St. Francis Institute of Technology, Mumbai-400 103

Department of Information Technology

A.Y. 2024-2025

Class: TE-ITA/B, Semester: VI

Subject: **Data Science Lab**

Experiment – 4: To implement statistical hypothesis tests.

1. **Aim: To implement statistical hypothesis tests Objectives:**
2. **After study of this experiment, the student will be able to**
   - Understand what are the various statistical tests available for evaluating probability of distribution of data
   - Steps involved in hypothesis testing
   - Various functions available in scipy package for hypothesis testing
   - Understand and practice analytical methods for solving real life problems based on Statistical analysis
3. **Outcomes:** After study of this experiment, the student will be able to
   - Understand the techniques of performing statistical tests available for evaluating probability of distribution of data
   - Various tests available in scipy package for the tests
   - Understand and practice analytical methods for solving real life problems based on Statistical analysis
   - Analyze the data using different statistical techniques
4. **Prerequisite:** Fundamentals of Python Programming and Database Management System.
5. **Requirements:** Python Installation, Personal Computer, Windows operating system, Internet Connection, Microsoft Word.
6. **Pre-Experiment Exercise:**

Brief Theory:

1) Normality tests

An important decision point when working with a sample of data is whether to use parametric or nonparametric statistical methods.

Parametric statistical methods assume Gaussian distribution. If a data sample is not Gaussian, then nonparametric statistical methods must be used.

Range of techniques that you can use to check if your data sample deviates from a Gaussian distribution, called normality tests.

--How whether a sample is normal dictates the types of statistical methods to use with a data sample.

--Graphical methods for qualifying deviations from normal, such as histograms and the Q-Q plot.

--Statistical normality tests for quantifying deviations from normal.

Normality Assumption There is also some middle ground where we can assume that the data is Gaussian-enough to use parametric methods or that we can use data preparation techniques to transform the data to be sufficiently Gaussian to use the parametric methods.

There are three main areas where you may need to make this evaluation of a data sample in a machine learning project; they are:

--Input data to the model in the case of fitting models.

--Model evaluation results in the case of model selection.

--Residual errors from model predictions in the case of regression.

Two classes of techniques for checking whether a sample of data is Gaussian:

1. Graphical Methods. These are methods for plotting the data and qualitatively evaluating whether the data looks Gaussian.
2. Statistical Tests. These are methods that calculate statistics on the data and quantify how likely it is that the data was drawn from a Gaussian distribution.

a) Shapiro-Wilk Test

Tests whether a data sample has a Gaussian distribution.

Assumptions

- Observations in each sample are independent and identically distributed (iid).

Interpretation

- H0: the sample has a Gaussian distribution.
- H1: the sample does not have a Gaussian distribution.

This test is named for Samuel Shapiro and Martin Wilk. The shapiro() SciPy function will calculate the Shapiro-Wilk on a given dataset. The function returns both the W-statistic calculated by the test and the p-value.

b) D'Agostino's K^2 Test

- The D'Agostino's K^2 test calculates summary statistics from the data, namely kurtosis and skewness, to determine if the data distribution departs from the normal distribution, named for Ralph D'Agostino.
- Skew is a quantification of how much a distribution is pushed left or right, a measure of asymmetry in the distribution.
- Kurtosis quantifies how much of the distribution is in the tail.
- It is a simple and commonly used statistical test for normality.(pointed means positive and flat means negative)
- The D'Agostino's K^2 test is available via the normaltest() SciPy function and returns the test statistic and the p-value.

2) Correlation tests

Variables within a dataset can be related for lots of reasons.

For example:

One variable could cause or depend on the values of another variable. One variable could be lightly associated with another variable. Two variables could depend on a third unknown variable.

A correlation could be positive, meaning both variables move in the same direction, or negative, meaning that when one variable's value increases, the other variables' values decrease. Correlation can also be neutral or zero, meaning that the variables are unrelated.

Positive Correlation: both variables change in the same direction. Neutral Correlation: No relationship in the change of the variables. Negative Correlation: variables change in opposite directions.

a) **Pearson's Correlation** 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

b) **Spearman's Correlation** (non paramtric test)

   Two variables may be related by a nonlinear relationship, such that the relationship is stronger or weaker across the distribution of the variables.

   Further, the two variables being considered may have a non-Gaussian distribution.

   If you are unsure of the distribution and possible relationships between two variables, Spearman correlation coefficient is a good tool to use.

   The spearmanr() SciPy function can be used to calculate the Spearman's correlation coefficient between two data samples with the same length.

7. Laboratory Exercise

A. **Procedure:** (separate sheet of codes is attached with the experiment)

Paste Screenshots of performance of the above commands.

