23917-MDSC-102-ESE

Name: Anirudh Yandrapu

Reg no: 23917

Sub: Inferential Statistics(P)

About Dataset:

The dataset 'supermarket_sales' consists of 17 features and 1000 rows. This dataset records sales transactions in a store, including details like product, price, and customer information. It helps track sales performance, customer behaviour, and profit margins. It captures sales of various product categories in branches located in specific cities at particular times and dates. Key columns include 'Product line', 'Total', and 'Rating'. The 'Customer type' feature indicates whether a customer is a member of the supermarket or a regular shopper, while the 'Payment' feature describes the payment method used, including Credit card, Cash, or Ewallet.

Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin percentage	gross income	Rating
750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	13:08	Ewallet	522.83	4.761905	26.1415	9.1
226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	10:29	Cash	76.40	4.761905	3.8200	9.6
631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	13:23	Credit card	324.31	4.761905	16.2155	7.4
123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	20:33	Ewallet	465.76	4.761905	23.2880	8.4
373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2/8/2019	10:37	Ewallet	604.17	4.761905	30.2085	5.3

Information:

```
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Invoice ID               1000 non-null   object
 1   Branch                   1000 non-null   object
 2   City                     1000 non-null   object
 3   Customer type            1000 non-null   object
 4   Gender                   1000 non-null   object
 5   Product line             1000 non-null   object
 6   Unit price               1000 non-null   float64
 7   Quantity                 1000 non-null   int64
 8   Tax 5%                   1000 non-null   float64
 9   Total                    1000 non-null   float64
 10  Date                     1000 non-null   object
 11  Time                     1000 non-null   object
 12  Payment                  1000 non-null   object
 13  cogs                     1000 non-null   float64
 14  gross margin percentage  1000 non-null   float64
 15  gross income             1000 non-null   float64
 16  Rating                   1000 non-null   float64
```

We can observe that many of the columns are of Object dtype and Some of the object dtype columns cannot be pre-processed like City, Product line which are meaning and helpful for analysis.

Pre-processing:

- Dropping un-useful column for analysis
- Changing the format of Date
- Converting Time into Hours

```python
# Invoice ID is not so important so we can remove that column
df = df.drop('Invoice ID', axis=1)
```

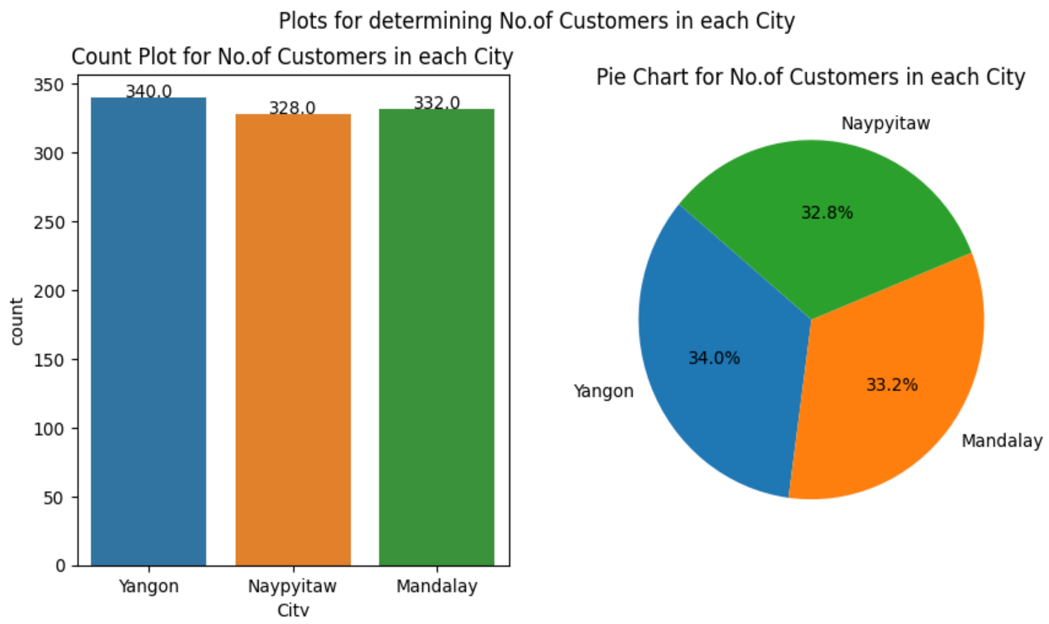
```python
# Changing the date format
if df['Date'].dtype == 'O':
    df['Date'] = df['Date'].map(pd.to_datetime)
# Conveting the time to hours only
if df['Time'].dtype == 'O':
    df['Time'] = pd.to_datetime(df['Time'], format='%H:%M').dt.hour
```

```python
# Branch are Object types with values A, B and C hence replacing them with 1,2,3.
df['Branch'] = df['Branch'].replace({'A': 1, 'B':2, 'C':3})
```

