In [36]:
```
# Standard imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from statsmodels.stats.proportion import proportions_ztest, proportion_confint, pro
from statsmodels.stats.power import NormalIndPower
from scipy.stats import mannwhitneyu, shapiro

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
np.random.seed(42)
N = 50000
variants = np.random.choice(['A','B'], size=N, p=[0.5,0.5])
conversion_rate_A = 0.118
conversion_rate_B = 0.132
conversion = []
order_value = []
for v in variants :
    if v == 'A':
        c = np.random.binomial(1, conversion_rate_A)
        conversion.append(int(c))
        order_value.append(round(float(np.random.lognormal(mean=3.4, sigma=0.45)),2
    else:
        c = np.random.binomial(1, conversion_rate_B)
        conversion.append(int(c))
        order_value.append(round(float(np.random.lognormal(mean=3.45, sigma=0.43)),
df = pd.DataFrame({'user_id': range(1, N+1), 'variant': variants, 'ordered': conver
csv_path = 'C:/Users/shubh/Downloads/ABTestData'
df.to_csv(csv_path, index=False)
print('Saved dataset to', csv_path)
df.head()
```

```
Saved dataset to C:/Users/shubh/Downloads/ABTestData
```

Out[2]:

	user_id	variant	ordered	order_value
0	1	A	0	0.0
1	2	B	0	0.0
2	3	B	0	0.0
3	4	B	0	0.0
4	5	A	0	0.0

We simulated a clean A/B testing dataset with 50,000 user sessions, split evenly between Variant A and Variant B. Each user receives a binary conversion outcome and a corresponding order value (0 for non-converters, lognormal for converters).

- A conversion event is modeled as a **Bernoulli / Binomial process**, using the true conversion rates (`pA = 0.118` , `pB = 0.132`).
- Order values follow a **lognormal distribution**, which reflects real e-commerce spending patterns (zero-inflated, heavy right-tail).
- Large sample size (50k) gives **high statistical power**, stable estimates for proportions, and reliable bootstrap confidence intervals later.

This dataset emulates a real checkout A/B experiment where a "time-limited discount banner" (Variant B) is expected to slightly improve both conversion rate and revenue per session. The synthetic data preserves realistic behavior: most sessions do not convert (order_value = 0), while buyers exhibit moderate-to-heavy skew in purchase amount.

In [4]:
```
df['order_value'].nunique()
```

Out[4]:
```
3551
```

In [5]:
```
df['order_value'].max()
```

Out[5]:
```
np.float64(155.71)
```

In [6]:
```
df['order_value'].min()
```

Out[6]:
```
np.float64(0.0)
```

In [7]:
```
print(df.shape)
```

```
(50000, 4)
```

In [8]:
```
df.isna().sum()
```

Out[8]:
```
user_id        0
variant        0
ordered        0
order_value    0
dtype: int64
```

In [9]:
```
print('\nvariant counts:\n', df['variant'].value_counts())
```

```
variant counts:
 variant
A    25053
B    24947
Name: count, dtype: int64
```

In [21]:
```
df.describe()
```

Out[21]:

