Cleaning Data

```
import pandas as pd
df = pd.read_csv('food2.csv')
df
```

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	...	Q32	Q33	Q34	Age	Grade	Gender	Level	Place	Instagram	Page
0	4	4	3	4	3	4	2	2	3	2	...	2	4	2	34	3	2	2	1	3	3
1	5	5	4	4	5	5	4	3	4	4	...	5	5	5	17	1	1	2	2	2	3
2	4	2	1	2	2	1	1	1	1	2	...	3	4	3	37	3		3	2	4	2
3	4	5	4	4	4	2	2	3	3	3	...	2	4	1	33	3	1	2	2	3	2
4	2	4	3	4	5	2	2	4	2	1	...	2	2	1	21	3	1	2	1	4	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
372	4	4	4	3	3	4	3	4	3	2	...	2	2	2	36	3	2	2	2	2	1
373	4	4	4	4	4	1	5	3	2	4	...	2	2	2	29	3	1	2	1	3	2
374	5	5	4	5	3	4	5	4	4	4	...	3	4	2	36	3	2	2	1	2	2
375	3	2	4	3	2	4	2	2	4	2	...	2	4	2	26	3	2	2	1	2	2
376	2	2	2	3	2	2	2	3	3	2	...	2	3	2	27	3	1	2	2	3	1

377 rows × 41 columns

```
#dq = df
dq = df.iloc[:, 0:34]
missing = (dq == ' ').sum(axis=1)
# Rows with at least one missing value
people_with_missing = dq[missing > 0]
missing2 = (people_with_missing == ' ').sum(axis=1)
print(missing2)
```

```
15     1
17     1
23     1
54     1
62     1
85     1
91     6
131    1
271    1
353    1
355    1
dtype: int64
```

```
same_ratio = dq.apply(
    lambda row: row.value_counts(normalize=True, dropna=True).max(),
    axis=1
)

# More than 90% of answers are identical
careless = dq[same_ratio >= 0.90]

print(careless)
```

```
Empty DataFrame
Columns: [Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13, Q14, Q15, Q16, Q17, Q18, Q19, Q20, Q21, Q22, Q23, Q24, Q25, Q2
Index: []

[0 rows x 34 columns]
```

```
# Number of unique answers in each row
unique_answers = dq.nunique(axis=1, dropna=True)

# Respondents with only one unique answer
same_answer = dq[unique_answers == 1]

print(same_answer)
```

```
Empty DataFrame
Columns: [Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13, Q14, Q15, Q16, Q17, Q18, Q19, Q20, Q21, Q22, Q23, Q24, Q25, Q2
Index: []

[0 rows x 34 columns]
```

```
print(df.duplicated().sum())
```

```
127
```

```
df = df.drop_duplicates(keep='first')
df = df.drop(index=91)
```

```
print(df.duplicated().sum())
```

```
0
```

```
df = df.apply(pd.to_numeric, errors='coerce')
```

```
constructs = {
    "Expertise": ["Q1", "Q2", "Q4", "Q5"],
    "Trust": ["Q6", "Q7"],
    "Parasocial": ["Q3", "Q8", "Q9", "Q10", "Q11", "Q12", "Q13", "Q14", "Q15"],
    "Loyalty": ["Q16", "Q17", "Q18", "Q19", "Q20", "Q21", "Q22", "Q23", "Q24"],
    "Intention": ["Q25", "Q26", "Q27", "Q28", "Q30", "Q31"],
    "Behavior": ["Q29", "Q32", "Q33", "Q34"]
}

for name, questions in constructs.items():
    print(f"{name}:")
    print(df[questions].head())
    print("-" * 40)
```

```
Expertise:
   Q1   Q2  Q4  Q5
0   4  4.0   4   3
1   5  5.0   4   5
2   4  2.0   2   2