- There are no null values so no need of imputing values

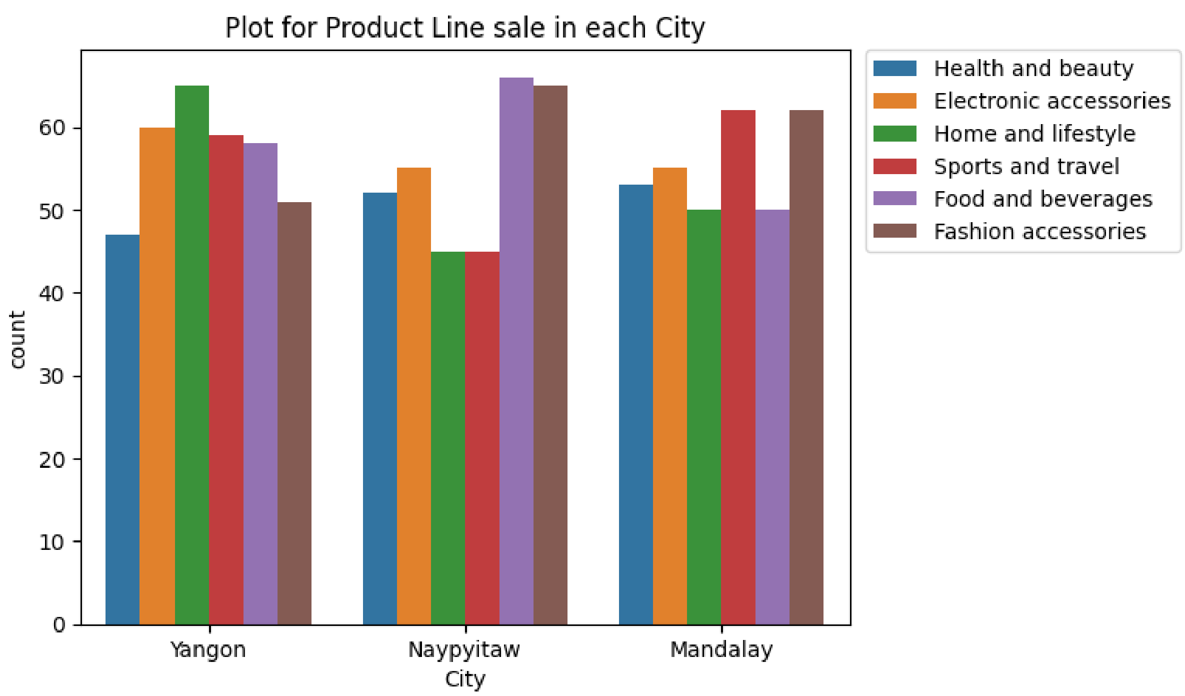
Plots Visualization:

1) Let’s see how many customers are there in each City:



We can see that there are 340 customers in Yangon, 328 in Naypyidaw and 332 in Mandalay

2) City wise Products sale:

