Documentation for dastats

Documentation

▼ Module 1 - Read_file

Function: `read_csv(filename)`

This function reads a CSV file and returns the column headers and data as separate lists.

Parameters:

- `filename` (str): The name of the CSV file to be read.

Returns:

- `headers` (list): A list containing the column headers.
- `data` (list): A list of lists containing the data rows.

Example Usage:

```
headers, data = read_csv('data.csv')
```

Function: `view_complete_data(filename)`

This function reads a CSV file and displays the complete data, including column headers.

Parameters:

- `filename` (str): The name of the CSV file to be viewed.

Returns:

- None

Example Usage:

```
view_complete_data('data.csv')
```

Function: `tail(filename, n=10)`

This function reads a CSV file and displays the last 'n' data rows, including column headers.

Parameters:

- `filename` (str): The name of the CSV file to be viewed.
- `n` (int, optional): The number of data rows to display. Defaults to 10.

Returns:

- None

Example Usage:

```
tail('data.csv', 5)
```

Function: `head(filename, n=10)`

This function reads a CSV file and displays the first 'n' data rows, including column headers.

Parameters:

- `filename` (str): The name of the CSV file to be viewed.
- `n` (int, optional): The number of data rows to display. Defaults to 10.

Returns:

- None

Example Usage:

```
head('data.csv', 5)
```

▼ Module 2- Descriptive_Statistics

Functions:

1. frequency(data)

- **Definition:** Calculates the frequency of each unique value in the data.
- **Formula:** For each unique value `x` in the input data, count the number of occurrences of `x`

```
# Example usage for frequency function
data_list = [1, 2, 3, 1, 2, 3, 1, 2, 1, 1]
frequency(data_list)

"""
    Calculate the frequency of each unique value in the data.

    Parameters:
    - data (list or dict): A list or dictionary containing the data.

    Returns:
    - None: Prints the result in a tabular format.
    """
```

2. mean(data)

1. **Definition:** Calculates the mean (average) of a list of numbers.
2. **Formula:** Mean=Sum of all values/Number of valuesMean

```
# Example usage for mean function
data_mean = [1, 2, 3, 4, 5]
print("Mean:", mean(data_mean))

"""
    Calculate the mean of a list of numbers.

    Parameters:
    - data (list): List of numeric values.

    Returns:
    - float: Mean of the input data.
    """
```

3. median(data)

- **Definition:** Calculates the median of a list of numbers.

- **Formula:** To calculate the median, the data must first be sorted in ascending order. If the number of data points is odd, the median is the middle value. If the number of data points is even, the median is the average of the two middle values.

```
data_median = [1, 3, 5, 2, 4]
print("Median:", median(data_median))

"""
    Calculate the median of a list of numbers.

    Parameters:
    - data (list): List of numeric values.

    Returns:
    - float: Median of the input data.
    """
```

4. variance(data)

- **Definition:** Calculates the variance of a list of numbers.
- **Variance** = Σ(x - Mean)^2 / (n - 1)

```
# Example usage for variance function
data_variance = [1, 2, 3, 4, 5]
print("Variance:", variance(data_variance))

"""
    Calculate the variance of a list of numbers.

    Parameters:
    - data (list): List of numeric values.

    Returns:
    - float: Variance of the input data.
    """
```

5. standard_deviation(data)

- **Definition:** Calculates the standard deviation of a list of numbers.
- **Formula:** Standard Deviation = $\sqrt{(\Sigma(x - \text{Mean})^2 / (n - 1))}$

```
data_std_dev = [1, 2, 3, 4, 5]
print("Standard Deviation:", standard_deviation(data_std_dev))
"""
    Calculate the standard deviation of a list of numbers.

    Parameters:
    - data (list): List of numeric values.

    Returns:
    - float: Standard deviation of the input data.
    """
```

6. mode(data)

Definition: Calculates the mode (most frequently occurring value) of a list of numbers.

Formula: The mode is the value(s) that appear(s) most frequently in the data.

```
# Example usage for mode function
data_mode = [1, 2, 3, 2, 4, 3, 5]
print("Mode:", mode(data_mode))
"""
    Calculate and display a summary of common descriptive statistics.

    Parameters:
    - data (list): List of numeric values.

    Returns:
    - dict: A dictionary containing summary statistics (mean, median, mode, variance, std deviation).
    """
```

7. summary_statistics(data)

The **summary statistics** function calculates and displays a summary of common descriptive statistics for a given list of numeric values. The function returns a dictionary containing the following statistics:

- Mean: The average value of the data.
- Median: The middle value in the sorted data.
- Mode: The most frequently occurring value(s) in the data.
- Variance: The measure of how spread out the data is.
- Standard Deviation: The square root of the variance, representing the average amount of deviation from the mean.

