameters
    ----------
    df_pairs : pandas DataFrame
        A DataFrame containing pairs of attributes and the corresponding prices.

    df_attribute : pandas DataFrame
        A DataFrame containing a list of unique attributes and prices.

    Returns
    -------
    result : bool
        True if the unique values of columns in 'df_pairs' are the same as those
        in 'df_attribute' for all columns, False otherwise.
    """
    # Get the column names of the DataFrame df_pairs
    cols = df_pairs.columns

    # Loop through each column of df_pairs
    for col in cols:
      if set(df_pairs[col]) == set(df_attribute[col]):
        return True
      else:
        return False
```

In [22]:
```
# check attribute
check_attribute(survey_1, df_1)
```

Out[22]:
```
True
```

In [23]:
```
def check_distribution(df_pairs, threshold=1):
    """
    Check if the distribution of unique values in the columns of a DataFrame 'df_pairs' is within a speci

    Parameters
    ----------
    df_pairs : pandas DataFrame
        A DataFrame containing pairs of attributes and their corresponding prices.

    threshold : int
        The threshold value for the range of occurrence counts of unique values.
        The default value is 1.

    Returns
    -------
    results : list of bool
        A list of boolean values where each element corresponds to a column in 'df_pairs'.
    """
    # Get the column names of the DataFrame df_pairs
    cols = df_pairs.columns

    # Initialize an empty list to store results for each column
    results = []

    # Loop through each column of df_pairs
    for col in cols:
        # Get the unique values and their occurrence counts in the current column
        values, counts = np.unique(df_pairs[col], return_counts=True)
        # Calculate the range of occurrence counts (difference between max and min counts)
        range = np.max(counts) - np.min(counts)
        # Append the result (True or False) to the results list
        results.append(range <= threshold)

    return results
```

In [24]:
```
# check distribution
check_distribution(survey_1, threshold=2)
```

Out[24]:
```
[True, True]
```

Generate Survey Question

In [26]:
```
def generateVariant(combi_data, n_question, n_choices=3):
    """
    Function to generate question variants.

    Parameters
    ----------
    combi_data : pandas DataFrame
        DataFrame containing conjoint attribute combinations.

    n_question : int
        Number of questions in the conjoint survey.

    n_choices : int, optional
        Number of options per question, default is 3.

    Returns
    -------
    varian : dict
        Dictionary containing question variants.
```

```
    """
    # Extract combination data
    total_combi, n_attribute = combi_data.shape
    required_profiles = n_question * n_choices

    # Shuffle data for randomness
    combi_data = combi_data.sample(frac=1, random_state=52).reset_index(drop=True)

    # Handle case where total_combi < required profiles
    if total_combi < required_profiles:
        extra_needed = required_profiles - total_combi
        duplicated_samples = combi_data.sample(n=extra_needed, replace=True, random_state=52)
        combi_data = pd.concat([combi_data, duplicated_samples]).reset_index(drop=True)

    # Distribute profiles evenly across questions
    combi_chunks = [combi_data.iloc[i::n_question].reset_index(drop=True) for i in range(n_question)]

    # Extract variant
    options = "ABCD"
    varian = {}

    for no, chunk in enumerate(combi_chunks):
        # reset index chunk
        chunk = chunk.reset_index(drop = True)

        # extract every combination in each question
        question_dict = {}
        for i in range(min(n_choices, len(chunk))):
            question_dict[options[i]] = chunk.loc[i].tolist()

        question_dict["D"] = ["" for i in range(n_attribute)]

        # add question_dict to varian
        varian[no+1] = question_dict

    return varian
```

In [27]:

```
# Generate questions
varian_1 = generateVariant(combi_data = survey_1,
                           n_question = 25)

varian_1
```

Out[27]:
```
{1: {'A': ['Dedicated Account Manager', ' $50'],
  'B': ['Standard Phone Support', '$45'],
  'C': ['On-Site Support', '$99'],
  'D': ['', '']},
 2: {'A': ['Dedicated Account Manager', '$65'],
  'B': ['Custom Support Plan', '$90'],
  'C': ['Custom Support Plan', '$85'],
  'D': ['', '']},
 3: {'A': ['Account Manager', '$99'],
  'B': ['Dedicated Account Manager', '$85'],
  'C': ['24/7 Phone Support', '$45'],
  'D': ['', '']},
 4: {'A': ['Livechat Support', '$45'],
  'B': ['Standard Email Support', '$85'],
  'C': ['Livechat Support', '$90'],
  'D': ['', '']},
 5: {'A': ['Standard Email Support', '$90'],
  'B': ['Standard Phone Support', '$99'],
  'C': ['Standard Phone Support', '$99'],
  'D': ['', '']},
 6: {'A': ['Livechat Support', '$85'],
  'B': ['Standard Email Support', '$45'],
  'C': ['Livechat Support', '$99'],
  'D': ['', '']},
 7: {'A': ['Standard Email Support', '$45'],
  'B': ['Livechat Support', '$40'],
  'C': ['Custom Support Plan', '$40'],
  'D': ['', '']},
 8: {'A': ['On-Site Support', '$90'],
  'B': ['On-Site Support', '$65'],
  'C': ['Standard Email Support', '$90'],
  'D': ['', '']},
 9: {'A': ['24/7 Phone Support', '$90'],
  'B': ['On-Site Support', '$90'],
  'C': ['Standard Email Support', '$40'],
  'D': ['', '']},
 10: {'A': ['Standard Phone Support', '$65'],
  'B': ['Custom Support Plan', '$65'],
  'C': ['24/7 Phone Support', '$40'],
  'D': ['', '']},
 11: {'A': ['Custom Support Plan', '$95'],
  'B': ['Account Manager', '$95'],
  'C': ['Dedicated Account Manager', '$40'],
  'D': ['', '']},
 12: {'A': ['24/7 Phone Support', '$85'],
  'B': ['Standard Phone Support', ' $50'],
  'C': ['Standard Email Support', ' $50'],
  'D': ['', '']},
 13: {'A': ['Standard Phone Support', '$40'],
  'B': ['Standard Phone Support', '$85'],
  'C': ['On-Site Support', '$95'],
  'D': ['', '']},
 14: {'A': ['Custom Support Plan', '$45'],
  'B': ['Livechat Support', '$65'],
  'C': ['Account Manager', '$40'],
  'D': ['', '']},
 15: {'A': ['Livechat Support', ' $50'],
  'B': ['Livechat Support', '$95'],
  'C': ['Account Manager', '$40'],
  'D': ['', '']},
 16: {'A': ['Account Manager', '$90'],
  'B': ['Account Manager', '$85'],
  'C': ['Standard Email Support', '$99'],
  'D': ['', '']},
 17: {'A': ['24/7 Phone Support', ' $50'],
  'B': ['On-Site Support', '$40'],
  'C': ['On-Site Support', '$45'],
  'D': ['', '']},
```