3   4  5.0   4   4
4   2  4.0   4   5
----------------------------------------
Trust:
   Q6  Q7
0   4   2
1   5   4
2   1   1
3   2   2
4   2   2
----------------------------------------
Parasocial:
   Q3  Q8  Q9  Q10  Q11  Q12  Q13  Q14  Q15
0   3   2   3    2    4  1.0  4.0    4    4
1   4   3   4    4    4  3.0  3.0    4    4
2   1   1   1    2    2  1.0  1.0    2    4
3   4   3   3    3    4  3.0  4.0    4    4
4   3   4   2    1    2  2.0  5.0    4    4
----------------------------------------
Loyalty:
   Q16  Q17  Q18  Q19  Q20  Q21  Q22  Q23  Q24
0    4    4    4  3.0    4    4    4    4    4
1    5    4    3  5.0    5    5    4    5    4
2    2    3    1  1.0    1    1    4    2    2
3    4    4    2  3.0    4    4    3    3    1
4    2    2    1  2.0    4    4    1    2    2
----------------------------------------
Intention:
   Q25  Q26  Q27  Q28  Q30  Q31
0    4    4    2    2    4  4.0
1    5    4    2    5    3  5.0
2    3    4    2    4    4  5.0
3    3    3    2    3    3  3.0
4    2    2    2    4    2  4.0
----------------------------------------
Behavior:
   Q29  Q32  Q33  Q34
0    2    2    4  2.0
1    5    5    5  5.0
2    4    3    4  3.0
3    3    2    4  1.0
4    4    2    2  1.0
----------------------------------------
```

```
for construct, cols in constructs.items():

    if construct == "Trust":
        for col in cols:
            df[col] = df[col].fillna(df[col].mean())

    else:
        df[cols] = df[cols].T.fillna(df[cols].mean(axis=1)).T
```

```
print(df.isnull().sum()[df.isnull().sum() > 0])
```

```
Age       2
Gender    2
Place     2
dtype: int64
```

```
# Age (numerical)
df['Age'] = df['Age'].fillna(df['Age'].median())

# Gender (categorical)
df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])

# Place (categorical)
df['Place'] = df['Place'].fillna(df['Place'].mode()[0])
```

```
print(df.isnull().sum()[df.isnull().sum() > 0])
```

```
Series([], dtype: int64)
```

```
df.to_csv("output.csv", index=False)
```

Preprocessing Data

```
# Create empty columns
df['Expertise'] = None
```

```
df['Trust'] = None
df['Parasocial'] = None
df['Loyalty'] = None
df['Intention'] = None
df['Behavior'] = None

for name, questions in constructs.items():
    df[name] = df[questions].mean(axis=1)

df.head()
```

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	...	Level	Place	Instagram	Page	Expertise	Trust	Parasocial	Loyalty	Intenti
0	4.0	4.0	3.0	4.0	3.0	4	2	2.0	3.0	2.0	...	2	1.0	3	3	3.75	3.0	3.000000	3.888889	3.3333
1	5.0	5.0	4.0	4.0	5.0	5	4	3.0	4.0	4.0	...	2	2.0	2	3	4.75	4.5	3.666667	4.444444	4.0000
2	4.0	2.0	1.0	2.0	2.0	1	1	1.0	1.0	2.0	...	3	2.0	4	2	2.50	1.0	1.666667	1.888889	3.6666
