1. In the following, I will present a decision matrix designed for comparing universities, tailored to the context of Multi-Attribute Decision Making (MADM). This decision matrix will aid in evaluating the relative merits of several universities by using various attributes that are crucial for decision-making.

Note.
Let a_1 represent the **Tuition Fees (USD)**,
Let a_2 represent the **World Ranking**,
Let a_3 represent the **Average Graduate Salary (USD)**,
Let a_4 represent the **Campus Crime Rate (per 100,000)**.

Note.
Let o_1 represent **Harvard**,
Let o_2 represent **Stanford**,
Let o_3 represent **MIT**,
Let o_4 represent **UC Berkeley**.

The **Decision Matrix** is as follows:

$D_{4\times4} =$

Uni	a_1^-	a_2^-	a_3^+	a_4^-
o_1	$56,000$	3	$120,000$	290
o_2	$57,000$	2	$130,000$	250
o_3	$55,000$	1	$125,000$	310
o_4	$43,000$	15	$110,000$	350

In this document, the original decision matrix is denoted by $D_{4\times4}$. After applying the normalization methods, the resulting normalized matrix is denoted by $N_{4\times4}$, which will allow for easier comparison across the attributes.

---

1. Normalization Methods

In this section, we perform normalization on the original decision matrix $D_{4\times4}$ to create a normalized matrix $N_{4\times4}$. Each normalization method is detailed below, and the final normalized matrix for each method is presented.

1.1. **Linear Normalization.** The formula for Linear Normalization is:

$$n_{ij} = \frac{r_{ij}}{\sum_{i=1}^{m} r_{ij}}, \quad j = 1, \ldots, n, \quad i = 1, \ldots, m$$

After applying the formula, we obtain the following normalized matrix N:

$N_{4\times 4} =$

Uni	a_1^-	a_2^-	a_3^+	a_4^-
o_1	0.265	0.143	0.247	0.242
o_2	0.270	0.095	0.268	0.208
o_3	0.261	0.048	0.258	0.258
o_4	0.204	0.714	0.227	0.292

Here is corresponding python code for the discussed:

Listing 1. Code for Linear Normalization

```
# Python code for Linear Normalization Method
def linear_normalization(dm: pd.DataFrame):
    n = dm.copy().astype(float)
    for col_name in n.columns:
        n[col_name] /= n[col_name].sum()
    return n.round(3)
```

1.2. **Euclidean Normalization.** The formula for Euclidean Normalization is:

$$n_{ij} = \frac{r_{ij}}{\sqrt{\sum_{i=1}^{m} r_{ij}^2}}$$

After applying the formula, we obtain the following normalized matrix N:

$N_{4\times 4} =$

Uni	a_1^-	a_2^-	a_3^+	a_4^-
o_1	0.528	0.194	0.494	0.480
o_2	0.537	0.129	0.535	0.414
o_3	0.518	0.065	0.515	0.513
o_4	0.405	0.970	0.453	0.579

Here is corresponding python code for the discussed:

Listing 2. Code for Euclidean Normalization

```
# Python code for Euclidean Normalization Method
def euclidean_normalization(dm: pd.DataFrame):
```

```
    n = dm.copy().astype(float)
    for col_name in n.columns:
        col_sum = (n[col_name] ** 2).sum()
        n[col_name] /= math.sqrt(col_sum)
    return n.round(3)
```

1.3. **Max-Min Normalization.** This method depends on whether the attribute is beneficial (a_j^+) or non-beneficial (a_j^-):

$$\text{If } a_j^+ : \quad n_{ij} = \frac{r_{ij}}{r_j^{\max}}, \quad \text{and if } a_j^- : \quad n_{ij} = \frac{r_j^{\min}}{r_{ij}}$$

After applying the formula, we obtain the following normalized matrix N:

$N_{4\times4} =$

Uni	a_1^+	a_2^+	a_3^+	a_4^+
o_1	0.982	0.333	0.923	0.862
o_2	1.000	0.500	1.000	1.000
o_3	0.965	1.000	0.962	0.806
o_4	0.754	0.067	0.846	0.714

As you see, after applying this normalization method (which is the only one of the three), both a_j^+ and a_j^- are transformed into a_j^+.

Here is corresponding python code for the discussed:

LISTING 3. Code for Max Min Normalization

```
# Python code for Max Min Normalization Method
def max_min_normalization(dm: pd.DataFrame):
    n = dm.copy().astype(float)
    for col_name in n.columns:
        # criteria_types is defined before
        # accessing the criteria types df in order to find the type
        criteria_type = criteria_types.iloc[0, n.columns.get_loc(col_name)]
        if criteria_type == "-":
            n[col_name] = n[col_name].min() / n[col_name]
        # +
        else:
            n[col_name] /= n[col_name].max()
    return n.round(3)
```

Note.
Each method produces a unique normalized matrix N, providing a standardized basis for decision-making.