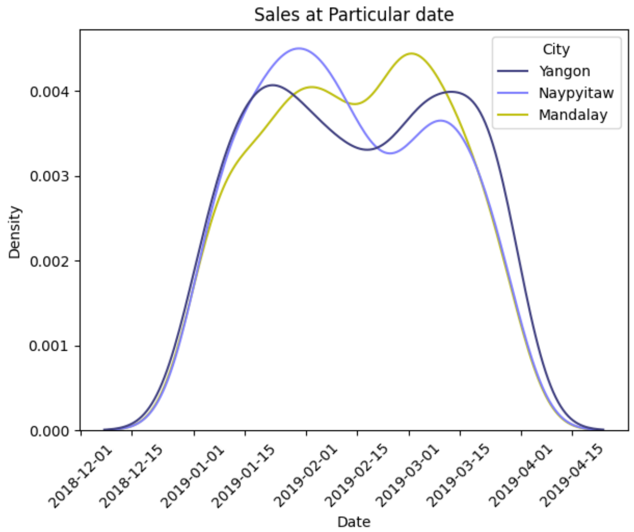


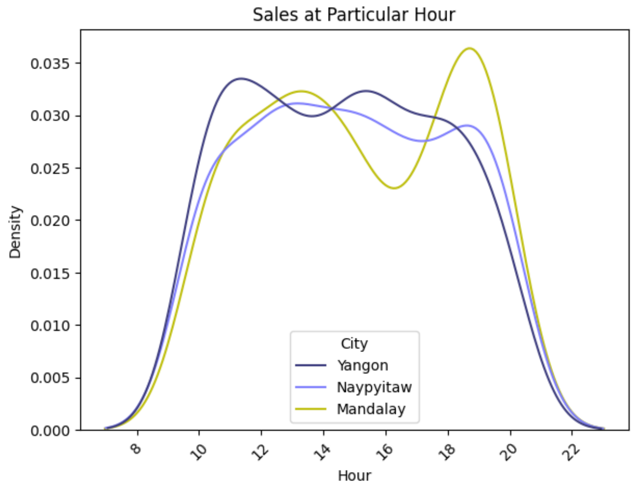
Yangon: The most purchased products are Home and lifestyle and the least purchased products are Health and beauty.

Naypyidaw: The most purchased products are Food and beverages and the least purchased products are Home and lifestyle, Sports and travel.

Mandalay: The most purchased products are Sports and travel, Fashion accessories and the Least purchased products are Home and lifestyle, Food and beverages.

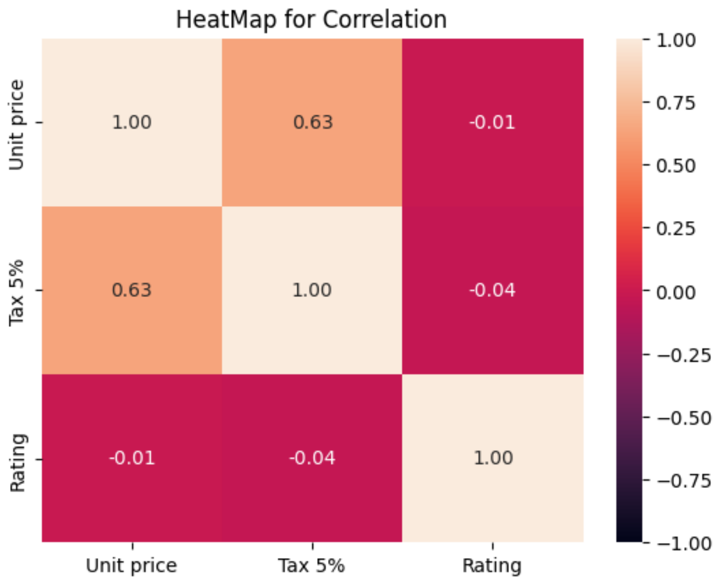
3) Sales Analysis at particular date and time:





