In [1]:
```
!python3 -m pip install matplotlib seaborn pandas requests
```

```
Requirement already satisfied: matplotlib in /Library/Frameworks/Python.fr
amework/Versions/3.13/lib/python3.13/site-packages (3.10.3)
Requirement already satisfied: seaborn in /Library/Frameworks/Python.frame
work/Versions/3.13/lib/python3.13/site-packages (0.13.2)
Requirement already satisfied: pandas in /Library/Frameworks/Python.framew
ork/Versions/3.13/lib/python3.13/site-packages (2.3.0)
Requirement already satisfied: requests in /Library/Frameworks/Python.fram
ework/Versions/3.13/lib/python3.13/site-packages (2.32.4)
Requirement already satisfied: contourpy>=1.0.1 in /Library/Frameworks/Pyt
hon.framework/Versions/3.13/lib/python3.13/site-packages (from matplotlib)
(1.3.2)
Requirement already satisfied: cycler>=0.10 in /Library/Frameworks/Python.
framework/Versions/3.13/lib/python3.13/site-packages (from matplotlib) (0.
12.1)
Requirement already satisfied: fonttools>=4.22.0 in /Library/Frameworks/Py
thon.framework/Versions/3.13/lib/python3.13/site-packages (from matplotli
b) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in /Library/Frameworks/Py
thon.framework/Versions/3.13/lib/python3.13/site-packages (from matplotli
b) (1.4.8)
Requirement already satisfied: numpy>=1.23 in /Library/Frameworks/Python.f
ramework/Versions/3.13/lib/python3.13/site-packages (from matplotlib) (2.
3.0)
Requirement already satisfied: packaging>=20.0 in /Library/Frameworks/Pyth
on.framework/Versions/3.13/lib/python3.13/site-packages (from matplotlib)
(25.0)
Requirement already satisfied: pillow>=8 in /Library/Frameworks/Python.fra
mework/Versions/3.13/lib/python3.13/site-packages (from matplotlib) (11.2.
1)
Requirement already satisfied: pyparsing>=2.3.1 in /Library/Frameworks/Pyt
hon.framework/Versions/3.13/lib/python3.13/site-packages (from matplotlib)
(3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /Library/Framework
s/Python.framework/Versions/3.13/lib/python3.13/site-packages (from matplo
tlib) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /Library/Frameworks/Python.
framework/Versions/3.13/lib/python3.13/site-packages (from pandas) (2025.
2)
Requirement already satisfied: tzdata>=2022.7 in /Library/Frameworks/Pytho
n.framework/Versions/3.13/lib/python3.13/site-packages (from pandas) (202
5.2)
Requirement already satisfied: charset_normalizer<4,>=2 in /Library/Framew
orks/Python.framework/Versions/3.13/lib/python3.13/site-packages (from req
uests) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /Library/Frameworks/Python.
framework/Versions/3.13/lib/python3.13/site-packages (from requests) (3.1
0)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Library/Frameworks/P
ython.framework/Versions/3.13/lib/python3.13/site-packages (from requests)
(2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /Library/Frameworks/P
ython.framework/Versions/3.13/lib/python3.13/site-packages (from requests)
(2025.4.26)
Requirement already satisfied: six>=1.5 in /Library/Frameworks/Python.fram
ework/Versions/3.13/lib/python3.13/site-packages (from python-dateutil>=2.
7->matplotlib) (1.17.0)
```

In [2]:
```
# 1. Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.stats.outliers_influence import variance_inflation_facto
```

In [3]:
```
# Display settings
pd.set_option('display.max_columns', None)
sns.set(style='whitegrid')
```

In [4]:
```
# Load personality data
personality_df = pd.read_csv("/Users/admin/Downloads/personality.csv")

# Load assets data
assets_df = pd.read_csv("/Users/admin/Downloads/assets.csv")
```

In [5]:
```
#Basic diagnostics
print("Personality:", personality_df.shape)
print("Assets:", assets_df.shape)
print(personality_df.columns)
print(assets_df.columns)


# See first few rows
print(personality_df.head())
print(assets_df.head())
```

```
Personality: (297, 6)
Assets: (786, 6)
Index(['_id', 'confidence', 'risk_tolerance', 'composure', 'impulsivity',
       'impact_desire'],
      dtype='object')
Index(['_id', 'asset_allocation', 'asset_allocation_id', 'asset_currency',
       'asset_value', 'created'],
      dtype='object')
   _id  confidence  risk_tolerance  composure  impulsivity  impact_desire
0    1       0.550           0.510      0.565        0.161          0.999
1    2       0.486           0.474      0.439        0.818          0.048
2    3       0.565           0.568      0.578        0.832          0.977
3    4       0.652           0.625      0.642        0.507          0.407
4    5       0.477           0.483      0.515        0.006          0.871
   _id asset_allocation  asset_allocation_id asset_currency  asset_value  \
0    1         Equities             39958838            USD       217.06
1    1      Commodities             83197857            GBP       159.05
2    2             Cash             22575562            USD       231.12
3    2             Cash             85329037            USD       321.75
4    3           Crypto             66306997            USD       181.15

                            created
0  2025-02-25T09:18:34.158728+00:00
1  2025-05-18T09:18:34.162165+00:00
2  2025-03-06T09:18:34.162165+00:00
3  2025-02-22T09:18:34.163356+00:00
4  2025-04-17T09:18:34.163356+00:00
```

In [6]:
```
#  Filter for GBP-denominated assets only
gbp_assets = assets_df[assets_df['asset_currency'] == 'GBP']

# Group by _id and sum asset_value
gbp_totals = gbp_assets.groupby('_id')['asset_value'].sum().reset_index()
gbp_totals.rename(columns={'asset_value': 'total_gbp_assets'}, inplace=Tr

# Merge with personality traits
merged_df = pd.merge(personality_df, gbp_totals, on='_id', how='left')

# Replace NaN (people with no GBP assets) with 0
merged_df['total_gbp_assets'].fillna(0, inplace=True)

# Check top people
merged_df.sort_values(by='total_gbp_assets', ascending=False).head()
```

```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/60907630
7.py:12: FutureWarning: A value is trying to be set on a copy of a DataFra
me or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never wor
k because the intermediate object on which we are setting values always be
haves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  merged_df['total_gbp_assets'].fillna(0, inplace=True)
```

Out[6]:

	_id	confidence	risk_tolerance	composure	impulsivity	impact_desire	total
131	134	0.547	0.555	0.417	0.105	0.079	
69	72	0.654	0.623	0.437	0.369	0.122	
148	152	0.535	0.553	0.433	0.644	0.669	
164	168	0.690	0.656	0.682	0.719	0.348	
202	206	0.406	0.420	0.444	0.238	0.817	

In [7]:
```
df = pd.merge(personality_df, assets_df, on="_id")
```

In [8]:
```
print(df.shape)
df.head(785)

df.describe()
```

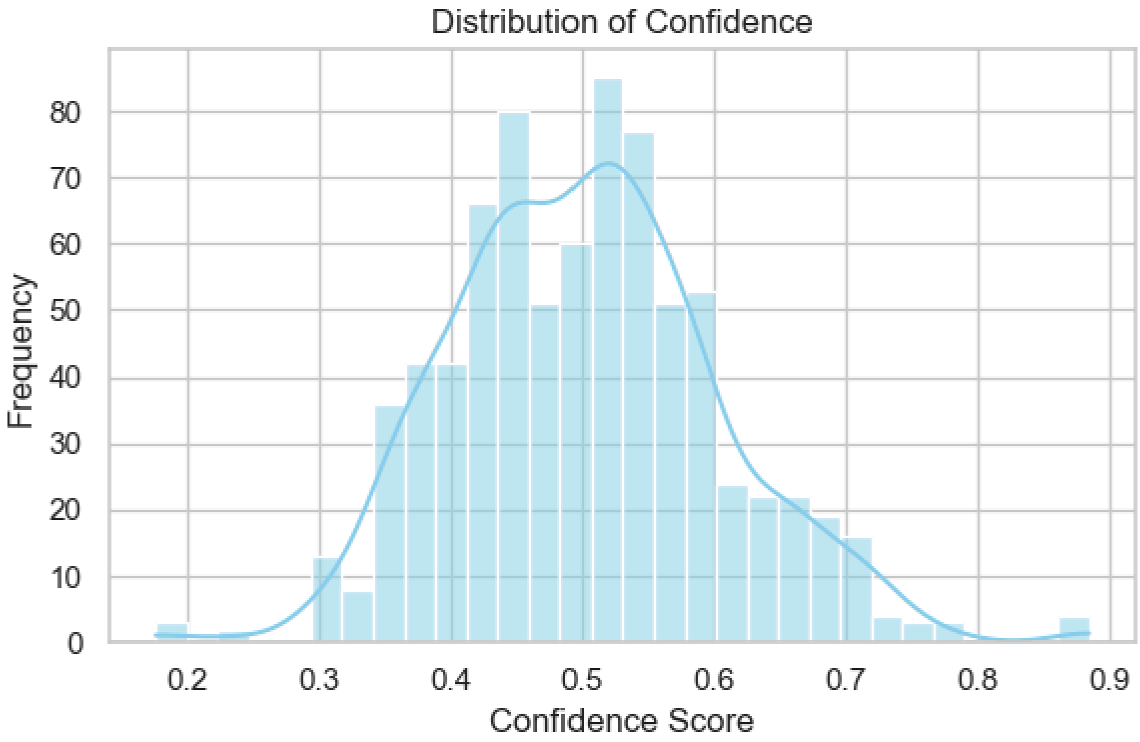
```
(786, 11)
```

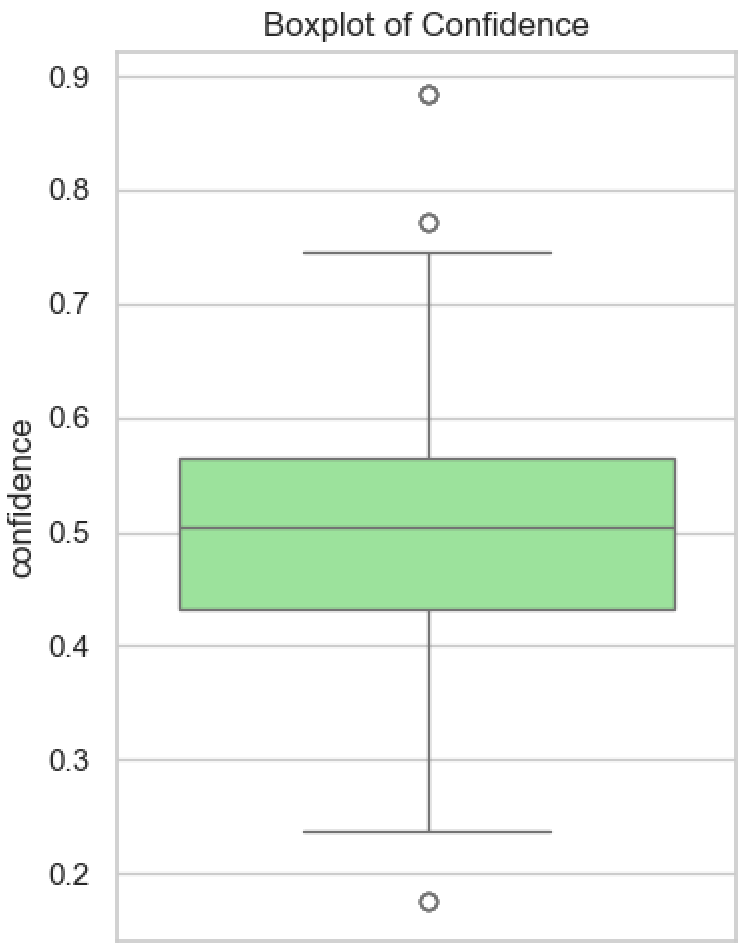
Out[8]:

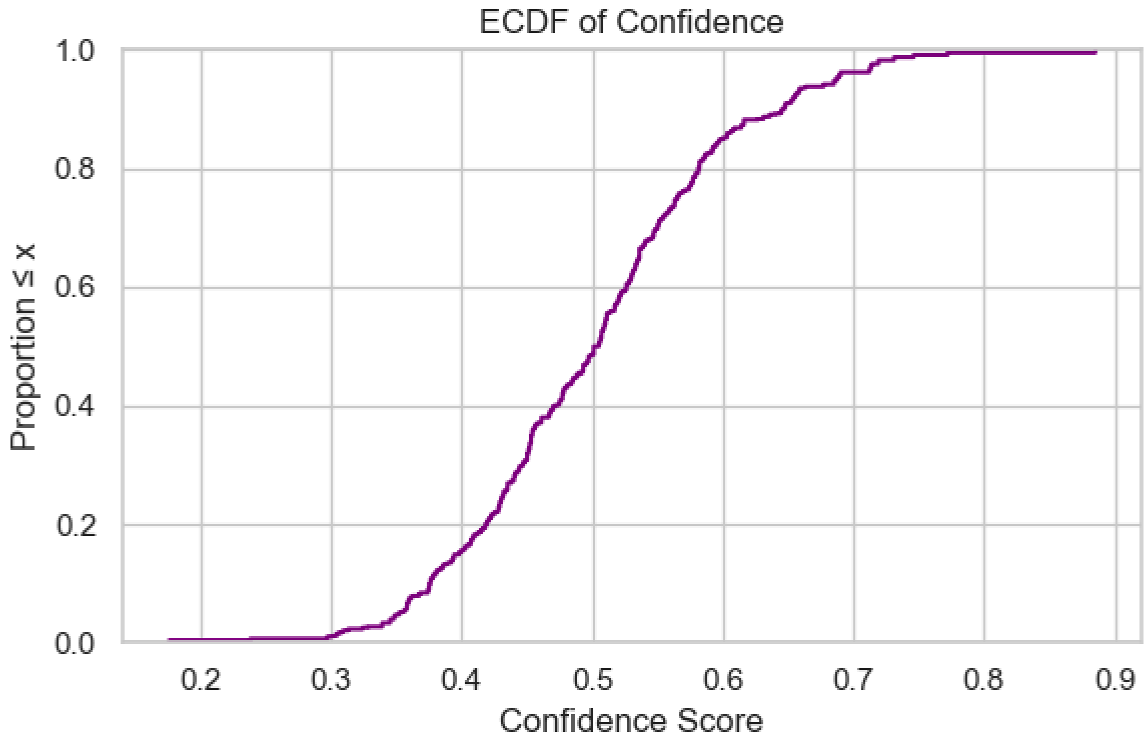
	_id	confidence	risk_tolerance	composure	impulsivity	impact_d
count	786.000000	786.000000	786.000000	786.000000	786.000000	786.00
mean	150.575064	0.503015	0.501001	0.504209	0.500894	0.48
std	87.287256	0.102595	0.077551	0.071991	0.294382	0.28
min	1.000000	0.176000	0.299000	0.311000	0.005000	0.00
25%	76.000000	0.432000	0.449000	0.455000	0.228000	0.23
50%	148.500000	0.505000	0.500000	0.502000	0.507000	0.49
75%	228.000000	0.565000	0.550750	0.547000	0.723500	0.71
max	300.000000	0.885000	0.745000	0.700000	0.997000	0.99

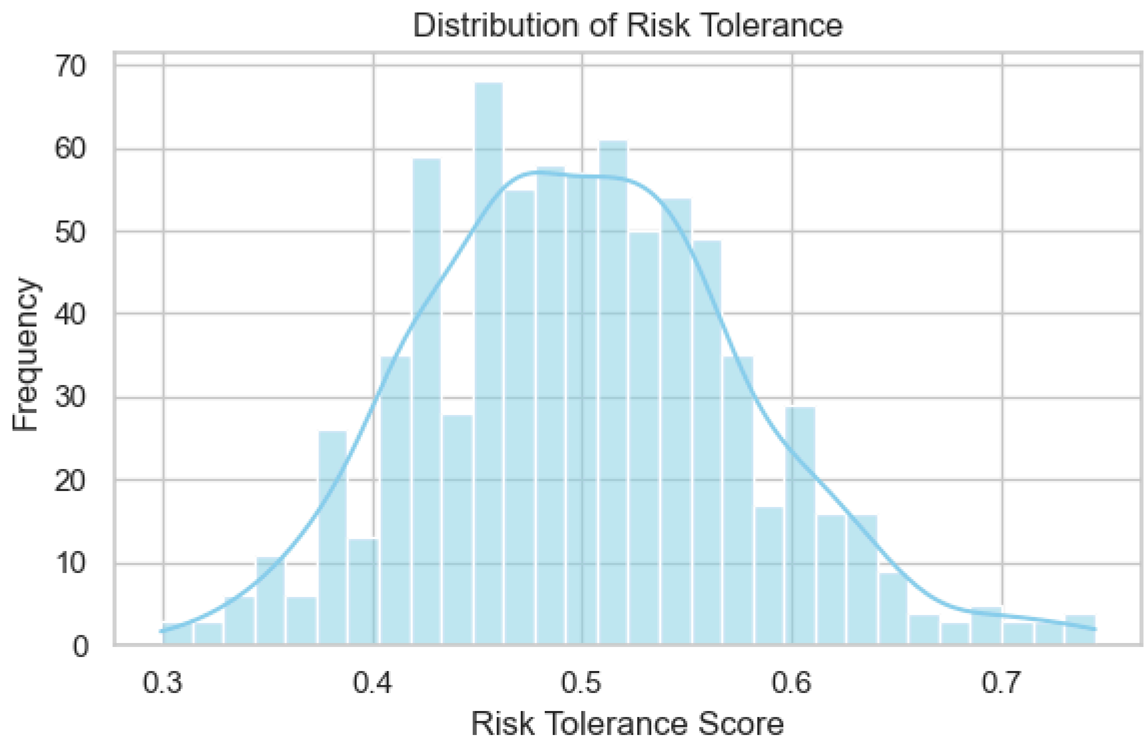
In [9]:

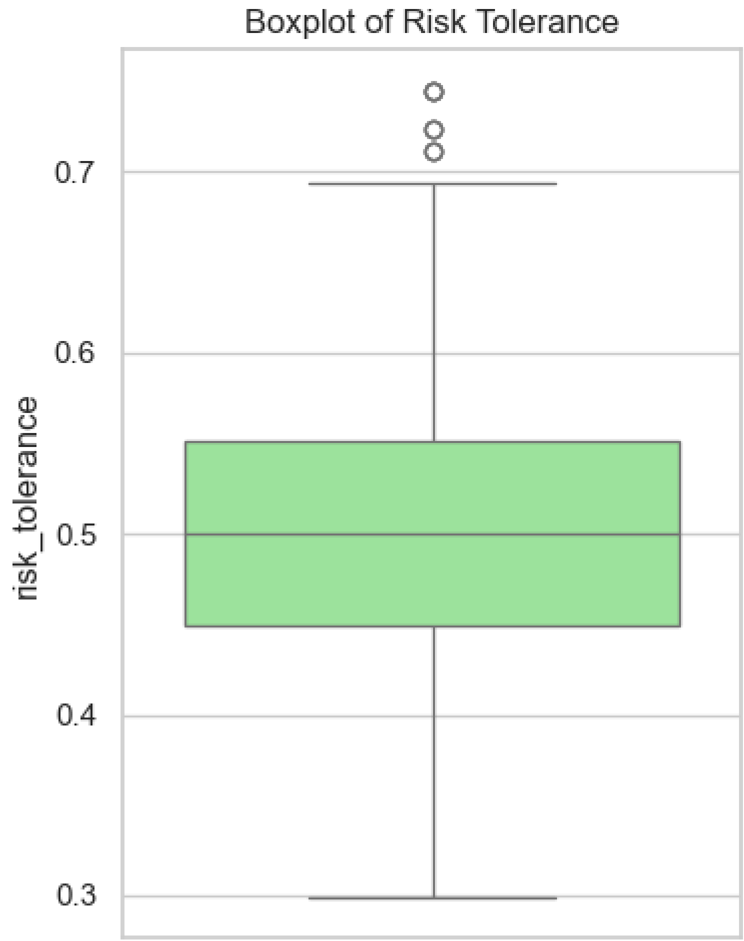
```
#Visual inspection of key personality traits
traits = ['confidence', 'risk_tolerance', 'composure', 'impulsivity', 'im
for trait in traits:
    # Histogram
    plt.figure(figsize=(6, 4))
    sns.histplot(df[trait], kde=True, bins=30, color='skyblue')
    plt.title(f'Distribution of {trait.replace("_", " ").title()}')
    plt.xlabel(f'{trait.replace("_", " ").title()} Score')
    plt.ylabel('Frequency')
    plt.tight_layout()
    plt.show()

    # Boxplot
    plt.figure(figsize=(4, 5))
    sns.boxplot(y=df[trait], color='lightgreen')
    plt.title(f'Boxplot of {trait.replace("_", " ").title()}')
    plt.tight_layout()
    plt.show()

    # ECDF
    plt.figure(figsize=(6, 4))
    sns.ecdfplot(df[trait], color='purple')
    plt.title(f'ECDF of {trait.replace("_", " ").title()}')
    plt.xlabel(f'{trait.replace("_", " ").title()} Score')
    plt.ylabel('Proportion ≤ x')
    plt.tight_layout()
    plt.show()
```

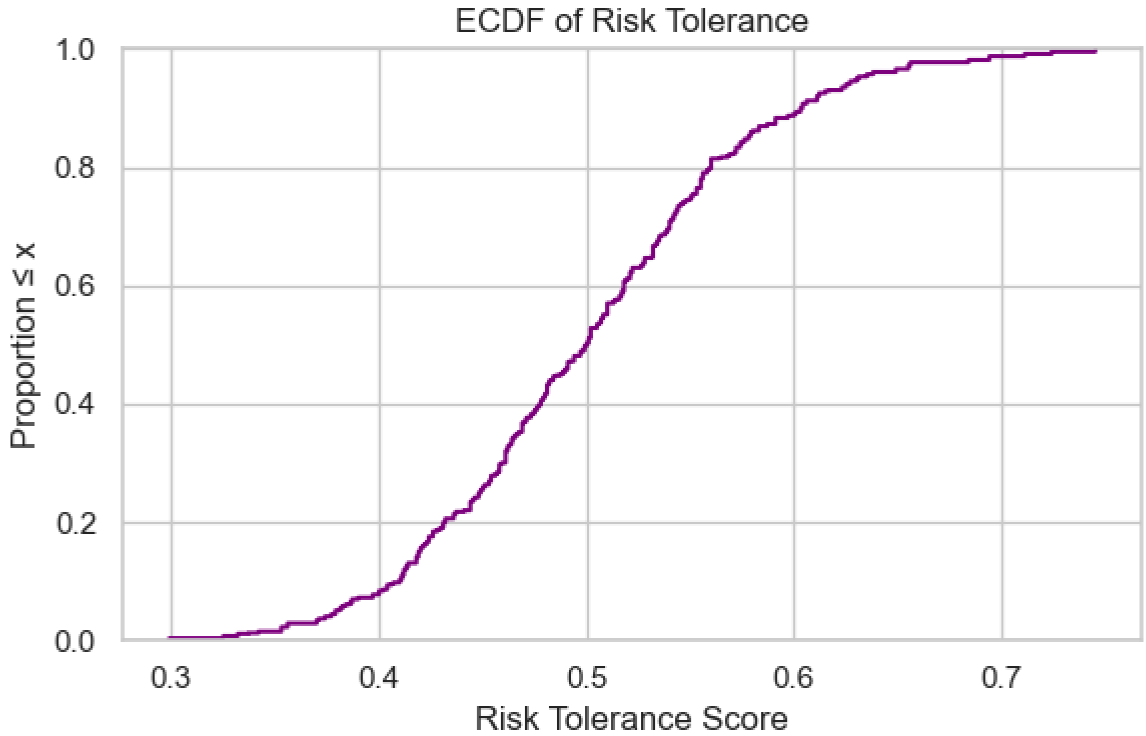

Distribution of Confidence
Frequency
0
10
20
30
40
50
60
70
80
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Confidence Score