```
18: {'A': ['24/7 Phone Support', '$95'],
 'B': ['On-Site Support', '$85'],
 'C': ['Custom Support Plan', ' $50'],
 'D': ['', '']},
19: {'A': ['Dedicated Account Manager', '$99'],
 'B': ['Standard Email Support', '$95'],
 'C': ['Custom Support Plan', '$95'],
 'D': ['', '']},
20: {'A': ['Livechat Support', '$45'],
 'B': ['Standard Email Support', '$65'],
 'C': ['24/7 Phone Support', '$99'],
 'D': ['', '']},
21: {'A': ['On-Site Support', ' $50'],
 'B': ['Account Manager', ' $50'],
 'C': ['Account Manager', '$95'],
 'D': ['', '']},
22: {'A': ['Dedicated Account Manager', '$95'],
 'B': ['Custom Support Plan', '$99'],
 'C': ['On-Site Support', ' $50'],
 'D': ['', '']},
23: {'A': ['Dedicated Account Manager', '$90'],
 'B': ['Standard Phone Support', '$95'],
 'C': ['Dedicated Account Manager', '$45'],
 'D': ['', '']},
24: {'A': ['Standard Phone Support', '$90'],
 'B': ['Account Manager', '$45'],
 'C': ['Dedicated Account Manager', ' $50'],
 'D': ['', '']},
25: {'A': ['Standard Phone Support', '$85'],
 'B': ['Account Manager', '$65'],
 'C': ['24/7 Phone Support', '$65'],
 'D': ['', '']}}
```

Data rows expectation

Assuming each row corresponds to a single choice presented to a respondent in a specific question in a OHE manner (refer to Task 3 expected ouput), then this rules are binding:

- Each respondent answers 25 questions.
- Each question has 3 product/profile choices + "D" (which is empty and not counted).
- Each choice shown in a question is recorded as a separate row.

Since there are 10,000 respondents, and each answers 25 questions with 3 choices, the total number of data rows is:

$$10.000 \times 25 \times 3 = 750.000$$

`Expected rows = 750.000`

Task 2: Conjoint Design Survey

The task is to find the optimum design survey for conjoint analysis for an Edutech company to understand users' most preferences on support options and certification, with given attribute as below: | Learning Support | Price | Certification | |--------------------|---------|-------------------------| | Self-Study Material | 40 | $Verified Certificate$ | | $Forum Access$ | 45 | Accredited Certificate | | Personal Mentorship | 50 | $Professional Certificate$ | | $Live Chat Support$ | 65 | - | | - | $70 | - |

Create survey

The survey form will consist of 30 questions, each question consists of A, B, C for the choices and D for none.

In [35]:
```
# Create and display dataframe

data_2 = {
    'Learning Support': ['Self-Study Materials', 'Forum Access', 'Personal Mentorship', 'Live Chat Suppor
    'Price': ['$40', '$45', '$50', '$65', '$70'],
    'Certification': ['Verified Certificate','Accredited Certificate','Professional Certificate']
}

df_2 = pd.DataFrame.from_dict(data_2, orient='index').transpose()

df_2
```

Out[35]:

	Learning Support	Price	Certification
0	Self-Study Materials	$40	Verified Certificate
1	Forum Access	$45	Accredited Certificate
2	Personal Mentorship	$50	Professional Certificate
3	Live Chat Support	$65	None
4	None	$70	None

Generate combinations

Since the attributes have different length, the dataframe will handle it by filling it with `None`. This will change the number of unique profile in generating possible combinations, however this issue has been addressed in `generate_combination` function.

- The data to analyze has 3 attributes consist of different level:
  - `Learning Support` : 4 level
  - `Price` : 5 level
  - `Certification` : 3 level
- The survey determined to have 30 questions
- Each of question will have 3 options with 1 additional None option

Based on calculation below:

Total Unique Profile $= 4 \times 5 \times 3 = 60$
Total Required Profiles $= 3 \times 30 = 90$

The `Total Required Profile` > `Total Unique Profile` so there must be profile being reused.

In [38]:
```
df_attr_2 = df_2
num_combo_2  = 90

survey_2 = generate_combinations(df_attributes = df_attr_2,
                                 num_combinations = num_combo_2)

survey_2
```

Out[38]:

	Learning Support	Price	Certification
0	Self-Study Materials	$50	Verified Certificate
1	Personal Mentorship	$50	Professional Certificate
2	Forum Access	$70	Verified Certificate
3	Self-Study Materials	$65	Verified Certificate
4	Live Chat Support	$45	Verified Certificate
...	...	...	...
85	Live Chat Support	$45	Accredited Certificate
86	Self-Study Materials	$50	Professional Certificate
87	Forum Access	$40	Accredited Certificate
88	Self-Study Materials	$50	Accredited Certificate
89	Live Chat Support	$50	Professional Certificate

90 rows × 3 columns

Uniform randomization

The combination preferred in conjoint survey has to be uniformly distributed in the sense of each frequency of occurrence is almost similar. While maintaining attribute level balance, some profiles are duplicated while some are not. Hence, the `distribution threshold` is increased to 2 since the attributes exceeding the number of maximum possible profile.

In [41]:
```
# check frequency of occurrence
print(
    survey_2['Learning Support'].value_counts(),
    '\n',
    survey_2['Price'].value_counts(),
    '\n',
    survey_2['Certification'].value_counts()
)
```

```
Learning Support
Self-Study Materials    23
Forum Access            23
Live Chat Support       23
Personal Mentorship     21
Name: count, dtype: int64
 Price
$50    19
$40    19
$70    18
$65    17
$45    17
Name: count, dtype: int64
 Certification
Accredited Certificate      31
Professional Certificate    30
Verified Certificate        29
Name: count, dtype: int64
```

The combination is perfectly distributed due to its maximum possibilities after dropping the `None` values.

Validate combinations

- A combination of attribute must appears in each question equally

- An evenly distributed combination means the frequency of occurrence is equal (or at least similar, threshold = 2)

In [45]:
```
# check distribution
check_distribution(survey_2, threshold=2)
```

Out[45]:
```
[True, True, True]
```

Generate Survey Question

In [47]:
```
# Generate questions
varian_2 = generateVariant(combi_data = survey_2,
                           n_question = 30)

