

MSc Dissertation Statistical Analysis

Dissertation: *Stated Click-Through Intention Across Two FocusForm Advertisement Conditions: An Online Experiment Comparing AI-Generated and Human-Written Copy*

This notebook:

1. imports and cleans the Qualtrics export;
2. documents the analytical-sample flow;
3. constructs and checks the four three-item measures;
4. reports reliability, demographics and descriptive statistics;
5. tests H1 and H2 using two-sided Welch independent-samples t-tests;
6. tests H3 and H4 using multiple linear regression;
7. evaluates multicollinearity, residual assumptions, heteroscedasticity and influential observations;
8. uses HC3 heteroscedasticity-robust standard errors for final regression inference.

Interpretation boundaries

- The study measures stated click-through intention, not actual clicks or CTR.
- It compares one selected AI-generated copy execution with one selected human-written copy execution.
- Relevance and trust were measured rather than experimentally manipulated; their regression coefficients are associations, not causal effects.

```
In [1]: # Core packages

from pathlib import Path
from importlib.metadata import version as package_version
import warnings

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

from IPython.display import display
from scipy import stats

import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.stats.diagnostic import (
    het_breuschpagan,
    het_white,
    linear_reset
)
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.stats.stattools import jarque_bera

pd.set_option("display.max_columns", 100)
pd.set_option("display.width", 140)
```

```
In [2]: # Record package versions used when the notebook is executed
package_versions = pd.DataFrame({
    "Package": ["Python", "NumPy", "pandas", "SciPy", "statsmodels", "Matplotlib"],
    "Version": [
        __import__("sys").version.split()[0],
        package_version("numpy"),
        package_version("pandas"),
        package_version("scipy"),
        package_version("statsmodels"),
        package_version("matplotlib")
    ]
})

display(package_versions)
```

	Package	Version
0	Python	3.13.9
1	NumPy	2.3.5
2	pandas	2.3.3
3	SciPy	1.16.3
4	statsmodels	0.14.5
5	Matplotlib	3.10.6

1. Import the anonymous Qualtrics export

The notebook uses the anonymous Qualtrics re-export generated on 24 July 2026. To reproduce the dataset analysed for the dissertation, it retains the responses contained in the original analysis export generated on 15 July 2026 at 10:14. The later-recorded sessions in the re-export were incomplete and were not part of that original analysis export.

```
In [3]: # Locate the anonymous Qualtrics re-export

csv_path = Path(
    "Consumer Responses to Digital Advertising_July 24, 2026_14.50.csv"
)

if not csv_path.exists():
    raise FileNotFoundError(
        f"The Qualtrics CSV was not found: {csv_path}"
    )

df_reexport = pd.read_csv(
    csv_path,
    dtype="string",
    encoding="utf-8-sig"
)

print(f"Source file: {csv_path}")
print(f"Full re-export shape: {df_reexport.shape}")
```

```

# Confirm that no direct-identifier, IP-address or Location columns are present

direct_identifier_columns = [
    "IPAddress",
    "LocationLatitude",
    "LocationLongitude",
    "RecipientLastName",
    "RecipientFirstName",
    "RecipientEmail",
    "ExternalReference"
]

unexpected_identifier_columns = [
    column
    for column in direct_identifier_columns
    if column in df_reexport.columns
]

assert not unexpected_identifier_columns, (
    "Unexpected identifying columns found: "
    f"{unexpected_identifier_columns}"
)

# Reproduce the dataset contained in the original analysis export

analysis_export_cutoff = pd.Timestamp("2026-07-15 10:14:00")
recorded_at = pd.to_datetime(
    df_reexport["RecordedDate"],
    format="%Y-%m-%d %H:%M:%S",
    errors="coerce"
)

later_recorded_sessions = (
    recorded_at > analysis_export_cutoff
)

assert df_reexport.loc[
    later_recorded_sessions,
    "Finished"
].eq("0").all(), (
    "A completed response was recorded after the original analysis export."
)

df_raw = (
    df_reexport.loc[~later_recorded_sessions]
    .copy()
    .reset_index(drop=True)
)

print(f"Analysis-export shape: {df_raw.shape}")
print(
    "Later-recorded incomplete sessions excluded:",
    int(later_recorded_sessions.sum())
)

# Show only non-identifying fields in the structural preview
display(
    df_raw[
        ["Progress", "Finished", "condition"]
    ]
)

```

```
].head(3)
)
```

Source file: Consumer Responses to Digital Advertising_July 24, 2026_14.50.csv
 Full re-export shape: (122, 30)
 Analysis-export shape: (95, 30)
 Later-recorded incomplete sessions excluded: 27

	Progress	Finished	condition
0	Progress	Finished	condition
1	{"ImportId":"progress"}	{"ImportId":"finished"}	{"ImportId":"condition"}
2	100	1	ai

2. Remove Qualtrics metadata rows and retain analytical variables

The metadata rows are identified by their contents rather than removed solely by row number:

- the question-text row contains `Start Date` ;
- the import row contains Qualtrics `ImportId` values.

The survey was configured to collect anonymous responses. The exported file contains no names, email addresses, IP-address fields or location fields.

```
In [4]: # Identify Qualtrics metadata rows

question_text_row = df_raw["StartDate"].eq("Start Date")

import_id_row = df_raw.apply(
    lambda column: column.str.contains(
        "ImportId:",
        regex=False,
        na=False
    )
).any(axis=1)

metadata_mask = question_text_row | import_id_row

print("Metadata rows identified:", int(metadata_mask.sum()))
display(
    df_raw.loc[
        metadata_mask,
        ["StartDate", "Finished", "cti_1", "attn_check", "condition"]
    ]
)

assert metadata_mask.sum() == 2, (
    "Expected exactly two Qualtrics metadata rows. Inspect the export."
)

df = (
    df_raw.loc[~metadata_mask]
    .copy()
```

```
.reset_index(drop=True)
)
```

Metadata rows identified: 2

	StartDate	Finished	cti_1
0	Start Date	Finished	Please indicate how much you agree or disagree...
1	{"ImportId":"startDate","timeZone":"Europe/Ber..."}	{"ImportId":"finished"}	{"ImportId":"QID8_1"}

In [5]: *# Retain only variables required for cleaning, descriptives and analysis*

```
columns_to_keep = [
    "ResponseId",
    "Progress",
    "Duration (in seconds)",
    "Finished",
    "age_eligibility",
    "consent",
    "condition",
    "cti_1", "cti_2", "cti_3",
    "auth_1", "auth_2", "auth_3",
    "rel_1", "rel_2", "rel_3",
    "trust_1", "trust_2", "trust_3",
    "attn_check",
    "age",
    "gender",
    "prior_app_use"
]

missing_columns = sorted(set(columns_to_keep) - set(df.columns))

if missing_columns:
    raise KeyError(f"Required columns are missing: {missing_columns}")

df = df[columns_to_keep].copy()

# Remove leading/trailing whitespace from text values
for column in df.select_dtypes(include=["string", "object"]).columns:
    df[column] = df[column].str.strip()
```

3. Convert variables to appropriate data types

Pandas' nullable `Int64` type permits integer coding while retaining missing values. The one written age response is corrected before age is converted.