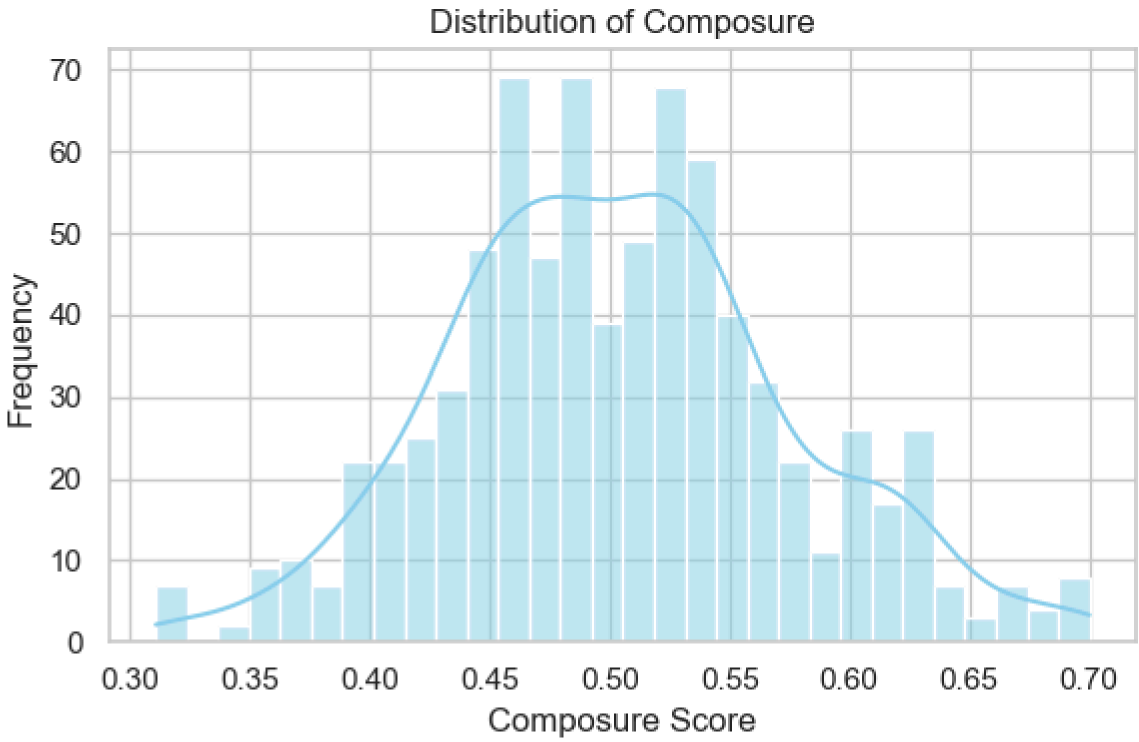

Boxplot of Confidence
confidence
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2

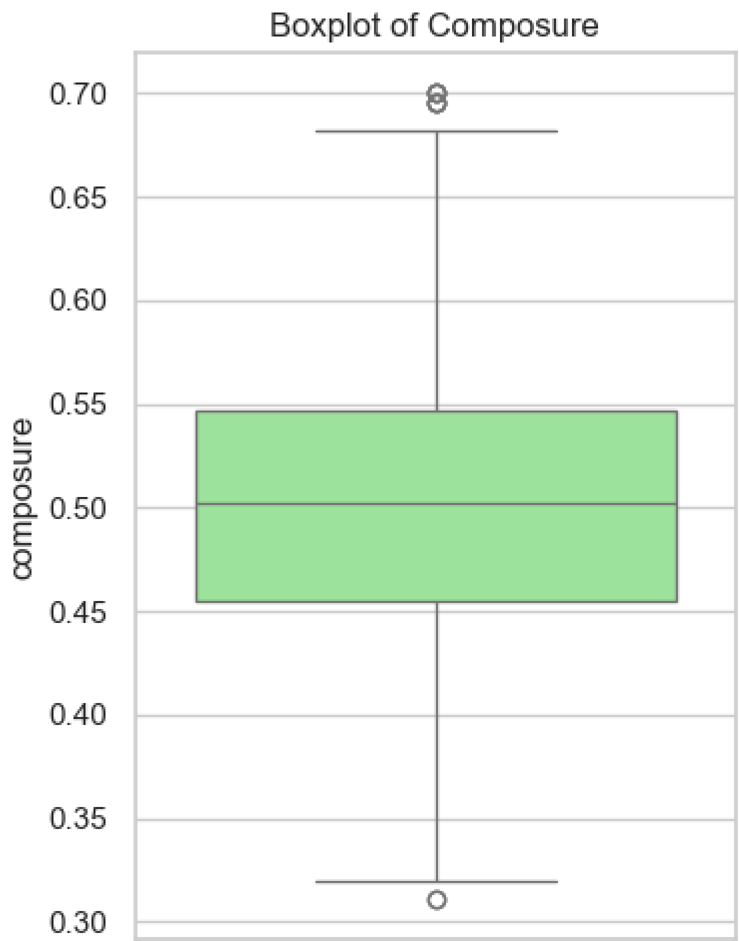

ECDF of Confidence
Proportion ≤ x
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Confidence Score

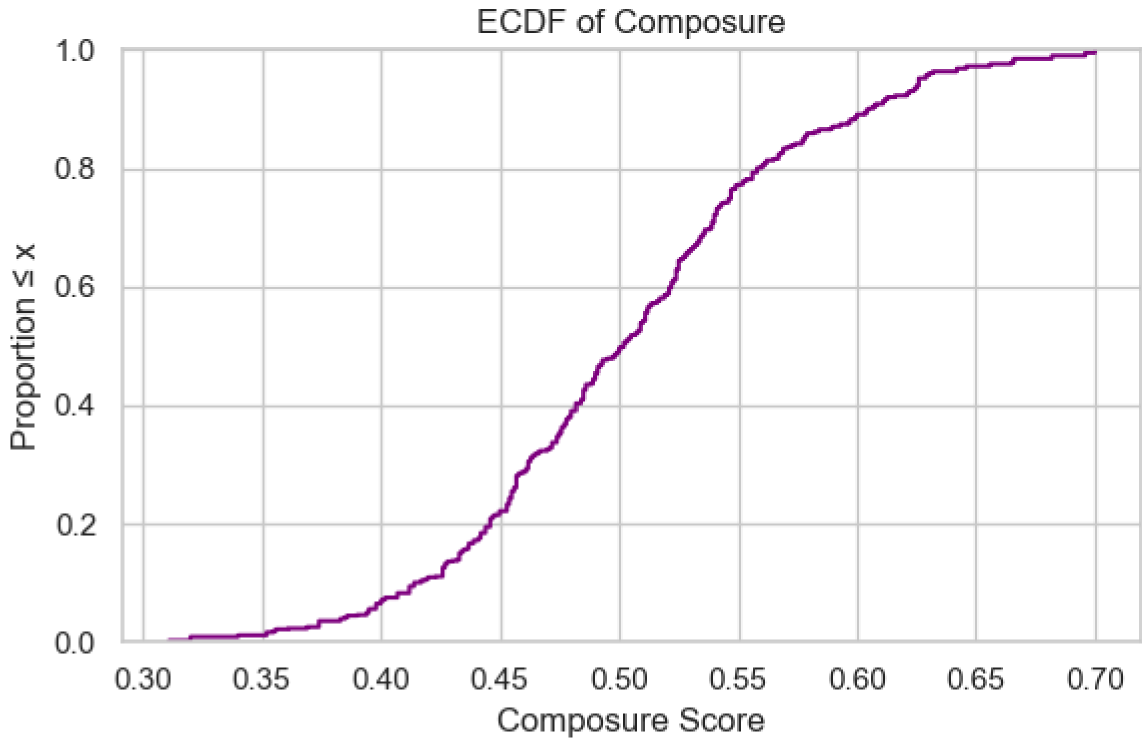

Distribution of Risk Tolerance
Frequency
70
60
50
40
30
20
10
0
0.3
0.4
0.5
0.6
0.7
Risk Tolerance Score

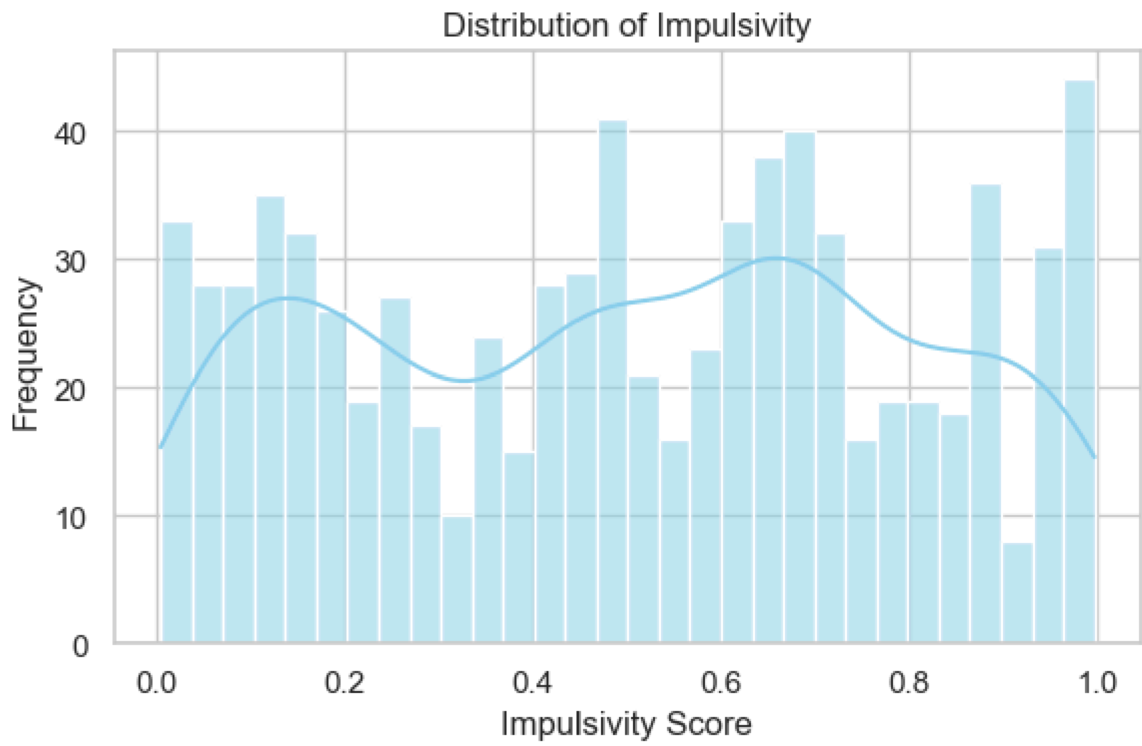

Boxplot of Risk Tolerance
risk_tolerance
0.3
0.4
0.5
0.6
0.7

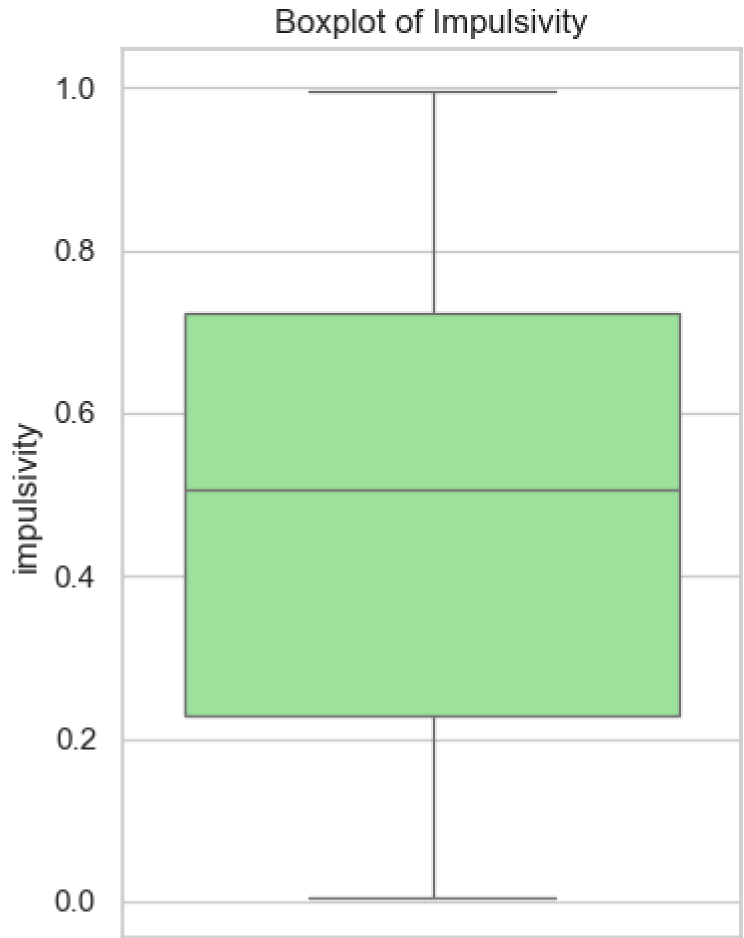

ECDF of Risk Tolerance
Proportion ≤ x
Risk Tolerance Score
0.0
0.2
0.4
0.6
0.8
1.0
0.3
0.4
0.5
0.6
0.7

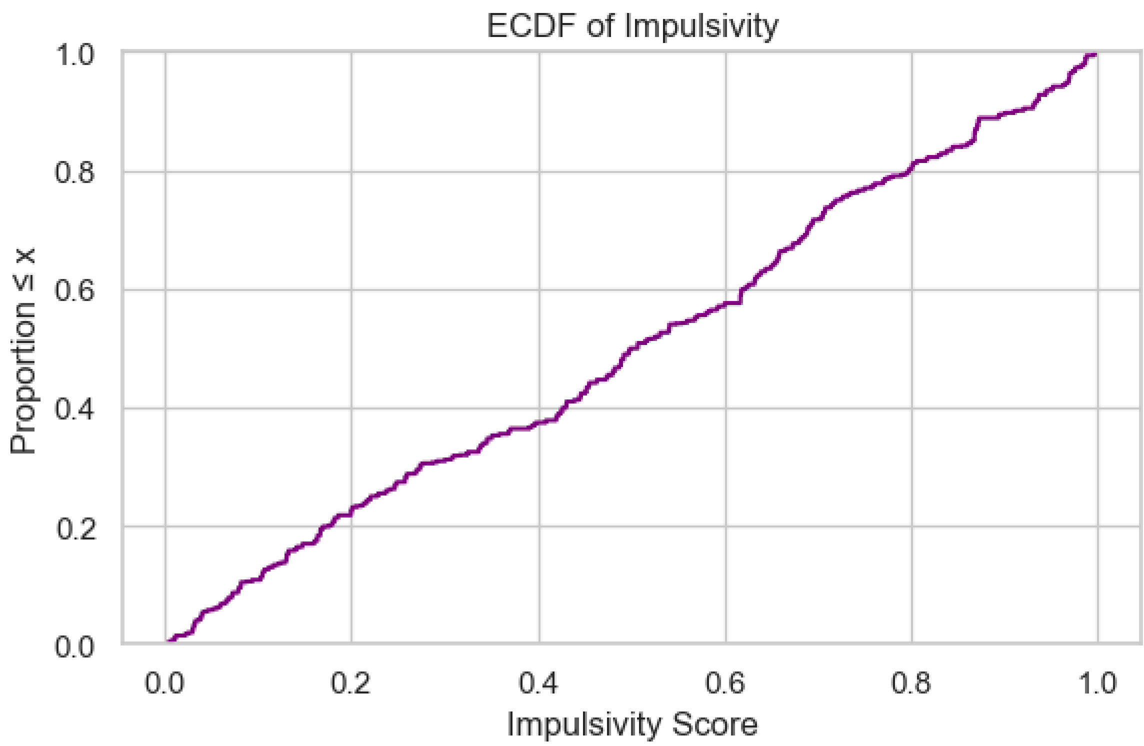

Distribution of Composure
Frequency
0
10
20
30
40
50
60
70
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65
0.70
Composure Score

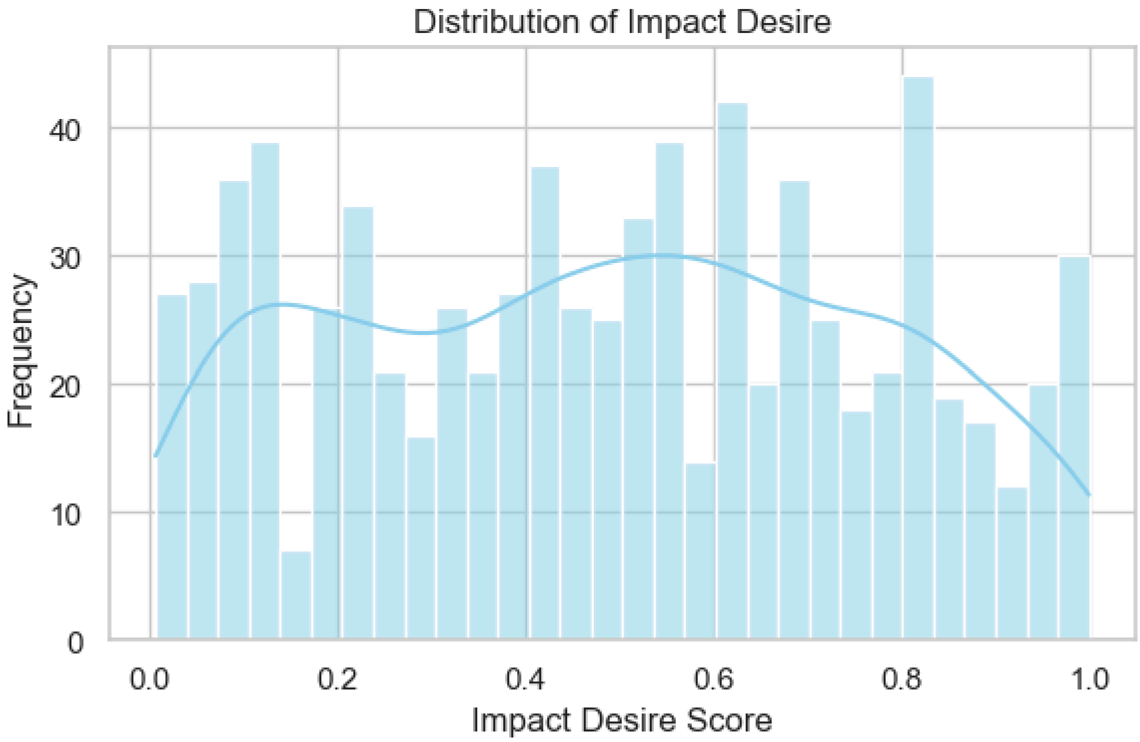

Boxplot of Composure
composure
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65
0.70

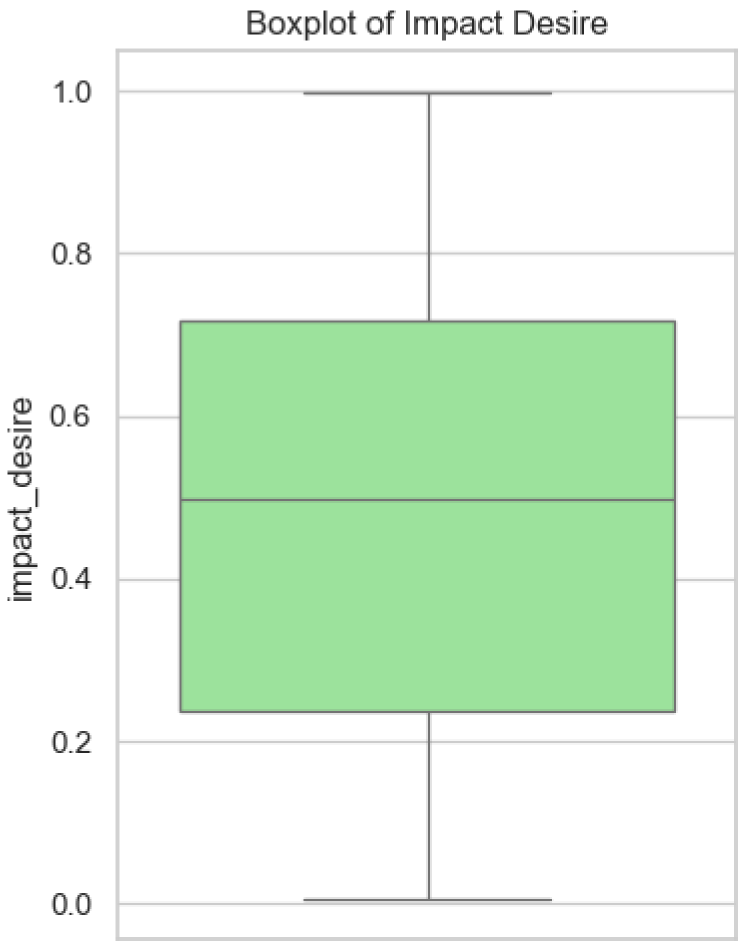

ECDF of Composure
Proportion ≤ x
1.0
0.8
0.6
0.4
0.2
0.0
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65
0.70
Composure Score

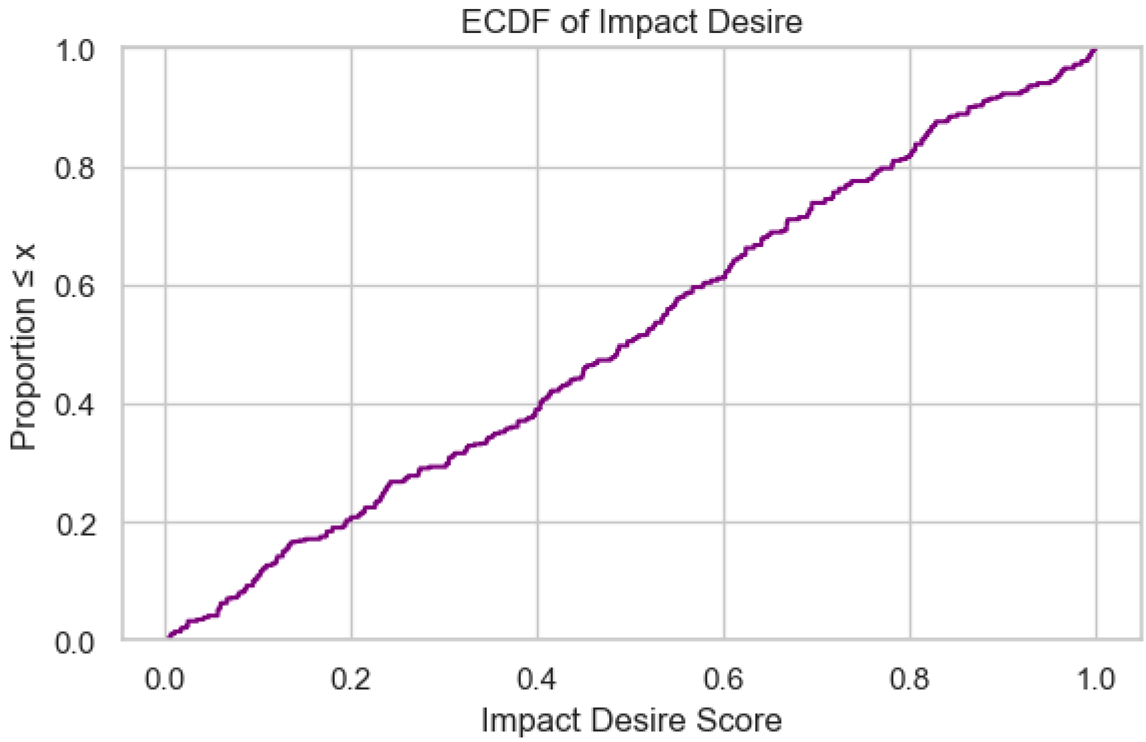

Distribution of Impulsivity
Frequency
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
Impulsivity Score