8. Post-Experiments Exercise

A. Extended Theory: (Soft Copy)

Mention Parametric Statistical Hypothesis Tests

- Student's t-test

Nonparametric Statistical Hypothesis Tests

- Mann-Whitney U Test

B. Questions:

Q.1) what is a normality test?

Q.2) What is the difference between parametric statistical methods and non-parametric statistical methods?

C. Conclusion:

Write the significance of the topic studied in the experiment.

D. References:

1) https://machinelearningmastery.com/statistical-hypothesis-tests-in-python-cheat-sheet/

Name: Vishal Rajesh Mahajan AIDS EXP 04
Class: TE IT A Roll No: 56

Post-Experiments Exercise

A. Extended Theory: (Soft Copy)

Parametric Statistical Hypothesis Tests

Parametric tests assume that the data follows a specific distribution, typically a normal distribution. They are more powerful when assumptions are met but may give misleading results if assumptions are violated.

Student's t-test

- Used to compare the means of two groups.
- Assumes that the data follows a normal distribution and that variances are equal (for independent samples).
- Types of t-tests:
  - **Independent t-test**: Compares the means of two independent groups.
  - **Paired t-test**: Compares the means of the same group before and after an intervention.
  - **One-sample t-test**: Compares the sample mean with a known population mean.

Nonparametric Statistical Hypothesis Tests

Nonparametric tests do not assume a normal distribution and are useful for skewed or ordinal data. They are often used when sample sizes are small or when the assumptions of parametric tests are violated.

Mann-Whitney U Test

- Also known as the Wilcoxon rank-sum test.
- Used to compare differences between two independent groups when data is not normally distributed.
- Instead of comparing means, it compares the ranks of values in the two groups.
- Suitable for ordinal or continuous data that do not meet normality assumptions.

In [ ]:

```
from scipy.stats import ttest_1samp
import numpy as np

# Creating a sample of ages
ages = [45, 89, 23, 46, 12, 69, 45, 24, 34, 67]
print(ages)

# Calculating the mean of the sample
mean = np.mean(ages)
print("mean :",mean)

# Performing the T-Test
t_test, p_val = ttest_1samp(ages, 30)
print("P-value is: ", p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print(" We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
[45, 89, 23, 46, 12, 69, 45, 24, 34, 67]
mean : 45.4
P-value is:  0.07179988272763561
We can accept the null hypothesis
```

In [ ]:

```
# Python program to implement Independent T-Test on the two independent samples

# Importing the required libraries
from scipy.stats import ttest_ind
import numpy as np

# Creating the data groups
data_group1 = np.array([12, 18, 12, 13, 15, 1, 7,
                        20, 21, 25, 19, 31, 21, 17,
                        17, 15, 19, 15, 12, 15])
data_group2 = np.array([23, 22, 24, 25, 21, 26, 21,
                        21, 25, 30, 24, 21, 23, 19,
                        14, 18, 14, 12, 19, 15])

# Calculating the mean of the two data groups
mean1 = np.mean(data_group1)
mean2 = np.mean(data_group2)

# Print mean values
print("Data group 1 mean value:", mean1)
print("Data group 2 mean value:", mean2)

# Calculating standard deviation
std1 = np.std(data_group1)
std2 = np.std(data_group2)

# Printing standard deviation values
print("Data group 1 std value:", std1)
print("Data group 2 std value:", std2)

# Implementing the t-test
t_test,p_val = ttest_ind(data_group1, data_group2)
print("The P-value is: ", p_val)

# taking the threshold value as 0.05 or 5%
```

```
if p_val < 0.05:
    print("We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
Data group 1 mean value: 16.25
Data group 2 mean value: 20.85
Data group 1 std value: 6.171507109288622
Data group 2 std value: 4.452808102759426
The P-value is:  0.012117171124028792
We can reject the null hypothesis
```

In [1]:

```
# Python program to implement Two Sample Z-Test

# Importing the required libraries
import pandas as pd
from scipy import stats
from statsmodels.stats import weightstats as stests

# Creating a dataset
data1 = [83, 85, 86, 90, 90, 93, 93, 95, 97, 97,
         106, 108, 106, 108, 111, 113, 113, 112, 116, 111]

data2 = [92, 92, 90, 93, 93, 97, 94, 98, 109, 108,
         110, 117, 110, 115, 114, 114, 130, 130, 149, 131]

# Implementing the two-sample z-test
z_test ,p_val = stests.ztest(data1, x2 = data2, value = 0, alternative = 'two-sided')
print(p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print("We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
0.04813782199434202
We can reject the null hypothesis
```

In [ ]:

```
# Python program to implement Paired Sample T-Test on the two dependent samples

# Importing the required libraries
import pandas as pd
from scipy import stats

# Creating two samples
sample1 = [29, 30, 33, 41, 38, 36,
       35, 31, 29, 30]
sample2 = [31, 32, 33, 39, 30, 33,
        30, 28, 29, 31]

# Performing paired sample t-test
t_test, p_val = stats.ttest_rel(sample1, sample2)
print("The P-value of the test is: ", p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print("We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
The P-value of the test is:  0.15266056244408904
We can accept the null hypothesis
```