In [6]: *# Convert integer-coded variables*

```
numeric_columns = [
    "Progress",
    "Duration (in seconds)",
    "Finished",
    "age_eligibility",
    "consent",
```

```

    "cti_1", "cti_2", "cti_3",
    "auth_1", "auth_2", "auth_3",
    "rel_1", "rel_2", "rel_3",
    "trust_1", "trust_2", "trust_3",
    "attn_check",
    "gender",
    "prior_app_use"
]

for column in numeric_columns:
    df[column] = pd.to_numeric(
        df[column],
        errors="coerce"
    ).astype("Int64")

# Correct and convert age
df["age_raw"] = df["age"].copy()

df["age"] = (
    df["age"]
    .str.lower()
    .str.replace(r"\s+", " ", regex=True)
    .str.strip()
    .replace({
        "twenty four years": "24",
        "twenty-four years": "24",
        "twenty four": "24",
        "twenty-four": "24"
    })
)

df["age"] = pd.to_numeric(
    df["age"],
    errors="coerce"
).astype("Int64")

# Standardise the experimental-condition labels
df["condition"] = (
    df["condition"]
    .str.lower()
    .str.strip()
)

unexpected_conditions = set(df["condition"].dropna().unique()) - {"ai", "human"}

if unexpected_conditions:
    raise ValueError(
        f"Unexpected condition labels found: {unexpected_conditions}"
    )

df["condition"] = pd.Categorical(
    df["condition"],
    categories=["human", "ai"]
)

```

Demographic coding

Prior productivity-app use is coded as:

- 0 = No
- 1 = Yes
- 2 = Not sure
- 3 = Prefer not to say

Gender is coded as:

- 1 = Male
- 2 = Female
- 3 = Non-binary / third gender
- 4 = Prefer not to say

```
In [7]: PRIOR_APP_USE_MAP = {
    0: "No",
    1: "Yes",
    2: "Not sure",
    3: "Prefer not to say"
}

# Gender labels confirmed from the Qualtrics response coding
GENDER_MAP = {
    1: 'Male',
    2: 'Female',
    3: 'Non-binary / third gender',
    4: 'Prefer not to say'
}

df["prior_app_use_label"] = (
    df["prior_app_use"]
    .map(PRIOR_APP_USE_MAP)
    .astype("string")
)

df["gender_label"] = (
    df["gender"]
    .map(GENDER_MAP)
    .fillna(
        df["gender"].map(
            lambda value: f"Code {int(value)}" if pd.notna(value) else pd.NA
        )
    )
    .astype("string")
)
```

4. Apply the pre-specified exclusion rules

Exclusions are applied sequentially:

1. under 18 / age ineligible;
2. did not consent;
3. incomplete response;
4. failed the attention check (`attn_check != 2`).

No response is excluded based on duration, outliers or unusual response patterns.

```
In [8]: # Sequential cleaning stages

stage_metadata_removed = df.copy()

stage_age_eligible = stage_metadata_removed.loc[
    stage_metadata_removed["age_eligibility"].eq(1)
].copy()

stage_consented = stage_age_eligible.loc[
    stage_age_eligible["consent"].eq(1)
].copy()

stage_complete = stage_consented.loc[
    stage_consented["Finished"].eq(1)
    & stage_consented["Progress"].eq(100)
].copy()

df_analysis = stage_complete.loc[
    stage_complete["attn_check"].eq(2)
].copy().reset_index(drop=True)

cleaning_flow = pd.DataFrame({
    "Stage": [
        "Raw CSV rows",
        "Qualtrics metadata rows removed",
        "Under-18 / age-ineligible responses removed",
        "Non-consenting responses removed",
        "Incomplete responses removed",
        "Attention-check failures removed",
        "Final analytical sample"
    ],
    "Excluded at stage": [
        0,
        len(df_raw) - len(stage_metadata_removed),
        len(stage_metadata_removed) - len(stage_age_eligible),
        len(stage_age_eligible) - len(stage_consented),
        len(stage_consented) - len(stage_complete),
        len(stage_complete) - len(df_analysis),
        0
    ],
    "Remaining": [
        len(df_raw),
        len(stage_metadata_removed),
        len(stage_age_eligible),
        len(stage_consented),
    ]
})
```

```

        len(stage_complete),
        len(df_analysis),
        len(df_analysis)
    ]
})

display(cleaning_flow)

assert len(df_analysis) == 60, (
    f"Expected 60 final observations, found {len(df_analysis)}."
)

assert df_analysis["condition"].value_counts().to_dict() == {
    "human": 30,
    "ai": 30
}, "Expected a 30/30 condition split."

```

	Stage	Excluded at stage	Remaining
0	Raw CSV rows	0	95
1	Qualtrics metadata rows removed	2	93
2	Under-18 / age-ineligible responses removed	1	92
3	Non-consenting responses removed	1	91
4	Incomplete responses removed	20	71
5	Attention-check failures removed	11	60
6	Final analytical sample	0	60

```

In [9]: # Additional data-quality checks

quality_summary = pd.DataFrame({
    "Check": [
        "Final observations",
        "AI condition",
        "Human condition",
        "Duplicate ResponseIds",
        "Missing construct-item values",
        "Minimum duration (seconds)",
        "Median duration (seconds)",
        "Maximum duration (seconds)"
    ],
    "Value": [
        len(df_analysis),
        int((df_analysis["condition"] == "ai").sum()),
        int((df_analysis["condition"] == "human").sum()),
        int(df_analysis["ResponseId"].duplicated().sum()),
        int(
            df_analysis[
                [
                    "cti_1", "cti_2", "cti_3",
                    "auth_1", "auth_2", "auth_3",
                    "rel_1", "rel_2", "rel_3",
                    "trust_1", "trust_2", "trust_3"
                ]
            ].isna().sum().sum()
        )
    ]
})

```

```

    ),
    int(df_analysis["Duration (in seconds)"].min()),
    float(df_analysis["Duration (in seconds)"].median()),
    int(df_analysis["Duration (in seconds)"].max())
]
})

display(quality_summary)

print(
    "No duration exclusion was applied. The maximum duration is retained "
    "because a long duration alone does not demonstrate an invalid response."
)

```

	Check	Value
0	Final observations	60.0
1	AI condition	30.0
2	Human condition	30.0
3	Duplicate Responselds	0.0
4	Missing construct-item values	0.0
5	Minimum duration (seconds)	97.0
6	Median duration (seconds)	175.5
7	Maximum duration (seconds)	88163.0

No duration exclusion was applied. The maximum duration is retained because a long duration alone does not demonstrate an invalid response.

5. Construct the four measures

Each score is the row-wise mean of its three pre-specified items. `skipna=False` ensures that a composite cannot be calculated from fewer than all three items.

```

In [10]: construct_items = {
    "cti": ["cti_1", "cti_2", "cti_3"],
    "auth": ["auth_1", "auth_2", "auth_3"],
    "rel": ["rel_1", "rel_2", "rel_3"],
    "trust": ["trust_1", "trust_2", "trust_3"]
}

all_items = [
    item
    for items in construct_items.values()
    for item in items
]

# Confirm all item responses use the intended 1-7 scale
item_range_check = pd.DataFrame({
    "Minimum": df_analysis[all_items].min(),
    "Maximum": df_analysis[all_items].max(),
    "Missing": df_analysis[all_items].isna().sum()
})

```

```

display(item_range_check)

assert (
    df_analysis[all_items]
    .apply(lambda column: column.dropna().between(1, 7).all())
    .all()
), "At least one construct item falls outside the expected 1-7 scale."

for construct, items in construct_items.items():
    df_analysis[construct] = (
        df_analysis[items]
        .mean(axis=1, skipna=False)
    )

assert (
    df_analysis[list(construct_items)]
    .apply(lambda column: column.dropna().between(1, 7).all())
    .all()
), "At least one composite score falls outside the 1-7 scale."

display(
    df_analysis[
        ["cti_1", "cti_2", "cti_3", "cti"]
    ].head()
)