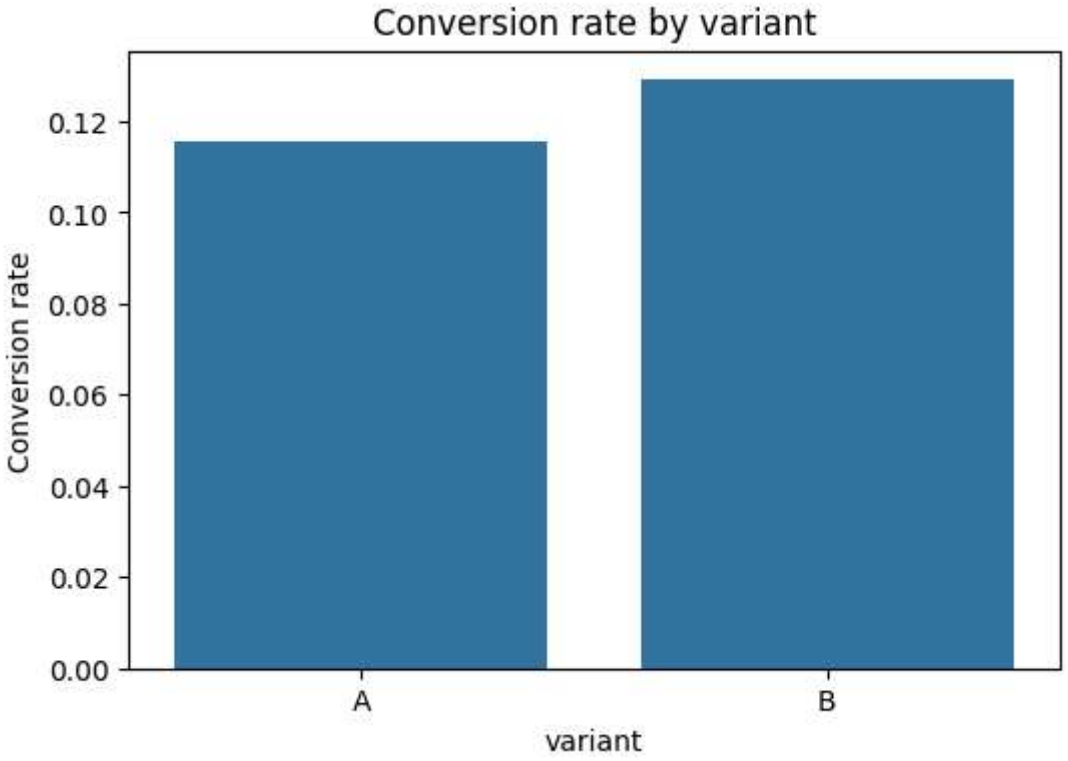
	user_id	ordered	order_value
count	50000.000000	50000.000000	50000.000000
mean	25000.500000	0.122360	4.190996
std	14433.901067	0.327704	12.524513
min	1.000000	0.000000	0.000000
25%	12500.750000	0.000000	0.000000
50%	25000.500000	0.000000	0.000000
75%	37500.250000	0.000000	0.000000
max	50000.000000	1.000000	155.710000

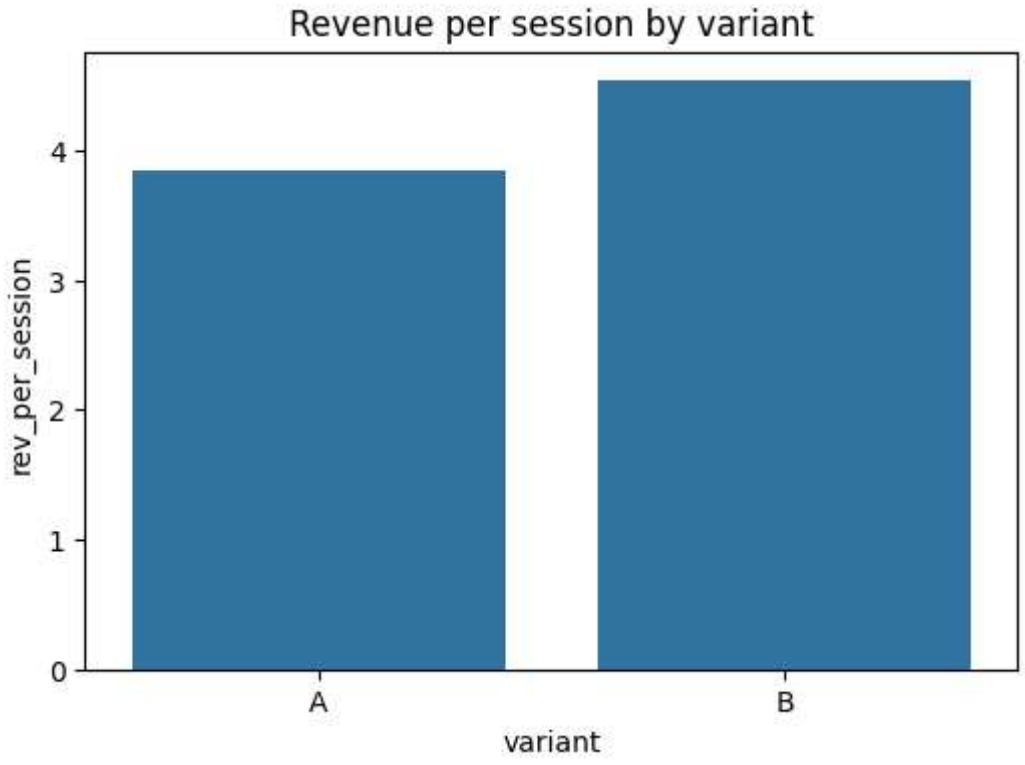
In [27]:
```
summary = df.groupby('variant').agg(
    sessions=('user_id','count'),
    conversions=('ordered','sum'),
    conversion_rate=('ordered','mean'),
    total_revenue=('order_value','sum'),
    rev_per_session=('order_value','mean'),             # includes zeros
    aov_converted=('order_value', lambda x: x[x>0].mean() if (x>0).sum()>0 else np.
).reset_index()

print(summary)
```