Here is an example usage of the summary statistics function:

```
data_summary = [1, 2, 3, 4, 5]
print("Summary Statistics:")
print(summary_statistics(data_summary))

"""
    Calculate and display a summary of common descriptive statistics.

    Parameters:
    - data (list): List of numeric values.

    Returns:
    - dict: A dictionary containing summary statistics (mean, median, mode, variance, std deviatio
n).
    """
```

8. quantiles(data, percentiles=None)

- **Definition:** Calculates the quantiles (including median, quartiles, and percentiles) of the data.
- **Formula:** Quantiles are values that divide a dataset into equal-sized groups. The median is the 50th percentile, dividing the data into two equal halves. The first quartile (Q1) is the 25th percentile, dividing the data into the lower 25%. The third quartile (Q3) is the 75th percentile, dividing the data into the upper 25%.

```
data_quantiles = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Quantiles:")
print(quantiles(data_quantiles, percentiles=[25, 50, 75]))

"""
    Calculate the quantiles (including median, quartiles, and percentiles) of the data.

    Parameters:
    - data (list): List of numeric values.
    - percentiles (list or tuple of floats, optional): The desired percentiles. Default is [25, 5
0, 75].

    Returns:
    - dict: A dictionary containing the specified quantiles.
    """
```

▼ Module 3 - Correlation_tests

1. rank(data)

Definition: The rank function assigns ranks to data points in ascending order.

Example usage:

```
data_rank = [3, 1, 4, 2, 5]
print("Ranks:", rank(data_rank))
```

This will return the ranks of the data points: `[2, 1, 3, 2, 4]`.

```
    """
    Assign ranks to data points.

    Parameters:
    - data (list or array-like): The input data.

    Returns:
    - list: The ranks assigned to each data point.
    """
```

2. spearman_rank_correlation(data_x, data_y)

Definition: The Spearman rank correlation coefficient (ρ) is a non-parametric measure of the monotonic relationship between two variables. It assesses how well the relationship between two variables can be described using a monotonic function.

Formula:

1. Calculate the ranks of each data point in both data_x and data_y.
2. Compute the differences between the ranks of corresponding data points: d = rank_x - rank_y.
3. Calculate the sum of squared differences of ranks: Σ(d^2).
4. Calculate the Spearman's rank correlation coefficient:

   ρ = 1 - (6 * Σ(d^2)) / (n * (n^2 - 1))

where n is the number of data points.

```
# Example usage for spearman_rank_correlation function
data_x = [1, 2, 3, 4, 5]
data_y = [5, 4, 3, 2, 1]
print("Spearman Rank Correlation:", spearman_rank_correlation(data_x, data_y))
```

```
"""
    Calculate the Spearman rank correlation coefficient between two variables.

    Parameters:
    - data_x (list or array-like): The first set of data points.
    - data_y (list or array-like): The second set of data points.

    Returns:
    - float: The Spearman rank correlation coefficient between the two variables.
    """
```

Statistical Function and Conclusion

The `spearman_rank_correlation()` function calculates the Spearman rank correlation coefficient between two sets of data points. It measures the strength and direction of the monotonic relationship between the variables. The coefficient ranges from -1 to 1, where a value close to 1 indicates a strong positive monotonic relationship, a value close to -1 indicates a strong negative monotonic relationship, and a value close to 0 indicates a weak or no monotonic relationship.

In conclusion, the Spearman rank correlation coefficient provides a useful measure for analyzing the monotonic relationship between two variables, even when the relationship is not linear. It is particularly valuable when dealing with ordinal or ranked data.

3. kendalls_rank_correlation(data1, data2)

Definition: The Kendall's rank correlation coefficient (τ) is a non-parametric measure of the strength and direction of the monotonic relationship between two variables. It measures the similarity of the ranking order of the data points.

