Experiment 04 - Statistical Hypothesis Test

| Roll No. |  |
| --- | --- |
| Name |  |
| Class | D15-A |
| Subject | DS using Python Lab |
| LO Mapped | LO4: Apply the different unsupervised machine learning algorithms like Clustering or Association to solve the problems. |
|  |  |

**Aim**:

To implement Statistical Hypothesis using Z-Test with Scipy and Sci-kit learn

**Introduction**:

**Scikit-learn**

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy, and Matplotlib.

**Origin of Scikit-Learn**

It was originally called scikits.learn and was initially developed by David Cournapeau as a Google Summer of Code project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel, from FIRCA (French Institute for Research in Computer Science and Automation), took this project to another level and made the first public release (v0.1 beta) on 1st Feb. 2010.

**Features of Scikit-Learn**

Rather than focusing on loading, manipulating, and summarizing data, the Scikit-Learn library is focused on modeling the data. Some of the most popular groups of models provided by Sklearn are as follows:

* **Supervised Learning algorithms** − Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree, etc., are part of sci-kit-learn.
* **Unsupervised Learning algorithms** − On the other hand, it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks.
* **Clustering** − This model is used for grouping unlabeled data.
* **Cross-Validation** − It is used to check the accuracy of supervised models on unseen data.
* **Dimensionality Reduction** − It is used for reducing the number of attributes in data which can be further used for summarization, visualization, and feature selection.
* **Ensemble methods** − As the name suggests, it is used for combining the predictions of multiple supervised models.
* **Feature extraction** − It is used to extract the features from data to define the attributes in image and text data.
* **Feature selection** − It is used to identify useful attributes to create supervised models.
* **Open Source** − It is an open-source library and is also commercially usable under the BSD license.

**SciPy**

SciPy, a scientific library for Python, is an open-source, BSD-licensed library for mathematics, science, and engineering. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The main reason for building the SciPy library is that it should work with NumPy arrays. It provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization.

**Data Structure of SciPy**

The basic data structure used by SciPy is a multidimensional array provided by the NumPy module. NumPy provides some functions for Linear Algebra, Fourier Transforms, and Random Number Generation, but not with the generality of the equivalent functions in SciPy.

**Test of Normality**

**Shapiro-Wilk Test**

The Shapiro-Wilk test is a test of normality, it determines whether the given sample comes from the normal distribution or not. Shapiro-Wilk’s test or Shapiro test is a normality test in frequentist statistics. The null hypothesis of Shapiro’s test is that the population is distributed normally.

**Shapiro-Wilk test using the shapiro() function**

In this approach, the user needs to call the shapiro() function with the required parameters from the scipy.stats library to conduct the Shapiro-Wilk test on the given data in the python programming language.

This is a hypothesis test and the two hypotheses are as follows:

Ho (Accepted): The sample is from the normal distributions. (Po>0.05)

Ha (Rejected): The sample is not from the normal distributions.

from scipy.stats import shapiro

stat, p = shapiro(df["Price"])

print("stat=%.3f, p=%.3f\n" % (stat, p))

if p>0.05:

print("Probably Gaussian")

else:

print("Probably not Gaussian")

**Output Interpretation**

****

From the above calculations, the p-value is 0.954 which is more than alpha(0.05) then we fail to reject the null hypothesis i.e. we have sufficient evidence to say that sample does not come from a normal distribution.

**D’Agostino’s K^2 Test**

D’Agostino’s K-squared test check’s normality of a variable based on skewness and kurtosis. It was named by Ralph D’Agostino. Skewness is a measure of symmetry. Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution.

**D’Agostino’s K^2 test using the normaltest() function**

In this approach, the user needs to call the normaltest() function with the required parameters from the scipy.stats library to conduct the D’Agostino’s K^2 test on the given data in the python programming language.

This is a hypothesis test and the two hypotheses are as follows:

Ho (Accepted): The sample distribution is from the normal distributions. (Po>0.05)

Ha (Rejected): The sample distribution is not from the normal distributions.

from scipy.stats import normaltest

stat, p = normaltest(df["Price"])

print("stat=%.3f, p=%.3f\n" % (stat, p))

if p>0.05:

print("Probably Gaussian")

else:

print("Probably not Gaussian")

**Output Interpretation**

****

From the above calculations, the p-value is 0.734 which is more than alpha(0.05) then we fail to reject the null hypothesis i.e. we have sufficient evidence to say that sample does not come from a normal distribution.