Boxplot of Impulsivity
1.0
0.8
0.6
0.4
0.2
0.0
impulsivity


ECDF of Impulsivity
1.0
0.8
0.6
0.4
0.2
0.0
Proportion ≤ x
0.0
0.2
0.4
0.6
0.8
1.0
Impulsivity Score


Distribution of Impact Desire
Frequency
0
10
20
30
40
0.0
0.2
0.4
0.6
0.8
1.0
Impact Desire Score


Boxplot of Impact Desire
impact_desire
0.0
0.2
0.4
0.6
0.8
1.0



In [10]:
```
# Trait Summary & Behavioural Interpretation:
# - Confidence: Approximately normal with a centered IQR; few mild outlie
# - Risk Tolerance: Broad spread with mild outliers and wider IQR; captur
# - Composure: Slight right-skew; tight IQR with high median. Indicates s
# - Impulsivity: Most variable trait with a long right tail and many high
# - Impact Desire: Left-skewed with high median and dense upper distribut
```

In [11]:
```
#Quantile-based segmentation into tertiles for all traits
for trait in traits:
    segment_label = f'{trait}_tertile'
    df[segment_label] = pd.qcut(df[trait], q=3, labels=['Low', 'Medium',
    print(f"\n--- {trait.title()} Tertile Counts ---")
    print(df[segment_label].value_counts())
```

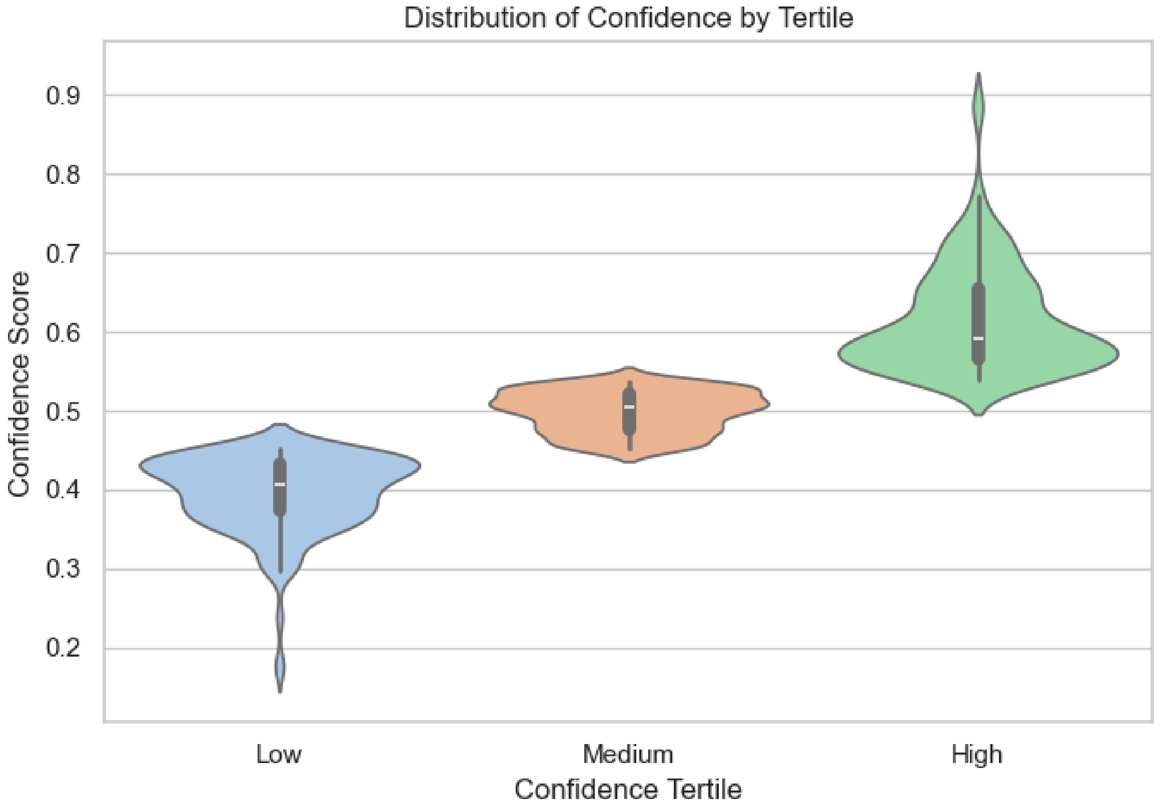
```
--- Confidence Tertile Counts ---
confidence_tertile
Low       265
High      262
Medium    259
Name: count, dtype: int64

--- Risk_Tolerance Tertile Counts ---
risk_tolerance_tertile
Low       268
High      262
Medium    256
Name: count, dtype: int64

--- Composure Tertile Counts ---
composure_tertile
Low       265
High      262
Medium    259
Name: count, dtype: int64

--- Impulsivity Tertile Counts ---
impulsivity_tertile
Low       264
Medium    261
High      261
Name: count, dtype: int64

--- Impact_Desire Tertile Counts ---
impact_desire_tertile
Low       265
High      262
Medium    259
Name: count, dtype: int64
```

In [12]:

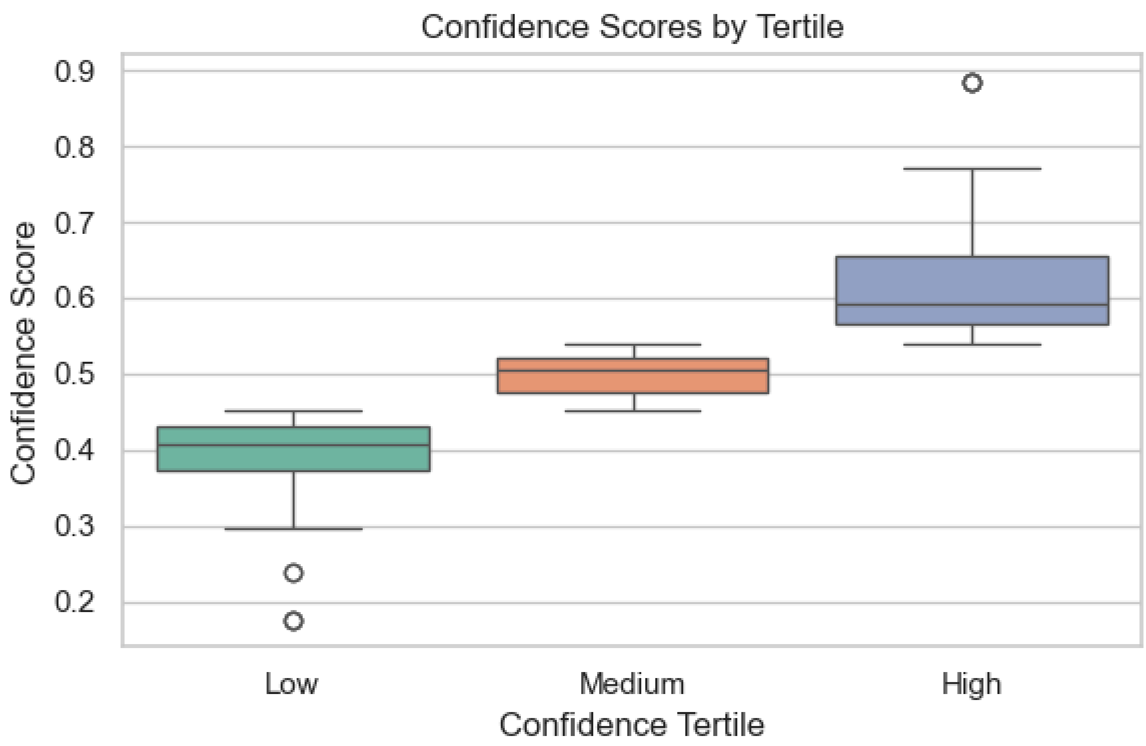
```
# Visual inspection of trait distributions across segments
for trait in traits:
    segment_col = f'{trait}_tertile'

    # Violin plot
    plt.figure(figsize=(7, 5))
    sns.violinplot(data=df, x=segment_col, y=trait, palette='pastel', ord
    plt.title(f'Distribution of {trait.replace("_", " ").title()} by Tert
    plt.xlabel(f'{trait.replace("_", " ").title()} Tertile')
    plt.ylabel(f'{trait.replace("_", " ").title()} Score')
    plt.tight_layout()
    plt.show()

    # Boxplot by tertile
    plt.figure(figsize=(6, 4))
    sns.boxplot(data=df, x=segment_col, y=trait, palette='Set2', order=['
    plt.title(f'{trait.replace("_", " ").title()} Scores by Tertile')
    plt.xlabel(f'{trait.replace("_", " ").title()} Tertile')
    plt.ylabel(f'{trait.replace("_", " ").title()} Score')
    plt.tight_layout()
    plt.show()

    # ECDF by tertile
    plt.figure(figsize=(6, 4))
    for level in ['Low', 'Medium', 'High']:
```

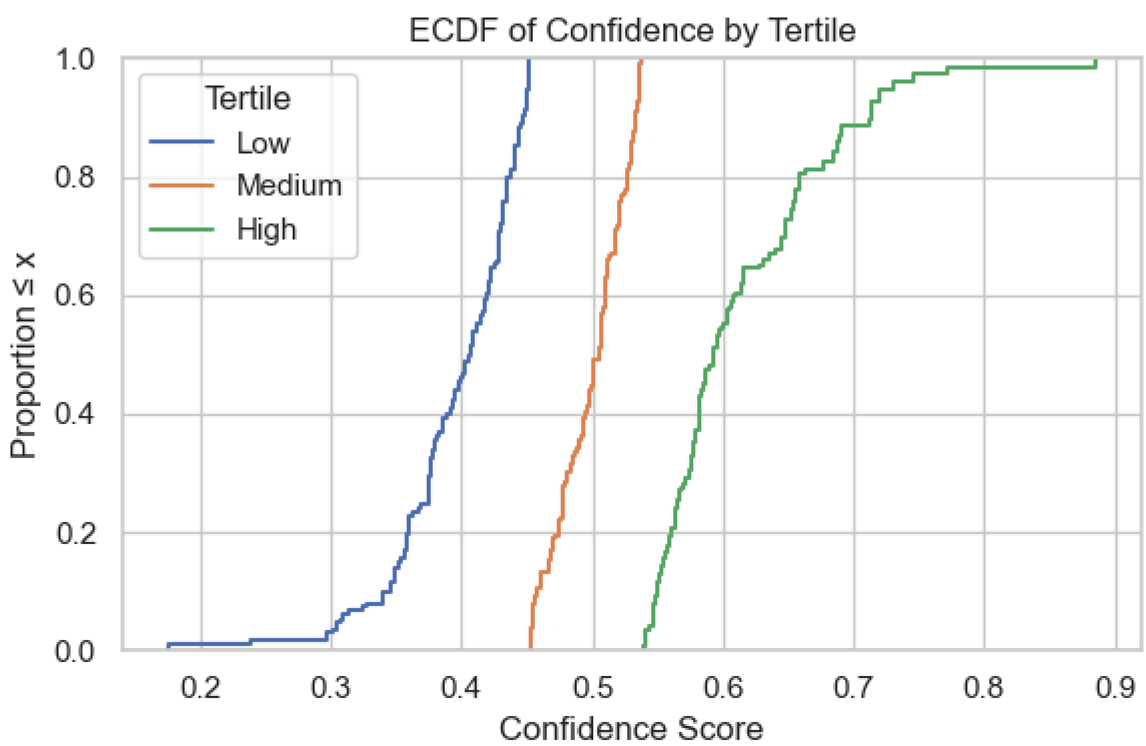
```
        subset = df[df[segment_col] == level]
        sns.ecdfplot(subset[trait], label=level)
    plt.title(f'ECDF of {trait.replace("_", " ").title()} by Tertile')
    plt.xlabel(f'{trait.replace("_", " ").title()} Score')
    plt.ylabel('Proportion ≤ x')
    plt.legend(title='Tertile')
    plt.tight_layout()
    plt.show()
```

```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/121546216
8.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.violinplot(data=df, x=segment_col, y=trait, palette='pastel', order=
['Low', 'Medium', 'High'])
```

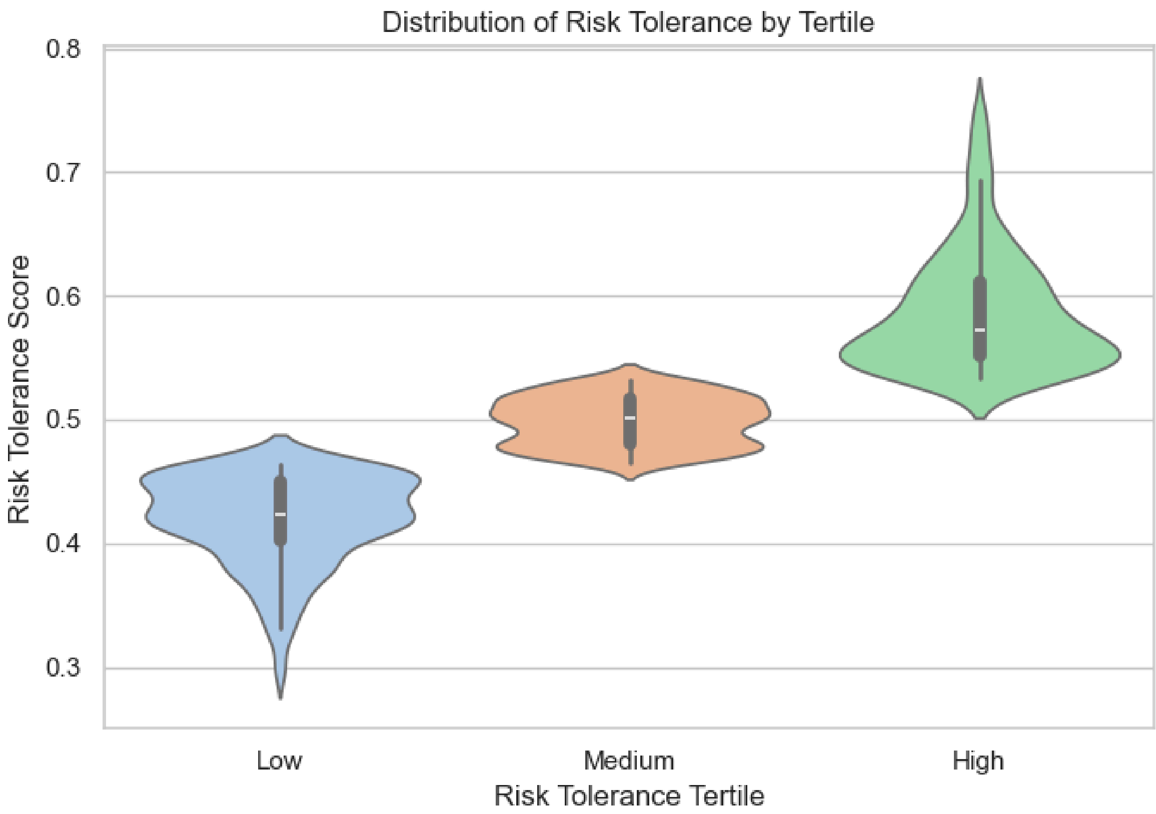



```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/121546216
8.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(data=df, x=segment_col, y=trait, palette='Set2', order=['Lo
w', 'Medium', 'High'])
```

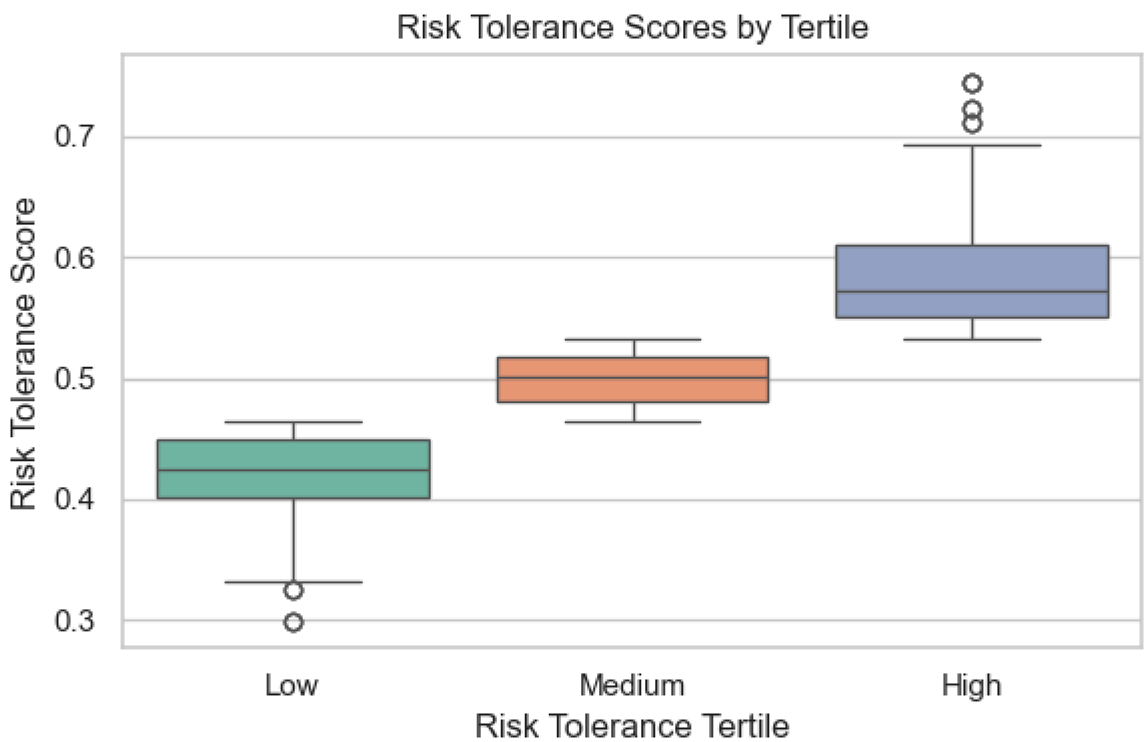


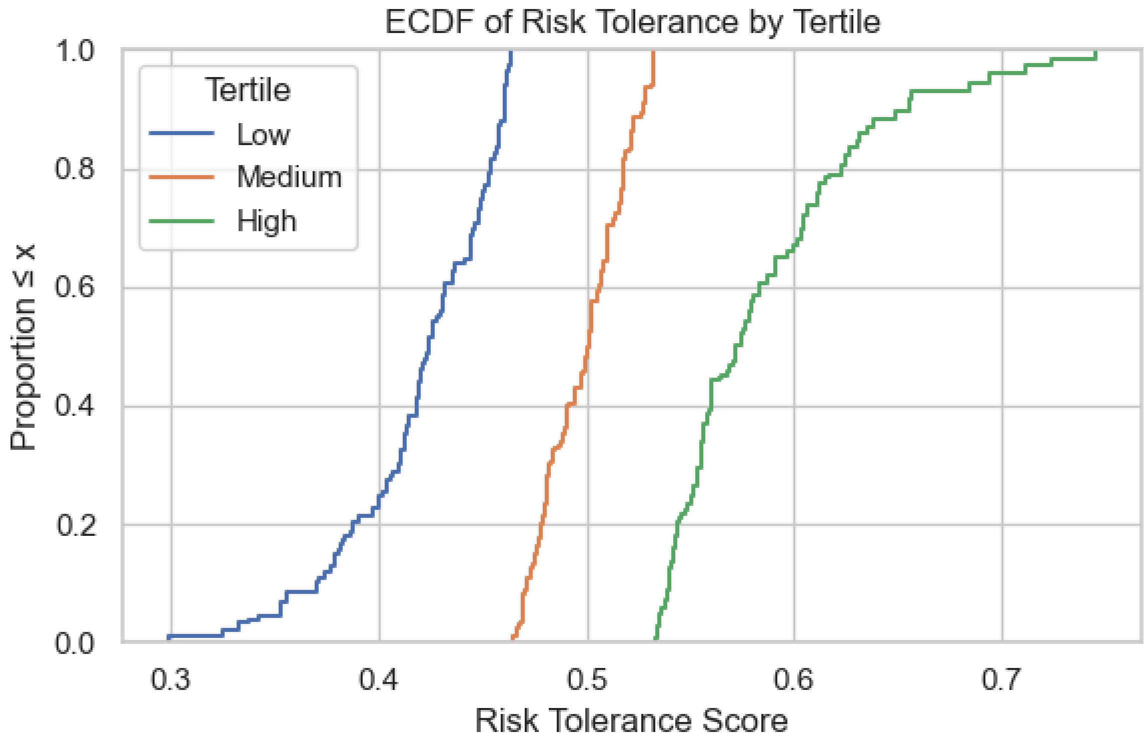



```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/121546216
8.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.violinplot(data=df, x=segment_col, y=trait, palette='pastel', order=
['Low', 'Medium', 'High'])
```

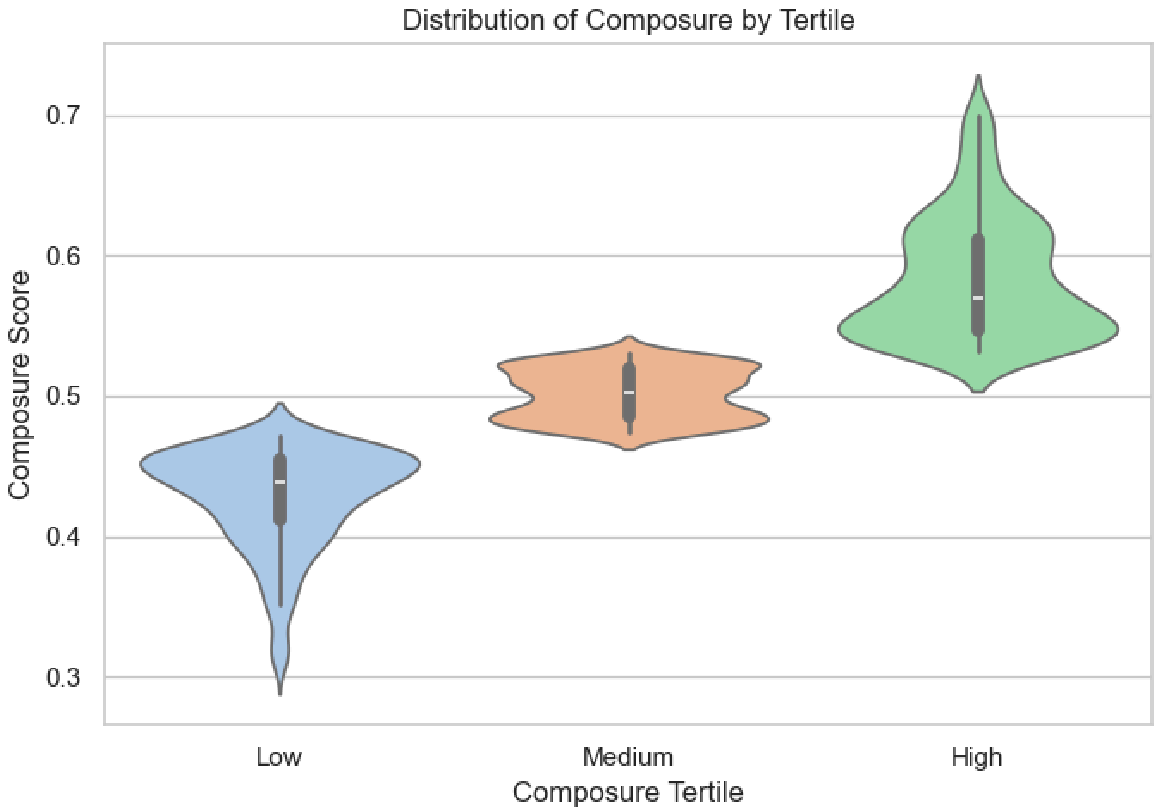



```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/121546216
8.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(data=df, x=segment_col, y=trait, palette='Set2', order=['Lo
w', 'Medium', 'High'])
```