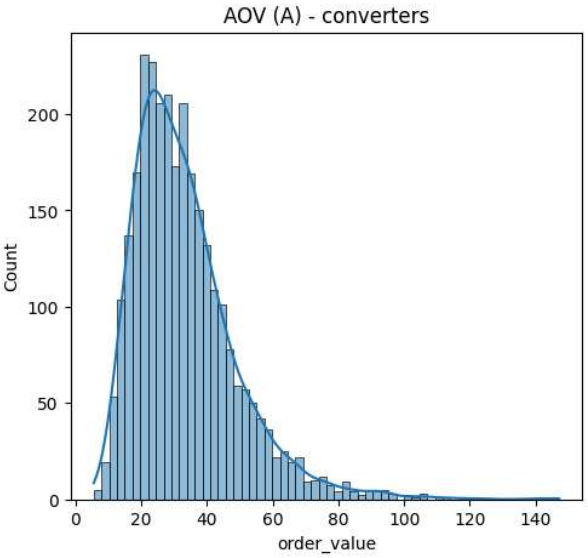
```
  variant  sessions  conversions  conversion_rate  total_revenue  \
0       A     25053         2898         0.115675       96444.10
1       B     24947         3220         0.129074      113105.71

   rev_per_session  aov_converted
0         3.849603      33.279538
1         4.533840      35.125997
```

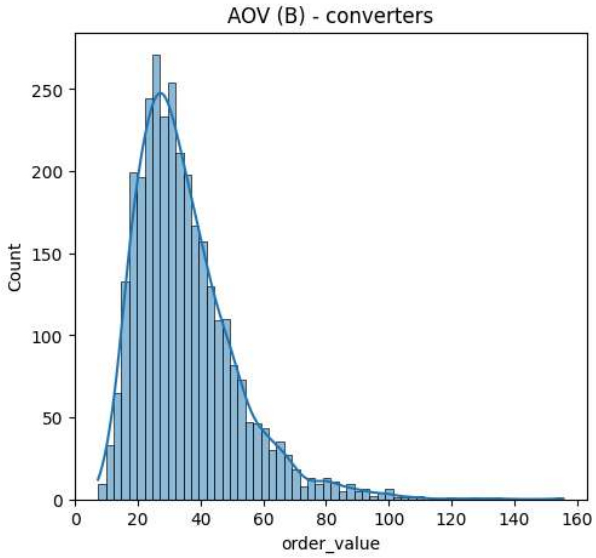
In [28]:
```
# Conversion bar with counts
plt.figure(figsize=(6,4))
sns.barplot(data=summary, x='variant', y='conversion_rate')
plt.ylabel('Conversion rate')
plt.title('Conversion rate by variant')
plt.show()

# Revenue per session
plt.figure(figsize=(6,4))
sns.barplot(data=summary, x='variant', y='rev_per_session')
plt.title('Revenue per session by variant')
plt.show()

# AOV among converters distribution (KDE + boxplot)
conv_df = df[df.ordered==1]
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.histplot(conv_df[conv_df.variant=='A'].order_value, kde=True, bins=60)
plt.title('AOV (A) - converters')
plt.subplot(1,2,2)
```