**Anderson-Darling Test**

Anderson-Darling Normality Test is another general normality test designed to determine if the data comes from a specified distribution, in our case, the normal distribution. It was developed in 1952 by Theodore Anderson and Donald Darling.

**Anderson-Darling test using the anderson() function**

In this approach, the user needs to call the anderson() function with the required parameters from the scipy.stats library to conduct the Anderson-Darling test on the given data in the python programming language.

It gives a range of critical values, at which the null hypothesis can fail to be rejected if the calculated statistic is less than the critical value.

H0 : Price data has normal distribution

H1 : Price data does not have normal distribution

from scipy.stats import anderson

result = anderson(df["Price"])

print("stat=%.3f" % (result.statistic))

for i in range(len(result.critical\_values)):

sig\_lev, crit\_val = result.significance\_level[i], result.critical\_values[i]

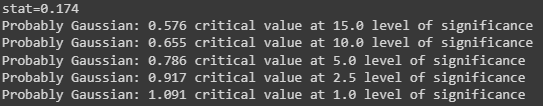
if result.statistic < crit\_val:

print(f"Probably Gaussian: {crit\_val} critical value at {sig\_lev} level of significance")

else:

print(f"Probably not Gaussian: {crit\_val} critical value at {sig\_lev} level of significance")

**Output Interpretation**

****

From above output, we see that, at each significance level, the Price data has a Gaussian Normal Distribution

**Test of Correlation**

**Pearson Correlation coefficient**

The Pearson correlation coefficient (named for Karl Pearson) can be used to summarize the strength of the linear relationship between two data samples. The Pearson’s correlation coefficient is calculated as the covariance of the two variables divided by the product of the standard deviation of each data sample. It is the normalization of the covariance between the two variables to give an interpretable score.

**Pearson Correlation test using the pearsonr() function**

In this approach, the user needs to call the pearsonr() function with the required parameters from the scipy.stats library to conduct the Pearson Correlation test on the given data in the python programming language.

The coefficient returns a value between -1 and 1 that represents the limits of correlation from a full negative correlation to a full positive correlation. A value of 0 means no correlation. The value must be interpreted, where often a value below -0.5 or above 0.5 indicates a notable correlation, and values below those values suggest a less notable correlation.

H0 : Attributes are uncorrelated

H1 : Attributes are correlated

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = df['Avg. Area Income']

list2 = df['Price']

# Apply the pearsonr()

corr, p = pearsonr(list1, list2)

print("Pearson's correlation: %.3f" % corr)

alpha = 0.05

if p > alpha:

print('Attributes are uncorrelated (fail to reject H0) p=%.3f' % p)

else:

print('Attributes are correlated (reject H0) p=%.3f' % p)

**Output Interpretation**

****

The statistical test reports a strong positive correlation with a value of 0.640. The p-value is close to zero, which means that the likelihood of observing the data given that the samples are uncorrelated is very unlikely and that we can reject the null hypothesis that the samples are uncorrelated.

**Spearman’s Rank Correlation**

It may also be called Spearman’s correlation coefficient and is denoted by the lowercase greek letter rho (p). As such, it may be referred to as Spearman’s rho. This statistical method quantifies the degree to which ranked variables are associated by a monotonic function, meaning an increasing or decreasing relationship. As a statistical hypothesis test, the method assumes that the samples are uncorrelated (fail to reject H0).

**Spearman’s Rank Correlation test using the spearmanr() function**

In this approach, the user needs to call the spearmanr() function with the required parameters from the scipy.stats library to conduct the Spearman’s Rank Correlation test on the given data in the python programming language.

The function takes two real-valued samples as arguments and returns both the correlation coefficient in the range between -1 and 1 and the p-value for interpreting the significance of the coefficient.

H0 : Attributes are uncorrelated

H1 : Attributes are correlated

from scipy.stats import spearmanr

# Convert dataframe into series

list1 = df['Avg. Area Income']

list2 = df['Price']

# Apply the pearsonr()

corr, p = spearmanr(list1, list2)

print("Spearman's correlation: %.3f" % corr)

alpha = 0.05

if p > alpha:

print('Attributes are uncorrelated (fail to reject H0) p=%.3f' % p)

else:

print('Attributes are correlated (reject H0) p=%.3f' % p)

**Output Interpretation**

****

The statistical test reports a strong positive correlation with a value of 0.620. The p-value is close to zero, which means that the likelihood of observing the data given that the samples are uncorrelated is very unlikely and that we can reject the null hypothesis that the samples are uncorrelated.