varian_2
```

Out[47]:
```
{1: {'A': ['Forum Access', '$70', 'Verified Certificate'],
  'B': ['Live Chat Support', '$65', 'Accredited Certificate'],
  'C': ['Live Chat Support', '$45', 'Professional Certificate'],
  'D': ['', '', '']},
 2: {'A': ['Personal Mentorship', '$65', 'Accredited Certificate'],
  'B': ['Self-Study Materials', '$65', 'Professional Certificate'],
  'C': ['Self-Study Materials', '$40', 'Verified Certificate'],
  'D': ['', '', '']},
 3: {'A': ['Forum Access', '$65', 'Accredited Certificate'],
  'B': ['Live Chat Support', '$40', 'Accredited Certificate'],
  'C': ['Forum Access', '$50', 'Professional Certificate'],
  'D': ['', '', '']},
 4: {'A': ['Self-Study Materials', '$70', 'Professional Certificate'],
  'B': ['Forum Access', '$40', 'Accredited Certificate'],
  'C': ['Personal Mentorship', '$45', 'Professional Certificate'],
  'D': ['', '', '']},
 5: {'A': ['Self-Study Materials', '$50', 'Verified Certificate'],
  'B': ['Live Chat Support', '$65', 'Professional Certificate'],
  'C': ['Live Chat Support', '$50', 'Professional Certificate'],
  'D': ['', '', '']},
 6: {'A': ['Self-Study Materials', '$65', 'Accredited Certificate'],
  'B': ['Live Chat Support', '$50', 'Accredited Certificate'],
  'C': ['Forum Access', '$70', 'Verified Certificate'],
  'D': ['', '', '']},
 7: {'A': ['Self-Study Materials', '$45', 'Professional Certificate'],
  'B': ['Forum Access', '$50', 'Accredited Certificate'],
  'C': ['Self-Study Materials', '$45', 'Professional Certificate'],
  'D': ['', '', '']},
 8: {'A': ['Live Chat Support', '$45', 'Verified Certificate'],
  'B': ['Self-Study Materials', '$40', 'Professional Certificate'],
  'C': ['Forum Access', '$40', 'Accredited Certificate'],
  'D': ['', '', '']},
 9: {'A': ['Live Chat Support', '$45', 'Accredited Certificate'],
  'B': ['Self-Study Materials', '$45', 'Verified Certificate'],
  'C': ['Personal Mentorship', '$70', 'Verified Certificate'],
  'D': ['', '', '']},
 10: {'A': ['Forum Access', '$65', 'Professional Certificate'],
  'B': ['Personal Mentorship', '$50', 'Verified Certificate'],
  'C': ['Live Chat Support', '$40', 'Verified Certificate'],
  'D': ['', '', '']},
 11: {'A': ['Live Chat Support', '$70', 'Verified Certificate'],
  'B': ['Self-Study Materials', '$50', 'Professional Certificate'],
  'C': ['Personal Mentorship', '$45', 'Accredited Certificate'],
  'D': ['', '', '']},
 12: {'A': ['Live Chat Support', '$40', 'Professional Certificate'],
  'B': ['Forum Access', '$40', 'Accredited Certificate'],
  'C': ['Forum Access', '$65', 'Accredited Certificate'],
  'D': ['', '', '']},
 13: {'A': ['Personal Mentorship', '$65', 'Professional Certificate'],
  'B': ['Personal Mentorship', '$50', 'Accredited Certificate'],
  'C': ['Forum Access', '$70', 'Accredited Certificate'],
  'D': ['', '', '']},
 14: {'A': ['Personal Mentorship', '$70', 'Professional Certificate'],
  'B': ['Personal Mentorship', '$70', 'Accredited Certificate'],
  'C': ['Self-Study Materials', '$50', 'Verified Certificate'],
  'D': ['', '', '']},
 15: {'A': ['Forum Access', '$45', 'Accredited Certificate'],
  'B': ['Self-Study Materials', '$70', 'Verified Certificate'],
  'C': ['Live Chat Support', '$50', 'Verified Certificate'],
  'D': ['', '', '']},
 16: {'A': ['Live Chat Support', '$40', 'Verified Certificate'],
  'B': ['Live Chat Support', '$65', 'Professional Certificate'],
  'C': ['Forum Access', '$50', 'Accredited Certificate'],
  'D': ['', '', '']},
 17: {'A': ['Forum Access', '$65', 'Verified Certificate'],
  'B': ['Live Chat Support', '$70', 'Accredited Certificate'],
  'C': ['Personal Mentorship', '$40', 'Verified Certificate'],
  'D': ['', '', '']},
```

```
18: {'A': ['Live Chat Support', '$40', 'Verified Certificate'],
 'B': ['Forum Access', '$45', 'Verified Certificate'],
 'C': ['Personal Mentorship', '$40', 'Verified Certificate'],
 'D': ['', '', '']},
19: {'A': ['Self-Study Materials', '$45', 'Accredited Certificate'],
 'B': ['Live Chat Support', '$70', 'Professional Certificate'],
 'C': ['Forum Access', '$50', 'Verified Certificate'],
 'D': ['', '', '']},
20: {'A': ['Personal Mentorship', '$40', 'Professional Certificate'],
 'B': ['Forum Access', '$45', 'Professional Certificate'],
 'C': ['Self-Study Materials', '$50', 'Accredited Certificate'],
 'D': ['', '', '']},
21: {'A': ['Self-Study Materials', '$65', 'Verified Certificate'],
 'B': ['Live Chat Support', '$40', 'Professional Certificate'],
 'C': ['Live Chat Support', '$65', 'Verified Certificate'],
 'D': ['', '', '']},
22: {'A': ['Personal Mentorship', '$70', 'Professional Certificate'],
 'B': ['Self-Study Materials', '$40', 'Accredited Certificate'],
 'C': ['Live Chat Support', '$45', 'Accredited Certificate'],
 'D': ['', '', '']},
23: {'A': ['Personal Mentorship', '$65', 'Verified Certificate'],
 'B': ['Personal Mentorship', '$65', 'Verified Certificate'],
 'C': ['Self-Study Materials', '$50', 'Accredited Certificate'],
 'D': ['', '', '']},
24: {'A': ['Personal Mentorship', '$45', 'Accredited Certificate'],
 'B': ['Personal Mentorship', '$40', 'Accredited Certificate'],
 'C': ['Self-Study Materials', '$45', 'Accredited Certificate'],
 'D': ['', '', '']},
25: {'A': ['Personal Mentorship', '$70', 'Professional Certificate'],
 'B': ['Forum Access', '$40', 'Verified Certificate'],
 'C': ['Self-Study Materials', '$70', 'Professional Certificate'],
 'D': ['', '', '']},
26: {'A': ['Forum Access', '$40', 'Professional Certificate'],
 'B': ['Self-Study Materials', '$50', 'Professional Certificate'],
 'C': ['Live Chat Support', '$50', 'Professional Certificate'],
 'D': ['', '', '']},
27: {'A': ['Forum Access', '$40', 'Professional Certificate'],
 'B': ['Personal Mentorship', '$50', 'Professional Certificate'],
 'C': ['Self-Study Materials', '$70', 'Verified Certificate'],
 'D': ['', '', '']},
28: {'A': ['Live Chat Support', '$65', 'Accredited Certificate'],
 'B': ['Self-Study Materials', '$50', 'Verified Certificate'],
 'C': ['Personal Mentorship', '$65', 'Professional Certificate'],
 'D': ['', '', '']},
29: {'A': ['Forum Access', '$45', 'Accredited Certificate'],
 'B': ['Live Chat Support', '$70', 'Verified Certificate'],
 'C': ['Forum Access', '$70', 'Professional Certificate'],
 'D': ['', '', '']},
30: {'A': ['Self-Study Materials', '$70', 'Accredited Certificate'],
 'B': ['Forum Access', '$50', 'Accredited Certificate'],
 'C': ['Personal Mentorship', '$45', 'Verified Certificate'],
 'D': ['', '', '']}}
```

Minimum target audience

Assuming each row corresponds to a single choice presented to a respondent in a specific question in a OHE manner (refer to Task 3 expected ouput), then this rules are binding:

- Each respondent answers 25 questions.
- Each question has 3 product/profile choices + "D" (which is empty and not counted).
- Each choice shown in a question is recorded as a separate row.

Since there are 20.000 data rows, and each answers 30 questions with 3 choices, the total number of audiences:

$$\frac{20.000}{30 \times 3} = 222.22 \approx 222$$