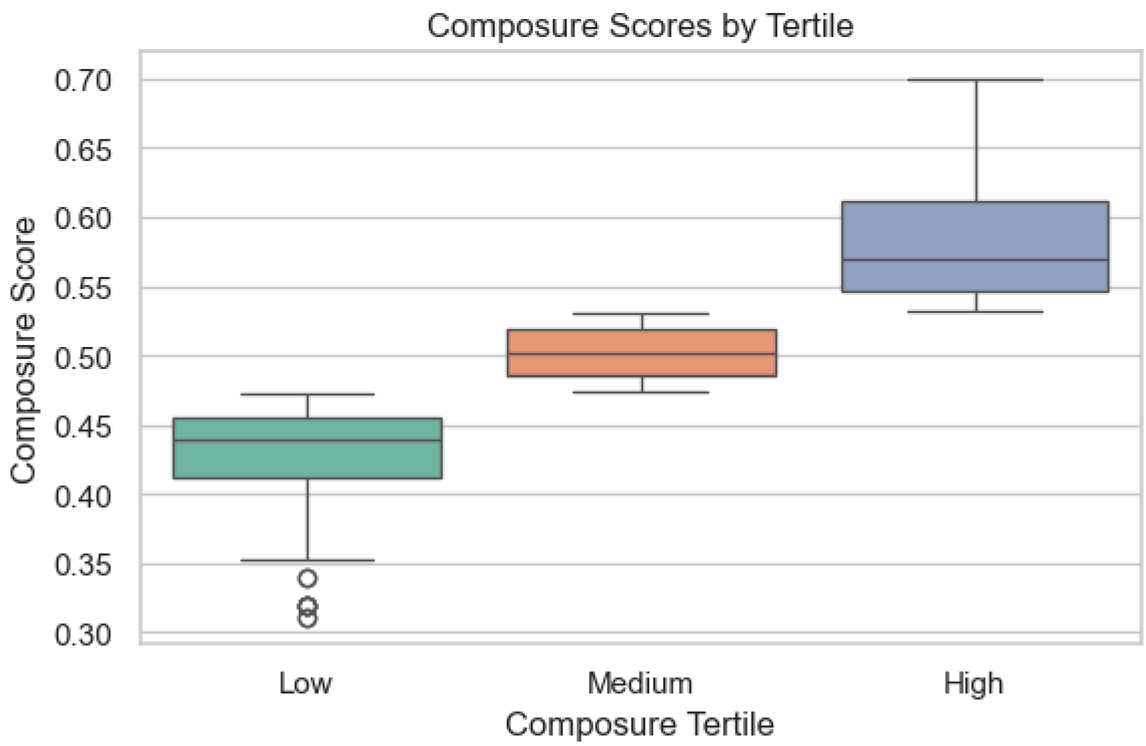


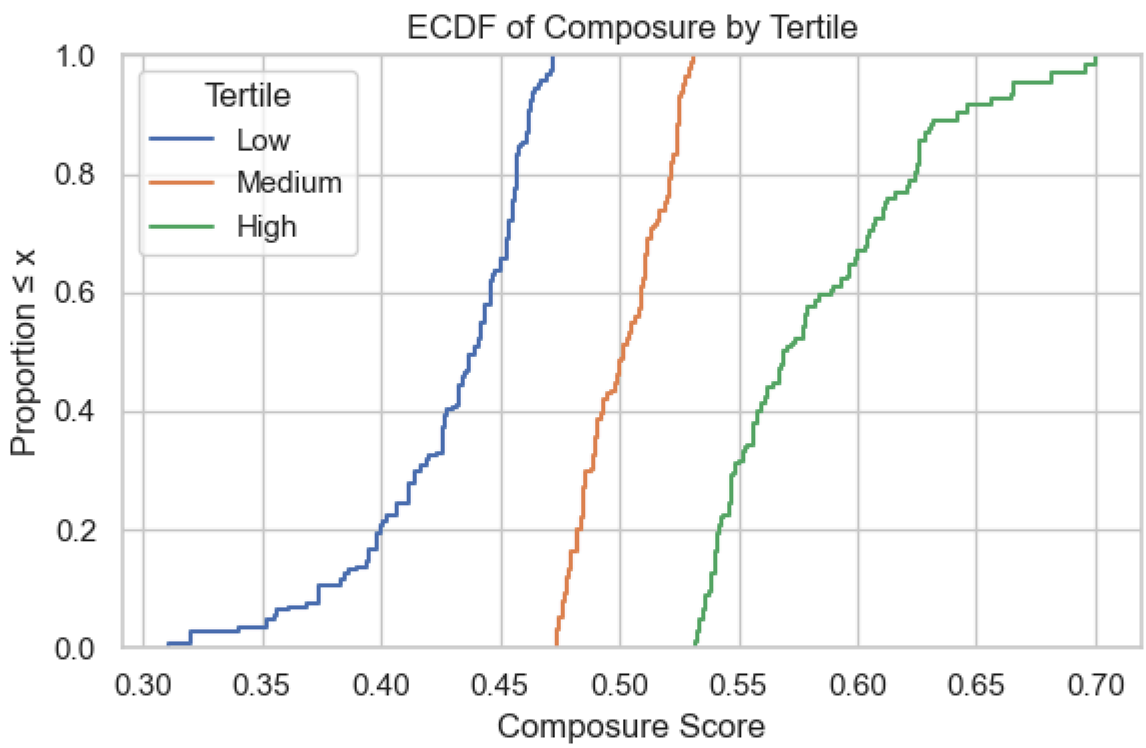



```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/121546216
8.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.violinplot(data=df, x=segment_col, y=trait, palette='pastel', order=
['Low', 'Medium', 'High'])
```

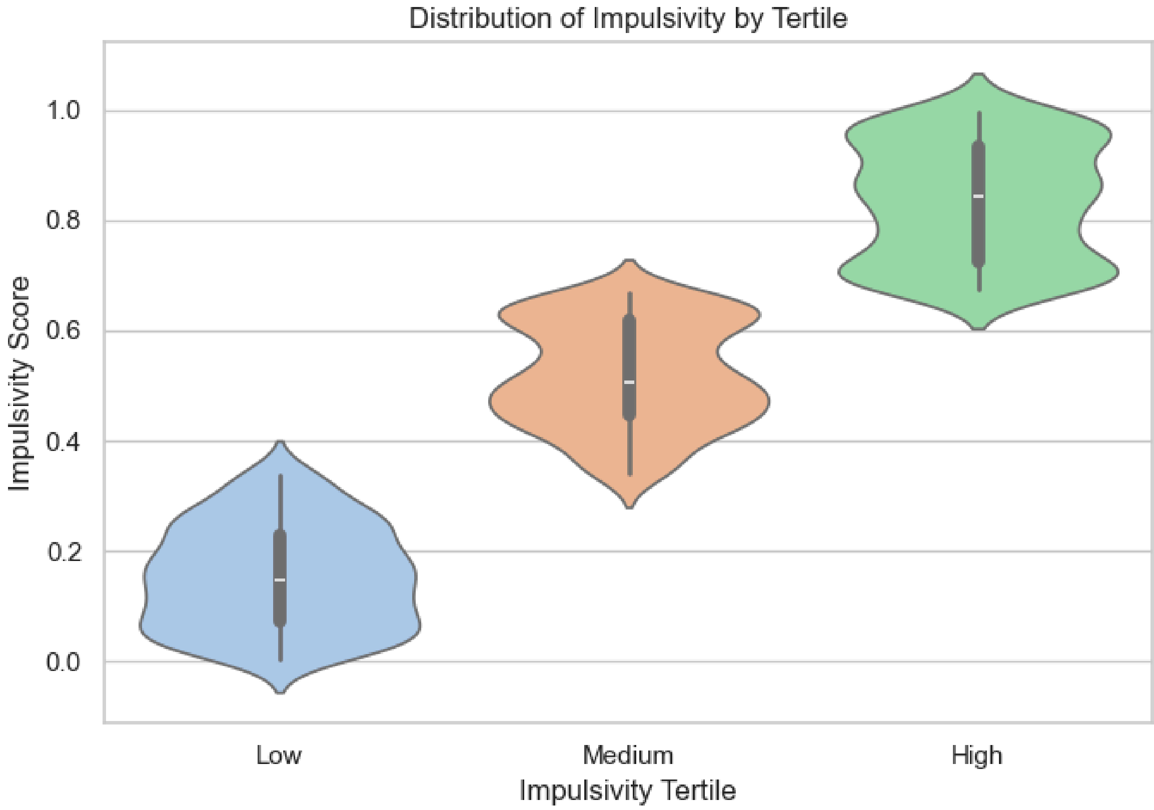


```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/121546216
8.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(data=df, x=segment_col, y=trait, palette='Set2', order=['Lo
w', 'Medium', 'High'])
```

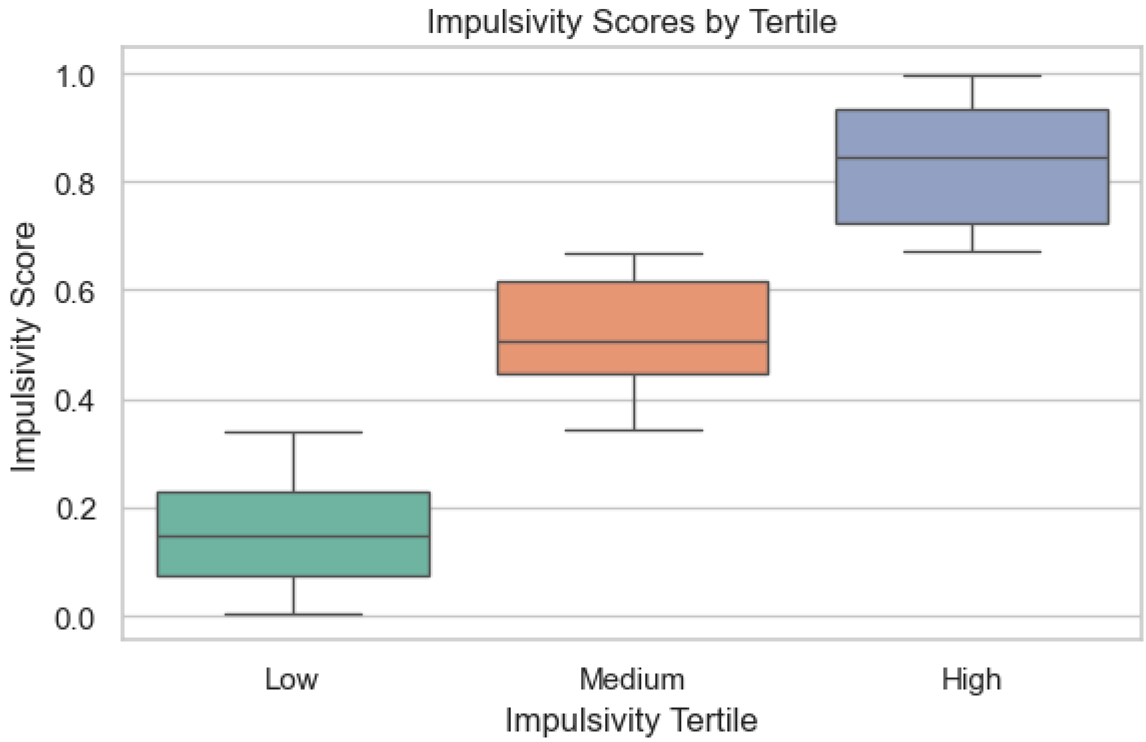


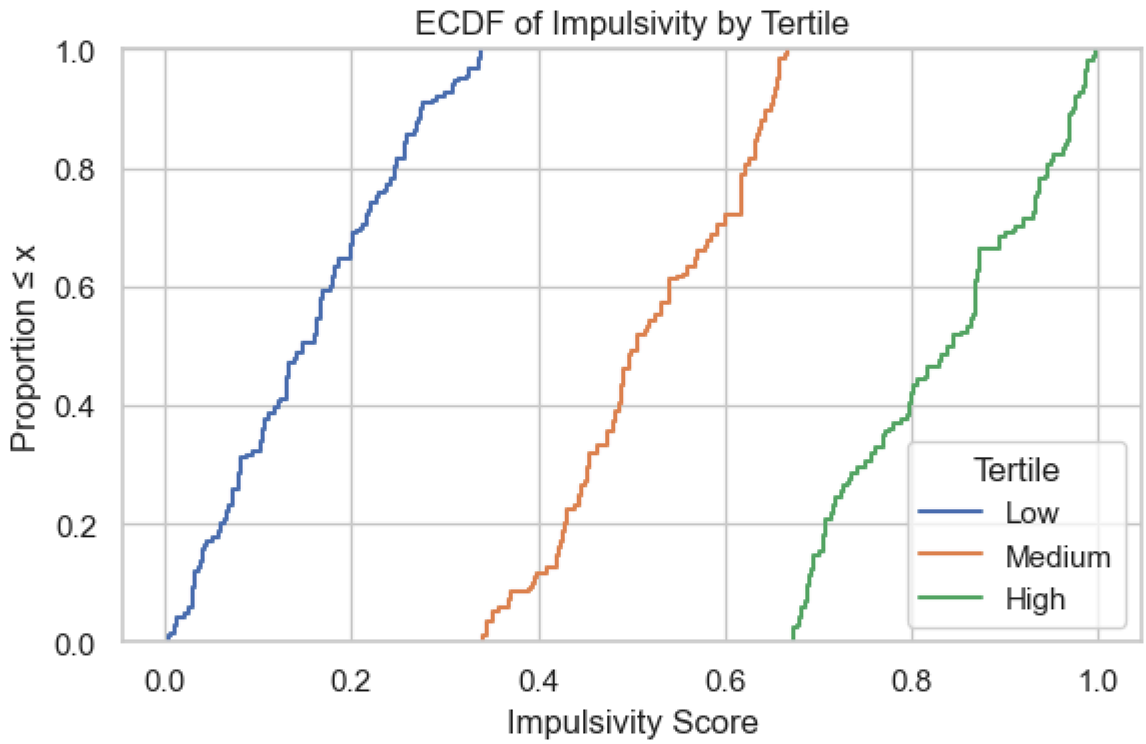


```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/121546216
8.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.violinplot(data=df, x=segment_col, y=trait, palette='pastel', order=
['Low', 'Medium', 'High'])
```

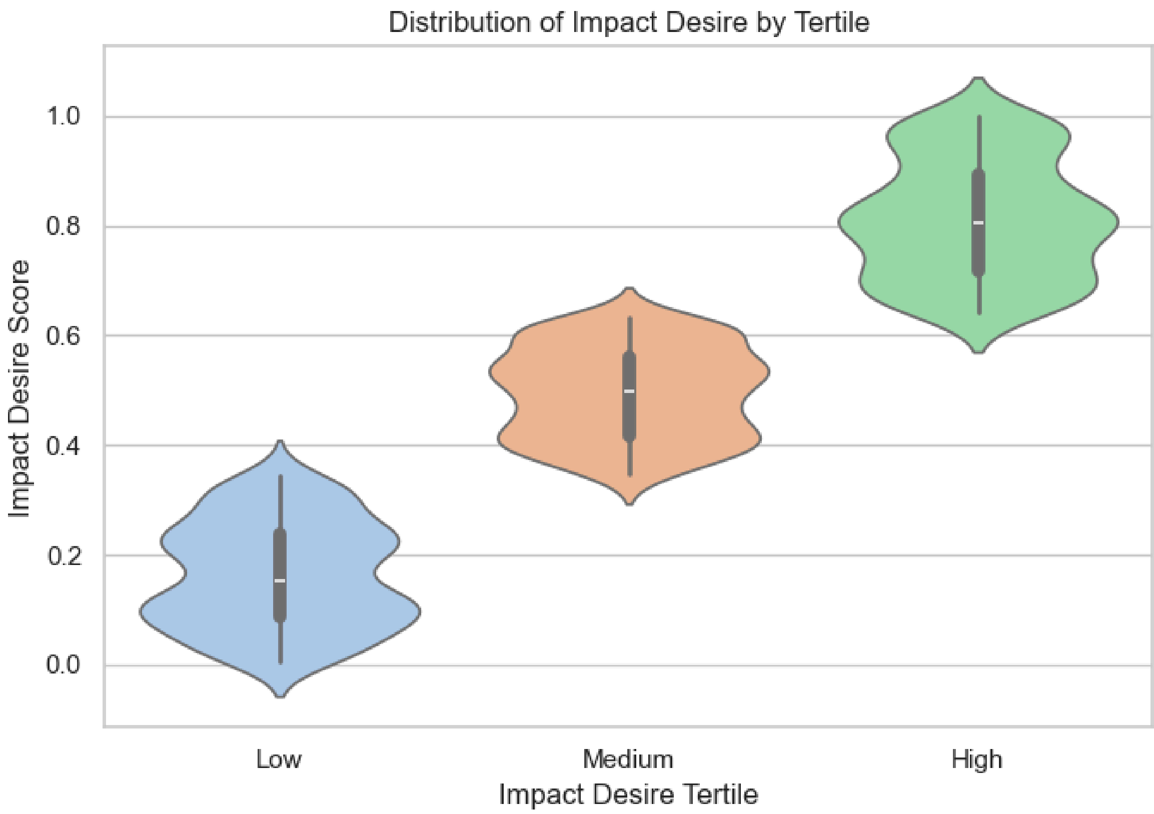



```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/121546216
8.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(data=df, x=segment_col, y=trait, palette='Set2', order=['Lo
w', 'Medium', 'High'])
```

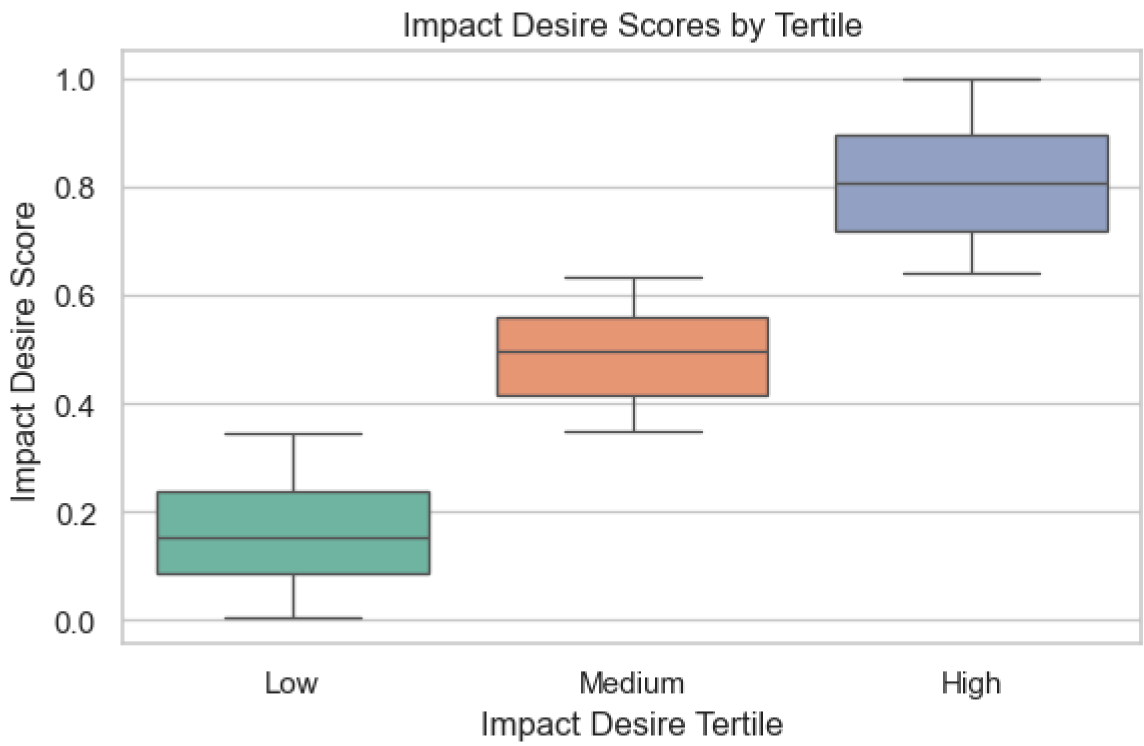


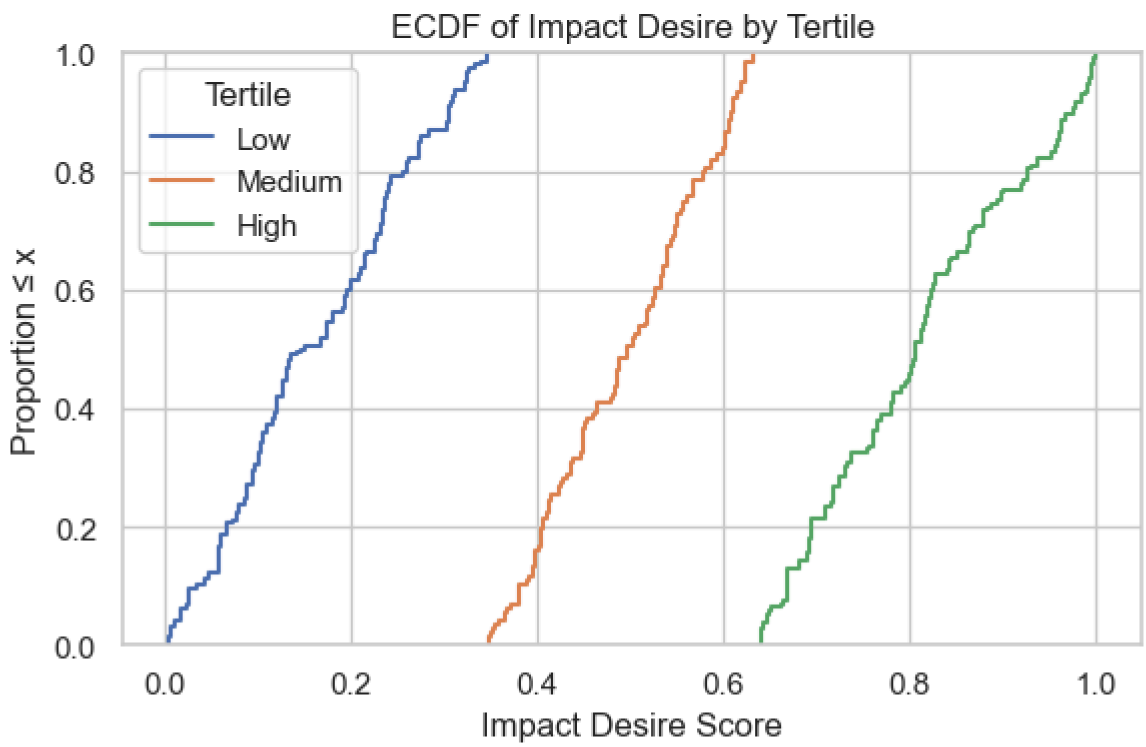



```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/121546216
8.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.violinplot(data=df, x=segment_col, y=trait, palette='pastel', order=
['Low', 'Medium', 'High'])
```