Formula:

1. Calculate the number of concordant pairs (C) and discordant pairs (D) in the data.
2. Calculate the Kendall's rank correlation coefficient:

   τ = (C - D) / sqrt((C + D + T) * (C + D + U))

   where T is the number of ties in data1 and U is the number of ties in data2.

```
# Example usage for kendalls_rank_correlation function
data1 = [1, 2, 3, 4, 5]
data2 = [5, 4, 3, 2, 1]
print("Kendall's Rank Correlation:", kendalls_rank_correlation(data1, data2))

"""
    Calculate the Kendall's rank correlation coefficient between two variables.

    Parameters:
    - data1 (list or array-like): The first set of data points.
```

```
    - data2 (list or array-like): The second set of data points.

    Returns:
    - float: The Kendall's rank correlation coefficient between the two variables.
    """
```

The `kendalls_rank_correlation()` function calculates the Kendall's rank correlation coefficient between two sets of data points. It measures the strength and direction of the monotonic relationship between the variables. The coefficient ranges from -1 to 1, where a value close to 1 indicates a strong positive monotonic relationship, a value close to -1 indicates a strong negative monotonic relationship, and a value close to 0 indicates a weak or no monotonic relationship.

In conclusion, the Kendall's rank correlation coefficient is a valuable measure for analyzing the monotonic relationship between two variables, particularly when dealing with ranked or ordinal data.

▼ Module 4 - Hypothesis_testing

1. ttest_ind(data1, data2)

- **Definition:** Performs an independent two-sample t-test to compare the means of two groups of data.
- **Assumptions:** The t-test assumes that the populations being compared are normally distributed and have equal variances.
- **Null Hypothesis (H0):** The means of the two groups are equal.
- **Alternative Hypothesis (H1):** The means of the two groups are not equal.

```
# Example usage for ttest_ind function
group1 = [1, 2, 3, 4, 5]
group2 = [2, 4, 6, 8, 10]
t_statistic, p_value = ttest_ind(group1, group2)
print("T-Statistic:", t_statistic)
print("P-Value:", p_value)

"""
    Perform an independent two-sample t-test to compare the means of two groups of data.

    Parameters:
    - data1 (list or array-like): The first group of data points.
    - data2 (list or array-like): The second group of data points.

    Returns:
    - float: The t-statistic calculated from the data.
    - float: The p-value associated with the t-statistic.
    """
```

The `ttest_ind()` function is used to perform an independent two-sample t-test on two groups of data. It calculates the t-statistic and the corresponding p-value to assess whether the means of the two groups are

significantly different. The function assumes that the populations being compared are normally distributed and have equal variances.

2. chi2_contingency(observed)

- **Definition:** Performs a chi-square test of independence to determine whether there is a significant association between two categorical variables.
- **Assumptions:** The chi-square test assumes that the observed frequencies are independent and that the expected frequencies are greater than 5 for each cell in the contingency table.
- **Null Hypothesis (H0):** There is no significant association between the two variables.
- **Alternative Hypothesis (H1):** There is a significant association between the two variables.

```
# Example usage for chi2_contingency function
observed = [[10, 20, 30], [15, 25, 35]]
chi2_statistic, p_value, degrees_of_freedom, expected = chi2_contingency(observed)
print("Chi-Square Statistic:", chi2_statistic)
print("P-Value:", p_value)
print("Degrees of Freedom:", degrees_of_freedom)
print("Expected Frequencies:", expected)

"""
    Perform a chi-square test of independence to determine whether there is a significant associat
ion between two categorical variables.

    Parameters:
    - observed (array-like): The observed frequencies in a contingency table.

    Returns:
    - float: The chi-square test statistic.
    - float: The p-value associated with the chi-square test.
    - int: The degrees of freedom.
    - array: The expected frequencies under the null hypothesis.
    """
```

The `chi2_contingency()` function is used to perform a chi-square test of independence on a contingency table. It calculates the chi-square test statistic, the corresponding p-value, the degrees of freedom, and the expected frequencies under the null hypothesis. The function assesses whether there is a significant association between two categorical variables.

3. ztest(data, value=None, alternative='two-sided')

- **Definition:** Performs a z-test to compare a sample mean to a known population mean or to test the difference between two sample means.
- **Assumptions:** The z-test assumes that the sample data is normally distributed and that the population standard deviation is known.