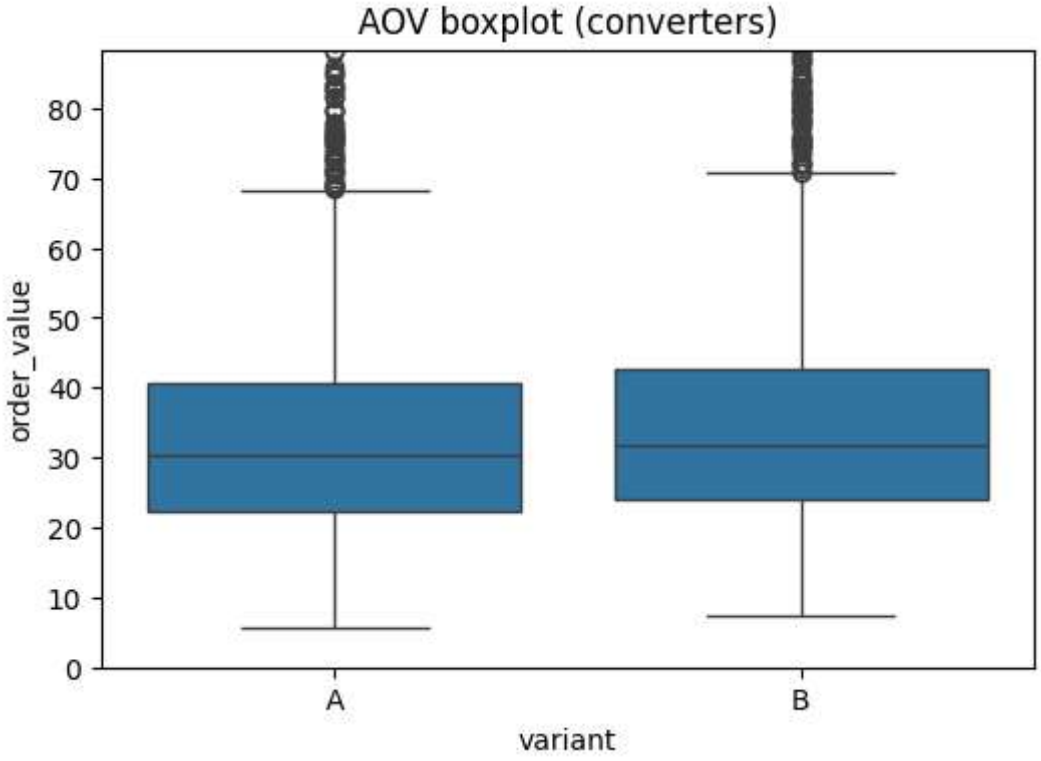
```
sns.histplot(conv_df[conv_df.variant=='B'].order_value, kde=True, bins=60)
plt.title('AOV (B) - converters')
plt.show()

plt.figure(figsize=(6,4))
sns.boxplot(x='variant', y='order_value', data=conv_df)
plt.ylim(0, conv_df.order_value.quantile(0.99))  # zoom outlier cap
plt.title('AOV boxplot (converters)')
plt.show()
```











Variant B achieves a higher conversion rate and greater revenue per session than Variant A The boxplots shows that buyers spend roughly the same amount in both variants,

PRIMARY METRIC : Conversion Rate Statistical Testing

This section evaluates if the difference in conversion rates between Variant A and Variant B is statistically significant or simply due to random sampling variation.

Null Hypothesis (H_0)

There is no difference in conversion rate between Variant A and Variant B.

H0:pA=pB

Alternative Hypothesis (H_1)

There is a difference in conversion rate between the two variants.

$$H_1 : p_A \neq p_B$$

This is a two-sided test, appropriate when we want to detect any deviation—positive or negative.

In [73]:
```
counts = summary['conversions'].values
nobs = summary['sessions'].values

# Perform the z-test
z_stat, p_val = proportions_ztest(counts, nobs, alternative='two-sided')
print('z_stat, p_val =', z_stat, p_val)

# Wilson CI
for i , row in summary.iterrows():
    ci_low, ci_up = proportion_confint(int(row['conversions']), int(row['sessions']
    print(row['variant'], 'rate=', row['conversion_rate'], '95% CI=', (ci_low, ci_u

# absolute and relative lift
p_A = float(summary.loc[summary.variant=='A','conversion_rate'])
p_B = float(summary.loc[summary.variant=='B','conversion_rate'])
absolute_lift = p_B - p_A
relative_lift = absolute_lift / p_A
print('absolute lift is:', absolute_lift, 'relative lift is :', relative_lift)
```

```
z_stat, p_val = -4.571346654175001 4.845998486667871e-06
A rate= 0.11567476948868399 95% CI= (0.11177311457151573, 0.11969426603798208)
B rate= 0.12907363610855013 95% CI= (0.12497014497130532, 0.13329134370256251)
absolute lift is: 0.01339886661986614 relative lift is : 0.11583223099637902
```

The two-proportion z-test shows a clear difference between the variants.