```
Expected audience = 222
```

Task 3: Data Wrangling in Conjoint

- The Pacmann AI product team wants to launch a new program.
- The product idea is simple, they want to improve some specific skills that are currently

needed by the industry.

- The problem is, that there are so many choices of skills, prices, and learning methods

that the Pacmann AI product team has to prioritize.

Skill	Bentuk Program	Harga Program
Create Analytics Dashboard	Tutorial Based	Rp 250.000
Create Machine Learning Model	Mentoring Based	Rp 300.000
Deploy Machine Learning Model		Rp 350.000
Design AB Test Experimentation		Rp 400.000
Perform Customer Lifetime Analysis		Rp 450.000
Perform Churn Analytics		Rp 500.000
Perform Credit Scoring Analytics		Rp 550.000
Perform Customer Segmentation		
Perform Price Optimization		
Design Data Pipeline		

In [53]:
```
# Create and display attribtues dataframe

data_3 = {
    'Skill': ['Create Analytics Dashboard', 'Create Machine Learning Model', 'Deploy Machine Learning Mod
              'Perform Customer Lifetime Analysis', 'Perform Churn Analytics', 'Perform Credit Scoring A
             'Perform Customer Segmentation', 'Perform Price Optimization', 'Design Data Pipeline'],
    'Bentuk Program': ['Tutorial Based', 'Mentoring Based'],
    'Harga Program': ['Rp 250.000','Rp 300.000','Rp 350.000', 'Rp 400.000', 'Rp 450.000', 'Rp 500.000', '
}

df_3 = pd.DataFrame.from_dict(data_3, orient='index').transpose()

df_3
```

Out[53]:

	Skill	Bentuk Program	Harga Program
0	Create Analytics Dashboard	Tutorial Based	Rp 250.000
1	Create Machine Learning Model	Mentoring Based	Rp 300.000
2	Deploy Machine Learning Model	None	Rp 350.000
3	Design AB Test Experimentation	None	Rp 400.000
4	Perform Customer Lifetime Analysis	None	Rp 450.000
5	Perform Churn Analytics	None	Rp 500.000
6	Perform Credit Scoring Analytics	None	Rp 550.000
7	Perform Customer Segmentation	None	None
8	Perform Price Optimization	None	None
9	Design Data Pipeline	None	None

Each user is given 10 questions, each question consist of 3 choices with additional `D. None` options. The 10 questions mock questionaire are as below.

In [55]:
```
# Create mock questionaire dictionary

varian_3 = {
    1: {'A': ['Create Analytics Dashboard', 'Tutorial Based', 'Rp 500.000'],
        'B': ['Perform Customer Segmentation', 'Mentoring Based', 'Rp 350.000'],
        'C': ['Design AB Test Experimentation', 'Mentoring Based', 'Rp 300.000'],
        'D': ['', '', '']},
    2: {'A': ['Create Analytics Dashboard', 'Tutorial Based', 'Rp 500.000'],
        'B': ['Design Data Pipeline', 'Mentoring Based', 'Rp 300.000'],
        'C': ['Perform Credit Scoring Analytics', 'Mentoring Based', 'Rp 550.000'],
        'D': ['', '', '']},
    3: {'A': ['Perform Customer Segmentation', 'Mentoring Based', 'Rp 350.000'],
        'B': ['Perform Customer Segmentation', 'Tutorial Based', 'Rp 450.000'],
        'C': ['Design Data Pipeline', 'Mentoring Based', 'Rp 250.000'],
        'D': ['', '', '']},
    4: {'A': ['Design AB Test Experimentation', 'Mentoring Based', 'Rp 500.000'],
        'B': ['Perform Price Optimization', 'Tutorial Based', 'Rp 350.000'],
        'C': ['Perform Credit Scoring Analytics', 'Mentoring Based', 'Rp 350.000'],
        'D': ['', '', '']},
    5: {'A': ['Design Data Pipeline', 'Mentoring Based', 'Rp 400.000'],
        'B': ['Perform Customer Lifetime Analysis', 'Tutorial Based', 'Rp 300.000'],
        'C': ['Design AB Test Experimentation', 'Tutorial Based', 'Rp 300.000'],
        'D': ['', '', '']},
    6: {'A': ['Perform Churn Analytics', 'Tutorial Based', 'Rp 450.000'],
        'B': ['Perform Customer Segmentation', 'Mentoring Based', 'Rp 300.000'],
        'C': ['Create Machine Learning Model', 'Mentoring Based', 'Rp 300.000'],
        'D': ['', '', '']},
    7: {'A': ['Perform Customer Lifetime Analysis', 'Tutorial Based', 'Rp 500.000'],
        'B': ['Design Data Pipeline', 'Mentoring Based', 'Rp 550.000'],
        'C': ['Deploy Machine Learning Model', 'Tutorial Based', 'Rp 350.000'],
        'D': ['', '', '']},
    8: {'A': ['Perform Credit Scoring Analytics', 'Mentoring Based', 'Rp 300.000'],
        'B': ['Design Data Pipeline', 'Mentoring Based', 'Rp 550.000'],
        'C': ['Create Machine Learning Model', 'Tutorial Based', 'Rp 550.000'],
        'D': ['', '', '']},
    9: {'A': ['Create Analytics Dashboard', 'Mentoring Based', 'Rp 250.000'],
        'B': ['Design AB Test Experimentation', 'Tutorial Based', 'Rp 550.000'],
        'C': ['Perform Customer Lifetime Analysis', 'Mentoring Based', 'Rp 350.000'],
        'D': ['', '', '']},
    10: {'A': ['Perform Credit Scoring Analytics', 'Mentoring Based', 'Rp 400.000'],
        'B': ['Perform Churn Analytics', 'Mentoring Based', 'Rp 450.000'],
        'C': ['Perform Churn Analytics', 'Tutorial Based', 'Rp 500.000'],
        'D': ['', '', '']}
            }
```

```
varian_3
```

Out[55]:
```
{1: {'A': ['Create Analytics Dashboard', 'Tutorial Based', 'Rp 500.000'],
  'B': ['Perform Customer Segmentation', 'Mentoring Based', 'Rp 350.000'],
  'C': ['Design AB Test Experimentation', 'Mentoring Based', 'Rp 300.000'],
  'D': ['', '', '']},
 2: {'A': ['Create Analytics Dashboard', 'Tutorial Based', 'Rp 500.000'],
  'B': ['Design Data Pipeline', 'Mentoring Based', 'Rp 300.000'],
  'C': ['Perform Credit Scoring Analytics', 'Mentoring Based', 'Rp 550.000'],
  'D': ['', '', '']},
 3: {'A': ['Perform Customer Segmentation', 'Mentoring Based', 'Rp 350.000'],
  'B': ['Perform Customer Segmentation', 'Tutorial Based', 'Rp 450.000'],
  'C': ['Design Data Pipeline', 'Mentoring Based', 'Rp 250.000'],
  'D': ['', '', '']},
 4: {'A': ['Design AB Test Experimentation', 'Mentoring Based', 'Rp 500.000'],
  'B': ['Perform Price Optimization', 'Tutorial Based', 'Rp 350.000'],
  'C': ['Perform Credit Scoring Analytics', 'Mentoring Based', 'Rp 350.000'],
  'D': ['', '', '']},
 5: {'A': ['Design Data Pipeline', 'Mentoring Based', 'Rp 400.000'],
  'B': ['Perform Customer Lifetime Analysis', 'Tutorial Based', 'Rp 300.000'],
  'C': ['Design AB Test Experimentation', 'Tutorial Based', 'Rp 300.000'],
  'D': ['', '', '']},
 6: {'A': ['Perform Churn Analytics', 'Tutorial Based', 'Rp 450.000'],
  'B': ['Perform Customer Segmentation', 'Mentoring Based', 'Rp 300.000'],
  'C': ['Create Machine Learning Model', 'Mentoring Based', 'Rp 300.000'],
  'D': ['', '', '']},
 7: {'A': ['Perform Customer Lifetime Analysis',
   'Tutorial Based',
   'Rp 500.000'],
  'B': ['Design Data Pipeline', 'Mentoring Based', 'Rp 550.000'],
  'C': ['Deploy Machine Learning Model', 'Tutorial Based', 'Rp 350.000'],
  'D': ['', '', '']},
 8: {'A': ['Perform Credit Scoring Analytics',
   'Mentoring Based',
   'Rp 300.000'],
  'B': ['Design Data Pipeline', 'Mentoring Based', 'Rp 550.000'],
  'C': ['Create Machine Learning Model', 'Tutorial Based', 'Rp 550.000'],
  'D': ['', '', '']},
 9: {'A': ['Create Analytics Dashboard', 'Mentoring Based', 'Rp 250.000'],
  'B': ['Design AB Test Experimentation', 'Tutorial Based', 'Rp 550.000'],
  'C': ['Perform Customer Lifetime Analysis', 'Mentoring Based', 'Rp 350.000'],
  'D': ['', '', '']},
 10: {'A': ['Perform Credit Scoring Analytics',
   'Mentoring Based',
   'Rp 400.000'],
  'B': ['Perform Churn Analytics', 'Mentoring Based', 'Rp 450.000'],
  'C': ['Perform Churn Analytics', 'Tutorial Based', 'Rp 500.000'],
  'D': ['', '', '']}}
```

Import data

Data are distributed through two channels: `Organic` and `Ads`. The final data contains a combination of organic data and ads data.

In [58]:
```python
# read dataset function
def read_data(path):
    """
    Reads a CSV file at the given path
    and returns its contents as a pandas DataFrame.