```

	Minimum	Maximum	Missing
cti_1	1	7	0
cti_2	1	7	0
cti_3	1	7	0
auth_1	1	7	0
auth_2	1	7	0
auth_3	1	7	0
rel_1	1	7	0
rel_2	1	7	0
rel_3	2	7	0
trust_1	1	7	0
trust_2	1	7	0
trust_3	1	7	0

	cti_1	cti_2	cti_3	cti
0	5	5	6	5.333333
1	5	6	3	4.666667
2	5	5	6	5.333333
3	7	7	7	7.0
4	6	6	6	6.0

6. Reliability analysis

Cronbach's alpha is calculated separately for each three-item construct. Alpha assesses internal consistency, not construct validity or unidimensionality.

```
In [11]: def cronbach_alpha(items: pd.DataFrame) -> float:
    """Calculate Cronbach's alpha using complete rows and sample variances."""
    complete_items = items.dropna(axis=0, how="any")
    k = complete_items.shape[1]

    if k < 2:
        raise ValueError("Cronbach's alpha requires at least two items.")

    item_variances = complete_items.var(axis=0, ddof=1)
    total_score = complete_items.sum(axis=1)
    total_variance = total_score.var(ddof=1)

    if total_variance == 0:
        return np.nan

    return (k / (k - 1)) * (
        1 - item_variances.sum() / total_variance
    )

construct_labels = {
    "cti": "Click-through intention",
    "auth": "Perceived authenticity",
    "rel": "Perceived relevance",
    "trust": "Trust"
}

reliability_rows = []

for construct, items in construct_items.items():
    reliability_rows.append({
        "Construct": construct_labels[construct],
        "Items": len(items),
        "n": df_analysis[items].dropna().shape[0],
        "Cronbach's alpha": cronbach_alpha(df_analysis[items])
    })

reliability_table = pd.DataFrame(reliability_rows)

display(
    reliability_table.style.format({
```

```

        "Cronbach's alpha": "{:.3f}"
    })
)

```

	Construct	Items	n	Cronbach's alpha
0	Click-through intention	3	60	0.935
1	Perceived authenticity	3	60	0.897
2	Perceived relevance	3	60	0.915
3	Trust	3	60	0.873

```

In [12]: # Item-level descriptive statistics

item_descriptives = (
    df_analysis[all_items]
    .agg(["count", "mean", "std", "min", "max"])
    .T
    .rename(columns={
        "count": "n",
        "mean": "Mean",
        "std": "SD",
        "min": "Minimum",
        "max": "Maximum"
    })
)

display(
    item_descriptives.style.format({
        "n": "{:.0f}",
        "Mean": "{:.2f}",
        "SD": "{:.2f}",
        "Minimum": "{:.0f}",
        "Maximum": "{:.0f}"
    })
)

```

	n	Mean	SD	Minimum	Maximum
cti_1	60	4.13	1.87	1	7
cti_2	60	4.48	1.91	1	7
cti_3	60	3.87	1.93	1	7
auth_1	60	4.82	1.49	1	7
auth_2	60	4.85	1.40	1	7
auth_3	60	5.00	1.40	1	7
rel_1	60	4.25	1.70	1	7
rel_2	60	4.20	1.75	1	7
rel_3	60	4.63	1.48	2	7
trust_1	60	5.05	1.33	1	7
trust_2	60	4.85	1.39	1	7
trust_3	60	5.07	1.23	1	7

7. Sample demographics

The table reports age descriptives and categorical distributions for condition, gender codes and prior productivity-app use.

```
In [13]: # Age summary

age_summary = pd.DataFrame({
    "Statistic": [
        "n", "Mean", "SD", "Median", "Minimum", "Maximum"
    ],
    "Value": [
        df_analysis["age"].count(),
        df_analysis["age"].mean(),
        df_analysis["age"].std(ddof=1),
        df_analysis["age"].median(),
        df_analysis["age"].min(),
        df_analysis["age"].max()
    ]
})

display(
    age_summary.style.format({
        "Value": "{:.2f}"
    })
)
```

	Statistic	Value
0	n	60.00
1	Mean	30.65
2	SD	11.83
3	Median	26.00
4	Minimum	22.00
5	Maximum	64.00

```
In [14]: def categorical_distribution(
    data: pd.DataFrame,
    column: str,
    variable_name: str
) -> pd.DataFrame:
    counts = data[column].value_counts(dropna=False)
    percentages = data[column].value_counts(
        dropna=False,
        normalize=True
    ) * 100

    result = pd.DataFrame({
        "Variable": variable_name,
        "Category": counts.index.astype(str),
        "n": counts.values,
        "Percentage": percentages.reindex(counts.index).values
    })

    return result

condition_distribution = categorical_distribution(
    df_analysis.assign(
        condition_label=df_analysis["condition"].map({
            "human": "Human-written copy",
            "ai": "AI-generated copy"
        })
    ),
    "condition_label",
    "Experimental condition"
)

gender_distribution = categorical_distribution(
    df_analysis,
    "gender_label",
    "Gender"
)

prior_use_distribution = categorical_distribution(
    df_analysis,
    "prior_app_use_label",
    "Prior productivity-app use"
)

demographic_distribution = pd.concat(
    [
```

```

        condition_distribution,
        gender_distribution,
        prior_use_distribution
    ],
    ignore_index=True
)

display(
    demographic_distribution.style.format({
        "n": "{:.0f}",
        "Percentage": "{:.1f}%"
    })
)

```

	Variable	Category	n	Percentage
0	Experimental condition	Human-written copy	30	50.0%
1	Experimental condition	AI-generated copy	30	50.0%
2	Gender	Male	33	55.0%
3	Gender	Female	27	45.0%
4	Prior productivity-app use	No	36	60.0%
5	Prior productivity-app use	Yes	20	33.3%
6	Prior productivity-app use	Not sure	4	6.7%

8. Overall and condition-level descriptive statistics

```

In [15]: measures = ["cti", "auth", "rel", "trust"]

overall_descriptives = (
    df_analysis[measures]
    .agg(["count", "mean", "std", "min", "max"])
    .T
    .reset_index()
    .rename(columns={
        "index": "Measure",
        "count": "n",
        "mean": "Mean",
        "std": "SD",
        "min": "Minimum",
        "max": "Maximum"
    })
)

overall_descriptives["Measure"] = (
    overall_descriptives["Measure"]
    .map(construct_labels)
)

display(
    overall_descriptives.style.format({
        "n": "{:.0f}",
        "Mean": "{:.2f}",
        "SD": "{:.2f}",

```

```

        "Minimum": "{:.2f}",
        "Maximum": "{:.2f}"
    })
)

```

	Measure	n	Mean	SD	Minimum	Maximum
0	Click-through intention	60	4.16	1.79	1.00	7.00
1	Perceived authenticity	60	4.89	1.30	1.00	7.00
2	Perceived relevance	60	4.36	1.53	1.67	7.00
3	Trust	60	4.99	1.18	1.00	7.00