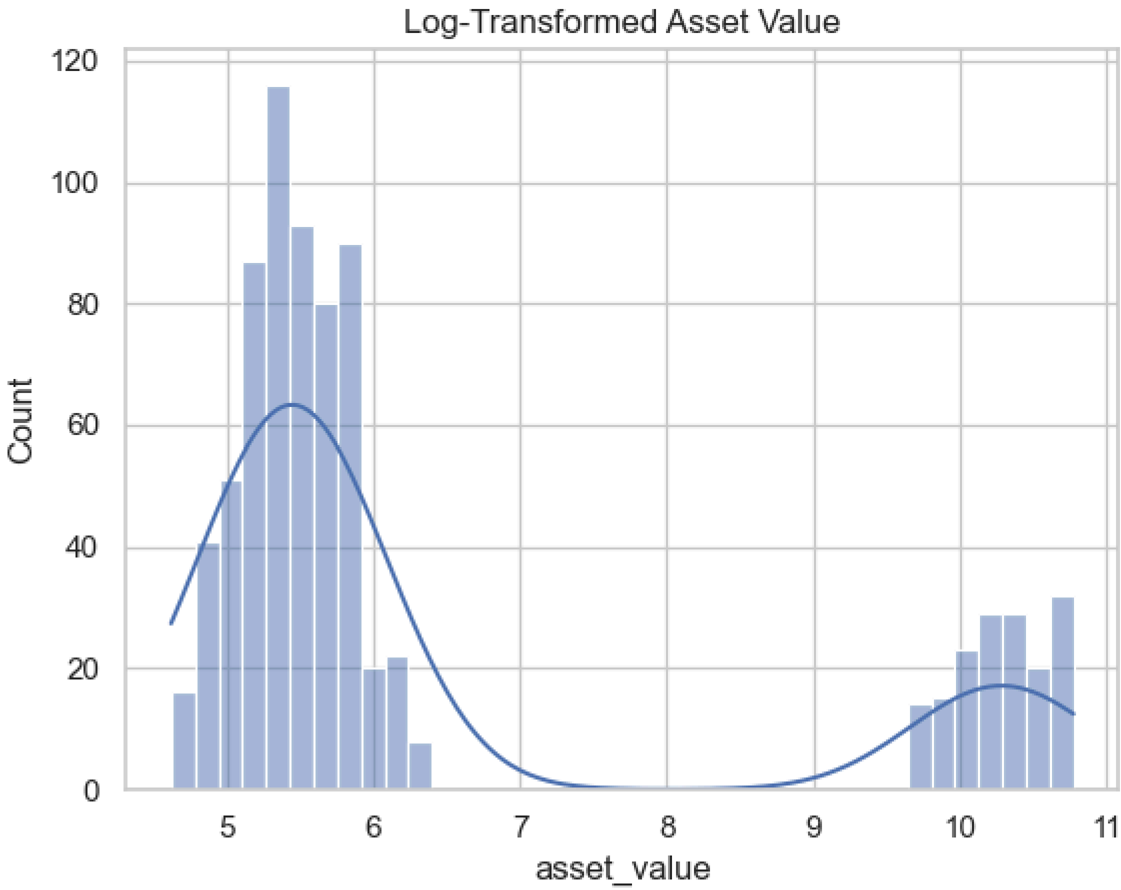


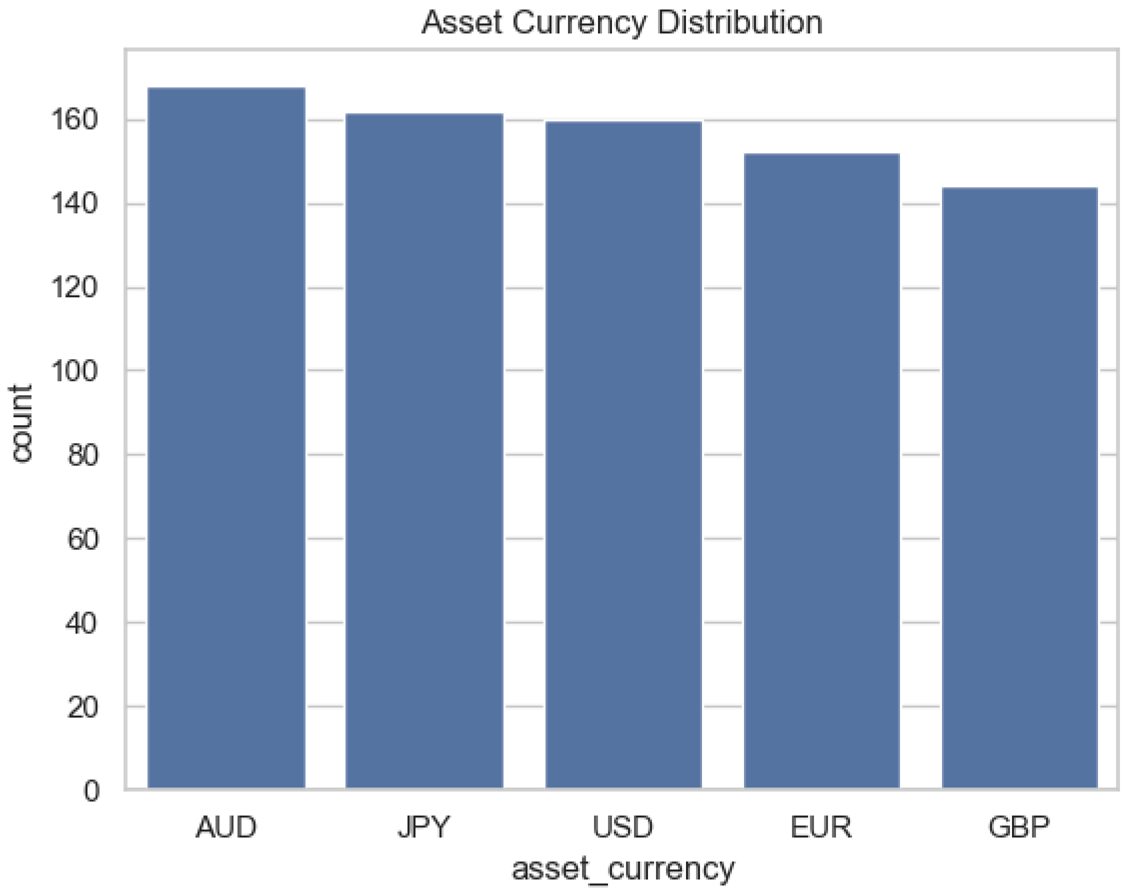

```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/121546216
8.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(data=df, x=segment_col, y=trait, palette='Set2', order=['Lo
w', 'Medium', 'High'])
```

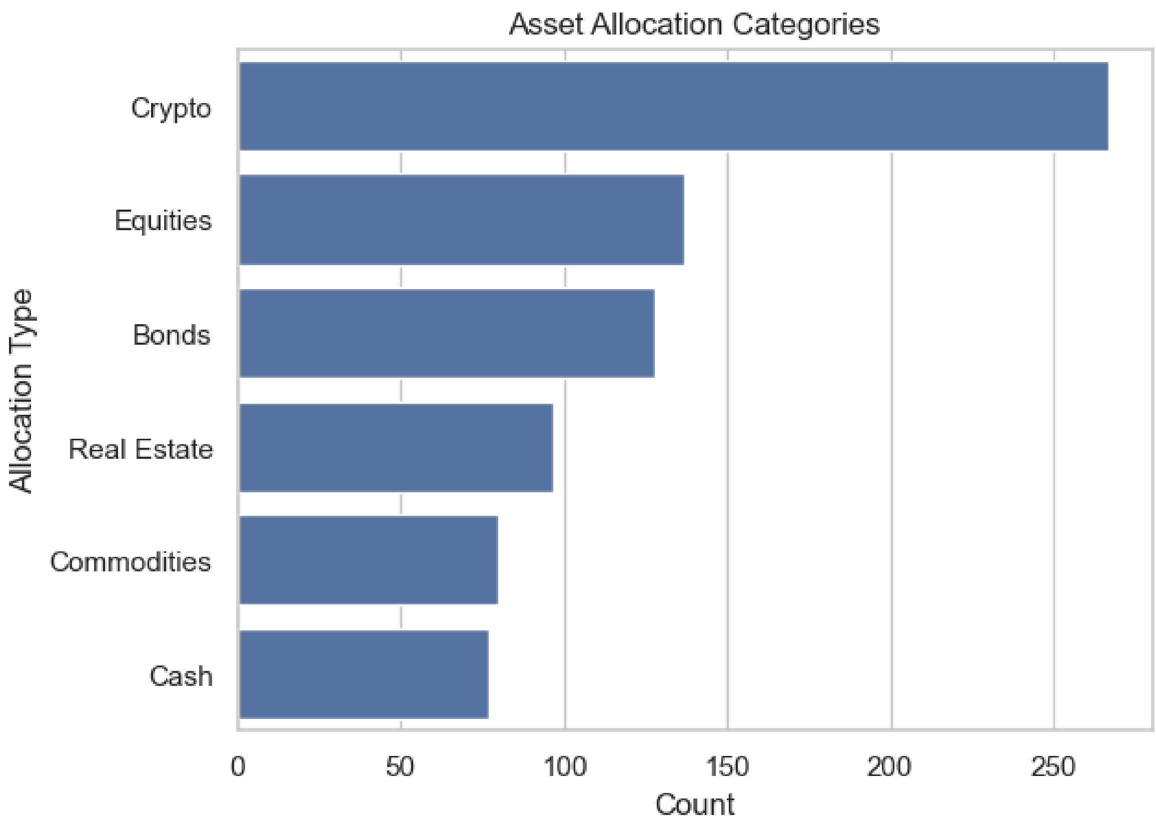




In [13]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

#Log-transformed distribution of asset value
sns.histplot(np.log1p(df['asset_value']), kde=True)
plt.title('Log-Transformed Asset Value')
plt.show()

#Distribution of asset currency
sns.countplot(data=df, x='asset_currency', order=df['asset_currency'].val
plt.title('Asset Currency Distribution')
plt.show()

#Distribution of asset allocation
sns.countplot(data=df, y='asset_allocation', order=df['asset_allocation']
plt.title('Asset Allocation Categories')
plt.xlabel('Count')
plt.ylabel('Allocation Type')
plt.show()
(df['asset_allocation'].value_counts(normalize=True) * 100).round(2)
```


Log-Transformed Asset Value
Count
120
100
80
60
40
20
0
5
6
7
8
9
10
11
asset_value


Asset Currency Distribution
count
160
140
120
100
80
60
40
20
0
AUD
JPY
USD
EUR
GBP
asset_currency

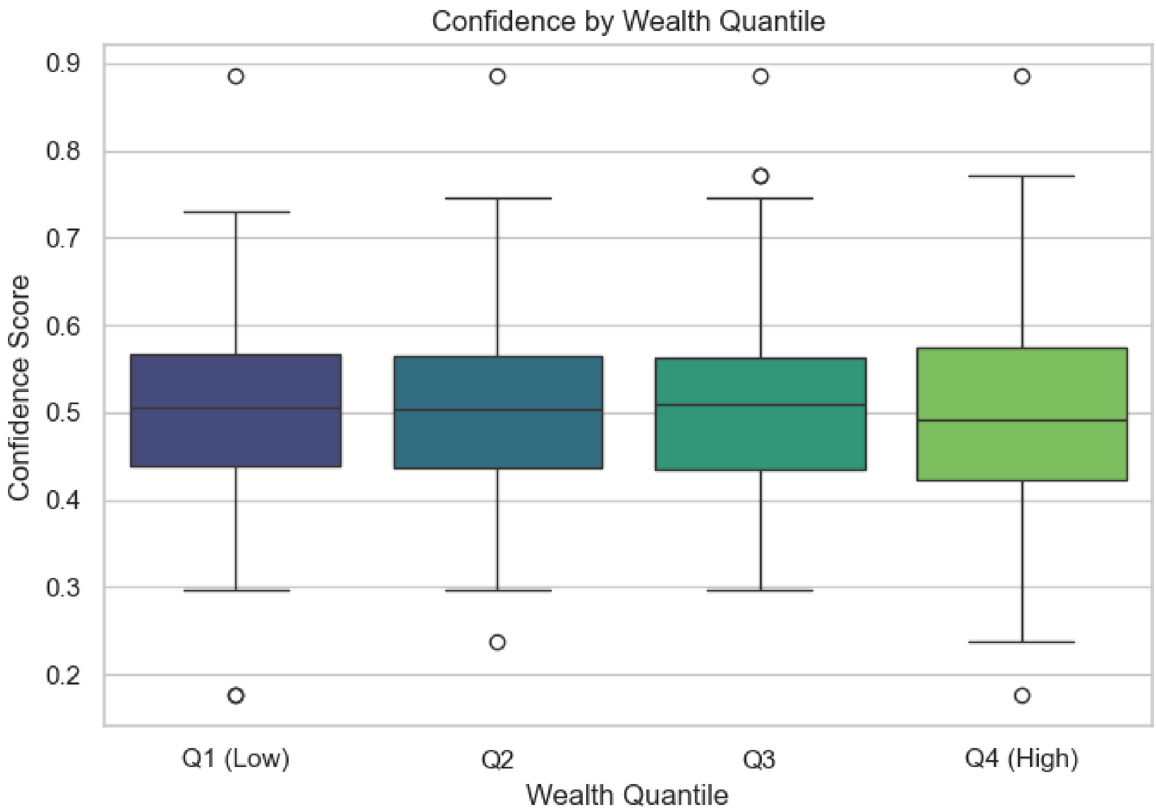


Out[13]:
```
asset_allocation
Crypto          33.97
Equities        17.43
Bonds           16.28
Real Estate     12.34
Commodities     10.18
Cash             9.80
Name: proportion, dtype: float64
```

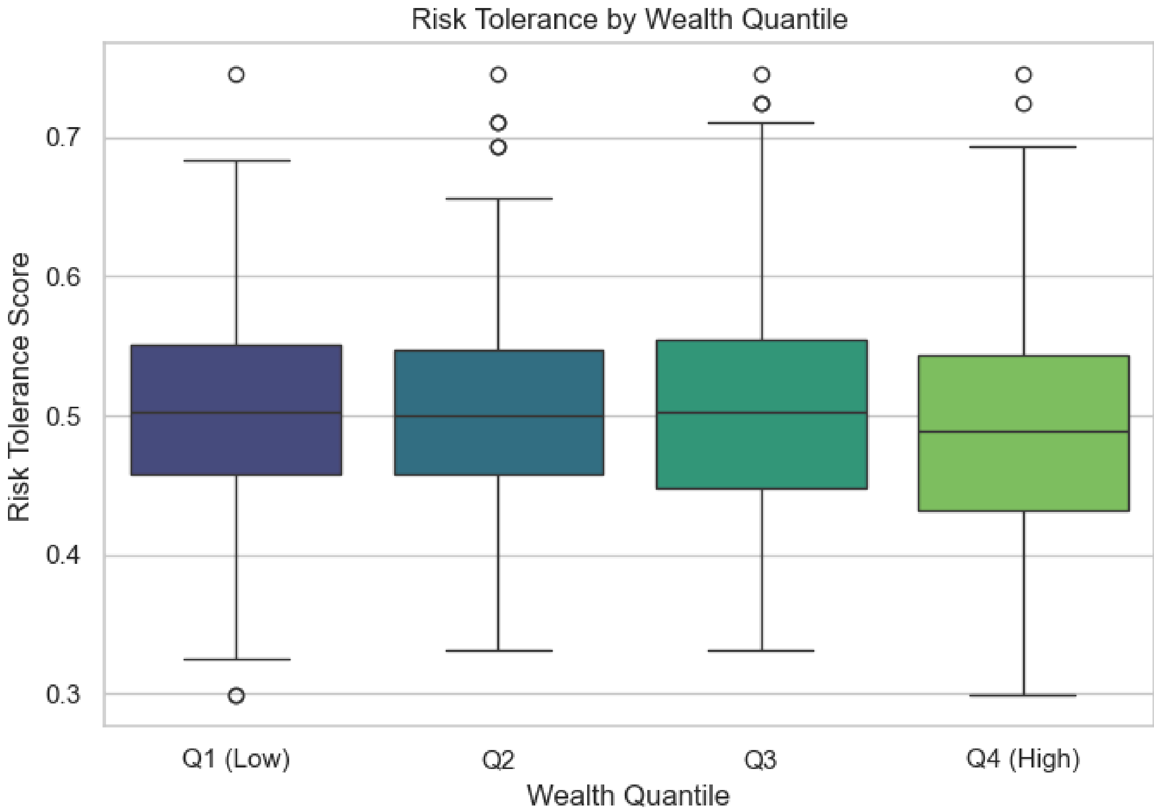
In [14]:
```
# Asset Insights Summary:
# - Log-Transformed Asset Value: Bimodal distribution reveals two financi
# - Asset Currency Distribution: Even spread across major currencies (AUD
# - Asset Allocation Categories: Crypto dominates all asset types — indic
```

In [15]:
```
#Visual inspection of traits by wealth quantiles

df['wealth_quantile'] = pd.qcut(df['asset_value'], q=4, labels=['Q1 (Low)
#Boxplots of each personality trait by wealth quantile
for trait in traits:
    plt.figure(figsize=(7, 5))
    sns.boxplot(data=df, x='wealth_quantile', y=trait, palette='viridis')
    plt.title(f'{trait.replace("_", " ").title()} by Wealth Quantile')
    plt.xlabel('Wealth Quantile')
    plt.ylabel(f'{trait.replace("_", " ").title()} Score')
    plt.tight_layout()
    plt.show()
```

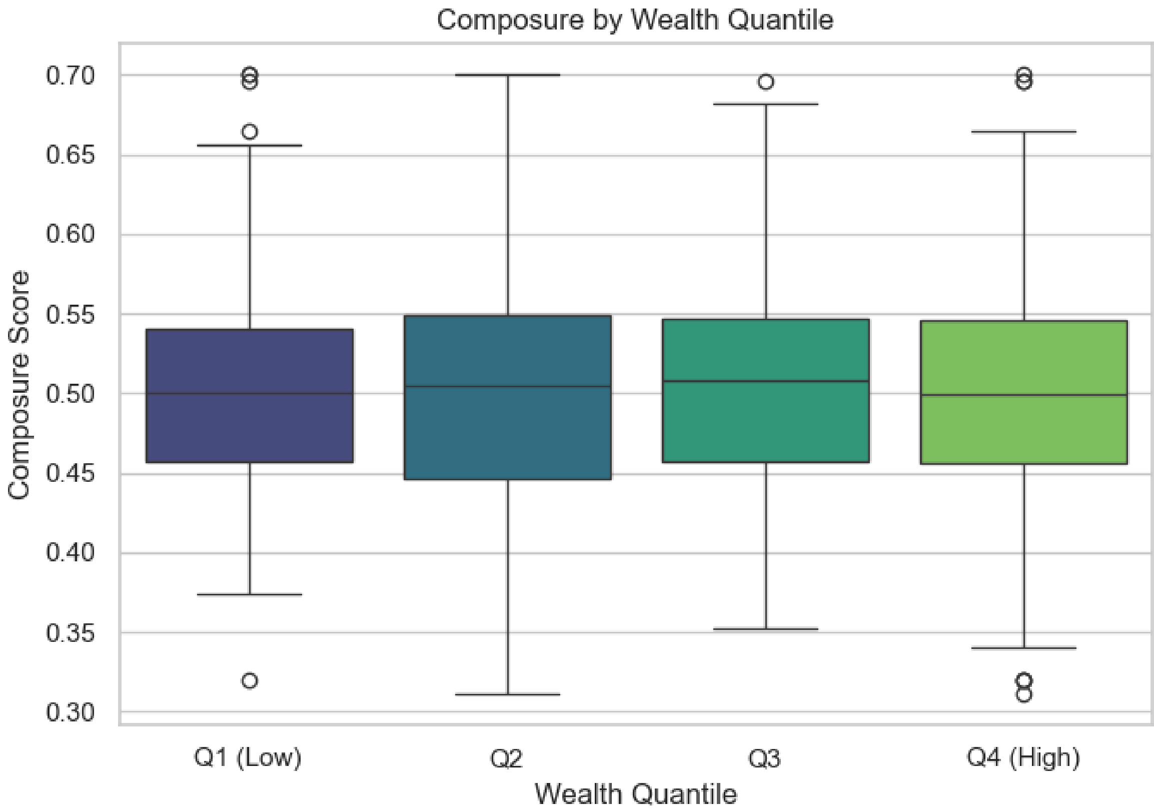
```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/10955406
9.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(data=df, x='wealth_quantile', y=trait, palette='viridis')
```



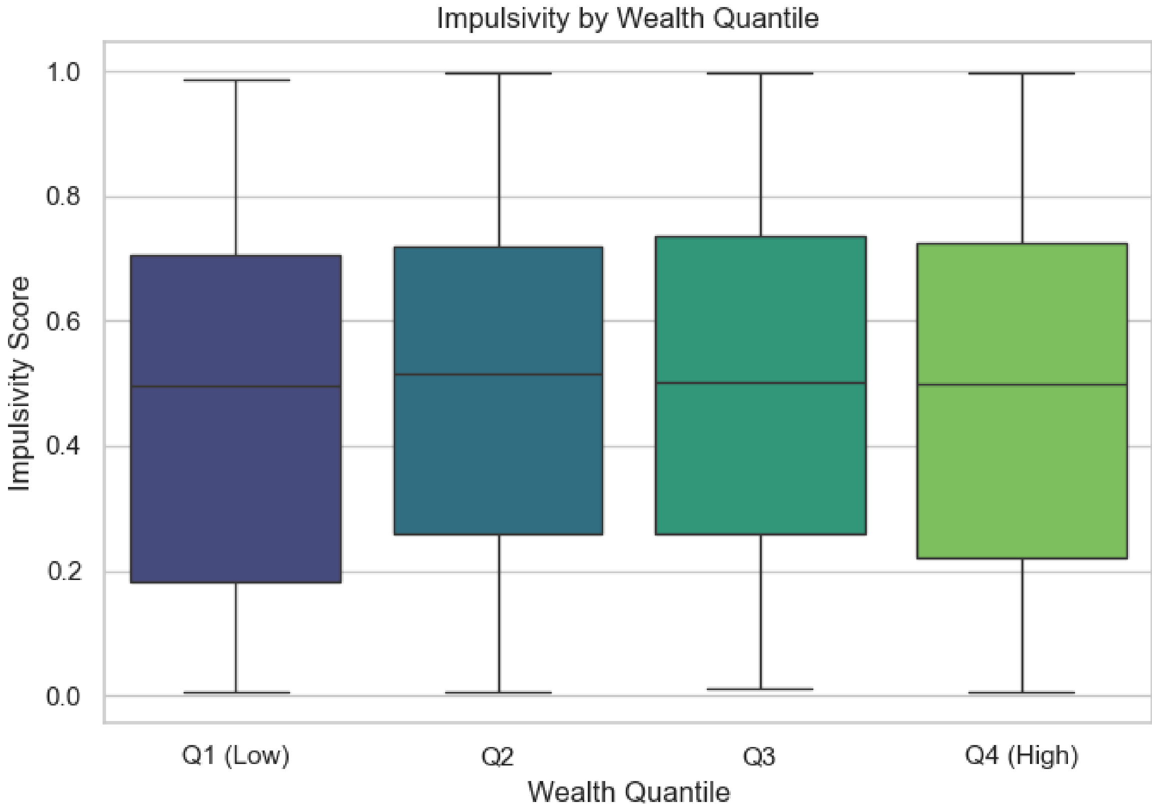