**Kendall’s Rank Correlation**

It is also called Kendall’s correlation coefficient, and the coefficient is often referred to by the lowercase Greek letter tau (t). In turn, the test may be called Kendall’s tau. The intuition for the test is that it calculates a normalized score for the number of matching or concordant rankings between the two samples. As such, the test is also referred to as Kendall’s concordance test.

**Kendall’s Rank Correlation test using the kendalltau() function**

In this approach, the user needs to call the kendalltau() function with the required parameters from the scipy.stats library to conduct the Kendall’s Rank Correlation test on the given data in the python programming language.

The test takes the two data samples as arguments and returns the correlation coefficient and the p-value. As a statistical hypothesis test, the method assumes (H0) that there is no association between the two samples.

H0 : Attributes are uncorrelated

H1 : Attributes are correlated

from scipy.stats import kendalltau

# Convert dataframe into series

list1 = df['Avg. Area Income']

list2 = df['Price']

# Apply the pearsonr()

corr, p = kendalltau(list1, list2)

print("Kendall's Rank correlation: %.3f" % corr)

alpha = 0.05

if p > alpha:

print('Attributes are uncorrelated (fail to reject H0) p=%.3f' % p)

else:

print('Attributes are correlated (reject H0) p=%.3f' % p)

**Output Interpretation**

****

The statistical test reports a positive correlation with a value of 0.441. The p-value is close to zero, which means that the likelihood of observing the data given that the samples are uncorrelated is very unlikely and that we can reject the null hypothesis that the samples are uncorrelated.

**Chi-Squared Test**

Chi-Square test is a statistical method to determine if two categorical variables have a significant correlation between them. Both those variables should be from the same population and they should be categorical.

**Chi-Squared Test using the chisquare() function**

In this approach, the user needs to call the chisquare() function with the required parameters from the scipy.stats library to conduct the Chi-Squared Test on the given data in the python programming language.

We define a significance factor to determine whether the relation between the variables is of considerable significance. Generally a significance factor or alpha value of 0.05 is chosen. This alpha value denotes the probability of erroneously rejecting H0 when it is true. A lower alpha value is chosen in cases where we expect more precision. If the p-value for the test comes out to be strictly greater than the alpha value, then H0 holds true.

If our calculated value of chi-square is less or equal to the tabular(also called critical) value of chi-square, then H0 holds true.

H0 : Attributes are uncorrelated

H1 : Attributes are correlated

from scipy.stats import chi2\_contingency

# Convert dataframe into series

list1 = df['Avg. Area Number of Bedrooms']

list2 = df['Price']

stat, p, dof, expected = chi2\_contingency(list1, list2)

print("stat=%.3f, p=%.3f\n" % (stat, p))

alpha = 0.05

if p > alpha:

print('Samples are uncorrelated (fail to reject H0) p=%.3f' % p)

else:

print('Samples are correlated (reject H0) p=%.3f' % p)

**Output Interpretation**

****

From the above calculations, the p-value is 1.000 which is more than the alpha(0.5) then we fail to reject the null hypothesis i.e. we have sufficient evidence to say that sample do not have a significant relation

**Test of Stationarity**

**Augmented Dickey-Fuller**

The Augmented Dickey-Fuller test is a type of statistical test called a unit root test. The intuition behind a unit root test is that it determines how strongly a time series is defined by a trend. It uses an autoregressive model and optimizes an information criterion across multiple different lag values.

**Augmented Dickey-Fuller using the adfuller() function**

In this approach, the user needs to call the adfuller() function with the required parameters from the scipy.stats library to conduct the Augmented Dickey-Fuller on the given data in the python programming language.

The null hypothesis of the test is that the time series can be represented by a unit root, that it is not stationary (has some time-dependent structure). The alternate hypothesis (rejecting the null hypothesis) is that the time series is stationary.