    Parameters
    ----------
    path : str
        input path

    Returns
```

```
    -------
    df : pandas Dataframe
        Sample dataframe
    """
    # Read data
    data = pd.read_csv(path)
    print('Original data shape :', data.shape)

    return data
```

In [59]:
```
# read data
df_ads_raw = read_data(path = 'conjoint_survey_ads.csv')
df_org_raw = read_data(path = 'conjoint_survey_organic.csv')
```

```
Original data shape : (229, 12)
Original data shape : (56, 12)
```

In [60]:
```
# Concatenate both dataframe
df_ads_concat = pd.concat([df_ads_raw, df_org_raw], ignore_index = True)

print('Data shape :',df_ads_concat.shape)
df_ads_concat.head()
```

```
Data shape : (285, 12)
```

Out[60]:

	Timestamp	Berapa nomer telepon anda? Nomer ini akan digunakan untuk membagikan GoPay Rp 50.000 per orang, hasil undian untuk 100 orang. Kami hanya akan mengirimkan ke pengisi kuisioner yang valid, i.e. jawaban tidak random.	1. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	2. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	3. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	4. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	5. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	6. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	7. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	8. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	r
0	2023-03-15 00:36:25	08xx8743xxx	B	B	A, C	A, B	A, B	B, C	C	C	
1	2023-03-15 00:47:43	08xx17856xxx	D. Tidak memilih semua product	A	A, C	A, B	A, C	A, B, C	D. Tidak memilih semua product	B, C	
2	2023-03-15 03:33:26	08xx15899xxx	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	
3	2023-03-15 03:49:05	08xx81429xxx	D. Tidak memilih semua product	B	B, C	A	A	A	C	C	
4	2023-03-15 06:36:28	08xx95479xxx	B, C	A, B	A, C	A, B	A	A, B, C	C	B, C	

Drop unnecessary column

`Timestamp` is currently irrelevant for the analysis. It will be dropped.

In [63]:
```
def drop_column(data, column_to_drop):
    """
    Function to drop columns

    Parameters
    ----------

    data: pandas dataframe
        sampel data

    column_to_drop: list
        Columns name to drop

    Return
    ------

    final_data: pandas dataframe
        Final sampel data
    """
    # Copy data
    final_data = data.copy()

    # Drop data
    final_data.drop(columns = column_to_drop,
                    inplace = True)

    return final_data
```

In [64]:
```
df_ads = drop_column (data = df_ads_concat,
                      column_to_drop = 'Timestamp')

df_ads
```

Out[64]:

	Berapa nomer telepon anda? Nomer ini akan digunakan untuk membagikan GoPay Rp 50.000 per orang, hasil undian untuk 100 orang. Kami hanya akan mengirimkan ke pengisi kuisioner yang valid, i.e. jawaban tidak random.	1. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	2. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	3. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	4. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	5. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	6. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	7. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	8. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	9. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)
0	08xx8743xxx	B	B	A, C	A, B	A, B	B, C	C	C	C
1	08xx17856xxx	D. Tidak memilih semua product	A	A, C	A, B	A, C	A, B, C	D. Tidak memilih semua product	B, C	A, C
2	08xx15899xxx	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product
3	08xx81429xxx	D. Tidak memilih semua product	B	B, C	A	A	A	C	C	B
4	08xx95479xxx	B, C	A, B	A, C	A, B	A	A, B, C	C	B, C	A, C
...	...	...	...	...	...	...	...	...	...	...
280	08xx25488xxx	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product
281	08xx165007xxx	D. Tidak memilih semua product	B	C	A	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product
282	08xx30266xxx	C, D. Tidak memilih semua product	B	A, C	A, B	B	D. Tidak memilih semua product	C	D. Tidak memilih semua product	A
283	08xx17271xxx	D. Tidak memilih semua product	D. Tidak memilih semua product	C	A	D. Tidak memilih semua product	C	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product
284	08xx26725xxx	D. Tidak memilih semua product	D. Tidak memilih semua product	A	D. Tidak memilih semua product	D. Tidak memilih semua product	B	D. Tidak memilih semua product	C	A, C

285 rows × 11 columns

Remove invalid data

In this questionaire, respondent can choose more than 1 answer. However this not apply for `D. Tidak memilih semua product` since the option means neither `A, B, C` are preferred. If `D` are selected along with other options, then the questionaire response is considered invalid.

In [67]:
```
# Checking any values containing 'D' but not exactly 'D. Tidak memilih semua product'
ads_validate = (df_ads.astype(str).apply(lambda x: x.str.contains('D', na=False))) & (df_ads != 'D. Tidak

# Find rows where this condition is True
ads_invalid = df_ads[ads_validate.any(axis=1)]

# Display the invalid rows
print('Total invalid response =',ads_invalid.shape[0])
print('Invalid index:', ads_invalid.index)
ads_invalid.head()
```