3	4.0	5.0	4.0	4.0	4.0	2	2	3.0	3.0	3.0	...	2	2.0	3	2	4.25	2.0	3.555556	3.111111	2.8333
4	2.0	4.0	3.0	4.0	5.0	2	2	4.0	2.0	1.0	...	2	1.0	4	1	3.75	2.0	3.000000	2.222222	2.6666

5 rows × 47 columns

```
cons = [
    'Expertise',
    'Trust',
    'Parasocial',
    'Loyalty',
    'Intention',
    'Behavior'
]
desc = df[cons].describe().T
print(desc)


desc = pd.DataFrame({
    'Mean': df[cons].mean(),
    'Std. Dev.': df[cons].std(),
    'Min': df[cons].min(),
    'Max': df[cons].max(),
    'Skewness': df[cons].skew(),
    'Kurtosis': df[cons].kurt()
})

print(desc.round(3))
```

```
            count      mean       std       min       25%       50%       75%  \
Expertise   249.0  3.860107  0.800676  1.750000  3.250000  4.000000  4.500000
Trust       249.0  3.116466  0.986037  1.000000  2.000000  3.000000  4.000000
Parasocial  249.0  3.294288  0.728788  1.444444  2.666667  3.333333  3.888889
Loyalty     249.0  3.439145  0.885029  1.000000  2.777778  3.666667  4.222222
Intention   249.0  3.414190  0.730422  1.000000  2.833333  3.500000  4.000000
Behavior    249.0  3.187416  0.820884  1.000000  2.500000  3.000000  3.750000

            max
Expertise   5.0
Trust       5.0
Parasocial  5.0
Loyalty     5.0
Intention   5.0
Behavior    5.0
             Mean  Std. Dev.    Min  Max  Skewness  Kurtosis
Expertise   3.860      0.801  1.750  5.0    -0.586    -0.361
Trust       3.116      0.986  1.000  5.0    -0.179    -0.924
Parasocial  3.294      0.729  1.444  5.0    -0.268    -0.561
Loyalty     3.439      0.885  1.000  5.0    -0.698    -0.459
Intention   3.414      0.730  1.000  5.0    -0.497     0.170
Behavior    3.187      0.821  1.000  5.0    -0.091    -0.354
```

```
from scipy.stats import pearsonr

corr_table = pd.DataFrame(index=constructs, columns=constructs)

for c1 in cons:
    for c2 in cons:
        r, p = pearsonr(df[c1], df[c2])

        if p < 0.001:
            corr_table.loc[c1, c2] = f"{r:.3f}"
        elif p < 0.01:
            corr_table.loc[c1, c2] = f"{r:.3f}"
        elif p < 0.05:
            corr_table.loc[c1, c2] = f"{r:.3f}"
        else:
            corr_table.loc[c1, c2] = f"{r:.3f}"

print(corr_table)
```

```
            Expertise  Trust Parasocial Loyalty Intention Behavior
Expertise       1.000  0.705      0.730   0.736     0.585    0.506
Trust           0.705  1.000      0.793   0.737     0.590    0.541
Parasocial      0.730  0.793      1.000   0.841     0.675    0.605
Loyalty         0.736  0.737      0.841   1.000     0.798    0.674
Intention       0.585  0.590      0.675   0.798     1.000    0.840
Behavior        0.506  0.541      0.605   0.674     0.840    1.000
```

```
corr = pd.DataFrame(index=cons, columns=cons)