H0 : Time series can be represented by a unit root i.e. it is not stationary

H1 : Time series is stationary

from statsmodels.tsa.stattools import adfuller

import pandas as pd

import numpy as np

%matplotlib inline

url = 'https://raw.githubusercontent.com/selva86/datasets/master/a10.csv'

df = pd.read\_csv(url, parse\_dates=['date'], index\_col='date')

series = df.loc[:, 'value'].values

df.plot(figsize=(10,6), legend=None, title='a10 - Drug Sales Series');

# ADF Test

result = adfuller(series, autolag='AIC')

print(f'ADF Statistic: {result[0]}')

print(f'n\_lags: {result[2]}')

print(f'p-value: {result[1]}')

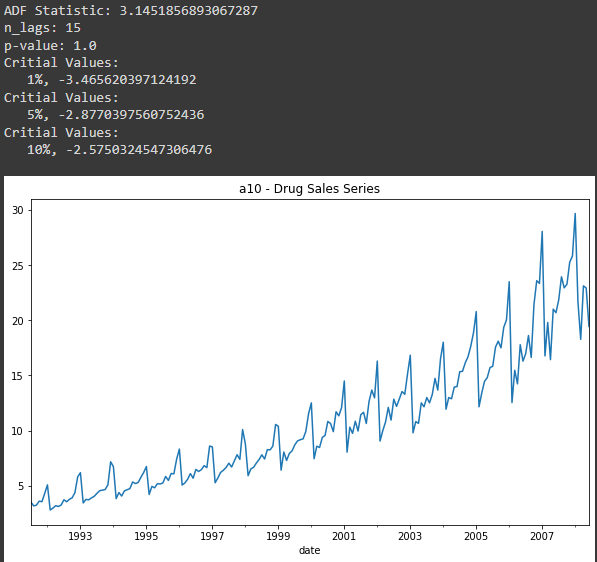
for key, value in result[4].items():

print('Critial Values:')

print(f' {key}, {value}')

print()

**Output Interpretation**



The p-value obtained is greater than the significance level of 0.05 and the ADF statistic is higher than any of the critical values. Hence, there we fail to reject the null hypothesis. So, the time series is in fact non-stationary.

**Kwiatkowski-Phillips-Schmidt-Shin**

The KPSS test, short for Kwiatkowski-Phillips-Schmidt-Shin (KPSS), is a type of Unit root test that tests for the stationarity of a given series around a deterministic trend. In other words, the test is somewhat similar in spirit with the ADF test.

**Kwiatkowski-Phillips-Schmidt-Shin using the kpss() function**

In this approach, the user needs to call the kpss() function with the required parameters from the scipy.stats library to conduct the Kwiatkowski-Phillips-Schmidt-Shin on the given data in the python programming language.

The authors of the KPSS test have defined the null hypothesis as the process is trend stationary, to an alternate hypothesis of a unit root series.

H0 : Time series trend is stationary

H1 : Time series can be represented by a unit root i.e. it is not stationary

from statsmodels.tsa.stattools import kpss

import pandas as pd

import numpy as np

%matplotlib inline

url = 'https://raw.githubusercontent.com/selva86/datasets/master/a10.csv'

df = pd.read\_csv(url, parse\_dates=['date'], index\_col='date')

series = df.loc[:, 'value'].values

df.plot(figsize=(10,6), legend=None, title='a10 - Drug Sales Series');

# KPSS Test

result = kpss(series)

print(f'KPSS Statistic: {result[0]}')

print(f'n\_lags: {result[2]}')

print(f'p-value: {result[1]}')

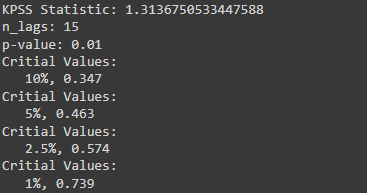
for key, value in result[3].items():

print('Critial Values:')

print(f' {key}, {value}')

print()

**Output Interpretation**

****

The p-value (0.01) obtained is lesser than the significance level of 0.05 and the KPSS statistic is higher than any of the critical values. Hence, there we reject the null hypothesis. So, the time series is non-stationary.

**Parametric Test**

**Student’s t-test**

The Student’s t-Test is a statistical hypothesis test for testing whether two samples are expected to have been drawn from the same population. It is named for the pseudonym “Student” used by William Gosset, who developed the test. The test works by checking the means from two samples to see if they are significantly different from each other.

**Student’s t-test using the ttest\_ind() function**

In this approach, the user needs to call the ttest\_ind() function with the required parameters from the scipy.stats library to conduct the Student’s t-test on the given data in the python programming language.