In [16]: *# Long-format dissertation-ready descriptive table*

```

condition_descriptives = []

for condition_value, condition_label in [
    ("human", "Human-written copy"),
    ("ai", "AI-generated copy")
]:
    subset = df_analysis.loc[
        df_analysis["condition"] == condition_value
    ]

    for measure in measures:
        condition_descriptives.append({
            "Measure": construct_labels[measure],
            "Group": condition_label,
            "n": subset[measure].count(),
            "Mean": subset[measure].mean(),
            "SD": subset[measure].std(ddof=1),
            "Minimum": subset[measure].min(),
            "Maximum": subset[measure].max()
        })

condition_descriptives = pd.DataFrame(
    condition_descriptives
)

display(
    condition_descriptives.style.format({
        "n": "{:.0f}",
        "Mean": "{:.2f}",
        "SD": "{:.2f}",
        "Minimum": "{:.2f}",
        "Maximum": "{:.2f}"
    })
)

```

	Measure	Group	n	Mean	SD	Minimum	Maximum
0	Click-through intention	Human-written copy	30	4.07	1.65	1.00	6.67
1	Perceived authenticity	Human-written copy	30	4.97	1.42	1.00	7.00
2	Perceived relevance	Human-written copy	30	4.22	1.49	2.00	6.33
3	Trust	Human-written copy	30	4.97	1.25	1.00	7.00
4	Click-through intention	AI-generated copy	30	4.26	1.94	1.00	7.00
5	Perceived authenticity	AI-generated copy	30	4.81	1.19	2.00	6.67
6	Perceived relevance	AI-generated copy	30	4.50	1.58	1.67	7.00
7	Trust	AI-generated copy	30	5.01	1.13	2.00	6.67

In [17]: *# Pearson correlation matrix for the four composite measures*

```
correlation_matrix = df_analysis[
    ["cti", "auth", "rel", "trust"]
].corr(method="pearson")

correlation_matrix.index = [
    construct_labels[index]
    for index in correlation_matrix.index
]

correlation_matrix.columns = [
    construct_labels[column]
    for column in correlation_matrix.columns
]

display(correlation_matrix.style.format("{:.3f}"))
```

	Click-through intention	Perceived authenticity	Perceived relevance	Trust
Click-through intention	1.000	0.531	0.465	0.449
Perceived authenticity	0.531	1.000	0.049	0.764
Perceived relevance	0.465	0.049	1.000	0.003
Trust	0.449	0.764	0.003	1.000

9. H1 and H2: Welch independent-samples t-tests

Both hypotheses are tested using two-sided Welch tests at ($\alpha = .05$). Differences are defined as AI minus human. Hedges' (g) reports the standardised magnitude and direction of each observed difference.

In [18]: **def** welch_group_comparison(
 data: pd.DataFrame,
 outcome: str,

```

outcome_label: str
) -> dict:
    ai = data.loc[
        data["condition"] == "ai",
        outcome
    ].dropna()

    human = data.loc[
        data["condition"] == "human",
        outcome
    ].dropna()

    n_ai = ai.count()
    n_human = human.count()

    mean_ai = ai.mean()
    mean_human = human.mean()

    sd_ai = ai.std(ddof=1)
    sd_human = human.std(ddof=1)

    mean_difference = mean_ai - mean_human

    t_statistic, p_value = stats.ttest_ind(
        ai,
        human,
        equal_var=False,
        alternative="two-sided"
    )

    standard_error = np.sqrt(
        sd_ai**2 / n_ai
        + sd_human**2 / n_human
    )

    welch_df = (
        (sd_ai**2 / n_ai + sd_human**2 / n_human) ** 2
        /
        (
            (sd_ai**2 / n_ai) ** 2 / (n_ai - 1)
            + (sd_human**2 / n_human) ** 2 / (n_human - 1)
        )
    )

    critical_t = stats.t.ppf(
        0.975,
        df=welch_df
    )

    ci_lower = mean_difference - critical_t * standard_error
    ci_upper = mean_difference + critical_t * standard_error

    pooled_sd = np.sqrt(
        (
            (n_ai - 1) * sd_ai**2
            + (n_human - 1) * sd_human**2
        )
        / (n_ai + n_human - 2)
    )

```

```

cohens_d = mean_difference / pooled_sd

correction_factor = 1 - (
    3 / (4 * (n_ai + n_human) - 9)
)

hedges_g = correction_factor * cohens_d

return {
    "Outcome": outcome_label,
    "AI n": n_ai,
    "AI Mean": mean_ai,
    "AI SD": sd_ai,
    "Human n": n_human,
    "Human Mean": mean_human,
    "Human SD": sd_human,
    "Mean difference (AI - human)": mean_difference,
    "CI lower": ci_lower,
    "CI upper": ci_upper,
    "Welch t": t_statistic,
    "df": welch_df,
    "p": p_value,
    "Cohen's d": cohens_d,
    "Hedges' g": hedges_g
}

h1_result = welch_group_comparison(
    df_analysis,
    "cti",
    "Click-through intention"
)

h2_result = welch_group_comparison(
    df_analysis,
    "auth",
    "Perceived authenticity"
)

ttest_results = pd.DataFrame([
    h1_result,
    h2_result
])

display(
    ttest_results.style.format({
        "AI n": "{:.0f}",
        "AI Mean": "{:.2f}",
        "AI SD": "{:.2f}",
        "Human n": "{:.0f}",
        "Human Mean": "{:.2f}",
        "Human SD": "{:.2f}",
        "Mean difference (AI - human)": "{:.2f}",
        "CI lower": "{:.2f}",
        "CI upper": "{:.2f}",
        "Welch t": "{:.2f}",
        "df": "{:.2f}",
        "p": "{:.3f}",
        "Cohen's d": "{:.3f}",
        "Hedges' g": "{:.3f}"
    })
)

```

```

    })
)

```

	Outcome	AI n	AI Mean	AI SD	Human n	Human Mean	Human SD	Mean difference (AI - human)	CI lower	CI upper	Welch's
0	Click-through intention	30	4.26	1.94	30	4.07	1.65	0.19	-0.74	1.12	0.4
1	Perceived authenticity	30	4.81	1.19	30	4.97	1.42	-0.16	-0.83	0.52	-0.4

In [19]: *# Distribution summaries by outcome and condition*

```

distribution_checks = []

for outcome in ["cti", "auth"]:
    for condition_value in ["human", "ai"]:
        values = df_analysis.loc[
            df_analysis["condition"] == condition_value,
            outcome
        ]

        distribution_checks.append({
            "Outcome": construct_labels[outcome],
            "Condition": condition_value,
            "n": values.count(),
            "Skewness": stats.skew(
                values,
                bias=False
            ),
            "Excess kurtosis": stats.kurtosis(
                values,
                fisher=True,
                bias=False
            ),
            "Variance": values.var(ddof=1)
        })

distribution_checks = pd.DataFrame(
    distribution_checks
)

display(
    distribution_checks.style.format({
        "n": "{:.0f}",
        "Skewness": "{:.3f}",
        "Excess kurtosis": "{:.3f}",
        "Variance": "{:.3f}"
    })
)

print(
    "Welch's test is the primary analysis and does not require equal "
    "group variances. With 30 observations per balanced group, it is "