In [ ]:

```python
# Python program to implement One Sample Z-Test

# Importing the required libraries
import pandas as pd
from scipy import stats
from statsmodels.stats import weightstats as stests

# Creating a dataset
data = [89, 93, 95, 93, 97, 98, 96, 99, 93, 97,
        110, 104, 119, 105, 104, 110, 110, 112, 115, 114]

# Performing the z-test
z_test ,p_val = stests.ztest(data, x2 = None, value = 160)
print(p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print("We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
2.417334226169332e-186
We can reject the null hypothesis
```

In [ ]:

```python
# Python program to implement One-Way f-test

# Importing the required libraries
import scipy.stats

# Creating sample data
data1 = [0.0842, 0.0368, 0.0847, 0.0935, 0.0376, 0.0963, 0.0684,
             0.0758, 0.0854, 0.0855]
data2 = [0.0785, 0.0845, 0.0758, 0.0853, 0.0946, 0.0785, 0.0853,
           0.0685]
data3 = [0.0864, 0.2522, 0.0894, 0.2724, 0.0853, 0.1367, 0.853]

# Performing the F-Test
f_test, p_val = scipy.stats.f_oneway(data1, data2, data3)
print("p-value is: ", p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print(" We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
p-value is:  0.04043792126789144
 We can reject the null hypothesis
```

In [ ]:

```python
# Python program to perform 2-way F-test

# Importing the required modules
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Create a dataframe
```

```
df = pd.DataFrame({'Frequency_fertilizers': np.repeat(['daily', 'weekly'], 16),
                           'Frequency_Watering': np.repeat(['daily', 'weekly'], 16),
                           'Crop_height': [12, 14, 15, 12, 17, 21, 19, 8,
                                           5, 12, 19, 23, 23, 14, 16, 21,
                                           25, 16, 17, 13, 24, 9, 19, 4,
                                           12, 14, 15, 12, 17, 15, 18, 14]})


# Performing the two-way ANOVA test
f_test = ols('Crop_height ~ C(Frequency_fertilizers) * C(Frequency_Watering) +C(Frequency_ferti
result = sm.stats.anova_lm(f_test, type = 2)

# Printing the result
print(result)
```

```
                                                df      sum_sq     mean_sq
  \
  C(Frequency_fertilizers)                     1.0    1.531250    1.531250
  C(Frequency_Watering)                        1.0  117.945205  117.945205
  C(Frequency_fertilizers):C(Frequency_Watering)   1.0    1.643988    1.643988
  Residual                                    30.0  802.437500   26.747917

                                                       F    PR(>F)
  C(Frequency_fertilizers)                      0.057247  0.812528
  C(Frequency_Watering)                         4.409510  0.044253
  C(Frequency_fertilizers):C(Frequency_Watering)  0.061462  0.805889
  Residual                                           NaN       NaN
```

In [ ]:

```
# Python program to perform a chi-square test

# Importing the required modules
from scipy.stats import chi2_contingency

# defining our data
data = [[231, 256, 321], [245, 312, 213]]

# Performing chi-square test
test, p_val, dof, expected_val = chi2_contingency(data)

# interpreting the p-value
alpha = 0.05
print("The p-value of our test is " + str(p_val))

# Checking the hypothesis
if p_val <= alpha:
    print('We can reject the null hypothesis')
else:
    print('We can accept the null hypothesis')
```

```
The p-value of our test is 1.4585823594475804e-06
We can reject the null hypothesis
```

In [ ]:

```
# Python program to implement Mann-Whitney U Test

# Importing the required dataset
from scipy.stats import mannwhitneyu

# Creating the dataset
data1 = [0.978, 2.792, 0.248, -0.820, -0.102, -1.203, 0.102, -1.392, -1.395, -1.928]
data2 = [1.283, -0.284, -0.821, -0.792, -0.793, -0.294, 0.600, 1.294, -1.183, -0.284]

# Implementing the test
k_test, p_val = mannwhitneyu(data1, data2)
```

```
print("P-value is: ", p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print(" We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
P-value is:  0.520366020896531
We can accept the null hypothesis
```

In [ ]:

```
#Python program to implement Wilcoxon Signed-Rank Test

# Importing the required dataset
from scipy.stats import wilcoxon

# Creating the dataset
data1 = [0.978, 2.792, 0.248, -0.820, -0.102, -1.203, 0.102, -1.392, -1.395, -1.928]
data2 = [1.283, -0.284, -0.821, -0.792, -0.793, -0.294, 0.600, 1.294, -1.183, -0.284]

# Implementing the test
w_test, p_val = wilcoxon(data1, data2)
print("P-value is: ", p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print(" We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
P-value is:  0.625
We can accept the null hypothesis
```