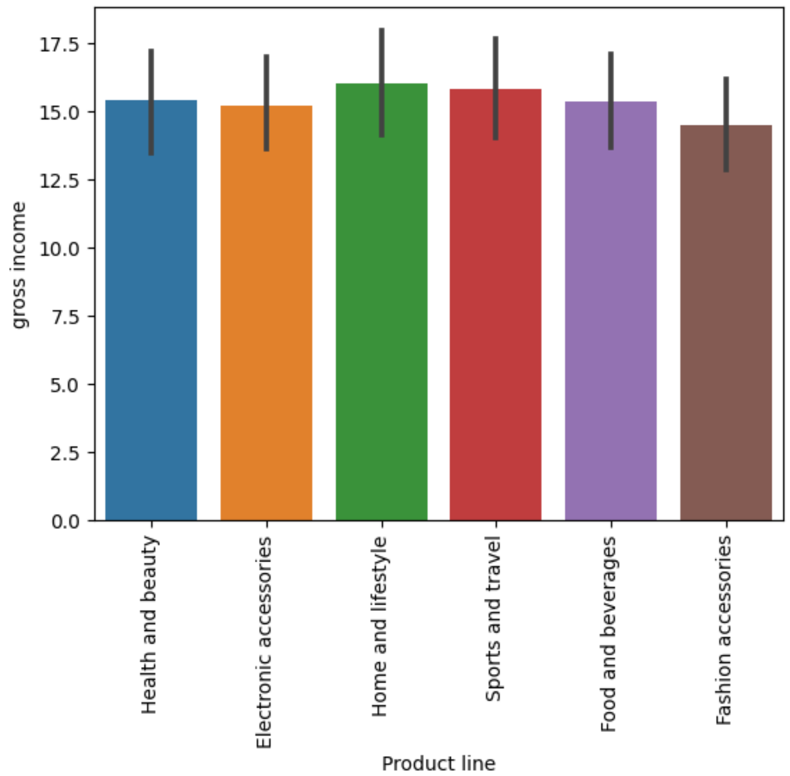
Date: The sales in the cities are almost same inconsideration of dates
Time: At 20:00 hrs the sales in the city Mandalay are very high compared to other cities and the remaining hours are almost same.

4) Correlation between the Columns product, tax and it rating:



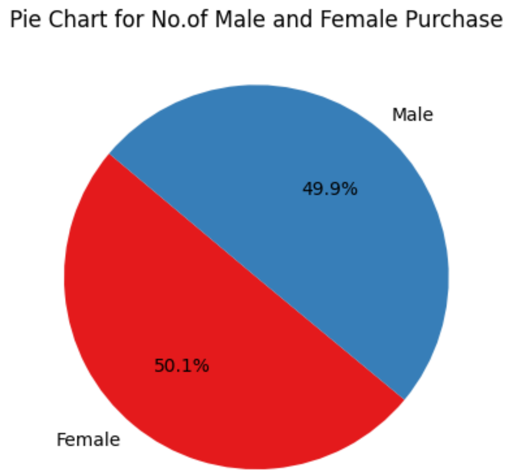
There is no correlation between the price of product and it tax to the final product rating, maybe the quality of product and time of deliver could influence this variable but it is an information not available in dataset unfortunately

5) Bar Plot for gross income feature:



From the above we can infer the gross income of each product line. If we observe the plot, Home and lifestyle has high gross income whereas fashion accessories have less gross income.

6) Plot of visualizing no of males and females:

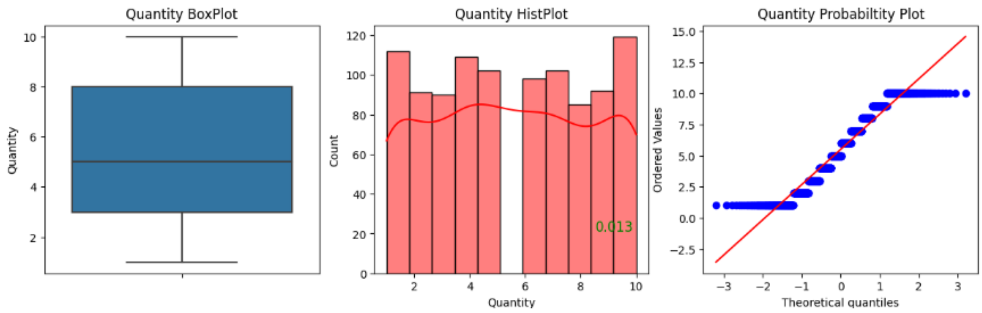


Female Purchases are more compared to male.

Inference of the features based on Hist, Box and Prob Plot:

1) Unit Price: Observing Boxplot we cannot say that it is Skewed and from histplot is not sure whether Unit Price is Skewed but, from Probability Plot we can say that Unit Price is skew.