- **Null Hypothesis (H0):** The sample mean is equal to the population mean (or the difference between the sample means is zero).
- **Alternative Hypothesis (H1):** The sample mean is not equal to the population mean (or the difference between the sample means is not zero).

```
# Example usage for ztest function
data = [1, 2, 3, 4, 5]
z_statistic, p_value = ztest(data, value=3.5, alternative='two-sided')
print("Z-Statistic:", z_statistic)
print("P-Value:", p_value)

"""
    Perform a z-test to compare a sample mean to a known population mean or to test the difference
between two sample means.

    Parameters:
    - data (array-like): The sample data.
    - value (float or None, optional): The known population mean (for one-sample z-test) or the hy
pothesized difference between sample means (for two-sample z-test). Default is None.
    - alternative (str, optional): The alternative hypothesis. Possible values are 'two-sided', 'l
ess', and 'greater'. Default is 'two-sided'.

    Returns:
    - float: The z-test statistic.
    - float: The p-value associated with the z-test.
    """
```

The `ztest()` function is used to perform a z-test on sample data. It compares the sample mean to a known population mean or tests the difference between two sample means. The function calculates the z-test statistic and the corresponding p-value to assess whether the sample mean(s) is significantly different from the population mean or whether the difference between the sample means is significantly different from zero. The z-test assumes that the sample data is normally distributed and that the population standard deviation is known.

4. ttest_rel(data1, data2)

- **Definition:** Performs a paired two-sample t-test to compare the means of two related groups of data.
- **Assumptions:** The t-test assumes that the differences between the paired observations are normally distributed.
- **Null Hypothesis (H0):** The means of the two groups are equal.
- **Alternative Hypothesis (H1):** The means of the two groups are not equal.

```
# Example usage for ttest_rel function
data1 = [1, 2, 3, 4, 5]
data2 = [2, 3, 4, 5, 6]
t_statistic, p_value = ttest_rel(data1, data2)
```

```
print("T-Statistic:", t_statistic)
print("P-Value:", p_value)

"""
    Perform a paired two-sample t-test to compare the means of two related groups of data.

    Parameters:
    - data1 (list or array-like): The first group of data points.
    - data2 (list or array-like): The second group of data points.

    Returns:
    - float: The t-statistic calculated from the data.
    - float: The p-value associated with the t-statistic.
    """
```

The `ttest_rel()` function is used to perform a paired two-sample t-test on two related groups of data. It calculates the t-statistic and the corresponding p-value to assess whether the means of the two groups are significantly different. The function assumes that the differences between the paired observations are normally distributed.

5. annova(*groups)

- **Definition:** Performs a one-way analysis of variance (ANOVA) to compare the means of two or more groups of data.
- **Assumptions:** The ANOVA assumes that the populations being compared are normally distributed and have equal variances.
- **Null Hypothesis (H0):** The means of all groups are equal.
- **Alternative Hypothesis (H1):** At least one group mean is significantly different from the others.

```
# Example usage for f_oneway function
group1 = [1, 2, 3, 4, 5]
group2 = [2, 4, 6, 8, 10]
group3 = [3, 6, 9, 12, 15]
f_statistic, p_value = f_oneway(group1, group2, group3)
print("F-Statistic:", f_statistic)
print("P-Value:", p_value)

"""
    Perform a one-way analysis of variance (ANOVA) to compare the means of two or more groups of d
ata.

    Parameters:
    - *groups (array-like): Two or more groups of data points.

    Returns:
    - float: The F-statistic calculated from the data.
    - float: The p-value associated with the F-statistic.
    """
```

The annova() function is used to perform a one-way analysis of variance (ANOVA) on two or more groups of data. It calculates the F-statistic and the corresponding p-value to assess whether the means of all groups are significantly different. The function assumes that the populations being compared are normally distributed and have equal variances.