```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/10955406
9.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(data=df, x='wealth_quantile', y=trait, palette='viridis')
```



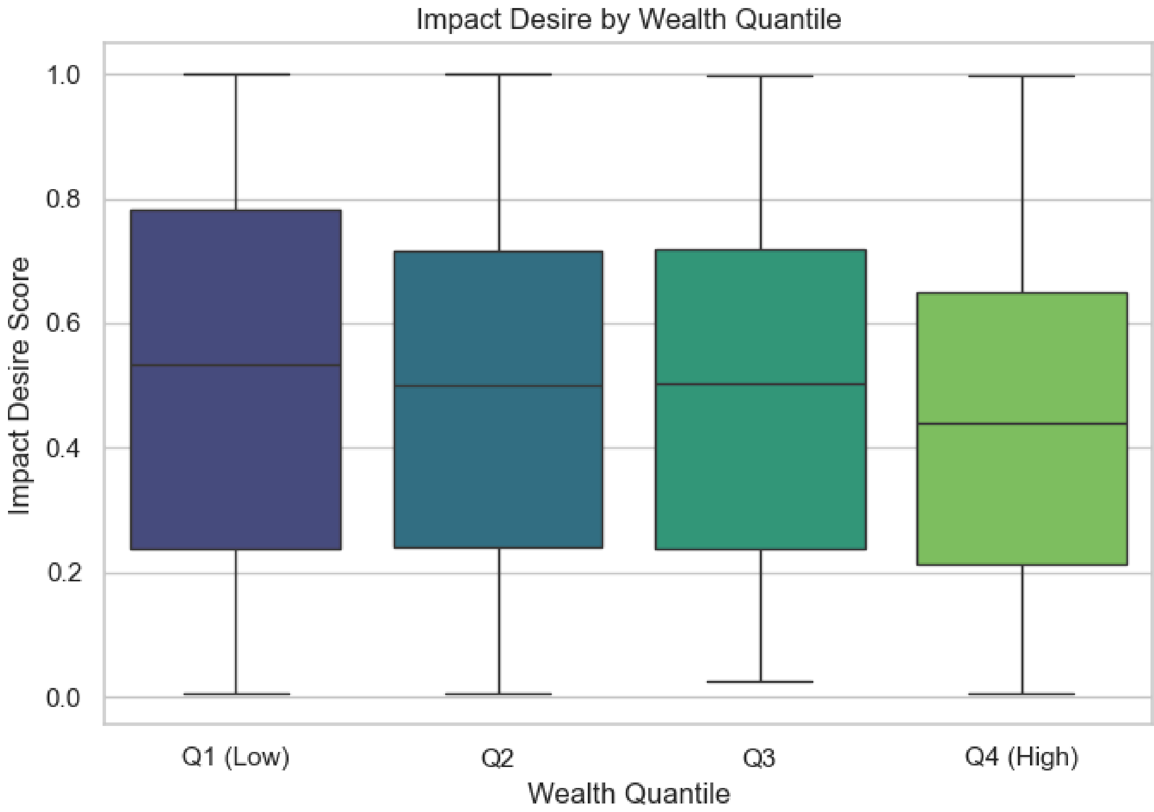

```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/10955406
9.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(data=df, x='wealth_quantile', y=trait, palette='viridis')
```



```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/10955406
9.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(data=df, x='wealth_quantile', y=trait, palette='viridis')
```




```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/10955406
9.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(data=df, x='wealth_quantile', y=trait, palette='viridis')
```



In [16]:

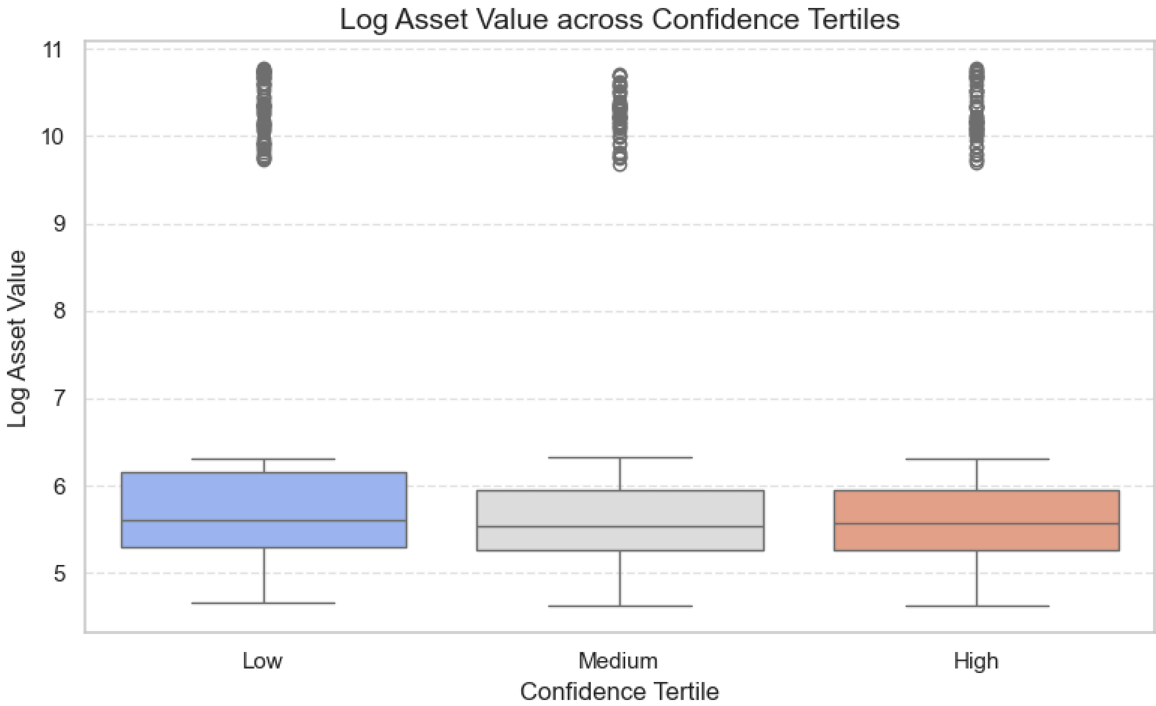
```
#Wealth Quartile × Trait Insights:
# Confidence: Median confidence increases from Q1 through Q3 then falls i
# Risk Tolerance: Risk-tolerance scores remain flat across all quartiles,
# Composure: Composure peaks in Q3 then slightly retreats in Q4, but Q4's
# Impulsivity: Impulsivity is flat from Q1 to Q3, dips in Q4, and Q4 show
#Impact Desire: Median impact-desire declines steadily from Q1 to Q4, wit
```

In [18]:

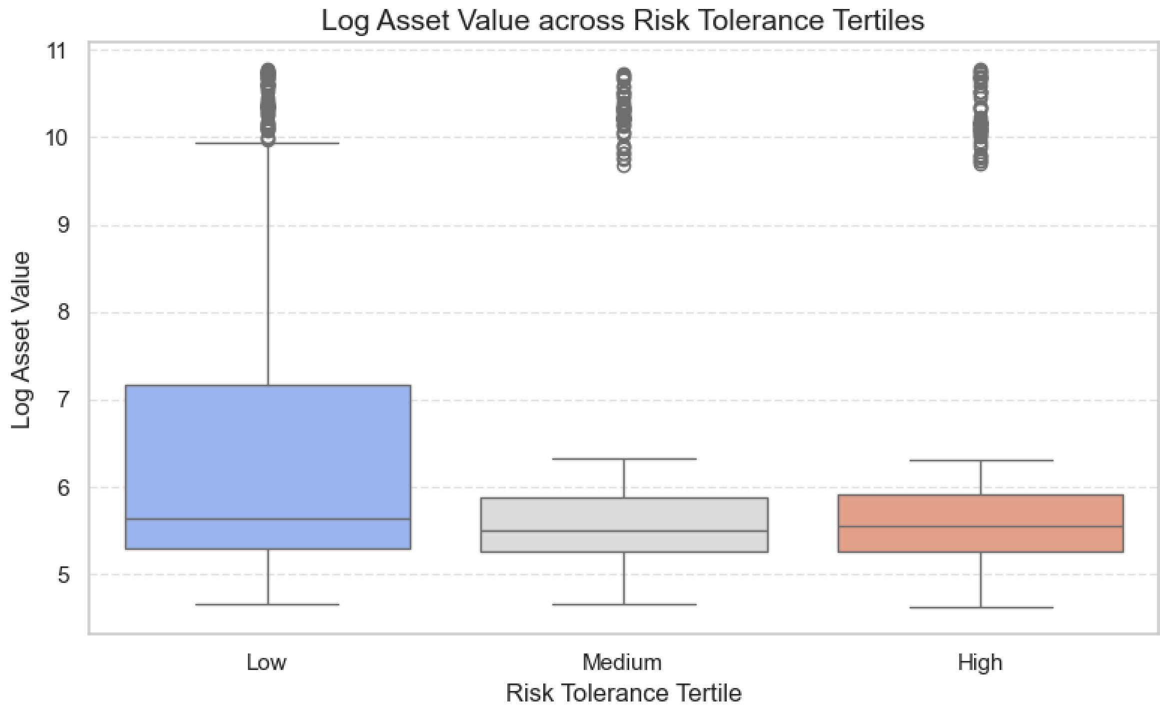
```
import numpy as np, seaborn as sns, matplotlib.pyplot as plt

# ------------------------------------------------------------------
# 1)  Make sure we have the column we're plotting
# ------------------------------------------------------------------
if 'log_asset_value' not in df.columns:
    df['log_asset_value'] = np.log1p(df['asset_value'])

# ------------------------------------------------------------------
# 2)  Helper to draw one box-plot
# ------------------------------------------------------------------
def tertile_boxplot(trait):
    col   = f"{trait}_tertile"
    title = f"Log Asset Value across {trait.replace('_',' ').title()} Ter

    # If tertile column doesn't exist, create it on the fly
    if col not in df.columns:
        df[col] = pd.qcut(df[trait], 3, labels=["Low","Medium","High"])

    plt.figure(figsize=(8, 5))
    sns.boxplot(
        data   = df,
        x      = col,
        y      = 'log_asset_value',
        order  = ['Low','Medium','High'],
        palette= 'coolwarm'
    )
```

```
    plt.title(title, fontsize=14)
    plt.xlabel(col.replace('_',' ').title(), fontsize=12)
    plt.ylabel("Log Asset Value",   fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()

# -------------------------------------------------------------------
# 3)  Loop through the five traits
# -------------------------------------------------------------------
for tr in ['confidence','risk_tolerance','composure','impulsivity','impac
    tertile_boxplot(tr)
```

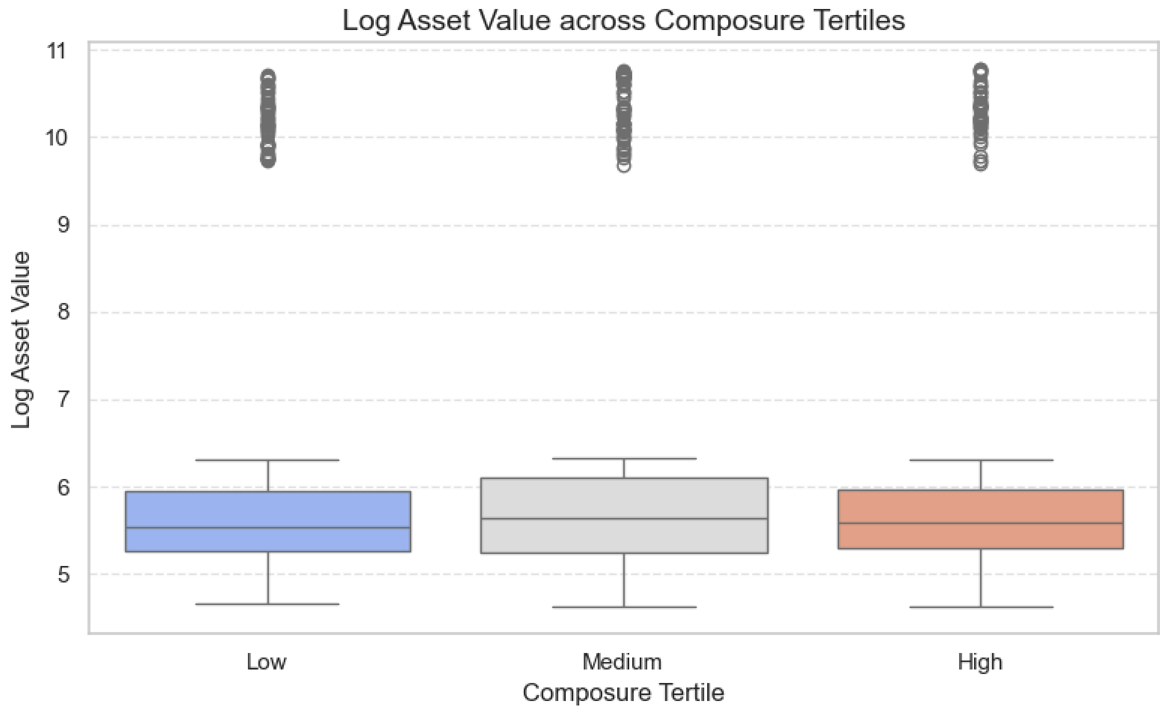
```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/88255300
0.py:21: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(
```



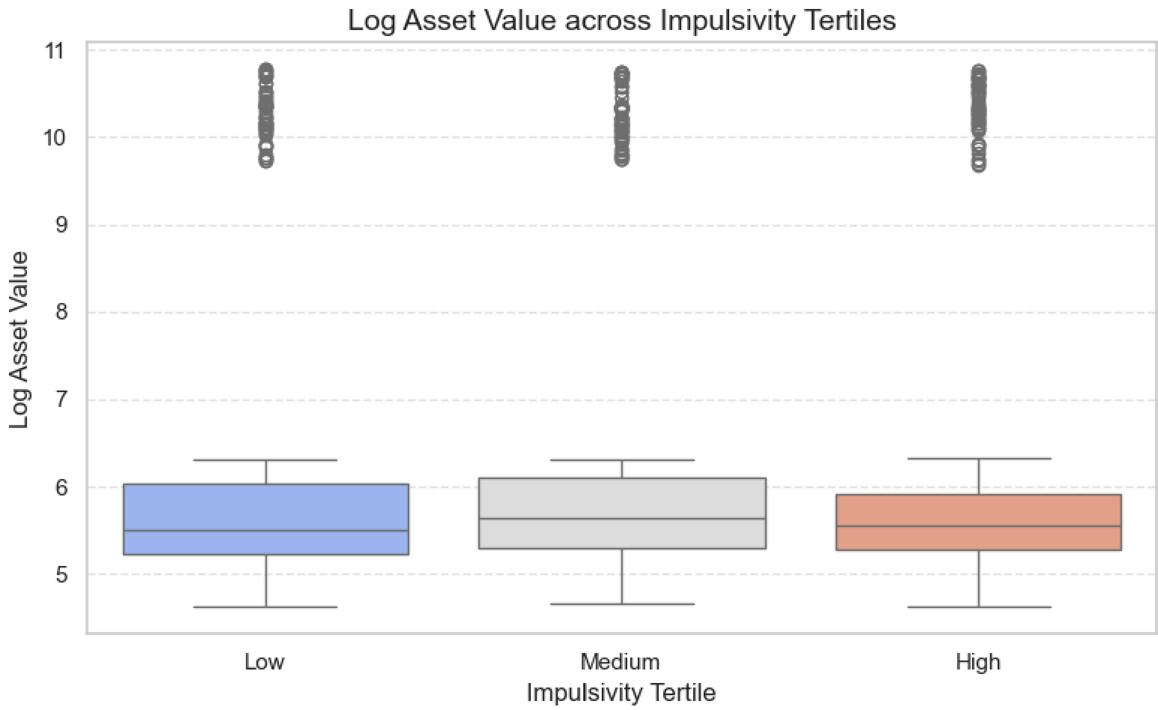

```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/88255300
0.py:21: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(
```




```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/88255300
0.py:21: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(
```




```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/88255300
0.py:21: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(
```




```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/88255300
0.py:21: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(
```

In [19]:
```python
import statsmodels.api as sm
from statsmodels.formula.api import ols

# List all tertile-based trait columns
tertile_vars = [
    'confidence_tertile',
    'composure_tertile',
    'impulsivity_tertile',
```

```
    'risk_tolerance_tertile',
    'impact_desire_tertile'
]

# Run one-way ANOVA for each trait’s tertiles against log_asset_value
for var in tertile_vars:
    print(f'\n=== ANOVA for {var.replace("_", " ").title()} ===')
    model = ols(f'log_asset_value ~ C({var})', data=df).fit()
    print(sm.stats.anova_lm(model, typ=2))
```

```
=== ANOVA for Confidence Tertile ===
                            sum_sq     df         F    PR(>F)
C(confidence_tertile)     5.759573    2.0  0.729379  0.482536
Residual               3091.496274  783.0       NaN       NaN

=== ANOVA for Composure Tertile ===
                           sum_sq     df         F    PR(>F)
C(composure_tertile)     4.891108    2.0  0.619225  0.538625
Residual              3092.364740  783.0       NaN       NaN

=== ANOVA for Impulsivity Tertile ===
                              sum_sq     df         F    PR(>F)
C(impulsivity_tertile)     2.502398    2.0  0.316564  0.728741
Residual                3094.753450  783.0       NaN       NaN

=== ANOVA for Risk Tolerance Tertile ===
                                 sum_sq     df         F    PR(>F)
C(risk_tolerance_tertile)     23.530047    2.0  2.997019  0.050509
Residual                    3073.725801  783.0       NaN       NaN

=== ANOVA for Impact Desire Tertile ===
                                sum_sq     df         F    PR(>F)
C(impact_desire_tertile)     17.903247    2.0  2.276167  0.103356
Residual                   3079.352600  783.0       NaN       NaN
```

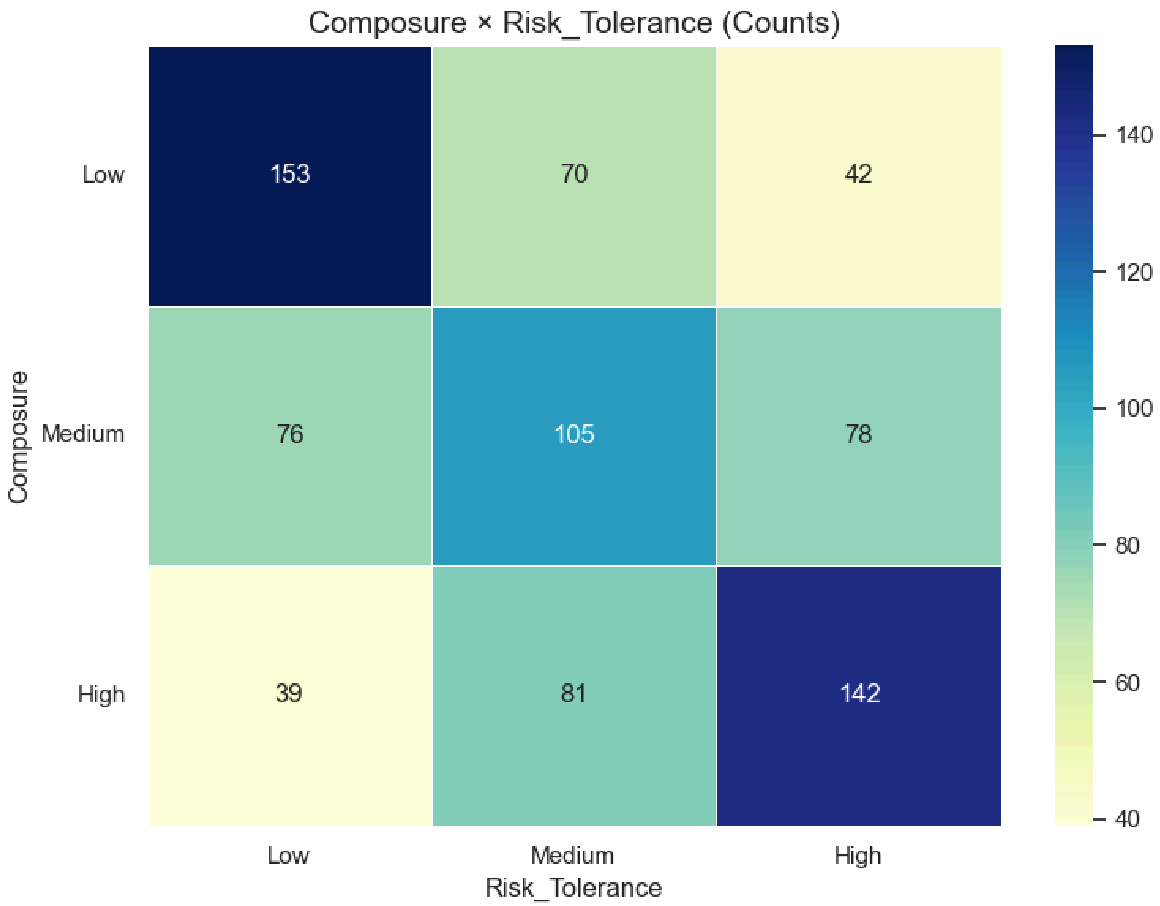
In [20]:
```
from statsmodels.stats.multicomp import pairwise_tukeyhsd