```

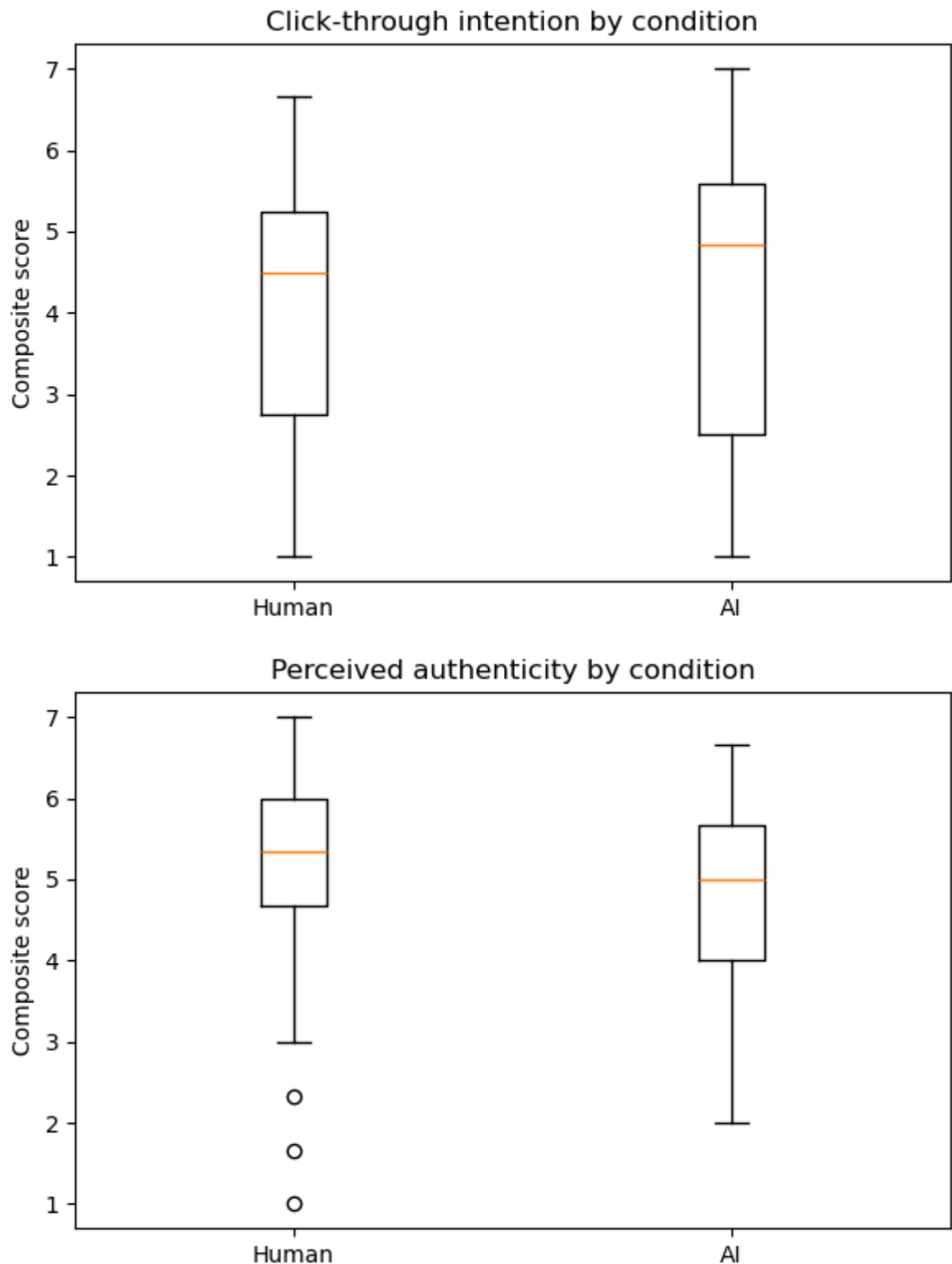
```
"reasonably robust to moderate deviations from normality."  
)
```

	Outcome	Condition	n	Skewness	Excess kurtosis	Variance
0	Click-through intention	human	30	-0.442	-0.789	2.723
1	Click-through intention	ai	30	-0.310	-1.161	3.775
2	Perceived authenticity	human	30	-1.282	1.593	2.026
3	Perceived authenticity	ai	30	-0.811	-0.053	1.415

Welch's test is the primary analysis and does not require equal group variances. With 30 observations per balanced group, it is reasonably robust to moderate deviations from normality.

```
In [20]: # Optional visual checks for H1 and H2 distributions
```

```
for outcome, title in [  
    ("cti", "Click-through intention by condition"),  
    ("auth", "Perceived authenticity by condition")  
]:  
    human_values = df_analysis.loc[  
        df_analysis["condition"] == "human",  
        outcome  
    ]  
  
    ai_values = df_analysis.loc[  
        df_analysis["condition"] == "ai",  
        outcome  
    ]  
  
    plt.figure(figsize=(6, 4))  
    plt.boxplot(  
        [human_values, ai_values],  
        tick_labels=["Human", "AI"]  
    )  
    plt.ylabel("Composite score")  
    plt.title(title)  
    plt.tight_layout()  
    plt.show()
```



H1 and H2 conclusions

- **H1 was not supported.** The study found no statistically significant evidence that stated click-through intention differed between the two selected copy conditions. This does not demonstrate that the population effect is exactly zero.
- **H2 was not supported.** The study found no statistically significant evidence that perceived authenticity differed between the two selected copy conditions.

10. H3 and H4: Multiple linear regression

The model is:

$$[CTI = \beta_0 + \beta_1(\text{Relevance}) + \beta_2(\text{Trust}) + \beta_3(\text{Condition}) + \epsilon]$$

Condition is coded 0 = human and 1 = AI. The relevance and trust coefficients are interpreted as associations while holding the other predictors constant.

```
In [21]: # Code condition for regression

df_analysis["condition_code"] = (
    df_analysis["condition"]
    .map({
        "human": 0,
        "ai": 1
    })
    .astype("Int64")
)

display(
    df_analysis[
        ["condition", "condition_code"]
    ].drop_duplicates()
)
```

	condition	condition_code
0	ai	1
1	human	0

```
In [22]: # Fit conventional OLS model for model diagnostics

ols_model = smf.ols(
    "cti ~ rel + trust + condition_code",
    data=df_analysis
).fit()

print(ols_model.summary())
```

OLS Regression Results

Dep. Variable:	cti	R-squared:	0.416
Model:	OLS	Adj. R-squared:	0.385
Method:	Least Squares	F-statistic:	13.31
Date:	Fri, 24 Jul 2026	Prob (F-statistic):	1.14e-06
Time:	15:23:07	Log-Likelihood:	-103.41
No. Observations:	60	AIC:	214.8
Df Residuals:	56	BIC:	223.2
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
Intercept	-1.5979	0.956	-1.671	0.100	-3.514	0.318
rel	0.5427	0.120	4.516	0.000	0.302	0.783
trust	0.6792	0.155	4.380	0.000	0.369	0.990
condition_code	0.0080	0.364	0.022	0.983	-0.721	0.737

Omnibus:	3.498	Durbin-Watson:	1.904
Prob(Omnibus):	0.174	Jarque-Bera (JB):	2.758
Skew:	0.303	Prob(JB):	0.252
Kurtosis:	3.858	Cond. No.	36.6

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [23]: `# Conventional OLS coefficient and model-fit tables`

```
ols_ci = ols_model.conf_int()

ols_coefficients = pd.DataFrame({
    "B": ols_model.params,
    "SE": ols_model.bse,
    "t": ols_model.tvalues,
    "p": ols_model.pvalues,
    "CI lower": ols_ci[0],
    "CI upper": ols_ci[1]
})

ols_model_fit = pd.DataFrame({
    "n": [int(ols_model.nobs)],
    "R-squared": [ols_model.rsquared],
    "Adjusted R-squared": [ols_model.rsquared_adj],
    "F": [ols_model.fvalue],
    "df model": [int(ols_model.df_model)],
    "df residual": [int(ols_model.df_resid)],
    "Model p": [ols_model.f_pvalue]
})
```

```
display(ols_coefficients.style.format("{:.3f}"))
display(ols_model_fit.style.format("{:.3f}"))
```