```
Total invalid response = 11
Invalid index: Index([12, 14, 136, 145, 158, 176, 177, 197, 230, 244, 282], dtype='int64')
```

Out[67]:

	Berapa nomer telepon anda? Nomer ini akan digunakan untuk membagikan GoPay Rp 50.000 per orang, hasil undian untuk 100 orang. Kami hanya akan mengirimkan ke pengisi kuisioner yang valid, i.e. jawaban tidak random.	1. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	2. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	3. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	4. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	5. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	6. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	7. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	8. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	9. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)
12	08xx29330xxx	D. Tidak memilih semua product	D. Tidak memilih semua product	A	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	C, D. Tidak memilih semua product	D. Tidak memilih semua product
14	08xx23015xxx	C	D. Tidak memilih semua product	C	D. Tidak memilih semua product	A	D. Tidak memilih semua product	B	C, D. Tidak memilih semua product	D. Tidak memilih semua product
136	08xx2446xxx	A, C, D. Tidak memilih semua product	A, C, D. Tidak memilih semua product	A, C, D. Tidak memilih semua product	A, C, D. Tidak memilih semua product	A, C, D. Tidak memilih semua product	A, C, D. Tidak memilih semua product	A, C, D. Tidak memilih semua product	A, C, D. Tidak memilih semua product	A, C, D. Tidak memilih semua product
145	08xx54168xxx	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	A	A	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	A, D. Tidak memilih semua product
158	08xx13567xxx	C, D. Tidak memilih semua product	D. Tidak memilih semua product	A, C	A, B	A	B	B	C	A, C

In [68]:
```
df_ads.index.duplicated().sum()
```

Out[68]: 0

In [69]:
```
df_ads.isnull().sum().sum()
```

Out[69]: 0

In [70]:
```
df_ads.duplicated().sum()
```

Out[70]: 0

Since there is no missing value nor duplicates. The one to be removed is the invalid data.

In [72]:
```
# Drop invalid index
df_ads_clean = df_ads.drop(ads_invalid.index)

# Display dataframe
print('Data shape:', df_ads_clean.shape)
df_ads_clean
```

```
Data shape: (274, 11)
```

Out[72]:

	Berapa nomer telepon anda? Nomer ini akan digunakan untuk membagikan GoPay Rp 50.000 per orang, hasil undian untuk 100 orang. Kami hanya akan mengirimkan ke pengisi kuisioner yang valid, i.e. jawaban tidak random.	1. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	2. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	3. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	4. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	5. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	6. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	7. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	8. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)	9. Produk manakah yang akan anda beli? (Anda bisa memilih membeli (klik) lebih dari 1 pilihan)
0	08xx8743xxx	B	B	A, C	A, B	A, B	B, C	C	C	C
1	08xx17856xxx	D. Tidak memilih semua product	A	A, C	A, B	A, C	A, B, C	D. Tidak memilih semua product	B, C	A, C
2	08xx15899xxx	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product
3	08xx81429xxx	D. Tidak memilih semua product	B	B, C	A	A	A	C	C	B
4	08xx95479xxx	B, C	A, B	A, C	A, B	A	A, B, C	C	B, C	A, C
...	...	...	...	...	...	...	...	...	...	...
279	08xx88828xxx	D. Tidak memilih semua product	D. Tidak memilih semua product	C	A	A	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product
280	08xx25488xxx	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product
281	08xx165007xxx	D. Tidak memilih semua product	B	C	A	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product
283	08xx17271xxx	D. Tidak memilih semua product	D. Tidak memilih semua product	C	A	D. Tidak memilih semua product	C	D. Tidak memilih semua product	D. Tidak memilih semua product	D. Tidak memilih semua product
284	08xx26725xxx	D. Tidak memilih semua product	D. Tidak memilih semua product	A	D. Tidak memilih semua product	D. Tidak memilih semua product	B	D. Tidak memilih semua product	C	A, C

274 rows × 11 columns

Generate conjoint data

Now the data is valid, the Choice-Based Conjoint (CBC) survey response dataset format will be transformed into a structured format where each question is expanded into multiple rows, with one row per choice. The structure is as follow:

Column Name	Description
Telepon	Respondent's identifier
Choice	1 = Chosen, 0 = Not Chosen
Skill	The specific skill/course options being offered
Bentuk Program	Format of the program options being offered
Harga Program	Price of the program options being offered

In [75]:
```
def get_user_answer_raw(user_data):
    """
    Function to get user answer (dirty data)

    Parameters
    ----------
    user_data: pandas Series
        Sample user data

    Returns
    -------
    user_answer_list: list
        User answer list
    """
    return list(user_data.iloc[1:-1])  # Exclude first and last row

def edit_answer(answer, remove_text=". Tidak memilih semua product"):
    """
    Function to edit the answer

    Parameters
    ----------
    answer: list
        Raw answer
    remove_text: str, optional
        Text to remove from the answer, by default ". Tidak memilih semua product"

    Returns
    -------
    edited_answer: list
        Edited answer
    """
    return str(answer).replace(remove_text, "").replace(" ", "").split(",")

def get_ohe_answer(answer, options):
    """
    Function to encode (OHE) the answer
    e.g.,:
    input: ["A"],       output: [1, 0, 0]
    input: ["A", "C"],  output: [1, 0, 1]
    input: ["D"],       output: [0, 0, 0]

    Parameters
    ----------
    answer: list
        Answer list
    options: list
```

```
        List of possible options

    Returns
    -------
    ohe_answer: list
        OHE encoded answer
    """
    return [1 if opt in answer else 0 for opt in options]

def convert_user_answer(user_answer_list, options, varian):
    """
    Function to convert user answer from choice to feature

    Parameters
    ----------
    user_answer_list: list
        User answer choices
    options: list
        List of possible options
    varian: dict
        Dictionary containing variant data

    Returns
    -------
    converted_user_answer_list: list
        Chosen feature by user
    """
    converted_user_answer_list = []

    for idx, answer in enumerate(user_answer_list):
        edited_answer = edit_answer(answer)
        ohe_answer = get_ohe_answer(edited_answer, options)

        for i, opt in enumerate(options):
            converted_answer = varian[idx + 1][opt].copy()
            converted_answer.insert(0, ohe_answer[i])
            converted_user_answer_list.append(converted_answer)

    return converted_user_answer_list

def convert_answer_to_df(user_data, user_answer, col_map):
    """
    Function to convert user answer to dataframe

    Parameters
    ----------
    user_data: pandas Series
        Sample user data
    user_answer: list
        User answer data
    column_mapping: dict
        Dictionary to map column names

    Returns
    -------
    user_answer_df: pandas DataFrame
        User answer dataframe
    """
    user_answer = np.array(user_answer)

    user_answer_df = pd.DataFrame({
        col_map['Choice']: user_answer[:, 0],
        col_map["Att_1"]: user_answer[:, 1],
        col_map["Att_2"]: user_answer[:, 2],
        col_map["Att_3"]: user_answer[:, 3]
    })

    user_answer_df[col_map["Telepon"]] = user_data[col_map["Telepon"]]
```

```
    return user_answer_df[[col_map["Telepon"], col_map["Choice"], col_map["Att_1"], col_map["Att_2"], col

def get_user_answer_clean(user_data, options, varian,col_map):
    """
    Function to get clean user answer

    Parameters
    ----------
    user_data: pandas Series
        Sample user data
    options: list
        List of possible options
    varian: dict
        Dictionary containing variant data
    col_map: dict
        Dictionary to map column names

    Returns
    -------
    clean_user_answer: pandas DataFrame
        User answer dataframe
    """
    raw_answers = get_user_answer_raw(user_data)
    cleaned_answers = convert_user_answer(raw_answers, options, varian)
    return convert_answer_to_df(user_data, cleaned_answers, col_map)

def generate_conjoint_data(raw_data, options, varian, col_map):
    """
    Function to generate conjoint data

    Parameters
    ----------
    raw_data: pandas DataFrame
        Sample raw data
    options: list
        List of possible options
    varian: dict
        Dictionary containing variant data
    col_map: dict
        Dictionary to map column names

