Chi-Squared Test of Independence Chi-Squared Test of Independence is a key concept in probability that describes a situation where knowing the value of one variable tells you nothing about the value of another. For instance, the month you were born probably doesn't tell you anything about which web browser you use, so we'd expect birth month and browser preference to be independent. On the other hand, your month of birth might be related to whether you excelled at sports in school, so month of birth and sports performance might not be independent.

The chi-squared test of independence tests whether two categorical variables are independent. The test of independence is commonly used to determine whether variables like education, political views and other preferences vary based on demographic factors like gender, race and religion. Let's generate some fake voter polling data and perform a test of independence.

Importing Libraries

```
1 import numpy as np
2 import pandas as pd
3 import scipy.stats as stats
```

""" METHODOLOGY 01: MANUAL CALCULATION """

STEP 1: GENERATE A RANDOM DATASET

Generate under a random factor

https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.seed.html

```
1 np.random.seed(10)
```

```
1 # Sample data randomly at fixed probabilities
2 voter_race = np.random.choice(a=["asian","black","hispanic","other","white"],
3                               p=[0.05, 0.15 ,0.25, 0.05, 0.5],
4                               size=1000)
```

```
1 # Sample data randomly at fixed probabilities
2 voter_party = np.random.choice(a=["democrat","independent","republican"],
3                               p=[0.4, 0.2, 0.4],
4                               size=1000)
```

```
# Binding 2 arrays (voter_race and voter_party) to make a DataFrame
voters = pd.DataFrame({"race":voter_race,
                       "party":voter_party})
# You can check the data of DataFrame by calling it
voters
```

	race	party
0	white	democrat
1	asian	republican
2	white	independent
3	white	republican
4	other	democrat
...	...	...
995	white	republican
996	hispanic	independent
997	black	independent
998	white	republican
999	black	democrat

1000 rows × 2 columns

```
# Create a CrossTab from DataFrame, Assign the column names and row names
voter_tab = pd.crosstab(voters.race, voters.party, margins=True)
voter_tab.columns = ["democrat", "independent", "republican", "row_totals"]
voter_tab.index = ["asian", "black", "hispanic", "other", "white", "col_totals"]
# You can check the data of CrossTab by calling it
voter_tab

```

	democrat	independent	republican	row_totals
asian	21	7	32	60
black	65	25	64	154
hispanic	107	50	94	251
other	15	8	15	38
white	189	96	212	497
col_totals	397	186	417	1000

STEP 2: GET THE "OBSERVED" TABLE AND "EXPECTED" TABLE

Calculate the "observed" table: "Observed" table can be extracted from our CrossTab by exclude the row_totals and col_totals You can see row_totals is in the index of 4 (in column) and col_totals is in the index of 6 (in row). [0:5, 0:3] means "we will take the rows from 0 index to 5 index and columns from 0 index to 3 index and assign to new CrossTab that named [observed]

```
observed = voter_tab.iloc[0:5, 0:3]
# You can check the data of observed table by calling it
observed
```

	democrat	independent	republican
asian	21	7	32
black	65	25	64
hispanic	107	50	94
other	15	8	15
white	189	96	212

Calculate the "expected" table: "Expected" table can be calculated using below formula: total_rows x total_columns / total_observations And these factors can be get by: - total_rows = voter_tab["row_totals"] - total_columns = voter_tab["col_totals"] - total_observations = 1000 Please note that the "loc" function in below code is used to switch the index base on column name to row name

```
expected =  np.outer(voter_tab["row_totals"][0:5],
                     voter_tab.loc["col_totals"][0:3]) / 1000
# Now convert into a DataFrame, Assign the column names and row names
expected = pd.DataFrame(expected)
expected.columns = ["democrat", "independent", "republican"]
expected.index = ["asian", "black", "hispanic", "other", "white"]
# You can check the data of expected table by calling it
expected

```

STEP 3: CALCULATE THE CHI SQUARE VALUE and CRITICAL VALUE

Chi square formula: chi square = total of [(observed - expected)^2]/expected

```
Note: We call .sum() twice: once to get the column sums
and a second time to add the column sums together,       returning the sum of the entire 2D table.
```

```
chi_squared_stat = (((observed-expected)**2)/expected).sum().sum()
print(chi_squared_stat)
```

```
7.169321280162059
```

Find the critical value for confidence of 95% and degree of freedom (df) of 8 Why df = 8? Degree of freedom formula: df = (total rows - 1) x (total columns - 1) = (5 - 1) x (3 - 1) = 4 x 2 = 8

```
crit = stats.chi2.ppf(q = 0.95, # Find the critical value for 95% confidence*
                      df = 8)   # *

print("Critical value")
print(crit)

p_value = 1 - stats.chi2.cdf(x=chi_squared_stat,  # Find the p-value
                             df=8)
print("P value")
print(p_value)
```

```
Critical value
15.50731305586545
P value
0.518479392948842
```

STEP 4: MAKE THE CONCLUSION

Because chi_squared_stat < crit

When your p-value is less than or equal to your significance level, you reject the null hypothesis. The data favors the

alternative hypothesis.

When your p value is greater than your significance level, you fail to reject the null hypothesis.

```
if chi_squared_stat < crit:
    print("""At 0.95 level of significance, we reject the null hypotheses and accept H1.
They are not independent.""" )
else:
    print("""At 0.95 level of significance, we accept the null hypotheses.
They are independent.""" )
```

```
At 0.95 level of significance, we reject the null hypotheses and accept H1.
They are not independent.
```

METHODOLOGY 02: CALCULATE USING SCIPY.STATS LIBRARY

```
""" METHODOLOGY 02: CALCULATE USING SCIPY.STATS LIBRARY"""
stats = stats.chi2_contingency(observed=observed)
# You can check the returned data by calling it
# The returned data includes: chi_squared_stat, p_value, df, expected_crosstab
print(stats)
```

```
(7.169321280162059, 0.518479392948842, 8, array([[ 23.82 ,  11.16 ,  25.02 ],
       [ 61.138,  28.644,  64.218],
       [ 99.647,  46.686, 104.667],
       [ 15.086,   7.068,  15.846],
       [197.309,  92.442, 207.249]]))
```

```
1
```