pval = pd.DataFrame(index=cons, columns=cons)

for c1 in cons:
```

```
    for c2 in cons:
        r, p = pearsonr(df[c1], df[c2])
        corr.loc[c1, c2] = round(r, 3)
        pval.loc[c1, c2] = p

print("Correlation Matrix")
print(corr)

print("\nP-values")
print(pval.astype(float).round(20))
```

```
Correlation Matrix
           Expertise  Trust Parasocial Loyalty Intention Behavior
Expertise        1.0  0.705       0.73   0.736     0.585    0.506
Trust          0.705    1.0      0.793   0.737      0.59    0.541
Parasocial      0.73  0.793        1.0   0.841     0.675    0.605
Loyalty        0.736  0.737      0.841     1.0     0.798    0.674
Intention      0.585   0.59      0.675   0.798       1.0     0.84
Behavior       0.506  0.541      0.605   0.674      0.84      1.0

P-values
              Expertise         Trust  Parasocial  Loyalty  Intention  \
Expertise   0.000000e+00  0.000000e+00         0.0      0.0        0.0
Trust       0.000000e+00  0.000000e+00         0.0      0.0        0.0
Parasocial  0.000000e+00  0.000000e+00         0.0      0.0        0.0
Loyalty     0.000000e+00  0.000000e+00         0.0      0.0        0.0
Intention   0.000000e+00  0.000000e+00         0.0      0.0        0.0
Behavior    1.437000e-17  2.000000e-20         0.0      0.0        0.0

               Behavior
Expertise   1.437000e-17
Trust       2.000000e-20
Parasocial  0.000000e+00
Loyalty     0.000000e+00
Intention   0.000000e+00
Behavior    0.000000e+00
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

constructs = [
    'Expertise',
    'Trust',
    'Parasocial',
    'Loyalty',
    'Intention',
    'Behavior'
]

corr = df[constructs].corr()

# Mask ONLY the upper triangle (keep the diagonal)
mask = np.triu(np.ones_like(corr, dtype=bool), k=1)

plt.figure(figsize=(8, 6))

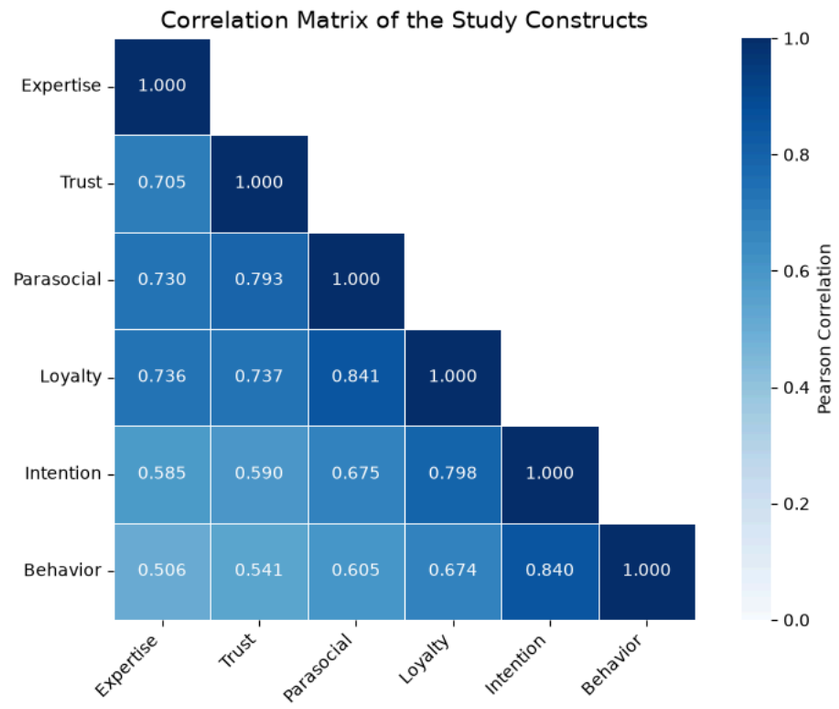
sns.heatmap(
    corr,
    mask=mask,
    annot=True,
    fmt=".3f",
    cmap="Blues",
    vmin=0,
    vmax=1,
    square=True,
    linewidths=0.5,
    cbar_kws={"label": "Pearson Correlation"},
    annot_kws={"size": 10}
)

plt.title("Correlation Matrix of the Study Constructs", fontsize=14)
plt.xticks(rotation=45, ha="right")
plt.yticks(rotation=0)
plt.tight_layout()

plt.tight_layout()

# Save the figure
plt.savefig(
    "correlation_heatmap.png",
    dpi=300,
    bbox_inches="tight"
)

plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Actual column names in your DataFrame
constructs = [
    'Expertise',
    'Trust',
    'Parasocial',
    'Loyalty',
    'Intention',
    'Behavior'
]

# Labels you want to DISPLAY on the graph
display_labels = [
    'Expertise',
    'Trust',
    'Parasocial Relationship',