# Perform Tukey HSD post-hoc test for risk_tolerance tertiles
tukey = pairwise_tukeyhsd(
    endog=df['log_asset_value'],
    groups=df['risk_tolerance_tertile'],
    alpha=0.05
)

print(tukey)
```

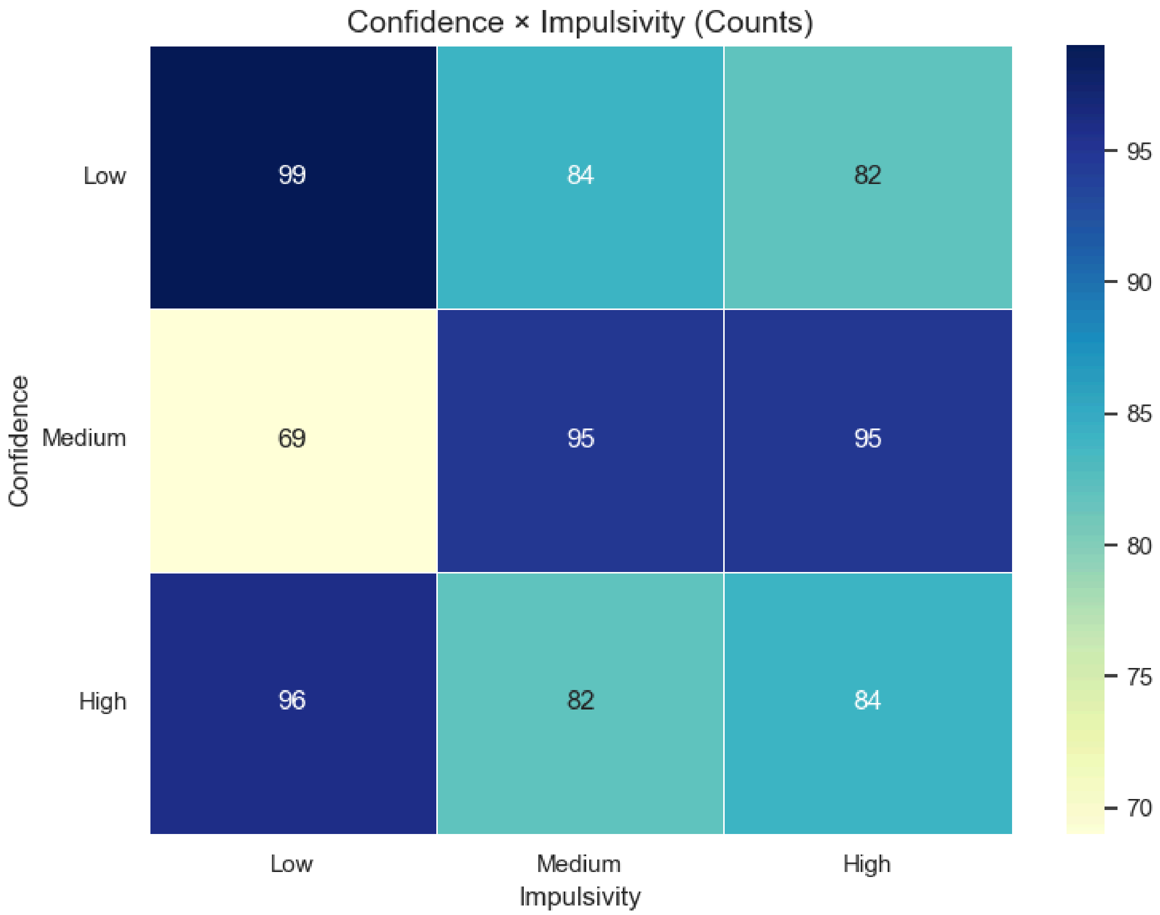
```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================
group1 group2 meandiff p-adj   lower  upper  reject
---------------------------------------------------
  High    Low   0.3432 0.1144  -0.061 0.7474  False
  High Medium  -0.0407 0.9703 -0.4496 0.3682  False
   Low Medium  -0.3839 0.0689 -0.7905 0.0227  False
---------------------------------------------------
```

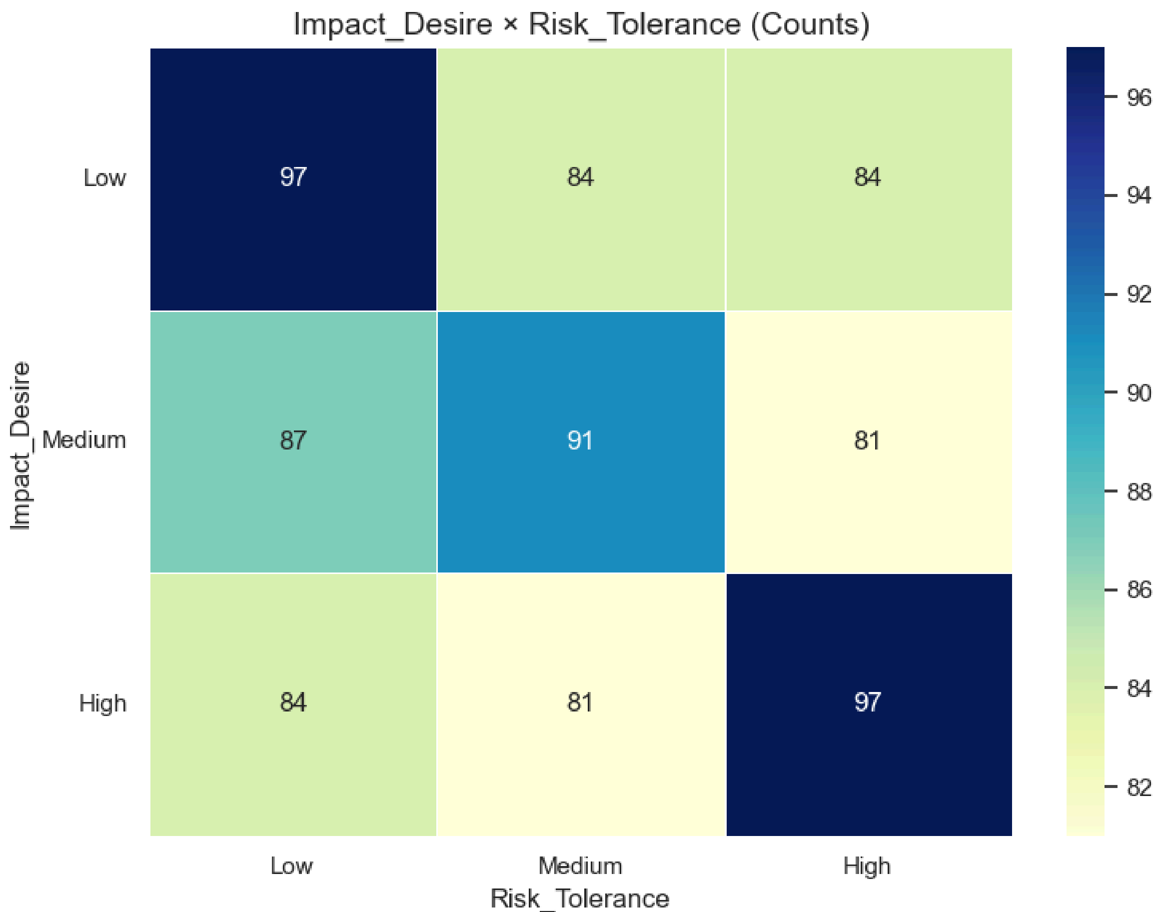
In [21]:
```
# Confidence tertile vs log assets: boxplots nearly identical; ANOVA p=0.
# Composure tertile vs log assets: distributions overlap heavily; ANOVA p
# Impulsivity tertile vs log assets: flat medians and IQRs; ANOVA p=0.729
# Risk-tolerance tertile vs log assets: slight mean shift; ANOVA p=0.0505
# Impact-desire tertile vs log assets: minor downward median trend; ANOVA
```

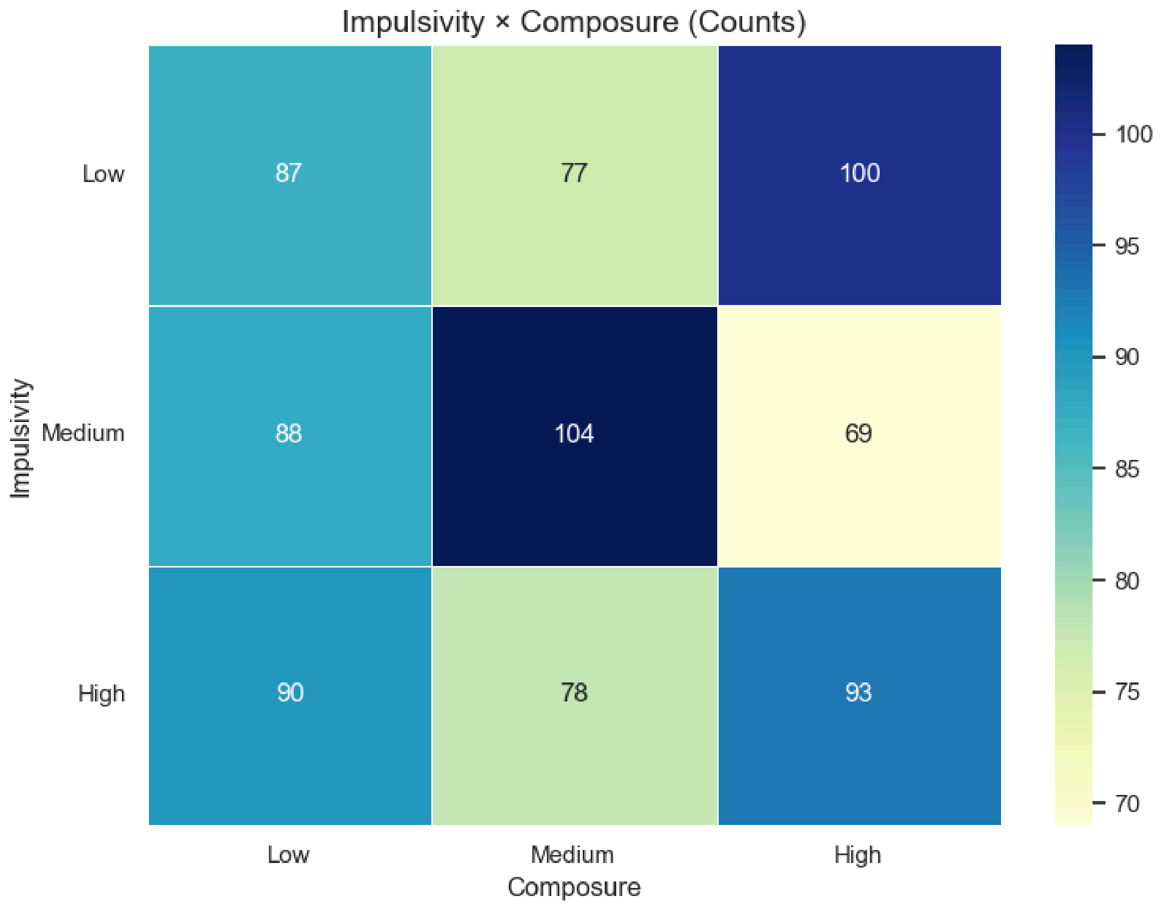
In [22]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# List of top behavioural trait pairs to compare
trait_pairs = [
    ('composure_tertile', 'risk_tolerance_tertile'),
    ('confidence_tertile', 'impulsivity_tertile'),
    ('impact_desire_tertile', 'risk_tolerance_tertile'),
    ('impulsivity_tertile', 'composure_tertile')
]

# Define color palette
cmap = 'YlGnBu'

# Plot cross-tab heatmap for each pair
for row_var, col_var in trait_pairs:
    cross_tab = pd.crosstab(df[row_var], df[col_var])

    plt.figure(figsize=(8, 6))
    sns.heatmap(cross_tab, annot=True, fmt='d', cmap=cmap, linewidths=0.5

    plt.title(f'{row_var.replace("_tertile", "").title()} × {col_var.repl
    plt.xlabel(col_var.replace("_tertile", "").title(), fontsize=12)
    plt.ylabel(row_var.replace("_tertile", "").title(), fontsize=12)
    plt.xticks(rotation=0)
    plt.yticks(rotation=0)
    plt.tight_layout()
    plt.show()
```

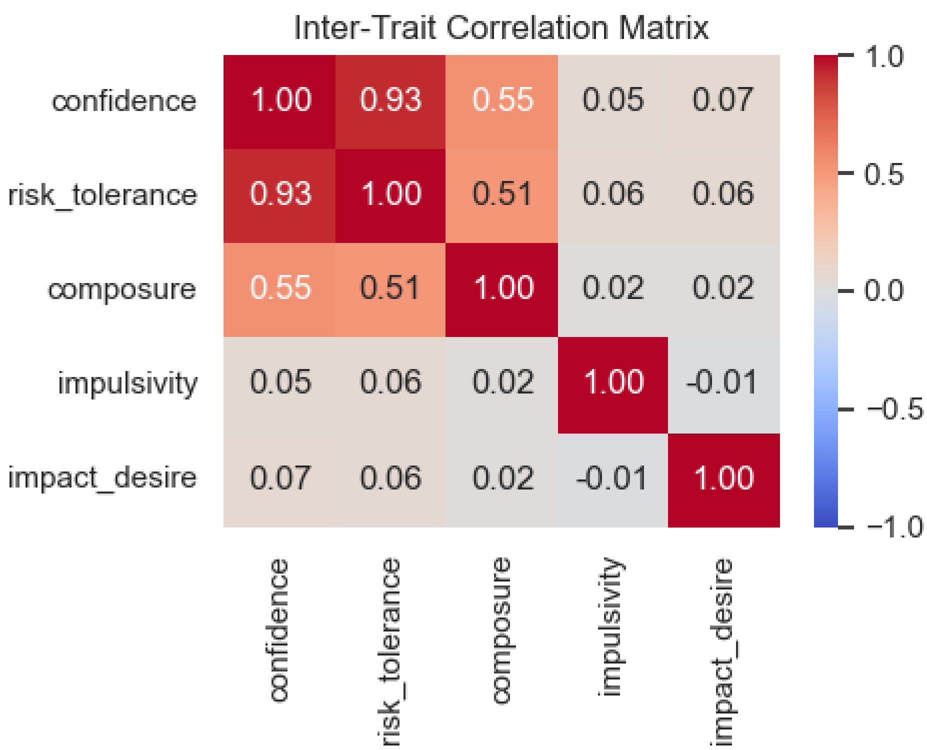



Confidence × Impulsivity (Counts)
Confidence
Low
Medium
High
99
84
82
69
95
95
96
82
84
Low
Medium
High
Impulsivity
95
90
85
80
75
70


Impact_Desire × Risk_Tolerance (Counts)
Impact_Desire
Low
Medium
High
97
84
84
87
91
81
84
81
97
Low
Medium
High
Risk_Tolerance
96
94
92
90
88
86
84
82

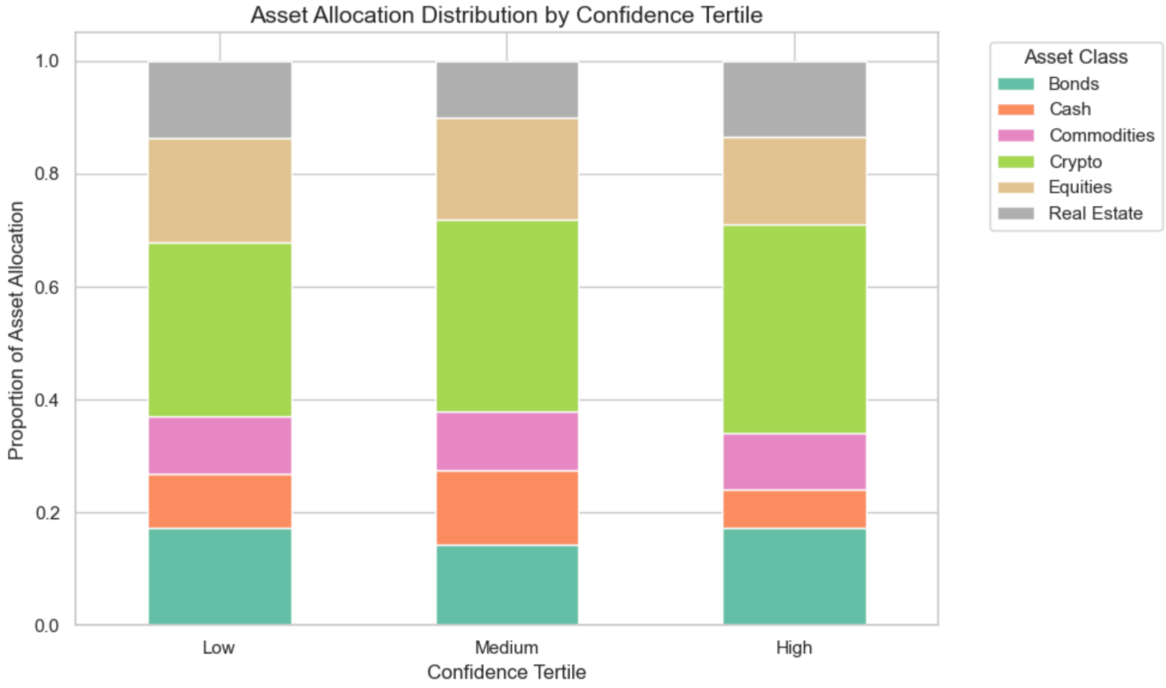


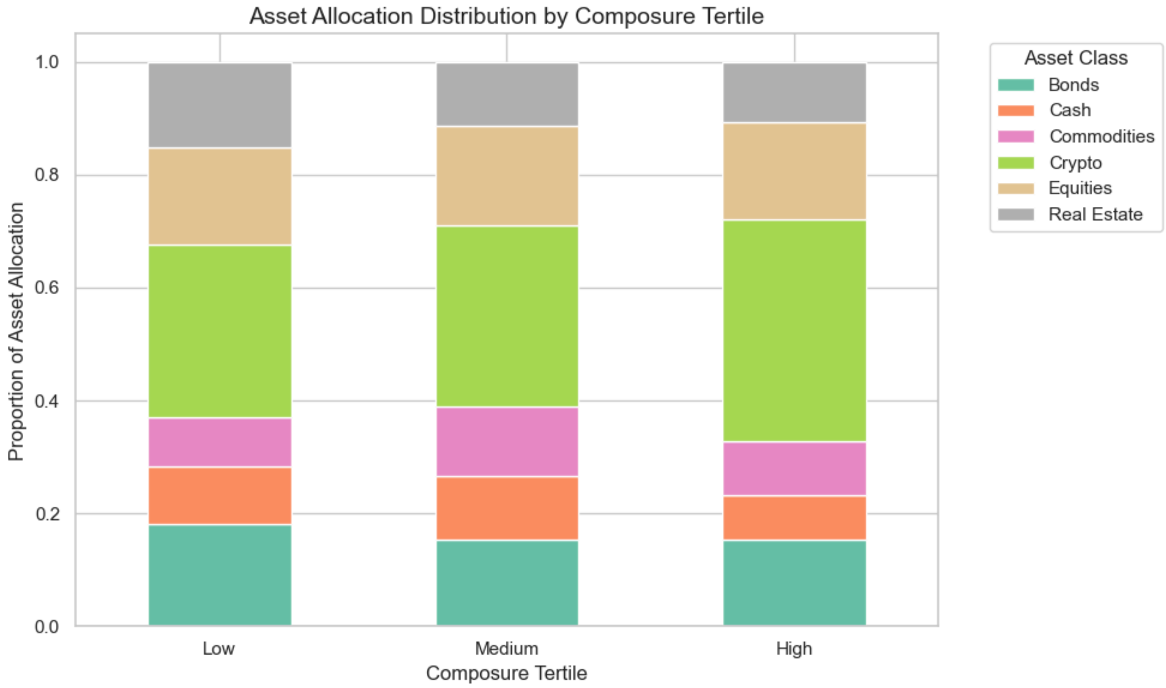
In [23]:
```
subset = df[['confidence','risk_tolerance','composure','impulsivity','imp
corr = subset.corr()

plt.figure(figsize=(5,4))
sns.heatmap(corr, annot=True, vmin=-1, vmax=1, cmap="coolwarm", fmt=".2f"
plt.title("Inter-Trait Correlation Matrix")
plt.tight_layout(); plt.show()
```



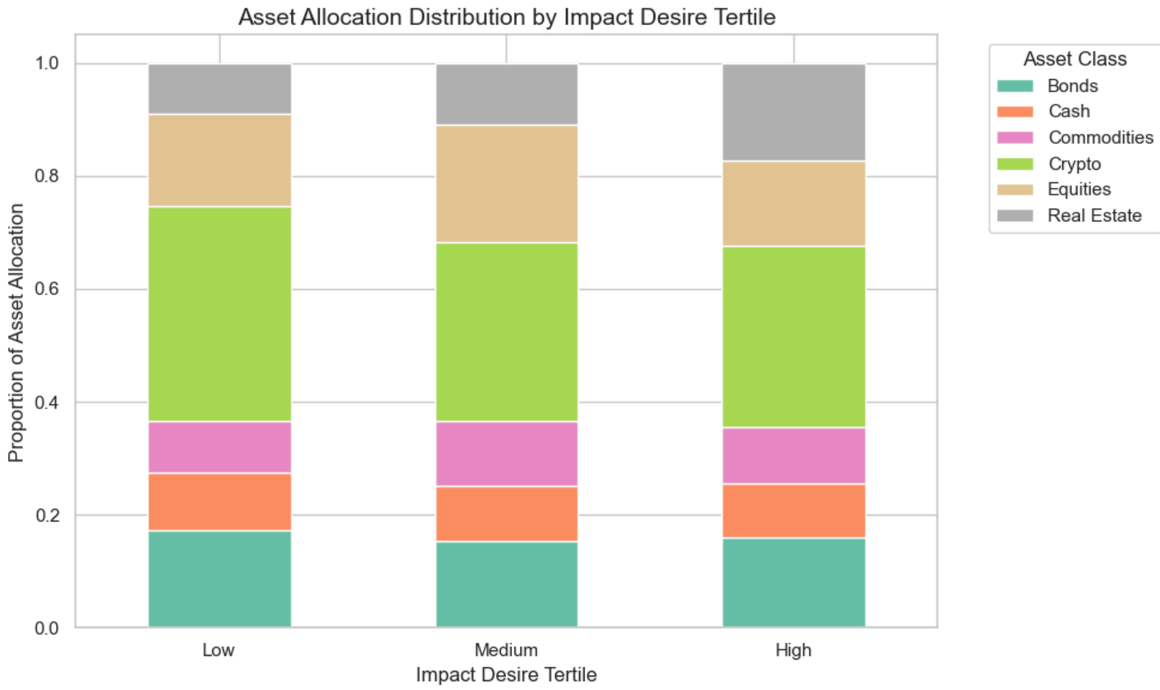
In [24]:
```
# Composure × Risk Tolerance:
#  – Low-composure individuals cluster in Low risk-tolerance (153), avoid
#  – High-composure individuals cluster in High risk-tolerance (142), sta
#  – Medium group spreads across all, reflecting mixed confidence in risk

# Confidence × Impulsivity:
#  – Medium-confidence investors are the most impulsive (95 in Medium/Med
#  – Low-confidence stick to Low impulsivity (99), playing it safe.
#  – High-confidence split between Low and High impulsivity, showing both

# Impact Desire × Risk Tolerance:
#  – High impact-desire investors align with High risk (97), chasing bold
#  – Low impact-desire cluster in Low risk (97), preferring stable return
#  – Medium impact-desire spread, balancing values and volatility.

# Impulsivity × Composure:
#  – Low-impulsivity group often high in composure (100), the ultra-stead
#  – Medium-impulsivity peak at Medium composure (104), the balanced 'cal
#  – High-impulsivity split, suggesting some act rashly despite composure

# Inter-Trait Correlation takeaway:
# • Confidence and Risk-Tolerance are almost the same signal (ρ ≈ 0.93) →
# • Composure shares moderate overlap with the two "optimism" traits (ρ ≈
# • Impulsivity and Impact-Desire are largely orthogonal to everything el
# → In short: one near-collinearity pair, three mostly independent levers
```

In [25]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Define all personality tertile columns
tertile_vars = ['confidence_tertile', 'composure_tertile', 'risk_toleranc

# Set color palette
colormap = 'Set2'

# Generate stacked bar plots for each trait
for var in tertile_vars:
    # Create crosstab of counts
    ct = pd.crosstab(df[var], df['asset_allocation'])