2) Quantity: Observing Boxplot we cannot say that it is Skewed and from histplot is not sure whether Quantity is Skewed. But, from Probability Plot we can say that Quantity is skew.



Similarly, for other features:

3) Tax 5%: Observing BoxPlot and HistPlot we say that Tax 5% is Right Skewed and from Probability Plot we can say that Tax 5% is skew and also having some outliers.

4) Total: Observing BoxPlot and HistPlot we say that Total is Right Skewed and from Probability Plot we can say that Total is skew and also having some outliers.

5) Time: Observing Boxplot we cannot say that it is Skewed and from histplot is not sure whether Time is Skewed. But, from Probability Plot we can say that Time is skew.

6) cogs: Observing BoxPlot and HistPlot we say that cogs are Right Skewed and from Probability Plot we can say that cogs are skew and also having some outliers.

7) Gross margin Percentage: Is a straight line, which means in every city the margin % is same.

8) Gross income: Observing BoxPlot and HistPlot we say that Gross income is Right Skewed and from Probability Plot we can say that Gross income is skew and also having some outliers.

9) Rating: Observing Boxplot we cannot say that it is Skewed and from histplot is not sure whether Rating is Skewed. But, from Probability Plot we can say that Rating is skew.

Conclusion:

The above features are Skewed. So we apply the transformation by boxcox function to transform the DataFrame(df).

Normalization:

- We apply boxcox function and transform the data into normal form. Here, are the few graphs after transformation:

For Tax 5%:

Transformed data Inference:

We cannot say that there is no skewness in each column but they are reduced compared to before and we can verify by Probability Plot which are almost normal and having less outliers.

Confidence Intervals:

1) Let's see the Confidence Intervals for MEAU for several features:(l.s = 0.05)

```
num_cols = ndf.select_dtypes(include=['int', 'float']) # Getting Integer and Float features
num_cols = numeric_cols.drop(columns=['Branch'])        # From that removing 'Branch' feature
# Calculate the mean for the selected columns
mean = num_cols.mean()
```

```
s = np.sqrt(num_cols.var(ddof=1))      # population Standard Deviation
```

- Function for Finding Confidence Intervals:

```
def Mean_Interval(data = None,obj_col = None,mean=None,s=None):   # Function for finding Mean_Inteval
    Intervals = []                                                # List of storing the Lower and upper bound
    temp = []                                                     # Temp list
    i = 0
    for cols in data:
        if(cols not in obj_col and cols != 'Branch'):
            n = len(data[cols])
            t_value = stats.t.ppf(0.975, df=(n-1))          # Finding the t_value
            lower = mean[i] - t_value*(s[i]/np.sqrt(n))     # Calculating the Lower Bound
            upper = mean[i] + t_value*(s[i]/np.sqrt(n))     # Calculating the upper Bound
            temp.append(lower)
            temp.append(upper)
            Intervals.append(temp)              # appending it to the list
            i += 1
            temp = []
    return Intervals

interval = Mean_Interval(ndf,obj_col,mean,s)          # Calling the function Mean_Interval()
features = ['Unit price','Quantity','Tax 5%','Total','Time','cogs','gross margin percentage','gross income','Rating' ]    # List
limits_mean = pd.DataFrame(interval, columns = ['Lower_Limit','Upper_Limit'] )    # Pandas dataframe for Mean interval
limits_mean.insert(0,"Features",features)
limits_mean
```

OUTPUT:

	Features	Lower_Limit	Upper_Limit
0	Unit price	22.798422	23.918769
1	Quantity	3.055236	3.287246
2	Tax 5%	3.498254	3.713995
3	Total	12.994208	13.499485
4	Time	7.269072	7.424371
5	cogs	12.769736	13.268168
6	gross margin percentage	4.761905	4.761905
7	gross income	3.498254	3.713995
8	Rating	4.267356	4.398420

2) Confidence Interval for Variance: (l.s = 0.05, meau is not known)