In [ ]:

```
# Python program to implement Kruskal-Wallis H Test

# Importing the required dataset
from scipy.stats import kruskal

# Creating the dataset
data1 = [0.978, 2.792, 0.248, -0.820, -0.102, -1.203, 0.102, -1.392, -1.395, -1.928]
data2 = [1.283, -0.284, -0.821, -0.792, -0.793, -0.294, 0.600, 1.294, -1.183, -0.284]

# Implementing the test
k_test, p_val = kruskal(data1, data2)
print("P-value is: ", p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print(" We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
P-value is:  0.49612971494281877
We can accept the null hypothesis
```

In [ ]:

```
# Python program to implement Friedman Test

# Importing the required dataset
from scipy.stats import friedmanchisquare
```

```
# Creating the dataset
data1 = [0.978, 2.792, 0.248, -0.820, -0.102, -1.203, 0.102, -1.392, -1.395, -1.928]
data2 = [1.283, -0.284, -0.821, -0.792, -0.793, -0.294, 1.100, 0.294, -0.183, -1.284]
data3 = [-0.324, 1.346, 1.148, -1.258, -0.233, 0.749, 0.157, 0.529, -0.240, -1.254]

# Implementing the test
f_test, p_val = friedmanchisquare(data1, data2, data3)
print("P-value is: ", p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print(" We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
P-value is:  0.496585303791408
We can accept the null hypothesis
```

In [ ]:

```
import pandas as pd
data = pd.read_csv('diameter.csv')
H0 = 'Data is normal'
Ha = 'Data is not normal'
alpha = 0.05
from scipy.stats import shapiro
p = round(shapiro(data)[1], 2)
if p > alpha:
    print(f"{p} > {alpha}. We fail to reject Null Hypothesis. {H0}")
else:
        print(f"{p} <= {alpha}. We reject Null Hypothesis. {Ha}")
```

```
0.0 <= 0.05. We reject Null Hypothesis. Data is not normal
```

In [ ]:

```
import pandas as pd
from scipy.stats import shapiro
#pd.DataFrame.iteritems = pd.DataFrame.items
pizzas = pd.read_csv('pizzas.csv')
alpha = 0.05
# Defining Null and Alternative Hypotheses
H0 = 'data is Normally distributed'
Ha = 'data is not Normally distributed'
columnName = pizzas.columns
columnData= pizzas.values
from scipy.stats import shapiro
def check_normality(df):
        for columnName, columnData in pizzas.items():
                print('\n' + "*** Shapiro Test Results of '{}' ***".format(columnName))
                p = round(shapiro(columnData.values)[1], 2)

                if p>alpha:
                        print(f"{p} > {alpha}. We fail to reject Null Hypothesis. '{columnName}
                else:
                        print(f"{p} <= {alpha}. We reject Null Hypothesis. '{columnName}' {Ha}'

check_normality(pizzas)
```

```
*** Shapiro Test Results of 'Making Unit 1' ***
0.32 > 0.05. We fail to reject Null Hypothesis. 'Making Unit 1' data is Norma
lly distributed

*** Shapiro Test Results of 'Making Unit 2' ***
0.52 > 0.05. We fail to reject Null Hypothesis. 'Making Unit 2' data is Norma
lly distributed
```

In [ ]:

```
H0 = 'Variances are equal'
Ha = 'Variances are not equal'

from scipy.stats import levene
def check_variances(df):
        print('\n' + "*** Variances Test Results' ***")
        p = round(levene(pizzas['Making Unit 1'], pizzas['Making Unit 1'])[1],2)

        if p>alpha:
                print(f"{p} > {alpha}. We fail to reject Null Hypothesis. {H0}")
        else:
                print(f"{p} <= {alpha}. We reject Null Hypothesis. {Ha}")

check_variances(pizzas)
```

```
*** Variances Test Results' ***
1.0 > 0.05. We fail to reject Null Hypothesis. Variances are equal
```

In [ ]:

```
H0 = 'There is no significant difference.'

Ha = 'There exist a significant difference.'


from scipy.stats import ttest_ind

def t_test(df):

    print('n' + "*** 2 Sample T Test Results ***")

    test_results = ttest_ind(pizzas['Making Unit 1'], pizzas['Making Unit 1'], equal_var=True)

    p = round(test_results[1],2)

    if p>alpha:

      print(f"{p} > {alpha}. We fail to reject Null Hypothesis. {H0}")

    else:

      print(f"{p} <= {alpha}. We reject Null Hypothesis. {Ha}")

t_test(pizzas)
```

```
n*** 2 Sample T Test Results ***
1.0 > 0.05. We fail to reject Null Hypothesis. There is no significant differ
ence.
```