▼ Module 5 - Data_Distribution_Test

1. test_parametric_test()

The `test_parametric_test()` function is used to perform a parametric test on a dataset. It assumes that the data follows a specific distribution, such as the normal distribution. This function can be used to test hypotheses about the population mean, population proportion, or the difference between two population means.

```
# Example usage for test_parametric_test function
data = [1, 2, 3, 4, 5]
result = test_parametric_test(data)
print("Parametric Test Result:", result)

"""
    Perform a parametric test on a dataset.

    Parameters:
    - data (array-like): The dataset to be tested.

    Returns:
    - str: The result of the parametric test.
    """
```

2. test_nonparametric_test()

The `test_nonparametric_test()` function is used to perform a nonparametric test on a dataset. It does not assume any specific distribution for the data and can be used when the data does not meet the assumptions of parametric tests. This function can be used to test hypotheses about the population median, the difference between two population medians, or the association between two variables.

```
# Example usage for test_nonparametric_test function
data = [1, 2, 3, 4, 5]
result = test_nonparametric_test(data)
print("Nonparametric Test Result:", result)

"""
    Perform a nonparametric test on a dataset.
```

```
    Parameters:
    - data (array-like): The dataset to be tested.

    Returns:
    - str: The result of the nonparametric test.
    """
```

The `test_parametric_test()` and `test_nonparametric_test()` functions are useful tools for testing hypotheses about datasets. Depending on the nature of the data and the assumptions that can be made, you can choose the appropriate function to perform the desired test.

▼ Module 6 - Non_Parametric_Test

6. Non_Parametric_Test

1. mann_whitney_u_test(sample1, sample2)

- **Definition:** The Mann-Whitney U test, also known as the Wilcoxon rank-sum test, is a non-parametric test used to determine whether there is a significant difference between the distributions of two independent samples.
- **Assumptions:** The Mann-Whitney U test assumes that the samples are independent and that the observations within each sample are independent and identically distributed.
- **Null Hypothesis (H0):** There is no significant difference between the distributions of the two samples.
- **Alternative Hypothesis (H1):** There is a significant difference between the distributions of the two samples.

```
# Example usage for mann_whitney_u_test function
sample1 = [1, 2, 3, 4, 5]
sample2 = [2, 4, 6, 8, 10]
u_statistic, p_value = mann_whitney_u_test(sample1, sample2)
print("U-Statistic:", u_statistic)
print("P-Value:", p_value)

"""
    Perform the Mann-Whitney U test to determine whether there is a significant difference between
the distributions of two independent samples.

    Parameters:
    - sample1 (array-like): The first sample.
    - sample2 (array-like): The second sample.

    Returns:
    - float: The U-statistic calculated from the samples.
    - float: The p-value associated with the U-statistic.
```

```
    """
```

2. wilcoxon_signed_rank_test(sample1, sample2)

- **Definition:** The Wilcoxon signed-rank test is a non-parametric test used to determine whether there is a significant difference between the distributions of two related samples (paired samples).
- **Assumptions:** The Wilcoxon signed-rank test assumes that the differences between the paired observations are independent and identically distributed.
- **Null Hypothesis (H0):** There is no significant difference between the distributions of the two related samples.
- **Alternative Hypothesis (H1):** There is a significant difference between the distributions of the two related samples.

```
# Example usage for wilcoxon_signed_rank_test function
sample1 = [1, 2, 3, 4, 5]
sample2 = [2, 3, 4, 5, 6]
w_statistic, p_value = wilcoxon_signed_rank_test(sample1, sample2)
print("W-Statistic:", w_statistic)
print("P-Value:", p_value)

"""
    Perform the Wilcoxon signed-rank test to determine whether there is a significant difference b
etween the distributions of two related samples.

    Parameters:
    - sample1 (array-like): The first sample.
    - sample2 (array-like): The second sample.

    Returns:
    - float: The W-statistic calculated from the samples.
    - float: The p-value associated with the W-statistic.
    """
```

3. friedman_test(*samples)

- **Definition:** The Friedman test is a non-parametric test used to determine whether there is a significant difference between the distributions of two or more related samples.
- **Assumptions:** The Friedman test assumes that the differences between the paired observations are independent and identically distributed.
- **Null Hypothesis (H0):** There is no significant difference between the distributions of the related samples.

- **Alternative Hypothesis (H1):** There is a significant difference between the distributions of the related samples.

```
# Example usage for friedman_test function
sample1 = [1, 2, 3, 4, 5]
sample2 = [2, 3, 4, 5, 6]
sample3 = [3, 4, 5, 6, 7]
chi2_statistic, p_value = friedman_test(sample1, sample2, sample3)
print("Chi-Square Statistic:", chi2_statistic)
print("P-Value:", p_value)

"""
    Perform the Friedman test to determine whether there is a significant difference between the d
istributions of two or more related samples.

    Parameters:
    - *samples (array-like): Two or more related samples.

    Returns:
    - float: The chi-square test statistic.
    - float: The p-value associated with the chi-square test.
    """
```

The `mann_whitney_u_test()`, `wilcoxon_signed_rank_test()`, and `friedman_test()` functions are used to perform non-parametric tests to assess whether there are significant differences between the distributions of samples. Depending on the nature of the data and the research question, you can choose the appropriate function to perform the desired test.