	B	SE	t	p	CI lower	CI upper
Intercept	-1.598	0.956	-1.671	0.100	-3.514	0.318
rel	0.543	0.120	4.516	0.000	0.302	0.783
trust	0.679	0.155	4.380	0.000	0.369	0.990
condition_code	0.008	0.364	0.022	0.983	-0.721	0.737

	n	R-squared	Adjusted R-squared	F	df model	df residual	Model p
0	60.000	0.416	0.385	13.306	3.000	56.000	0.000

11. Multicollinearity

```
In [24]: X = df_analysis[
    ["rel", "trust", "condition_code"]
].astype(float)

X_with_constant = sm.add_constant(X)

vif_results = pd.DataFrame({
    "Predictor": X_with_constant.columns,
    "VIF": [
        variance_inflation_factor(
            X_with_constant.values,
            index
        )
        for index in range(X_with_constant.shape[1])
    ]
})

display(vif_results.style.format({"VIF": "{:.3f}"}))

print(
    "Interpret the predictor VIFs only; the intercept VIF is not "
    "substantively meaningful."
)
```

	Predictor	VIF
0	const	27.859
1	rel	1.008
2	trust	1.000
3	condition_code	1.009

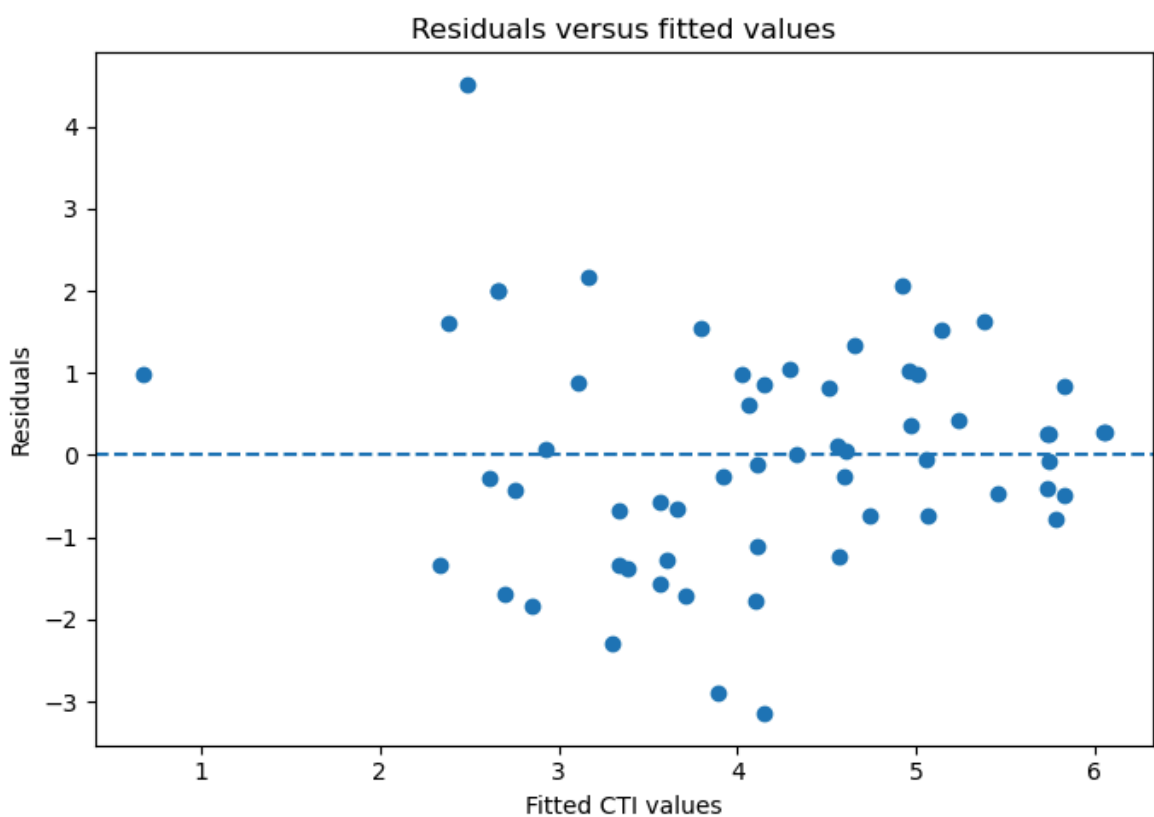
Interpret the predictor VIFs only; the intercept VIF is not substantively meaningful.

12. Regression diagnostics

Visual checks are considered alongside formal tests. Formal normality tests are not treated as pass/fail rules.

```
In [25]: # Residuals versus fitted values

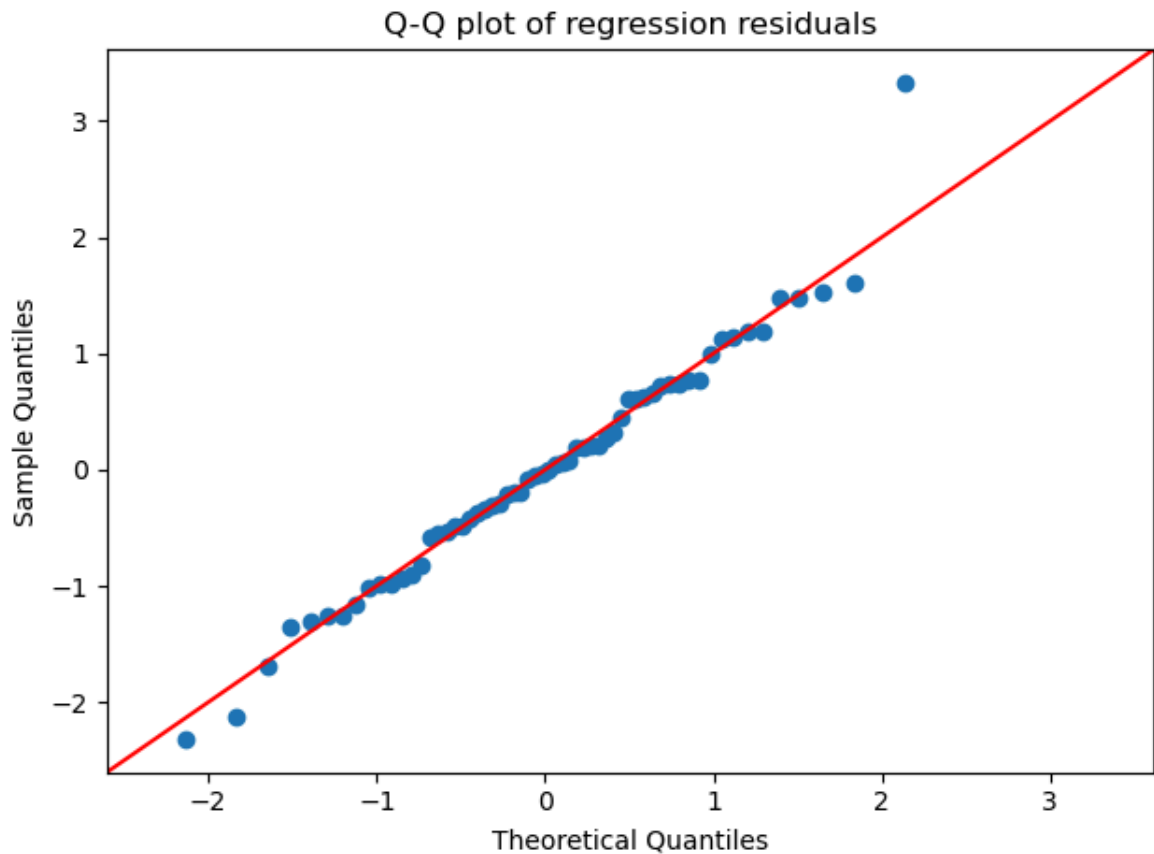
plt.figure(figsize=(7, 5))
plt.scatter(
    ols_model.fittedvalues,
    ols_model.resid
)
plt.axhline(0, linestyle="--")
plt.xlabel("Fitted CTI values")
plt.ylabel("Residuals")
plt.title("Residuals versus fitted values")
plt.tight_layout()
plt.show()
```



```
In [26]: # Normal Q-Q plot

sm.qqplot(
    ols_model.resid,
    line="45",
    fit=True
)

plt.title("Q-Q plot of regression residuals")
plt.tight_layout()
plt.show()
```



```
In [27]: # Formal diagnostic tests

bp_lm, bp_lm_p, bp_f, bp_f_p = het_breuschpagan(
    ols_model.resid,
    ols_model.model.exog
)

white_lm, white_lm_p, white_f, white_f_p = het_white(
    ols_model.resid,
    ols_model.model.exog
)

jb_stat, jb_p, residual_skew, residual_kurtosis = jarque_bera(
    ols_model.resid
)

reset_result = linear_reset(
    ols_model,
    power=2,
    use_f=True
)

diagnostic_tests = pd.DataFrame({
    "Diagnostic": [
        "Breusch-Pagan LM",
        "Breusch-Pagan F",
        "White LM",
        "White F",
        "Jarque-Bera residual normality",
        "Ramsey RESET"
    ],
    "Statistic": [
        bp_lm,
```

```

        bp_f,
        white_lm,
        white_f,
        jb_stat,
        float(reset_result.fvalue)
    ],
    "p": [
        bp_lm_p,
        bp_f_p,
        white_lm_p,
        white_f_p,
        jb_p,
        float(reset_result.pvalue)
    ]
})

display(
    diagnostic_tests.style.format({
        "Statistic": "{:.3f}",
        "p": "{:.3f}"
    })
)

print(
    "The heteroscedasticity tests indicate non-constant residual "
    "variance. HC3 robust standard errors are therefore used for "
    "the final coefficient inference."
)