Calculates the standard error in the difference between means, which can be interpreted to see how likely the difference is, if the two samples have the same mean (the null hypothesis).

* If **abs(t-statistic) <= critical value**: Accept null hypothesis that the means are equal.
* If **abs(t-statistic) > critical value**: Reject the null hypothesis that the means are equal.
* If **p > alpha**: Accept null hypothesis that the means are equal.
* If **p <= alpha**: Reject null hypothesis that the means are equal.

H0 : μ1 == μ2

H1 : μ1 != μ2

from scipy.stats import ttest\_ind

from sklearn.model\_selection import train\_test\_split

train, test = train\_test\_split(df,test\_size=0.3,random\_state=1)

stat, p = ttest\_ind(train, test)

print('stat=%.3f, p=%.3f' % (stat, p))

if p > 0.05:

print('Fail to reject null hypothesis i.e. the means are equal.')

else:

print('Reject the null hypothesis i.e. the means are not equal.')

**Output Interpretation**

****

The t-statistic and the p-value are then used to interpret the results of the test. We find that p-value (0.387) is greater than the significance level 0.05, hence we fail to reject the null hypothesis, i.e. the means are equal.

**Paired Student’s t-test**

The Student’s t-Test is a statistical hypothesis test for testing whether two samples are expected to have been drawn from the same population. It is named for the pseudonym “Student” used by William Gosset, who developed the test. The test works by checking the means from two dependent samples to see if they are significantly different from each other.

**Paired Student’s t-test using the ttest\_rel() function**

In this approach, the user needs to call the ttest\_rel() function with the required parameters from the scipy.stats library to conduct the Paired Student’s t-test on the given data in the python programming language.

Calculates the standard error in the difference between means, which can be interpreted to see how likely the difference is, if the two samples have the same mean (the null hypothesis).

* If **abs(t-statistic) <= critical value**: Accept null hypothesis that the means are equal.
* If **abs(t-statistic) > critical value**: Reject the null hypothesis that the means are equal.
* If **p > alpha**: Accept null hypothesis that the means are equal.
* If **p <= alpha**: Reject null hypothesis that the means are equal.

H0 : μ1 == μ2

H1 : μ1 != μ2

from scipy.stats import ttest\_rel

df = pd.read\_csv("https://raw.githubusercontent.com/Opensourcefordatascience/Data-sets/master/blood\_pressure.csv")

# Convert dataframe into series

list1 = df['bp\_before']

list2 = df['bp\_after']

stat, p = ttest\_rel(list1, list2)

print('stat=%.3f, p=%.3f' % (stat, p))

if p > 0.05:

print('Fail to reject null hypothesis i.e. the means are equal.')

else:

print('Reject the null hypothesis i.e. the means are not equal.')

**Output Interpretation**

****

The t-statistic and the p-value are then used to interpret the results of the test. Inference: We find that p-value (0.001) is lesser than the significance level 0.05, hence we reject the null hypothesis, i.e. the means are not equal.

**Analysis of Variance Test (ANOVA)**

An Analysis of Variance Test, or ANOVA, can be thought of as a generalization of the t-tests for more than 2 groups. The independent t-test is used to compare the means of a condition between two groups. ANOVA is used when we want to compare the means of a condition between more than two groups. ANOVA tests if there is a difference in the mean somewhere in the model (testing if there was an overall effect), but it does not tell us where the difference is (if there is one).

**Analysis of Variance Test using the anova\_lm() function**

In this approach, the user needs to call the anova\_lm() function with the required parameters from the scipy.stats library to conduct the Analysis of Variance Test on the given data in the python programming language.

The observations are obtained independently and randomly from the population defined by the factor levels. The data for each factor level is normally distributed. Sample cases should be independent of each other and the variance among the groups should be approximately equal.

Null hypothesis: Groups means are equal (no variation in means of groups) H0: μ1=μ2=…=μp

Alternative hypothesis: At least, one group mean is different from other groups H1: All μ are not equal

df = pd.read\_csv("https://reneshbedre.github.io/assets/posts/anova/onewayanova.txt", sep="\t")

# reshape the dataframe suitable for statsmodels package

df\_melt = pd.melt(df.reset\_index(), id\_vars=['index'], value\_vars=['A', 'B', 'C', 'D'])

# replace column names

df\_melt.columns = ['index', 'treatments', 'value']

import statsmodels.api as sm

from statsmodels.formula.api import ols

# Ordinary Least Squares (OLS) model

mod = ols('value ~ C(treatments)', data=df\_melt).fit()

aov\_table = sm.stats.anova\_lm(mod, typ=2)

print(aov\_table)

**Output Interpretation**

****

The p value obtained from ANOVA analysis is less than 0.05, and therefore, we reject H0 i.e. the group means are different from each other.