▼ Module 7 - Normality_Test

7. Hypothesis Testing

1. Kolmogorov-Smirnov Testdata)

The Kolmogorov-Smirnov test is a non-parametric test used to determine whether a sample follows a specified distribution. It compares the cumulative distribution function (CDF) of the sample with the theoretical CDF of the specified distribution.

1. Kolmogorov-Smirnov Test(data)

Definition: The Kolmogorov-Smirnov test assesses the goodness-of-fit between the observed sample data and the hypothesized distribution. It tests whether the differences between the empirical distribution function (EDF) and the theoretical distribution function are significant.

Formula:
The test statistic, denoted as D, is calculated as the maximum absolute difference between the EDF and the theoretical CDF:

D = max|F(x) - G(x)|

where F(x) is the EDF and G(x) is the theoretical CDF.

The p-value associated with the test statistic is calculated based on the distribution of the Kolmogorov-Smirnov test statistic under the null hypothesis. It indicates the probability of observing a test statistic as extreme as the one calculated if the null hypothesis is true.

The Kolmogorov-Smirnov test is useful for determining whether a given sample follows a specified distribution or for comparing two samples to assess whether they are drawn from the same distribution. It is commonly used in fields such as statistics, economics, and engineering.

```
# Example usage for kolmogorov_smirnov_test function
data = [1, 2, 3, 4, 5]
test_statistic, p_value = kolmogorov_smirnov_test(data, distribution='normal')
print("Test Statistic:", test_statistic)
print("P-Value:", p_value)

"""
    Perform the Kolmogorov-Smirnov test to determine whether a sample follows a specified distribu
tion.

    Parameters:
    - data (array-like): The sample data.
    - distribution (str, optional): The specified distribution. Possible values are 'normal', 'uni
form', 'exponential', etc. Default is 'normal'.

    Returns:
    - float: The test statistic calculated from the data.
    - float: The p-value associated with the test statistic.
    """
```

2. Shapiro-Wilk Test(

The Shapiro-Wilk test is a statistical test used to assess whether a sample of data follows a normal distribution. It is based on the calculation of the test statistic W and its comparison to critical values. The null hypothesis of the test is that the data is normally distributed.

The formula for calculating the test statistic W is as follows:

W = (Σ(ai * zi))^2 / Σ(ai^2)

where ai are the coefficients derived from the sample size and the expected values of the order statistics, and zi are the standardized values of the ordered observations.

The p-value associated with the test statistic is calculated based on the distribution of W under the null hypothesis. A small p-value indicates evidence against the null hypothesis, suggesting that the data does not follow a normal distribution.

The Shapiro-Wilk test is commonly used in various fields, including statistics, biology, and social sciences, to assess the normality assumption of data before applying certain statistical techniques.

```
# Example usage for shapiro_wilk_test function
data = [1, 2, 3, 4, 5]
test_statistic, p_value = shapiro_wilk_test(data)
print("Test Statistic:", test_statistic)
print("P-Value:", p_value)

"""
    Perform the Shapiro-Wilk test to determine whether a sample follows a normal distribution.

    Parameters:
    - data (array-like): The sample data.

    Returns:
    - float: The test statistic calculated from the data.
    - float: The p-value associated with the test statistic.
    """
```

The `kolmogorov_smirnov_test()` and `shapiro_wilk_test()` functions are useful tools for testing hypotheses about the distribution of data. The Kolmogorov-Smirnov test can be used to determine whether a sample follows a specified distribution, while the Shapiro-Wilk test is specifically designed to test for normality.

▼ Module 8 - Regression_analysis

1. linear_regression_fit(X, y):

Statistical Definition:
Linear regression is a statistical modeling technique used to determine the relationship between a dependent variable and one or more independent variables. It seeks to find the best-fitting line that represents the linear relationship between the variables.

Formula:

The formula for linear regression (x, y) is:

```
y = b0 + b1*x
```

Where:

- `y` is the dependent variable,
- `x` is the independent variable,
- `b0` is the intercept (the value of `y` when `x` is 0),

- `b1` is the slope (the change in `y` for a one-unit change in `x`).