```

	Diagnostic	Statistic	p
0	Breusch-Pagan LM	9.578	0.023
1	Breusch-Pagan F	3.546	0.020
2	White LM	19.717	0.011
3	White F	3.120	0.006
4	Jarque-Bera residual normality	2.758	0.252
5	Ramsey RESET	2.864	0.096

The heteroscedasticity tests indicate non-constant residual variance. HC3 robust standard errors are therefore used for the final coefficient inference.

13. HC3 robust regression results

HC3 changes the estimated standard errors, confidence intervals and significance tests, but not the OLS coefficient estimates or (R^2).

```

In [28]: hc3_model = smf.ols(
        "cti ~ rel + trust + condition_code",
        data=df_analysis
    ).fit(cov_type="HC3")

hc3_ci = hc3_model.conf_int()

hc3_coefficients = pd.DataFrame({

```

```

"B": hc3_model.params,
"HC3 SE": hc3_model.bse,
# Statsmodels uses asymptotic normal inference here because use_t=False
"z": hc3_model.tvalues,
"p": hc3_model.pvalues,
"CI lower": hc3_ci[0],
"CI upper": hc3_ci[1]
})

hc3_model_fit = pd.DataFrame({
    "n": [int(hc3_model.nobs)],
    "R-squared": [hc3_model.rsquared],
    "Adjusted R-squared": [hc3_model.rsquared_adj],
    "Robust F": [hc3_model.fvalue],
    "df model": [int(hc3_model.df_model)],
    "df residual": [int(hc3_model.df_resid)],
    "Robust model p": [hc3_model.f_pvalue]
})

display(hc3_coefficients.style.format("{:.3f}"))
display(hc3_model_fit.style.format("{:.3f}"))

```

	B	HC3 SE	z	p	CI lower	CI upper
Intercept	-1.598	1.056	-1.514	0.130	-3.667	0.471
rel	0.543	0.156	3.487	0.000	0.238	0.848
trust	0.679	0.156	4.363	0.000	0.374	0.984
condition_code	0.008	0.396	0.020	0.984	-0.767	0.783

	n	R-squared	Adjusted R-squared	Robust F	df model	df residual	Robust model p
0	60.000	0.416	0.385	12.567	3.000	56.000	0.000

H3 and H4 interpretation

The overall regression model is statistically significant and explains 41.6% of the observed variance in CTI.

- **H3 was supported by the observed data.** Holding trust and condition constant, a one-point increase in perceived relevance was associated with an estimated 0.543-point increase in CTI. The association was statistically significant using HC3 robust standard errors.
- **H4 was supported by the observed data.** Holding relevance and condition constant, a one-point increase in trust was associated with an estimated 0.679-point increase in CTI. The association was statistically significant using HC3 robust standard errors.
- The adjusted condition coefficient was not statistically significant.

These are associations rather than causal effects because relevance and trust were measured variables.

14. Influence and sensitivity checks

Common screening thresholds are used to identify observations for inspection, not automatic exclusion.

```
In [29]: influence = ols_model.get_influence()
influence_frame = influence.summary_frame()

influence_results = pd.DataFrame({
    "Analysis row": np.arange(1, len(df_analysis) + 1),
    "Cook's distance": influence_frame["cooks_d"],
    "Leverage": influence_frame["hat_diag"],
    "Externally studentised residual": influence_frame["student_resid"]
})

cook_threshold = 4 / len(df_analysis)
p_parameters = len(ols_model.params)
leverage_threshold = 2 * p_parameters / len(df_analysis)

influence_summary = pd.DataFrame({
    "Screening rule": [
        "Cook's distance > 4/n",
        "Leverage > 2p/n",
        "|Externally studentised residual| > 3"
    ],
    "Threshold": [
        cook_threshold,
        leverage_threshold,
        3
    ],
    "Flagged observations": [
        int(
            (
                influence_results["Cook's distance"]
                > cook_threshold
            ).sum()
        ),
        int(
            (
                influence_results["Leverage"]
                > leverage_threshold
            ).sum()
        ),
        int(
            (
                influence_results[
                    "Externally studentised residual"
                ].abs() > 3
            ).sum()
        )
    ]
})

display(
    influence_summary.style.format({
        "Threshold": "{:.3f}",
        "Flagged observations": "{:.0f}"
    })
)
```

```

    })
)

display(
  influence_results
  .sort_values(
    "Cook's distance",
    ascending=False
  )
  .head(10)
  .style.format({
    "Cook's distance": "{:.3f}",
    "Leverage": "{:.3f}",
    "Externally studentised residual": "{:.3f}"
  })
)

```

	Screening rule	Threshold	Flagged observations
0	Cook's distance > 4/n	0.067	3
1	Leverage > 2p/n	0.133	2
2	Externally studentised residual > 3	3.000	1