    'Loyalty',
    'Purchase Intention',
    'Purchase Behavior'
]

# Calculate correlation using the actual DataFrame column names
corr = df[constructs].corr()

# Mask ONLY the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool), k=1)

plt.figure(figsize=(10, 8))

sns.heatmap(
    corr,
    mask=mask,
    annot=True,
    fmt=".3f",
    cmap="Blues",
    vmin=0,
    vmax=1,
    square=True,
    linewidths=0.5,
    cbar_kws={"label": "Pearson Correlation"},
    annot_kws={"size": 10},
    xticklabels=display_labels,
    yticklabels=display_labels
)

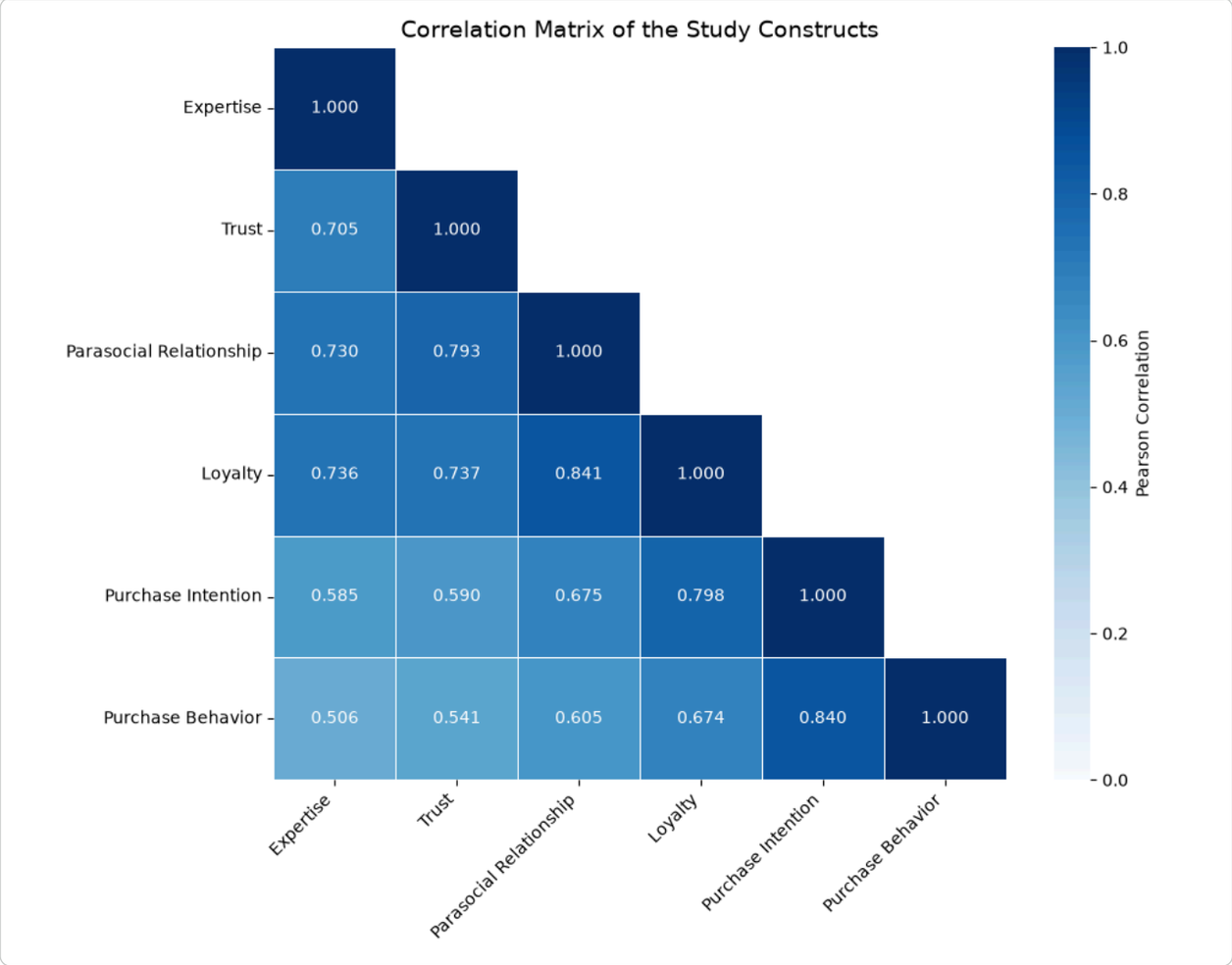
plt.title("Correlation Matrix of the Study Constructs", fontsize=14)

plt.xticks(rotation=45, ha="right")
plt.yticks(rotation=0)

plt.tight_layout()

# Save the figure
plt.savefig(
    "correlation_heatmap.png",
    dpi=300,
    bbox_inches="tight"
)
plt.savefig("figure.eps", format="eps", bbox_inches="tight")

plt.show()
```



pls-stage 1

```
from openpls import Plspm
import openpls.config as c
from openpls.mode import Mode
from openpls.scheme import Scheme

# ======================================================
# Structural Model
# ======================================================

structure = c.Structure()

# Expertise affects Parasocial and Trust
structure.add_path(["Expertise"], ["Parasocial"])
structure.add_path(["Expertise"], ["Trust"])

# Parasocial affects Trust
structure.add_path(["Parasocial"], ["Trust"])

# Trust affects Loyalty
structure.add_path(["Trust"], ["Loyalty"])

# Loyalty affects Purchase Intention
structure.add_path(["Loyalty"], ["Intention"])

# Purchase Intention affects Behavior
structure.add_path(["Intention"], ["Behavior"])


config = c.Config(structure.path(), scaled=True)

# Expertise
config.add_lv(
    "Expertise",
    Mode.A,
    c.MV("Q1"),
    c.MV("Q2"),
    c.MV("Q4"),
    c.MV("Q5"),
)

# Trust
config.add_lv(
    "Trust",
    Mode.A,
    c.MV("Q6"),
    c.MV("Q7"),
)

# Parasocial
config.add_lv(
    "Parasocial",
    Mode.A,
```

```
    c.MV("Q3"),
    c.MV("Q8"),
    c.MV("Q9"),
    c.MV("Q10"),
    c.MV("Q11"),
    c.MV("Q12"),
    c.MV("Q13"),
    c.MV("Q14"),
    c.MV("Q15"),
)

# Loyalty
config.add_lv(
    "Loyalty",
    Mode.A,
    c.MV("Q16"),
    c.MV("Q17"),
    c.MV("Q18"),
    c.MV("Q19"),
    c.MV("Q20"),
    c.MV("Q21"),
    c.MV("Q22"),
    c.MV("Q23"),
    c.MV("Q24"),
)

# Purchase Intention
config.add_lv(
    "Intention",
    Mode.A,
    c.MV("Q25"),
    c.MV("Q26"),
    c.MV("Q27"),
    c.MV("Q28"),
    c.MV("Q30"),
    c.MV("Q31"),
)

# Behavior
config.add_lv(
    "Behavior",
    Mode.A,
    c.MV("Q29"),
    c.MV("Q32"),
    c.MV("Q33"),
    c.MV("Q34"),
)