Python Example:

```
# Example usage
X = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

intercept, slope = linear_regression_fit(X, y)
```

Documentation:

```
def linear_regression_fit(X, y):
    """
    Fits a linear regression model to the given data.

    Args:
        X (array-like): The independent variable.
        y (array-like): The dependent variable.

    Returns:
        float: The intercept of the regression line.
        float: The slope of the regression line.
    """
    # Implementation goes here
```

2. linear_regression_predict(intercept, slope, X)

Statistical Definition:
Linear regression prediction is the process of using a trained linear regression model to estimate the values of the dependent variable based on the given independent variable(s) and the learned coefficients (intercept and slope).

Formula:

The formula for linear regression prediction is:

```
y = intercept + slope * x
```

Python Example:

```
# Example usage
intercept = 1.0
slope = 2.0
X = [1, 2, 3, 4, 5]
```

```
predicted_values = linear_regression_predict(intercept, slope, X)
```

Documentation:

```
def linear_regression_predict(intercept, slope, X):
    """
    Predicts the values using the linear regression model.

    Args:
        intercept (float): The intercept of the regression line.
        slope (float): The slope of the regression line.
        X (array-like): The independent variable.

    Returns:
        array-like: The predicted values.
    """
    # Implementation goes here
```

2. calculate_r_squared(actual_values, predicted_values)

Statistical Definition:
The coefficient of determination, commonly known as R-squared, is a statistical measure that represents the proportion of the variance in the dependent variable that can be explained by the independent variable(s). It provides an indication of the goodness of fit of a linear regression model.

Formula:

The formula for calculating the coefficient of determination (R-squared) is as follows:

```
R-squared = 1 - (SSR / SST)
```

Where:

- SSR (Sum of Squared Residuals) represents the sum of the squared differences between the predicted values and the mean of the actual values.
- SST (Total Sum of Squares) represents the sum of the squared differences between the actual values and the mean of the actual values.

Interpretation of R-squared:
R-squared is a statistical measure that ranges from 0 to 1. It represents the proportion of the variance in the dependent variable that can be explained by the independent variable(s).

- An R-squared value of 0 indicates that the independent variable(s) does not explain any of the variance in the dependent variable.
- An R-squared value of 1 indicates that the independent variable(s) perfectly explain the variance in the dependent variable.

In general, a higher R-squared value indicates a better fit of the regression model to the data. However, it is important to note that R-squared alone does not determine the validity or reliability of the regression model. It should be used in conjunction with other statistical measures and domain knowledge to evaluate the model's performance.

Python Example:

```
# Example usage
actual_values = [2, 4, 6, 8, 10]
predicted_values = [2.5, 4.2, 6.3, 8.1, 10.7]

r_squared = calculate_r_squared(actual_values, predicted_values)
```

Documentation:

```
def calculate_r_squared(actual_values, predicted_values):
    """
    Calculates the coefficient of determination (R-squared) for the predicted values.

    Args:
        actual_values (array-like): The actual values.
        predicted_values (array-like): The predicted values.

    Returns:
        float: The R-squared value.
    """
    # Implementation goes here
```

3. calculate_mse(actual_values, predicted_values)

Statistical Definition:
Mean squared error (MSE) is a statistical measure that calculates the average squared difference between the actual values and the predicted values. It provides a measure of the average squared deviation of the predicted values from the actual values.

Formula:

The formula for Mean Squared Error (MSE) is:

MSE = (1/n) * Σ(y - ŷ)^2

Where:

- n is the number of data points
- y is the actual value
- ŷ is the predicted value

Note: The Σ symbol represents the summation of the squared differences between the actual and predicted values for each data point.

Python Example:

```
# Example usage
actual_values = [2, 4, 6, 8, 10]
predicted_values = [2.5, 4.2, 6.3, 8.1, 10.7]

mse = calculate_mse(actual_values, predicted_values)
```

Documentation:

```
def calculate_mse(actual_values, predicted_values):
    """
    Calculates the mean squared error (MSE) for the predicted values.

    Args:
        actual_values (array-like): The actual values.
        predicted_values (array-like): The predicted values.

    Returns:
        float: The MSE value.
    """
    # Implementation goes here
```

4. calculate_rmse(actual_values, predicted_values)

Statistical Definition:
Root mean squared error (RMSE) is a statistical measure that calculates the square root of the average squared difference between the actual values and the predicted values. It provides a measure of the average deviation of the predicted values from the actual values, in the same unit as the dependent variable.