**Repeated Measures ANOVA Test**

Repeated measures ANOVA is the equivalent of the one-way ANOVA, but for related, not independent groups, and is the extension of the dependent t-test. A repeated measures ANOVA is also referred to as a within-subjects ANOVA or ANOVA for correlated samples. All these names imply the nature of the repeated measures ANOVA, that of a test to detect any overall differences between related means.

**Repeated Measures ANOVA Test using the AnovaRM() function**

In this approach, the user needs to call the AnovaRM() function with the required parameters from the scipy.stats library to conduct the Repeated Measures ANOVA Test on the given data in the python programming language.

Null hypothesis: Groups means are equal (no variation in means of groups) H0: μ1=μ2=…=μp

Alternative hypothesis: At least, one group mean is different from other groups H1: All μ are not equal

df = pd.read\_csv("https://reneshbedre.github.io/assets/posts/anova/plants\_leaves.csv")

# reshape the dataframe suitable for statsmodels package

df\_melt = pd.melt(df.reset\_index(), id\_vars=['Id'], value\_vars=['W1', 'W2', 'W3', 'W4', 'W5'])

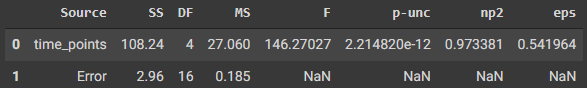
# replace column names

df\_melt.columns = ['Id', 'time\_points', 'leaves']

import pingouin as pg

pg.rm\_anova(dv='leaves', within='time\_points', subject='Id', data=df\_melt, detailed=True)

**Output Interpretation**

****

The p value obtained from Repeated Measures ANOVA analysis is less than 0.05, and therefore, we reject H0 i.e. the group means are different from each other.

**Non\_parametric Test**

**Mann-Whitney U Test**

Mann and Whitney’s U-test or Wilcoxon rank-sum test is the non-parametric statistical hypothesis test that is used to analyze the difference between two independent samples of ordinal data. In this test, we have provided two randomly drawn samples and we have to verify whether these two samples are from the same population.

**Mann-Whitney U Test using the mannwhitneyu() function**

In this approach, the user needs to call the mannwhitneyu() function with the required parameters from the scipy.stats library to conduct the Mann-Whitney U Test on the given data in the python programming language.

The default assumption or null hypothesis is that there is no difference between the distributions of the data samples. Rejection of this hypothesis suggests that there is likely some difference between the samples. More specifically, the test determines whether it is equally likely that any randomly selected observation from one sample will be greater or less than a sample in the other distribution. If violated, it suggests differing distributions.

H0 : There is no difference between the distributions of the data samples

H1 : There is likely some difference between the samples

from scipy.stats import mannwhitneyu

df = pd.read\_csv("https://raw.githubusercontent.com/Opensourcefordatascience/Data-sets/master/blood\_pressure.csv")

# Convert dataframe into series

list1 = df['bp\_before']

list2 = df['bp\_after']

stat, p = mannwhitneyu(list1, list2)

print('stat=%.3f, p=%.3f' % (stat, p))

if p > 0.05:

print('Fail to reject null hypothesis i.e. the means are equal.')

else:

print('Reject the null hypothesis i.e. the means are not equal.')

**Output Interpretation**

****

Since p value is less than 0.05, we reject the H0 i.e. there is difference between the samples.

**Wilcoxon Signed-Rank Test**

Examples of paired samples in machine learning might be the same algorithm evaluated on different datasets or different algorithms evaluated on exactly the same training and test data. The Wilcoxon signed-rank test is used, also called the Wilcoxon T test, named for Frank Wilcoxon. It is the equivalent of the paired Student T-test, but for ranked data instead of real valued data with a Gaussian distribution.

**Wilcoxon Signed-Rank Test using the wilcoxon() function**

In this approach, the user needs to call the wilcoxon() function with the required parameters from the scipy.stats library to conduct the Wilcoxon Signed-Rank Test on the given data in the python programming language.