	Analysis row	Cook's distance	Leverage	Externally studentised residual
6	7	0.295	0.094	3.753
12	13	0.114	0.077	-2.435
10	11	0.100	0.249	-1.101
54	55	0.065	0.055	-2.186
52	53	0.060	0.085	1.642
22	23	0.042	0.071	1.502
20	21	0.042	0.071	1.502
31	32	0.040	0.203	0.790
15	16	0.035	0.048	-1.705
37	38	0.035	0.091	-1.178

```

In [30]: # Leave-one-out HC3 sensitivity check

loo_rows = []

for excluded_index in df_analysis.index:
    reduced_data = df_analysis.drop(
        index=excluded_index
    )

    reduced_model = smf.ols(
        "cti ~ rel + trust + condition_code",
        data=reduced_data
    ).fit(cov_type="HC3")

    for predictor in [

```

```

        "rel",
        "trust",
        "condition_code"
    ]:
        loo_rows.append({
            "Excluded row": excluded_index,
            "Predictor": predictor,
            "B": reduced_model.params[predictor],
            "p": reduced_model.pvalues[predictor]
        })

loo_results = pd.DataFrame(loo_rows)

loo_summary = (
    loo_results
    .groupby("Predictor")
    .agg(
        Minimum_B=("B", "min"),
        Maximum_B=("B", "max"),
        Minimum_p=("p", "min"),
        Maximum_p=("p", "max")
    )
    .reset_index()
)

display(
    loo_summary.style.format({
        "Minimum_B": "{:.3f}",
        "Maximum_B": "{:.3f}",
        "Minimum_p": "{:.3f}",
        "Maximum_p": "{:.3f}"
    })
)

print(
    "The leave-one-out results show whether any single observation "
    "changes the direction or statistical conclusion for a predictor. "
    "No observation is removed solely because it is influential."
)

```

	Predictor	Minimum_B	Maximum_B	Minimum_p	Maximum_p
0	condition_code	-0.188	0.118	0.592	1.000
1	rel	0.509	0.646	0.000	0.002
2	trust	0.593	0.737	0.000	0.000

The leave-one-out results show whether any single observation changes the direction or statistical conclusion for a predictor. No observation is removed solely because it is influential.

15. Hypothesis summary

```

In [31]: hypothesis_summary = pd.DataFrame({
    "Hypothesis": ["H1", "H2", "H3", "H4"],
    "Analysis": [
        "Two-sided Welch independent-samples t-test",
        "Two-sided Welch independent-samples t-test",

```

```

    "Multiple regression with HC3 standard errors",
    "Multiple regression with HC3 standard errors"
  ],
  "Key result": [
    (
      f"t({h1_result['df']:.2f}) = "
      f"{h1_result['Welch t']:.2f}, "
      f"p = {h1_result['p']:.3f}, "
      f"Hedges' g = {h1_result['Hedges\' g']:.3f}"
    ),
    (
      f"t({h2_result['df']:.2f}) = "
      f"{h2_result['Welch t']:.2f}, "
      f"p = {h2_result['p']:.3f}, "
      f"Hedges' g = {h2_result['Hedges\' g']:.3f}"
    ),
    (
      f"B = {hc3_model.params['rel']:.3f}, "
      f"HC3 SE = {hc3_model.bse['rel']:.3f}, "
      f"p = {hc3_model.pvalues['rel']:.3f}"
    ),
    (
      f"B = {hc3_model.params['trust']:.3f}, "
      f"HC3 SE = {hc3_model.bse['trust']:.3f}, "
      f"p < .001"
    )
  ],
  "Conclusion": [
    "Not supported",
    "Not supported",
    "Supported by the observed data",
    "Supported by the observed data"
  ],
  "Interpretation": [
    "No statistically significant evidence of a CTI difference between condi",
    "No statistically significant evidence of an authenticity difference bet",
    "Relevance was positively associated with CTI, controlling for trust and",
    "Trust was positively associated with CTI, controlling for relevance and"
  ]
]
})

display(hypothesis_summary)

```

	Hypothesis	Analysis	Key result	Conclusion	Interpretation
0	H1	Two-sided Welch independent-samples t-test	$t(56.52) = 0.41$, $p = 0.686$, Hedges' $g = 0.103$	Not supported	No statistically significant evidence of a CTI...
1	H2	Two-sided Welch independent-samples t-test	$t(56.23) = -0.46$, $p = 0.648$, Hedges' $g = -0.117$	Not supported	No statistically significant evidence of an au...
2	H3	Multiple regression with HC3 standard errors	$B = 0.543$, HC3 $SE = 0.156$, $p = 0.000$	Supported by the observed data	Relevance was positively associated with CTI, ...
3	H4	Multiple regression with HC3 standard errors	$B = 0.679$, HC3 $SE = 0.156$, $p < .001$	Supported by the observed data	Trust was positively associated with CTI, cont...

16. Export anonymised analysis data and tables

The optional supplementary dataset excludes Responseld, age_raw, IP-address fields, location fields and other platform metadata. It contains only the variables required to reproduce the reported analyses.

```
In [32]: output_dir = Path("analysis_outputs")
output_dir.mkdir(exist_ok=True)

supplementary_columns = [
    "condition",
    "cti_1", "cti_2", "cti_3",
    "auth_1", "auth_2", "auth_3",
    "rel_1", "rel_2", "rel_3",
    "trust_1", "trust_2", "trust_3",
    "age", "gender", "prior_app_use",
    "cti", "auth", "rel", "trust",
    "condition_code"
]

df_analysis[supplementary_columns].to_csv(
    output_dir / "5754512_Anonymised_Analysis_Data.csv",
    index=False
)

cleaning_flow.to_csv(output_dir / "cleaning_flow.csv", index=False)
reliability_table.to_csv(output_dir / "reliability_table.csv", index=False)
demographic_distribution.to_csv(output_dir / "demographic_distribution.csv", index=False)
overall_descriptives.to_csv(output_dir / "overall_descriptives.csv", index=False)
condition_descriptives.to_csv(output_dir / "condition_descriptives.csv", index=False)
ttest_results.to_csv(output_dir / "h1_h2_results.csv", index=False)
hc3_coefficients.to_csv(output_dir / "hc3_regression_coefficients.csv")
vif_results.to_csv(output_dir / "vif_results.csv", index=False)
hypothesis_summary.to_csv(output_dir / "hypothesis_summary.csv", index=False)
package_versions.to_csv(output_dir / "package_versions.csv", index=False)

print(f"Outputs saved to: {output_dir.resolve()}")
```

Final analytical conclusion

The notebook supports the following conclusions:

- No statistically significant difference was observed between the selected AI-generated and human-written copy conditions for stated click-through intention.
- No statistically significant difference was observed between conditions for perceived authenticity.
- Perceived relevance and trust were each positively associated with stated click-through intention after controlling for the other predictor and experimental condition.
- The H3 and H4 associations should not be interpreted causally.
- The findings relate to the two selected advertisement executions and should not be generalised to all AI-generated and human-written advertising copy.

```
In [33]: from statsmodels.stats.power import TTestIndPower

# Create power-analysis object
analysis = TTestIndPower()

# Assumptions
n_per_group = 30      # participants in each condition
alpha = 0.05          # significance level
power = 0.80          # desired statistical power
ratio = 1             # equal group sizes
alternative = "two-sided"

# Calculate the minimum detectable standardised effect size
detectable_effect_size = analysis.solve_power(
    effect_size=None,
    nobs1=n_per_group,
    alpha=alpha,
    power=power,
    ratio=ratio,
    alternative=alternative
)

print(f"Sample per group: {n_per_group}")
print(f"Total sample: {n_per_group * 2}")
print(f"Minimum detectable effect size: Cohen's d = {detectable_effect_size:.3f}")
```

Sample per group: 30

Total sample: 60

Minimum detectable effect size: Cohen's d = 0.736