At a standard significance level of α = 0.05:

Reject H_0 if the p-value < 0.05

Fail to reject H_0 if the p-value ≥ 0.05

p-value = 4.85×10^{-6}, which is smaller than 0.05 , so we reject the null hypothesis.

The observed difference in conversion rate is statistically significant and extremely unlikely to be due to chance.

In [40]:
```
# quick Beta-Binomial posterior sampling
import numpy as np
sA = int(summary.loc[summary.variant=='A','conversions'])
nA = int(summary.loc[summary.variant=='A','sessions'])
sB = int(summary.loc[summary.variant=='B','conversions'])
nB = int(summary.loc[summary.variant=='B','sessions'])

Nsim = 50000
a_post = np.random.beta(1 + sA, 1 + nA - sA, size=Nsim)
b_post = np.random.beta(1 + sB, 1 + nB - sB, size=Nsim)
prob_B_superior = (b_post > a_post).mean()
print('P(B > A) =', prob_B_superior)
```

```
P(B > A) = 1.0
```

Based on the posterior distributions, there is effectively a ~100% probability that Variant B's true conversion rate exceeds Variant A's.

Secondary — AOV among converters only (conditional metric)

We will implement a reusable bootstrap function to compute distribution-free confidence intervals for mean differences—a necessity for skewed metrics like AOV, with compact wrapper around the two-proportion Z-test to streamline conversion rate testing.

In [78]:
```
# for bootstrap sampling distribution
def bootstrap_mean_diff_np(a, b, n_boot=5000, seed=42):
    range1 = np.random.RandomState(seed)
    diffs = np.empty(n_boot)
    for i in range(n_boot):
        sa = range1.choice(a, len(a), replace=True)
        sb = range1.choice(b, len(b), replace=True)
        diffs[i] = sb.mean() - sa.mean()
    return diffs.mean(), np.percentile(diffs, [2.5,97.5]), diffs

# run z-test wrapper
def run_z_test(summary):
    counts = summary['conversions'].values
    nobs = summary['sessions'].values
    return proportions_ztest(counts, nobs)
```

AOV Diagnosis and Non Parametric Testing

This section assesses whether Variant A and Variant B differ in Average Order Value (AOV) among users who completed a purchase. Because AOV is skewed, we will validate distributional assumptions and then apply the correct statistical tests.

In [49]:
```
a_conv = df[(df.variant=='A') & (df.ordered==1)]['order_value'].values
b_conv = df[(df.variant=='B') & (df.ordered==1)]['order_value'].values
```

```
# Shapiro Wilk Test for Normality Check
print('Shapiro A (converters):', shapiro(a_conv))
print('Shapiro B (converters):', shapiro(b_conv))

# Levene
lev_stat, lev_p = levene(a_conv, b_conv)
print('Levene p:', lev_p)
```

```
Shapiro A (converters): ShapiroResult(statistic=np.float64(0.9023136024054693), pval
ue=np.float64(1.3306481397950426e-39))
Shapiro B (converters): ShapiroResult(statistic=np.float64(0.9061672919193668), pval
ue=np.float64(1.0778996410276337e-40))
Levene p: 0.509328729882875
Mann-Whitney p: 2.748457550917563e-07
AOV among converters B - A mean: 1.843544896342305 95% CI: [1.04928339 2.64081836]
```

Both p-values are small , strong rejection of normality.
Therefore we canot use any parametric test like t test

Variances between A and B are not significantly different.

This suggests the spread of spending among converters is comparable across variants

In [79]:
```
# Mann-Whitney Test
mw_stat, mw_p = mannwhitneyu(a_conv, b_conv, alternative='two-sided')
print('Mann-Whitney p:', mw_p)

# use Bootstrap mean difference
mean_diff_conv, ci_conv, diffs_conv = bootstrap_mean_diff_np(a_conv, b_conv, n_boot
print('AOV among converters B - A mean:', mean_diff_conv, '95% CI:', ci_conv)
```

```
Mann-Whitney p: 2.748457550917563e-07
AOV among converters B - A mean: 1.843544896342305 95% CI: [1.04928339 2.64081836]
```

Interpretation p value from Mann Whitney Test is 2.748457550917563e-07
Statistically significant difference in distributions.Indicates that converters in Variant B tend to spend slightly more.