The default assumption or null hypothesis is that there is no difference between the distributions of the data samples. Rejection of this hypothesis suggests that there is likely some difference between the samples. More specifically, the test determines whether it is equally likely that any randomly selected observation from one sample will be greater or less than a sample in the other distribution. If violated, it suggests differing distributions.

H0 : There is no difference between the distributions of the data samples

H1 : There is likely some difference between the samples

from scipy.stats import wilcoxon

df = pd.read\_csv("https://raw.githubusercontent.com/Opensourcefordatascience/Data-sets/master/blood\_pressure.csv")

# Convert dataframe into series

list1 = df['bp\_before']

list2 = df['bp\_after']

stat, p = wilcoxon(list1, list2)

print('stat=%.3f, p=%.3f' % (stat, p))

if p > 0.05:

print('Fail to reject null hypothesis i.e. the means are equal.')

else:

print('Reject the null hypothesis i.e. the means are not equal.')

**Output Interpretation**

****

Since p value is less than 0.05, we reject the H0 i.e. there is difference between the samples.

**Kruskal-Wallis H Test**

The Kruskal-Wallis test is a nonparametric version of the one-way analysis of variance test or ANOVA for short. It is named for the developers of the method, William Kruskal and Wilson Wallis. This test can be used to determine whether more than two independent samples have a different distribution. It can be thought of as the generalization of the Mann-Whitney U test.

**Kruskal-Wallis H Test using the kruskal() function**

In this approach, the user needs to call the kruskal() function with the required parameters from the scipy.stats library to conduct the Kruskal-Wallis H Test on the given data in the python programming language.

The default assumption or the null hypothesis is that all data samples were drawn from the same distribution. Specifically, that the population medians of all groups are equal. A rejection of the null hypothesis indicates that there is enough evidence to suggest that one or more samples dominate another sample, but the test does not indicate which samples or by how much.

H0 : There is no difference between the distributions of the data samples

H1 : There is likely some difference between the samples

from scipy.stats import kruskal

df = pd.read\_csv("https://raw.githubusercontent.com/Opensourcefordatascience/Data-sets/master/blood\_pressure.csv")

# Convert dataframe into series

list1 = df['bp\_before']

list2 = df['bp\_after']

stat, p = kruskal(list1, list2)

print('stat=%.3f, p=%.3f' % (stat, p))

if p > 0.05:

print('Fail to reject null hypothesis i.e. the means are equal.')

else:

print('Reject the null hypothesis i.e. the means are not equal.')

**Output Interpretation**

****

Since p value is less than 0.05, we reject the H0 i.e. there is difference between the samples.

**Friedman Test**

If the samples are paired in some way, such as repeated measures, then the Kruskal-Wallis H test would not be appropriate. Instead, the Friedman test can be used, named for Milton Friedman. The Friedman test is the nonparametric version of the repeated measures analysis of variance test, or repeated measures ANOVA. The test can be thought of as a generalization of the Kruskal-Wallis H Test to more than two samples.

**Friedman Test using the friedmanchisquare() function**

In this approach, the user needs to call the friedmanchisquare() function with the required parameters from the scipy.stats library to conduct the Friedman Test on the given data in the python programming language.

The default assumption, or null hypothesis, is that the multiple paired samples have the same distribution. A rejection of the null hypothesis indicates that one or more of the paired samples has a different distribution.

H0 : Multiple paired samples have the same distribution

H1 : One or more of the paired samples has a different distribution

from scipy.stats import friedmanchisquare

# Convert dataframe into series

list1 = df['bp\_before']

list2 = df['bp\_after']

list3 = df['bp\_before']

stat, p = friedmanchisquare(list1, list2, list3)

print('stat=%.4f, p=%.4f' % (stat, p))

if p < 0.05:

print('Reject Null Hypothesis (Significant difference between two samples)')

else:

print('Do not Reject Null Hypothesis (No significant difference between two samples)')

**Output Interpretation**

****

From the above calculations, the p-value is less than 0.05, we reject H0 i.e. there is difference between one or more pairs of samples.

**Conclusion**:

Thus, we have learnt how to implement Statistical Hypothesis using Z-Test with Scipy and Sci-kit learn libraries in Python. We performed hands-on operations on Normality, Correlation Tests, Stationary Tests, Parametric Statistical Hypothesis Tests and Nonparametric Statistical Hypothesis Tests.