- Function for finding the CI for Variance:

```
def Variance_Interval(data = None,obj_col = None,s2 = None):  # Function for finding Variance_Inteval
    Intervals = []                                             # List of storing the Lower and upper bound
    temp = []                                                  # Temp list
    i = 0
    for cols in data:
        if(cols not in obj_col and cols != 'Branch'):
            n = len(data[cols])
            chi_1 = stats.chi2.ppf(0.975, df=(n-1))            # chi_square value for alpha/2
            chi_2 = stats.chi2.ppf(1-0.975, df=(n-1))          # chi_square value for 1 - alpha/2
            lower = ((n-1)*s2[i]) / chi_1                     # Calculating the lower Bound
            upper = ((n-1)*s2[i]) / chi_2                     # Calculating the upper Bound
            temp.append(lower)
            temp.append(upper)
            Intervals.append(temp)
            i += 1
            temp = []
    return Intervals

s2 = num_cols.var(ddof=1)      # population variance

interval_var = Variance_Interval(ndf,obj_col,s2)
limits_var = pd.DataFrame(interval_var, columns = ['Lower_Limit','Upper_Limit'] )   # Pandas dataframe for Variance interval
limits_var.insert(0, "Features", features)
limits_var
```

OUTPUT:

	Features	Lower_Limit	Upper_Limit
0	Unit price	74.789009	89.134764
1	Quantity	3.207353	3.822576
2	Tax 5%	2.773325	3.305294
3	Total	15.212182	18.130128
4	Time	1.437048	1.712697
5	cogs	14.802860	17.642290
6	gross margin percentage	0.000000	0.000000
7	gross income	2.773325	3.305294
8	Rating	1.023533	1.219864

Testing Hypothesis:

Function for Testing Hypothesis for μ:

The function inputs are:

Data: the feature of the dataset
Alpha: the confidence value
Ho: Null hypothesis
Var: Value of variance is known
S: sample variance

```
def hyp_testing(data=None,alpha=0.05,Ho=None,var=None,s = None): # Function for testing for μ for both sigma known and unknown
    mean = float(data.mean())                                    # Getting the sample mean of the data
    n = len(data)
    if(var != None):                                             # If variance is known
        z_cal = abs((mean - Ho)/(np.sqrt(var)/np.sqrt(n)))         # Calculating z_cal value
        p = stats.norm.cdf(z_cal)                                  # p value
    elif(n > 30):                                                # If the variance is unknowm and n > 30
        z_cal = abs((mean - Ho)/(np.sqrt(s)/np.sqrt(n)))           # Calculating z_cal value
        p = stats.norm.cdf(z_cal)                                  # p value
    else:                                                        # If the variance is unknowm and n <= 30
        t_cal = abs((mean - Ho)/(s/np.sqrt(n)))                    # Calculating t_cal value
        p = 2*(1-stats.t.cdf(t_cal,df=n))                          # p value
    if(p <= alpha):                                              # if p <= alpha reject Ho
        print("reject Ho")
    else:
        print("do not reject Ho")
```

1) Testing whether the μ of feature 'Total' is 320 or Not.

$$H0 : \mu = 320 vs H1 : \mu \neq 320$$

$$\alpha = 0.05$$
$$\sigma^2 is not known$$

```
# Declaring Ho
Ho = 320
# lets find the sample variance
s2 = df['Total'].var(ddof=1)     # s^2 value
s = np.sqrt(s2)                  # s value
print("The value of s =",s)
```

```
The value of s = 245.88533510097187
```

- Calling the above Function: hyp_testing ()

```
hyp_testing(data=df['Total'],Ho=Ho,s = s)          # Calling the Function: hyp_testing()
```

```
do not reject Ho
```

Result: Do not REJECT Ho

2) Testing whether the μ of feature 'gross income' is 20 or Not.

$$H0 : \mu = 20 vs H1 : \mu \neq 20$$

$$\alpha = 0.05$$
$$\sigma^2 is not known$$

- Declaration:

```
# Declaring Ho
Ho = 20
# lets find the sample variance
s2 = df['gross income'].var(ddof=1)    # s^2 value
s = np.sqrt(s2)                        # s value
print("The value of s =",s)
```

```
The value of s = 11.708825480998659
```

- Calling the above Function: hyp_testing ()

```
hyp_testing(data=df['gross income'],Ho=Ho,s = s)          # Calling the Function: hyp_testing()
```

```
do not reject Ho
```

Result: Do not REJECT Ho