    Returns
    -------
    conjoint_data: pandas DataFrame
        Final conjoint data
    """
    conjoint_data = pd.DataFrame()

    for user in raw_data.index:
        user_data = raw_data.loc[user]
        clean_answers = get_user_answer_clean(user_data, options, varian, col_map)
        conjoint_data = pd.concat([conjoint_data, clean_answers])

    return conjoint_data
```

In [76]:
```
options = "ABC"

col_map = {
    "Choice": "Choice",
    "Att_1" : "Skill",
    "Att_2" : "Bentuk Program",
    "Att_3" : "Harga Program",
    "Telepon": "Berapa nomer telepon anda? Nomer ini akan digunakan untuk membagikan GoPay Rp 50.000 per
}
```

In [77]:
```
conjoint_ads = generate_conjoint_data(raw_data = df_ads_clean,
                                      options = options,
```

```
                                    varian = varian_3,
                                    col_map = col_map)

conjoint_ads = conjoint_ads.rename(columns = {'Berapa nomer telepon anda? Nomer ini akan digunakan untuk

print(conjoint_ads.shape)
conjoint_ads
```

```
(7398, 5)
```

Out[77]:

	Telepon	Choice	Skill	Bentuk Program	Harga Program
0	08xx8743xxx	0	Create Analytics Dashboard	Tutorial Based	Rp 500.000
1	08xx8743xxx	1	Perform Customer Segmentation	Mentoring Based	Rp 350.000
2	08xx8743xxx	0	Design AB Test Experimentation	Mentoring Based	Rp 300.000
3	08xx8743xxx	0	Create Analytics Dashboard	Tutorial Based	Rp 500.000
4	08xx8743xxx	1	Design Data Pipeline	Mentoring Based	Rp 300.000
...	...	...	...	...	...
22	08xx26725xxx	0	Design Data Pipeline	Mentoring Based	Rp 550.000
23	08xx26725xxx	1	Create Machine Learning Model	Tutorial Based	Rp 550.000
24	08xx26725xxx	1	Create Analytics Dashboard	Mentoring Based	Rp 250.000
25	08xx26725xxx	0	Design AB Test Experimentation	Tutorial Based	Rp 550.000
26	08xx26725xxx	1	Perform Customer Lifetime Analysis	Mentoring Based	Rp 350.000

7398 rows × 5 columns

Task 4: Modeling Conjoint Analysis

Define target variable & Preprocess the data

The output target of the model is `Respondents' Choice`. The goal of the modelling task is to model customer choice. Before proceed to modelling, it is required to pre-process the data first.

In [81]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
```

In [82]:
```
class DataPreprocessor:
    """
    A class for preprocessing data, including encoding categorical features,
    standardizing numerical features, and splitting data into training and testing sets.

    Parameters
    ----------
    data_df : pandas.DataFrame
        The input dataset.
    attribute_df : pandas.DataFrame
        A dataframe containing categorical attributes for encoding.
    columns_to_drop : list, optional
        A list of column names to drop from the dataset (default is None).
    target_column : list, optional
        A list specifying the target output column (default is ['Choice']).
    """

    def __init__(self, data_df, attribute_df, columns_to_drop=None, target_column=['Choice']):
        if columns_to_drop is None:
            columns_to_drop = []

        self.data = data_df.drop(columns=columns_to_drop, errors='ignore')
```

```
        self.attribute_df = attribute_df
        self.target_column = target_column

        self._read_attribute()
        self._create_encoder()
        self._prepare()

        print(self.y_train.value_counts())

    def _read_attribute(self):
        """
        Reads and processes the attribute dataframe for encoding.
        Drops missing values to ensure only valid attributes are encoded.
        """
        self.list_attribute = self.attribute_df.fillna("Unknown")

    def _create_encoder(self):
        """
        Creates an encoder using one-hot encoding for categorical attributes.
        """
        self.ohe_enc = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
        self.ohe_enc.fit(self.list_attribute.fillna("Unknown"))

    def _split_input_output(self):
        """
        Separates the input (X) and output (y) from the dataset.

        Returns
        -------
        X : pandas.DataFrame
            The input features.
        y : pandas.Series
            The target output.
        """
        y = self.data[self.target_column]
        X = self.data.drop(self.target_column, axis=1)
        return X, y

    def _split_train_test(self, X, y, test_size=0.2, seed=123):
        """
        Splits the dataset into training and testing sets.

        Parameters
        ----------
        X : pandas.DataFrame
            The input features.
        y : pandas.Series
            The target output.
        test_size : float, optional
            The proportion of the dataset to include in the test split (default is 0.2).
        seed : int, optional
            The random seed for reproducibility (default is 123).

        Returns
        -------
        X_train : pandas.DataFrame
            The training input features.
        X_test : pandas.DataFrame
            The testing input features.
        y_train : pandas.Series
            The training target output.
        y_test : pandas.Series
            The testing target output.
        """
        return train_test_split(X, y, stratify=y, test_size=test_size, random_state=seed)

    def _transform_encoder(self, data):
        """
```

```
        Applies one-hot encoding transformation to categorical data.

        Parameters
        ----------
        data : pandas.DataFrame
            The input data containing categorical features.

        Returns
        -------
        pandas.DataFrame
            The one-hot encoded dataframe.
        """
        features = [feature for category in self.ohe_enc.categories_ for feature in category]

        data_enc = pd.DataFrame(self.ohe_enc.transform(data),
                                columns=features,
                                index=data.index)

        data_enc = data_enc.loc[:, ~data_enc.columns.str.contains('Unknown')]

        print('Data encoded shape:', data_enc.shape)

        return data_enc

    def _create_scaler(self, data):
        """
        Creates a standard scaler for numerical features.

        Parameters
        ----------
        data : pandas.DataFrame
            The input numerical data to fit the scaler.
        """
        self.scaler = StandardScaler()
        self.scaler.fit(data)

    def _transform_scaler(self, data):
        """
        Applies standardization to numerical features.

        Parameters
        ----------
        data : pandas.DataFrame
            The input numerical data to be standardized.

        Returns
        -------
        data_scaled : pandas.DataFrame
                    The standardized dataframe.
        """

        data_scaled = pd.DataFrame(self.scaler.transform(data))

        data_scaled.columns = data.columns
        data_scaled.index = data.index

        return data_scaled

    def _prepare(self):
        """
        Executes the full preprocessing pipeline.
        """
        X, y = self._split_input_output()

        X_train, X_test, y_train, y_test = self._split_train_test(X, y)

        X_train_enc = self._transform_encoder(X_train)
        X_test_enc = self._transform_encoder(X_test)
```

```
        self._create_scaler(X_train_enc)
        self.X_train_clean = self._transform_scaler(X_train_enc)
        self.X_test_clean = self._transform_scaler(X_test_enc)

        self.y_train = y_train
        self.y_test = y_test

    def get_train_test(self):
        """
        Returns the preprocessed training and testing sets.

        Returns
        -------
        tuple
            A tuple containing X_train_clean, X_test_clean, y_train, y_test.
        """

        return self.X_train_clean, self.X_test_clean, self.y_train, self.y_test
```