pls = Plspm(
    df,
    config,
    Scheme.PATH
)
```

```
scores = pls.scores()
print(type(scores))
print(scores.head())
```

```
<class 'pandas.DataFrame'>
   Expertise  Parasocial     Trust   Loyalty  Intention  Behavior
0  -0.123799   -0.489494 -0.149559  0.515782  -0.118951 -0.848059
1   1.100025    0.462110  1.388056  1.144461   0.720250  2.184804
2  -1.781591   -2.310830 -2.151053 -1.769147   0.303987  0.386606
3   0.521112    0.332898 -1.135708 -0.340020  -0.771052 -0.780899
4  -0.028228   -0.214422 -1.135708 -1.351656  -1.131914 -1.090314
```

```
reliability = pls.reliability()

print(reliability)
```

```
           mode  mvs  cronbach_alpha  dillon_goldstein_rho   eig_1st   eig_2nd
Expertise     A  4.0        0.846770              0.897137  2.743445  0.627306
Parasocial    A  9.0        0.875768              0.902432  4.626001  1.085243
Trust         A  2.0        0.800577              0.909330  1.667469  0.332531
Loyalty       A  9.0        0.928138              0.940156  5.726975  0.751517
Intention     A  6.0        0.800314              0.859887  3.091428  0.970232
Behavior      A  4.0        0.805593              0.873535  2.539119  0.639362
```

```
# Get Stage 1 outer model results
outer = pls.outer_model()

# Indicators used in the Stage 1 measurement model
constructs = {
    "Expertise": ["Q1", "Q2", "Q4", "Q5"],

    "Trust": ["Q6", "Q7"],

    "Parasocial": [
        "Q3", "Q8", "Q9", "Q10", "Q11",
        "Q12", "Q13", "Q14", "Q15"
    ],

    "Loyalty": [
        "Q16", "Q17", "Q18", "Q19", "Q20",
        "Q21", "Q22", "Q23", "Q24"
    ],

    "Purchase Intention": [
```

```
        "Q25", "Q26", "Q27", "Q28", "Q30", "Q31"
    ],

    "Behavior": [
        "Q29", "Q32", "Q33", "Q34"
    ]
}

results = []

for construct, indicators in constructs.items():

    # Extract outer loadings
    loadings = outer.loc[indicators, "loading"].astype(float).values

    # Composite Reliability (CR)
    CR = (np.sum(loadings) ** 2) / (
        (np.sum(loadings) ** 2) +
        np.sum(1 - loadings ** 2)
    )

    # Average Variance Extracted (AVE)
    AVE = np.mean(loadings ** 2)

    results.append({
        "Construct": construct,
        "CR": CR,
        "AVE": AVE
    })

# Convert results to DataFrame
cr_ave_stage1 = pd.DataFrame(results)


print(cr_ave_stage1)
```

```
            Construct        CR       AVE
0           Expertise  0.896594  0.684542
1               Trust  0.909116  0.833390
2          Parasocial  0.902177  0.513683
3             Loyalty  0.940116  0.636178
4  Purchase Intention  0.859157  0.514525
5            Behavior  0.873378  0.633906
```

```
Start coding or generate with AI.
```