Formula:

The formula for Root Mean Squared Error (RMSE) is:

RMSE = sqrt((1/n) * Σ(y - ŷ)^2)

Where:

- n is the number of data points
- y is the actual value
- ŷ is the predicted value

Note: The Σ symbol represents the summation of the squared differences between the actual and predicted values for each data point.

Python Example:

```
# Example usage
actual_values = [2, 4, 6, 8, 10]
predicted_values = [2.5, 4.2, 6.3, 8.1, 10.7]

rmse = calculate_rmse(actual_values, predicted_values)
```

Documentation:

```
def calculate_rmse(actual_values, predicted_values):
    """
    Calculates the root mean squared error (RMSE) for the predicted values.

    Args:
        actual_values (array-like): The actual values.
        predicted_values (array-like): The predicted values.

    Returns:
        float: The RMSE value.
    """
    # Implementation goes here
```

5. adf_test(time_series)

Statistical Definition:
The Augmented Dickey-Fuller (ADF) test is a statistical test used to determine the presence of a unit root in a time series data. It tests whether a time series is stationary or not, which is an important assumption in many time series analysis techniques.

Formula:

Formula:
The formula for the Augmented Dickey-Fuller (ADF) test is as follows:

ADF test statistic = (Estimated Test Statistic - Null Hypothesis Value) / Standard Error

Hypothesis:
The null hypothesis of the ADF test is that the time series has a unit root, implying that it is non-stationary. The alternative hypothesis is that the time series is stationary.

- Null Hypothesis (H0): The time series has a unit root (it is non-stationary).
- Alternative Hypothesis (HA): The time series is stationary.

Interpretation:

- If the p-value is less than the chosen significance level (e.g., 0.05), we reject the null hypothesis and conclude that the time series is stationary.
- If the p-value is greater than the chosen significance level, we fail to reject the null hypothesis and conclude that the time series has a unit root and is non-stationary.

Keep in mind that the ADF test also provides critical values at different confidence levels, which can be used to compare the test statistic against and make a more informed decision.

Python Example:

```
# Example usage
time_series = [1, 2, 3, 4, 5]

adf_statistic, p_value, num_lags, critical_values = adf_test(time_series)
```

Documentation:

```
def adf_test(time_series):
    """
    Performs the Augmented Dickey-Fuller (ADF) test on the given time series data.

    Args:
        time_series (array-like): The time series data.

    Returns:
        float: The ADF test statistic.
        float: The p-value.
        int: The number of lags used in the regression.
        dict: The critical values at different confidence levels.
    """
    # Implementation goes here
```

6. kpss_test(time_series)

Statistical Definition:
The Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test is a statistical test used to determine the presence of

a unit root in a time series data. It tests whether a time series is stationary or not, which is an important assumption in many time series analysis techniques.

Formula:

The formula for the KPSS test is as follows:

KPSS test statistic = (Estimated Test Statistic - Null Hypothesis Value) / Standard Error

Hypothesis:

The null hypothesis of the KPSS test is that the time series is stationary. The alternative hypothesis is that the time series has a unit root and is non-stationary.

- Null Hypothesis (H0): The time series is stationary.
- Alternative Hypothesis (HA): The time series has a unit root and is non-stationary.

Interpretation:

- If the p-value is less than the chosen significance level (e.g., 0.05), we reject the null hypothesis and conclude that the time series has a unit root and is non-stationary.
- If the p-value is greater than the chosen significance level, we fail to reject the null hypothesis and conclude that the time series is stationary.

Keep in mind that the KPSS test also provides critical values at different confidence levels, which can be used to compare the test statistic against and make a more informed decision.

Python Example:

```
# Example usage
time_series = [1, 2, 3, 4, 5]

kpss_statistic, p_value, num_lags, critical_values = kpss_test(time_series)
```

Documentation:

```
def kpss_test(time_series):
    """
    Performs the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test on the given time series data.

    Args:
        time_series (array-like): The time series data.

    Returns:
        float: The KPSS test statistic.
        float: The p-value.
        int: The number of lags used in the regression.
        dict: The critical values at different confidence levels.
```

```
    """
```