In [83]:
```
# Define columns to drop and target column
columns_to_drop = ["Telepon"]
target_column = ["Choice"]

# Preprocess the data
preprocessor = DataPreprocessor(data_df = conjoint_ads,
                                attribute_df = df_3,
                                columns_to_drop = columns_to_drop,
                                target_column = target_column)
X_train_clean, X_test_clean, y_train, y_test = preprocessor.get_train_test()
```

```
Data encoded shape: (5918, 19)
Data encoded shape: (1480, 19)
Choice
0         3969
1         1949
Name: count, dtype: int64
```

In [84]:
```
# Check the preprocessing result
X_train_clean.head()
```

Out[84]:

	Create Analytics Dashboard	Create Machine Learning Model	Deploy Machine Learning Model	Design AB Test Experimentation	Design Data Pipeline	Perform Churn Analytics	Perform Credit Scoring Analytics	Perform Customer Lifetime Analysis	Perform Customer Segmentation	Op
5	-0.354594	-0.278861	-0.189428	-0.422114	-0.47408	-0.197882	2.839505	-0.352174	-0.422392	
23	-0.354594	3.586012	-0.189428	-0.422114	-0.47408	-0.197882	-0.352174	-0.352174	-0.422392	
5	-0.354594	-0.278861	-0.189428	-0.422114	-0.47408	-0.197882	2.839505	-0.352174	-0.422392	
14	-0.354594	-0.278861	-0.189428	2.369027	-0.47408	-0.197882	-0.352174	-0.352174	-0.422392	
18	-0.354594	-0.278861	-0.189428	-0.422114	-0.47408	-0.197882	-0.352174	2.839505	-0.422392	

Modelling

This model is a classification task. Since it is required to have an interpretable model, the model will be used is `Logistic Regression`. The target response is rather imbalance so `f1-score` is to be considered as evaluation metric

In [87]:
```
class ModelTrainer:
    """
    A class for training machine learning models and evaluating performance.
```

```
    Parameters
    ----------
    model : object
        The machine learning model to train.
    X_train : pandas.DataFrame
        The training input features.
    X_test : pandas.DataFrame
        The testing input features.
    y_train : pandas.Series
        The training target output.
    y_test : pandas.Series
        The testing target output.
    """

    def __init__(self, model, X_train, X_test, y_train, y_test):
        self.model = model
        self.X_train = X_train
        self.X_test = X_test
        self.y_train = y_train
        self.y_test = y_test

    def fit(self):
        """
        Fits the model using the training data.
        """
        self.model.fit(self.X_train, self.y_train)

    def score(self):
        """
        Evaluates the model using F1-score.

        Prints
        ------
        F1-score for both training and testing datasets.
        """
        train_score = f1_score(self.y_train.astype(int), self.model.predict(self.X_train).astype(int))
        test_score = f1_score(self.y_test.astype(int), self.model.predict(self.X_test).astype(int))
        print('F1 score train:', train_score)
        print('F1 score test:', test_score)

    def weight_summary(self):
        summary = pd.DataFrame({'attributes': self.model.feature_names_in_.tolist() + ['constant'],
                                'weights': self.model.coef_[0].tolist() + self.model.intercept_.tolist()})
        summary = summary.sort_values(by='weights', ascending=False)

        return summary
```

In [88]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
```

In [89]:
```
# Define and train the model
model = LogisticRegression(class_weight='balanced')
trainer = ModelTrainer(model, X_train_clean, X_test_clean, y_train, y_test)

trainer.fit()
trainer.score()
trainer.weight_summary()
```

```
D:\Application\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1339: DataConversionWarning: A colu
mn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  y = column_or_1d(y, warn=True)
F1 score train: 0.4489992765854835
F1 score test: 0.44854368932038835
```

Out[89]:

	attributes	weights
12	Rp 250.000	0.230539
17	Rp 500.000	0.188729
15	Rp 400.000	0.175947
1	Create Machine Learning Model	0.145058
5	Perform Churn Analytics	0.120619
8	Perform Customer Segmentation	0.105740
9	Perform Price Optimization	0.050945
3	Design AB Test Experimentation	0.031499
10	Mentoring Based	0.031247
2	Deploy Machine Learning Model	0.020776
19	constant	-0.025377
4	Design Data Pipeline	-0.031098
11	Tutorial Based	-0.031247
18	Rp 550.000	-0.047861
7	Perform Customer Lifetime Analysis	-0.072023
13	Rp 300.000	-0.113622
14	Rp 350.000	-0.115933
6	Perform Credit Scoring Analytics	-0.127990
0	Create Analytics Dashboard	-0.153657
16	Rp 450.000	-0.166517

Coefficient Interpretation & Recommendation

The top three coefficients for each attribute indicate the choices most preferred by customers. Here are the results:

1. `Harga Program`:
   - Rp 250.000
   - Rp 500.000
   - Rp 400.000

2. `Skill` : - Create Machine Learning Model - Perform Churn Analytics - Perform Customer Segmentation

3. `Bentuk Program` : - Mentoring Based

Task 5: Find the Best Optimum Price

The additional information in the table below is the data that the company collected with 1-year trial trying to find the **optimum price to sell**. | Price |Quantity| |:----------|-------:| | Rp 250.000 | 350 | | Rp 300.000 | 321 | | Rp 350.000 | 420 | | Rp 400.000 | 230 | | Rp 450.000 | 240 | | Rp 500.000 | 220 | | Rp 550.000 | 110 |

The optimum price to sell based on these findings using the demand curve function where:

- `P` : Price
- `Q` : Quantity
- `Q(p)` : Quantity demanded at price (P)
- `a` : Intercept
- `b` : Slope

To find the price that maximizes revenue, use the derivative. The derivative is `R(p) with respect to p`. Set this derivative equal to zero to find the optimal price

The demand curve function:

$$Q(p) = a - b \cdot P$$

Calculate the slope `b` using the least squares formula:

$$b = \frac{\sum(P_i - \bar{P})(Q_i - \bar{Q})}{\sum(P_i - \bar{P})^2}$$

Calculate the intercept `a`:

$$a = \bar{Q} + b \cdot \bar{P}$$

The revenue function:

$$R = P * Q$$

$$R(p) = p \cdot Q(p) = p(a - b \cdot p)$$

Derivative revenue:

$$R'(p) = \frac{d}{dp}(ap - bp^2) = a - 2b \cdot p$$

To find the optimum price, set the `R'(p) = 0`:

$$0 = a - 2b \cdot p$$

Solving for (p):

$$p = \frac{a}{2b}$$

Since the demand function is typically written as `Q = a - bP`, we should ensure `b` is positive

In [95]:
```
# Create data list
price = [250000, 300000, 350000, 400000, 450000, 500000, 550000]
quantity = [350, 321, 420, 230, 240, 220, 110]
```

In [96]:
```
# Compute Demand Function using Least Squares
P_mean = np.mean(price)
Q_mean = np.mean(quantity)

# Compute slope (b)
b = (np.sum((price - P_mean) * (quantity - Q_mean)) / np.sum((price - P_mean) ** 2)) * -1

# Compute intercept (a)
a = (Q_mean + b * P_mean)

# Compute the optimum price
optimum_price = a / (2 * abs(b))
```

```
print(f'Demand function: Q(p) = {a:.2f} - {b:.5f} * P')
print(f'Optimum price: Rp{optimum_price:.2f}')
```

```
Demand function: Q(p) = 585.00 - 0.00079 * P
Optimum price: Rp371597.10
```