Mean Difference (B − A): +1.84 and 95% CI: [1.05, 2.64]
Bootstrap Estimate of Mean AOV Difference The uplift in AOV among converters is positive and well above zero.

Power Analysis Code

In [80]:
```
from statsmodels.stats.power import NormalIndPower
from statsmodels.stats.proportion import proportion_effectsize

# Instantiate the power analysis tool for comparing two independent proportions
analysis = NormalIndPower()

# Baseline conversion rate from Variant A (this is our current performance level)
pA = float(summary.loc[summary.variant=='A', 'conversion_rate'])
```

```
# Minimum Detectable Effect (MDE): the smallest uplift we care to reliably detect
# Here we choose +1 percentage point (0.01), which is realistic for product experim
mde = 0.01

# Convert absolute difference (pA vs pA+mde) into a standardized effect size
# This is required by the statsmodels power function
es = proportion_effectsize(pA, pA + mde)

# Calculate required per-arm sample size to detect the MDE with 80% power and 5% al
req_n = analysis.solve_power(
    effect_size=es,
    power=0.8,
    alpha=0.05,
    ratio=1  # equal-sized groups
)
print("Required per-arm n for MDE =", mde, "is", int(np.ceil(req_n)))

# Check the power of our *actual* experiment size
current_n = int(summary['sessions'].mean())  # average sample size per variant
achieved_power = analysis.power(
    effect_size=es,
    nobs1=current_n,
    alpha=0.05,
    ratio=1
)
print("Achieved power at current per-arm n =", current_n, "is", achieved_power)
```

```
Required per-arm n for MDE = 0.01 is 16651
Achieved power at current per-arm n = 25000 is 0.9296130437943907
```

In [52]:
```
# cap = df.order_value.quantile(0.999)
# df2 = df.copy()
# df2.loc[df2.order_value > cap, 'order_value'] = cap
# # re-run revenue-per-session bootstrap on df2
```

In [53]:
```
# # Bootstrap mean diff (converters)
# mean_diff_conv, ci_conv, diffs_conv = bootstrap_mean_diff_np(a_conv, b_conv, n_bo
# print('AOV among converters B - A mean:', mean_diff_conv, '95% CI:', ci_conv)
```

```
AOV among converters B - A mean: 1.843544896342305 95% CI: [1.04928339 2.64081836]
```

Business impact calculation

In [81]:
```
# Compute the mean order value among converters only
# (This represents the revenue contribution from each successful purchase)
mean_order_value_converted = df[df.ordered == 1]['order_value'].mean()

# Revenue per session assumes: rev/session = conversion_rate × mean_order_value_con
rev_per_session_A = pA * mean_order_value_converted
rev_per_session_B = pB * mean_order_value_converted

# Gross revenue uplift per 1,000 sessions (before subtracting any discount cost)
uplift_per_1000 = (rev_per_session_B - rev_per_session_A) * 1000
print("Gross uplift per 1,000 sessions:", uplift_per_1000)
```

```
# Compute net uplift (if discount has a monetary cost)


discount_cost = 5   # example: ₹5 or $5 cost per discount redemption; change as nee
conv_diff_per_1000 = (pB - pA) * 1000    # additional conversions because of Varian

# Total cost of additional conversions that received the discount
total_discount_cost = conv_diff_per_1000 * discount_cost

# Net revenue uplift after subtracting cost
net_uplift_per_1000 = uplift_per_1000 - total_discount_cost
print("Net uplift per 1,000 sessions:", net_uplift_per_1000)
```

```
Gross uplift per 1,000 sessions: 458.9293812370534
Net uplift per 1,000 sessions: 391.9350481377227
```

Calculations

We compute:

- **mean_order_value_converted**
  -> Average order value *among converters only*.
- **rev_per_session_A & rev_per_session_B**
  -> Expected revenue per visitor (conversion rate × AOV).
- **uplift_per_1000**
  -> Incremental revenue B generates over A for every 1,000 sessions.

---

2. Interpretation of the Result

Output:
Gross uplift per 1,000 sessions ≈ ₹458.93

Insight:
Variant B is expected to generate **~₹459 more revenue per 1,000 visitors** compared to Variant A.

.