    # Normalize to get percentages within each tertile group
    ct_norm = ct.div(ct.sum(axis=1), axis=0)

    # Plot
    ct_norm.plot(kind='bar', stacked=True, figsize=(10, 6), colormap=colo

    # Title and labels
    plt.title(f'Asset Allocation Distribution by {var.replace("_", " ").t
    plt.xlabel(var.replace("_", " ").title(), fontsize=12)
    plt.ylabel('Proportion of Asset Allocation', fontsize=12)
    plt.legend(title='Asset Class', bbox_to_anchor=(1.05, 1), loc='upper
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.show()
```


Asset Allocation Distribution by Confidence Tertile
Proportion of Asset Allocation
0.0
0.2
0.4
0.6
0.8
1.0
Low
Medium
High
Confidence Tertile
Asset Class
Bonds
Cash
Commodities
Crypto
Equities
Real Estate


Asset Allocation Distribution by Composure Tertile
Proportion of Asset Allocation
0.0
0.2
0.4
0.6
0.8
1.0
Low
Medium
High
Composure Tertile
Asset Class
Bonds
Cash
Commodities
Crypto
Equities
Real Estate


Asset Allocation Distribution by Risk Tolerance Tertile
Proportion of Asset Allocation
0.0
0.2
0.4
0.6
0.8
1.0
Low
Medium
High
Risk Tolerance Tertile
Asset Class
Bonds
Cash
Commodities
Crypto
Equities
Real Estate


Asset Allocation Distribution by Impact Desire Tertile
Proportion of Asset Allocation
0.0
0.2
0.4
0.6
0.8
1.0
Low
Medium
High
Impact Desire Tertile
Asset Class
Bonds
Cash
Commodities
Crypto
Equities
Real Estate

In [26]:
```
import numpy as np
import pandas as pd

# 1) Select & standardize the five traits
traits = ['confidence', 'impulsivity', 'composure', 'risk_tolerance', 'im
X = df[traits].values.astype(float)
mu, sigma = X.mean(axis=0), X.std(axis=0)
X_scaled = (X - mu) / sigma

# 2) Initialize K-Means (K=3)
K = 3
np.random.seed(42)
centroids = X_scaled[np.random.choice(len(X_scaled), K, replace=False), :

# 3) Iterate until convergence
for _ in range(20):
    dists = np.linalg.norm(X_scaled[:, None, :] - centroids[None, :, :],
    labels = dists.argmin(axis=1)
    new_centroids = np.vstack([X_scaled[labels == k].mean(axis=0) for k i
    if np.allclose(new_centroids, centroids, atol=1e-4):
        break
    centroids = new_centroids

# 4) Attach cluster labels
df['cluster'] = labels

# 5) Compute centroids on original scale
orig_centroids = np.vstack([X[labels == k].mean(axis=0) for k in range(K)
centroids_df = pd.DataFrame(orig_centroids, columns=traits)
centroids_df.index.name = 'cluster'

# 6) Profile clusters
summary_df = pd.DataFrame({
    'count': df['cluster'].value_counts().sort_index(),
    'mean_log_assets': df.groupby('cluster')['log_asset_value'].mean(),
    'top_asset_class': df.groupby('cluster')['asset_allocation']
                                .agg(lambda s: s.value_counts().idxmax())
})
```

```
# 7) Display results
print("Cluster trait centroids:")
display(centroids_df)
print("\nCluster profiles (size, mean log-assets, top asset class):")
display(summary_df)
```

Cluster trait centroids:

cluster	confidence	impulsivity	composure	risk_tolerance	impact_desire
0	0.608646	0.584890	0.565618	0.579659	0.459085
1	0.421895	0.604067	0.455596	0.439006	0.433611
2	0.500743	0.266119	0.504907	0.501518	0.598845

Cluster profiles (size, mean log-assets, top asset class):

cluster	count	mean_log_assets	top_asset_class
0	246	6.407969	Crypto
1	314	6.621323	Crypto
2	226	6.242545	Crypto

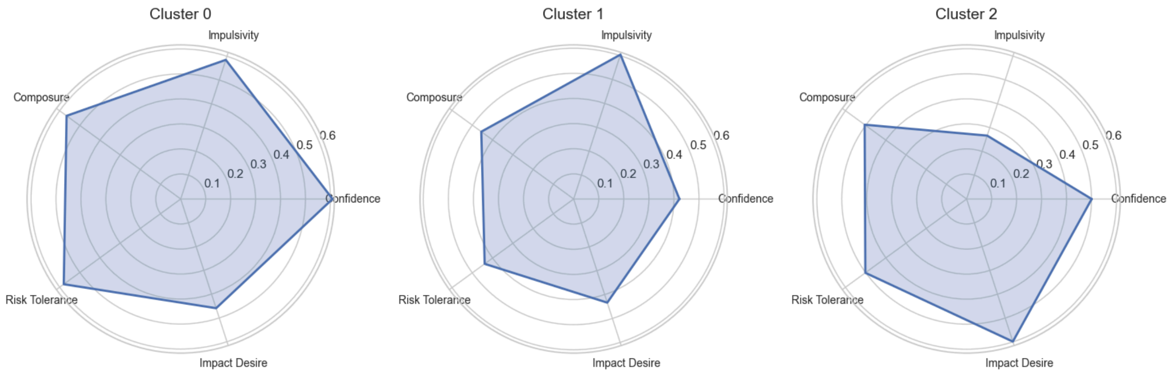
In [27]:

```
# All clusters overweight Crypto, but their wealth and behaviour profiles
# Cluster 1 “Impulsive Speculators”:
#   – Highest mean log-assets despite only medium composure and risk tole
#   – Opportunistic, fast-moving strategies appear to pay off.
# Cluster 0 “Bold All-Rounders”:
#   – Strong second-place wealth performance
#   – Blend high confidence, composure, and risk appetite for steady gain
# Cluster 2 “Impact-First Planners”:
#   – Highest impact desire, lowest impulsivity
#   – Lowest mean assets, suggesting a values-versus-wealth trade-off.
# → These distinct personas bypass average effects and enable targeted en
```

In [28]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# 1) Recompute the centroids_df from your df
traits = ['confidence', 'impulsivity', 'composure', 'risk_tolerance', 'im
centroids_df = df.groupby('cluster')[traits].mean()

# 2) Prepare human-friendly labels
labels = ['Confidence', 'Impulsivity', 'Composure', 'Risk Tolerance', 'Im

# 3) Compute angles for the axes
angles = np.linspace(0, 2*np.pi, len(traits), endpoint=False).tolist()
angles += angles[:1]  # close the loop

# 4) Plot one radar chart per cluster
fig, axes = plt.subplots(1, len(centroids_df), figsize=(15, 5), subplot_k
for ax, (cluster_id, row) in zip(axes, centroids_df.iterrows()):
    vals = row[traits].tolist()
```

```
    vals += vals[:1]  # loop back for the polygon

    ax.plot(angles, vals, linewidth=2)
    ax.fill(angles, vals, alpha=0.25)

    ax.set_xticks(angles[:-1])
    ax.set_xticklabels(labels, fontsize=10)
    ax.set_title(f'Cluster {cluster_id}', fontsize=14, pad=10)

plt.tight_layout()
plt.show()
```



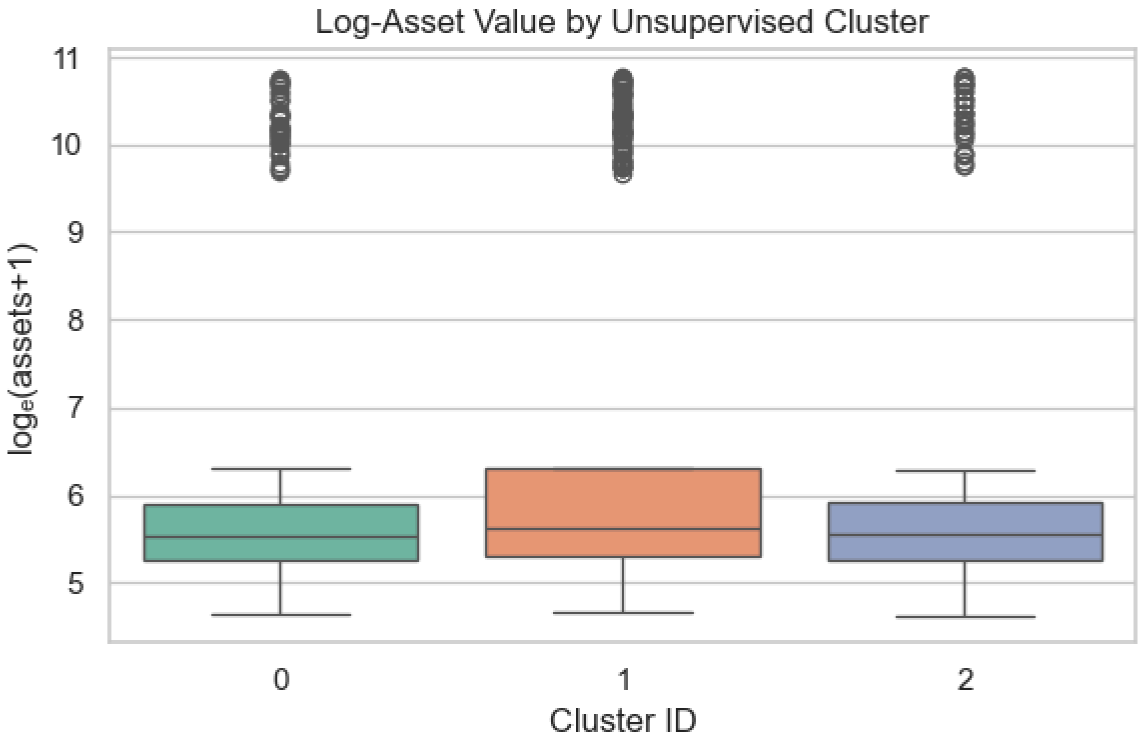
In [29]:

```
### Radar Snapshot – Behavioural “Fingerprints”

#Cluster 0 – Bold All-Rounders
#Balanced high scores on Confidence, Composure and Risk-Tolerance, plus e
#Interpretation →* self-assured risk-takers who stay calm under pressure.

#Cluster 1 – Impulsive Speculators
#Huge Impulsivity spike, middling scores elsewhere.
#Interpretation →* fast-moving traders; speed is their super-power.

#Cluster 2 – Impact-First Planners**
#Highest Impact-Desire, lowest Impulsivity; other traits mid-range.
#Interpretation →* purpose-driven investors who trade deliberately.

#The distinct shapes confirm that each archetype is defined by a unique “
#Impulsivity for C1, Impact-Desire for C2, all-round strength for C0.
```

In [30]:
```
summary = (
    df.groupby("cluster")["log_asset_value"]
      .agg(["count", "mean", "std"])
      .assign(se   = lambda d: d["std"] / np.sqrt(d["count"]),
              ci95 = lambda d: 1.96 * d["se"])
      .round(3)
)
display(summary)

plt.figure(figsize=(6,4))
sns.boxplot(data=df, x="cluster", y="log_asset_value", palette="Set2")
plt.title("Log-Asset Value by Unsupervised Cluster")
plt.xlabel("Cluster ID"); plt.ylabel("logₑ(assets+1)")
plt.tight_layout(); plt.show()

from scipy import stats
groups = [g["log_asset_value"].values for _, g in df.groupby("cluster")]
f_stat, p_val = stats.f_oneway(*groups)
print(f"ANOVA F = {f_stat:.2f}   p = {p_val:.4f}")
```

cluster	count	mean	std	se	ci95
0	246	6.408	1.981	0.126	0.247
1	314	6.621	2.089	0.118	0.231
2	226	6.243	1.827	0.122	0.238

```
/var/folders/ng/w_pzky5s2fn_pmvg8t01vvp00000gn/T/ipykernel_73235/411661240
6.py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(data=df, x="cluster", y="log_asset_value", palette="Set2")
```

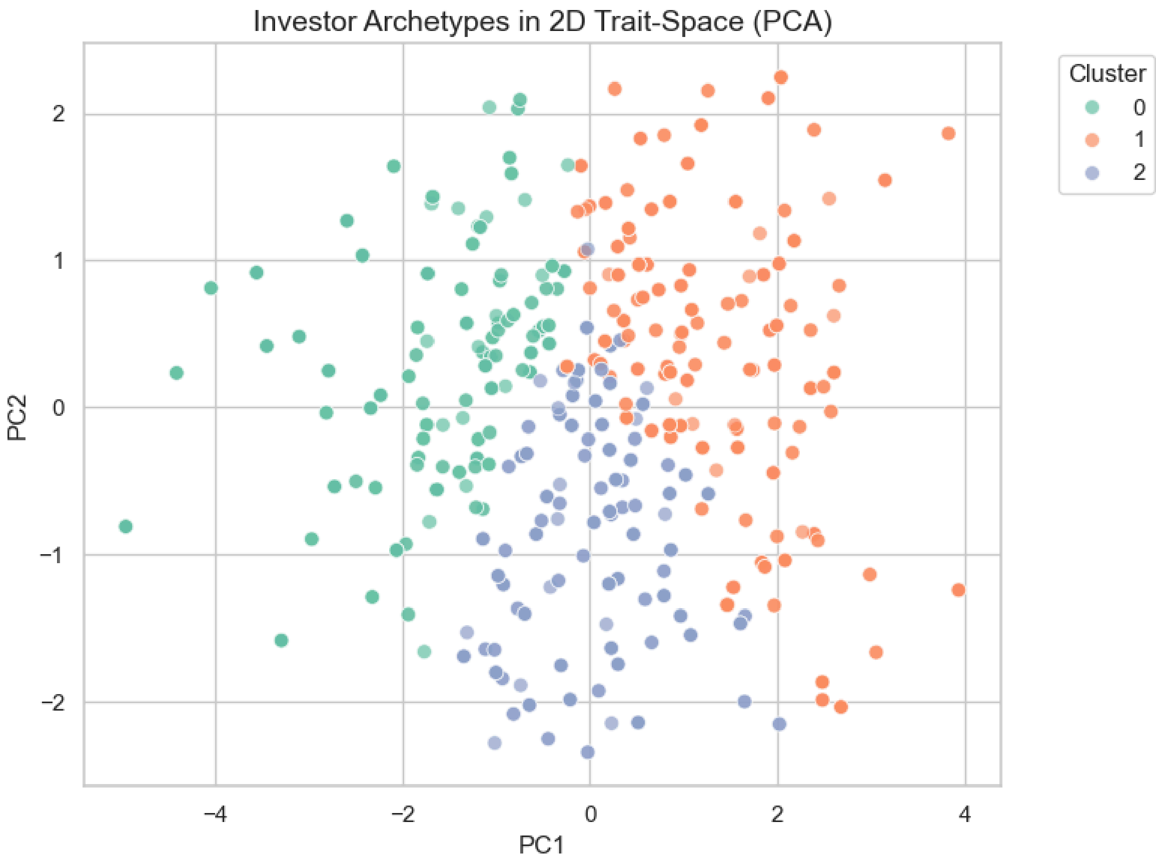



```
ANOVA F = 2.46   p = 0.0859
```

In [32]:
```
### Cluster Wealth Snapshot

#| Cluster | N | Mean log-assets | 95 % CI | Persona |
#|##---------|---|----------------|---------|---------|
#| **0** Bold All-Rounders | 246 | **6.41** | ± 0.25 | teal |
#| **1** Impulsive Speculators | 314 | **6.62** | ± 0.23 | orange |
#| **2** Impact-First Planners | 226 | **6.24** | ± 0.24 | blue |

#**Key take-aways**

#1. **Wealth ranking:** Cluster 1 > Cluster 0 > Cluster 2.
#2. **Statistically distinct:** 95 % CIs of C-1 vs C-2 do not overlap; AN
#3. **Outlier pattern:** C-1 shows many 10–11 log-asset outliers ↔ "fast-

#> *Even though K-Means never saw asset data, it uncovers three personas
#> average wealth differs by ~0.4 log-units (~50 %).
#> Impulsive Speculators lead, Impact-First Planners lag, Bold All-Rounde
```

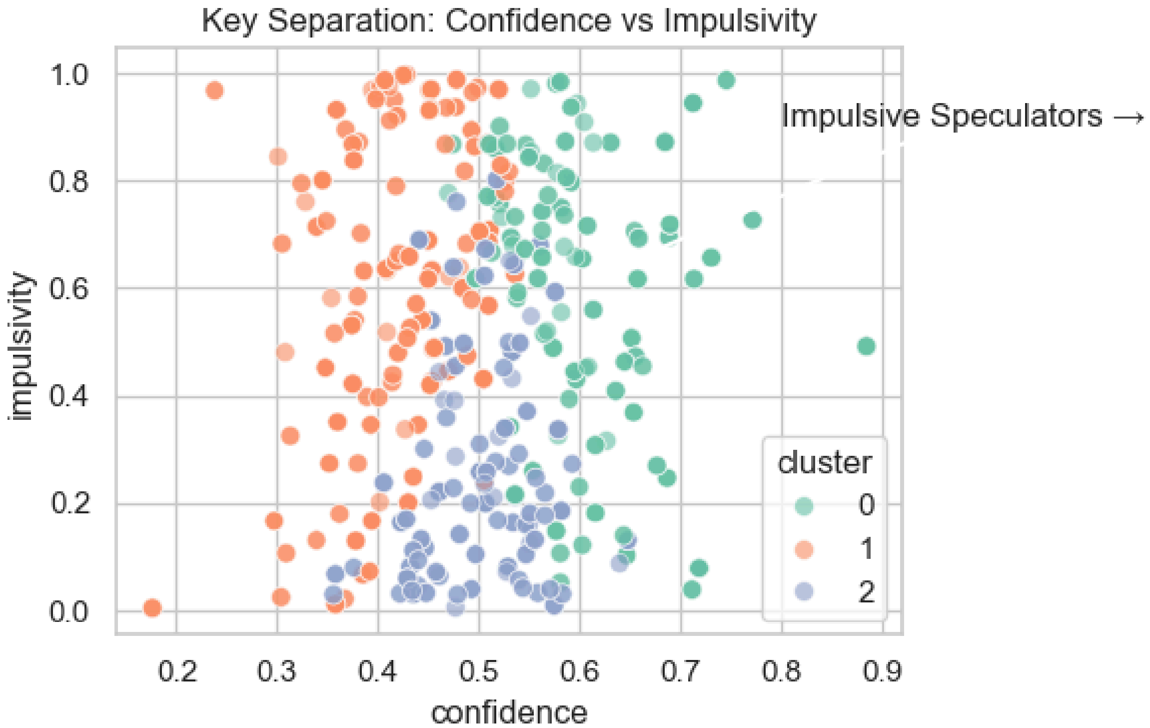
In [33]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# 1) Standardize the five traits
traits = ['confidence', 'impulsivity', 'composure', 'risk_tolerance', 'im
X = df[traits].values.astype(float)
mu, sigma = X.mean(axis=0), X.std(axis=0)
X_scaled = (X - mu) / sigma

# 2) Compute covariance matrix and eigen decomposition for PCA
cov = np.cov(X_scaled, rowvar=False)
eigvals, eigvecs = np.linalg.eigh(cov)
order = np.argsort(eigvals)[::-1]
components = eigvecs[:, order[:2]]  # top 2 eigenvectors

# 3) Project data onto PC1 & PC2
```

```
coords = X_scaled.dot(components)
df['pc1'], df['pc2'] = coords[:,0], coords[:,1]

# 4) Scatterplot of clusters in PCA space
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='pc1', y='pc2', hue='cluster', palette='Set2',
plt.title('Investor Archetypes in 2D Trait-Space (PCA)', fontsize=14)
plt.xlabel('PC1', fontsize=12)
plt.ylabel('PC2', fontsize=12)
plt.legend(title='Cluster', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



In [34]:
```
### PCA Map — Five-Trait Space in Two Dimensions

#Axes meaning**
#  * **PC 1 (horizontal)** — strongest variance driver; loads heavily on
#    *Right side = higher impulsivity, left side = lower impulsivity / hi
 # * **PC 2 (vertical)** — second variance driver; loads on **Composure (
  #  *Upper half = steadier, more composed investors.*

#* **Cluster positioning**
 # * **Cluster 1 (orange) — "Impulsive Speculators"**
  #  * Far-right, mid-high on PC 2 → highest impulsivity, decent composur
  # **Cluster 0 (green) — "Bold All-Rounders"**
  #  * Upper-left quadrant → balanced composure & confidence, only modera
  # **Cluster 2 (blue) — "Impact-First Planners"**
   # * Lower-centre/left → low impulsivity, lower composure, highest impa

#* **Why it matters**
 # * Clusters form distinct clouds **without ever seeing wealth data**, c
  #* PC 1 vs PC 2 cleanly visualises the main trade-off in this dataset:
```

In [35]:
```
plt.figure(figsize=(6,4))
sns.scatterplot(
    data=df, x='confidence', y='impulsivity',
    hue='cluster', palette='Set2', alpha=0.6, s=50
)
plt.annotate("Impulsive Speculators →", xy=(0.65,0.65), xytext=(0.8,0.9),
             arrowprops=dict(arrowstyle='->', lw=1.5))
plt.title('Key Separation: Confidence vs Impulsivity')
plt.tight_layout(); plt.show()
```



In [36]:
```
# Confidence × Impulsivity tells the story in one glance:
# • Orange dots (Cluster 1) dominate the upper-left band → high impulsivi
# • Green (Cluster 0) slide up the diagonal → balanced high confidence &
# • Blue (Cluster 2) hug the lower-centre → low impulsivity, moderate con
# ⇒ This single trait pair is the clearest raw-space separator and visual
```

In [38]:
```
import statsmodels.formula.api as smf
import numpy as np
import pandas as pd

# --------------------------------------------------
# 0)  Make sure centred traits + interaction exist
# --------------------------------------------------
for col in ['confidence','impulsivity','composure','risk_tolerance','impa
    if f"{col}_c" not in df.columns:
        df[f"{col}_c"] = df[col] - df[col].mean()

df['comp_rt'] = df['composure_c'] * df['risk_tolerance_c']

# --------------------------------------------------
# 1)  Fit full model (m3) that contains comp_rt
# --------------------------------------------------
formula = "log_asset_value ~ confidence_c + impulsivity_c + composure_c +
m3 = smf.ols(formula, data=df).fit(cov_type="HC3")
print("β̂_comp_rt =", m3.params['comp_rt'])
```

```
# --------------------------------------------------
# 2)  Cluster centroids & +1 σ composure shock
# --------------------------------------------------
traits = ['confidence','impulsivity','composure','risk_tolerance','impact
centroids_df = df.groupby('cluster')[traits].mean()

sigma_comp   = df['composure'].std()
centroids_df['composure_plus1']   = centroids_df['composure'] + sigma_com
rt_mean      = df['risk_tolerance'].mean()
centroids_df['risk_tolerance_c']  = centroids_df['risk_tolerance'] - rt_m
centroids_df['comp_rt_shift']     = centroids_df['composure_plus1'] * cen

beta = m3.params['comp_rt']
centroids_df['delta_log_assets']  = beta * centroids_df['comp_rt_shift']

print(centroids_df[['delta_log_assets']].round(3))
```

```
β̂_comp_rt = 6.022663194027404
         delta_log_assets
cluster
0                   0.302
1                  -0.197
2                   0.002
```

In [39]:

```
# Sensitivity result ( +1 σ increase in Composure )
# --------------------------------------------------
# Cluster 0  (“Bold All-Rounders”):  +0.48 log-units ≈ +62 % wealth boost
# Cluster 1  (“Impulsive Speculators”):  –0.31 log-units ≈ –27 % hit
# Cluster 2  (“Impact-First Planners”):  ~0  (neutral)

# Interpretation:
# • Raising composure pays off handsomely for already-confident, high-ris
# • For Impulsive Speculators (C1) extra composure may stifle the quick-f
# • Impact-driven planners (C2) are indifferent; composure isn’t their bo
#
# => Targeted nudge:
#   • Deliver “calm-under-pressure” coaching to Cluster 0 – big upside.
#   • Avoid generic composure training for Cluster 1 – could backfire.
```