Assignment1:

Isnull

Head sum

Remove null values: fillna

Type conversion: astype(‘int’)

Categorical to quantitive get\_dummies:

# Convert categorical column to dummy variables

dummies = pd.get\_dummies(df['categorical\_column'])

# Concatenate the original DataFrame with the dummy variables

df\_with\_dummies = pd.concat([df, dummies], axis=1)

# Print the updated DataFrame

print(df\_with\_dummies)

To convert a categorical column to a numeric representation in a pandas DataFrame, you can use various techniques. One common approach is to use the **LabelEncoder** class from the scikit-learn library. This class transforms categorical labels into numeric values.

Here's an example of how to convert a categorical column to numeric using **LabelEncoder**:

import pandas as pd

from sklearn.preprocessing import LabelEncoder

# Assuming your dataset is in a CSV file named 'dataset.csv'

df = pd.read\_csv('dataset.csv')

# Convert the 'categorical\_column' from categorical to numeric

le = LabelEncoder()

df['categorical\_column'] = le.fit\_transform(df['categorical\_column'])

A histogram is a graphical representation of the distribution of a dataset. It provides a visual summary of the frequencies or counts of data points falling within different ranges, also known as bins or intervals.

# Print the updated DataFrame

print(df)

**Here are two commonly used methods for data normalization:**

1. Min-Max Scaling (Normalization): This method scales the data to a fixed range, typically between 0 and 1. It can be achieved using the following formula:

X\_normalized = (X - X.min()) / (X.max() - X.min())

1. Standardization (Z-score normalization): Standardization transforms the data to have a mean of 0 and a standard deviation of 1. It can be calculated using the following formula:

X\_standardized = (X - X.mean()) / X.std()

To handle outliers using the interquartile range (IQR) method, you can follow these steps:

1. Calculate the IQR for the variable of interest. The IQR is the range between the 75th percentile (Q3) and the 25th percentile (Q1) of the data.
2. Define a threshold to determine outliers. Commonly, outliers are considered values that fall below Q1 - 1.5 \* IQR or above Q3 + 1.5 \* IQR. Adjust the threshold based on the specific requirements of your analysis.
3. Identify the outliers by comparing each data point to the threshold.
4. Decide how to handle the outliers. You can either remove them from the dataset or apply a transformation to minimize their impact.

Here's an example code snippet demonstrating how to handle outliers using the IQR method:

import pandas as pd

# Assuming your dataset is already loaded into a DataFrame named 'df'

# Define the variable of interest

variable\_of\_interest = 'column\_name'

# Calculate the IQR

Q1 = df[variable\_of\_interest].quantile(0.25)

Q3 = df[variable\_of\_interest].quantile(0.75)

IQR = Q3 - Q1

# Define the outlier threshold

outlier\_threshold = 1.5 \* IQR

# Identify the outliers

outliers = df[(df[variable\_of\_interest] < Q1 - outlier\_threshold) | (df[variable\_of\_interest] > Q3 + outlier\_threshold)]

# Decide how to handle the outliers

# Option 1: Remove outliers from the dataset

df\_filtered = df[~((df[variable\_of\_interest] < Q1 - outlier\_threshold) | (df[variable\_of\_interest] > Q3 + outlier\_threshold))]

# Option 2: Apply a transformation to minimize the impact of outliers

# Example: Cap the outliers at a certain threshold

# Apply a transformation to minimize the impact of outliers

df\_transformed = df.copy()

df\_transformed[variable\_of\_interest] = np.clip(df\_transformed[variable\_of\_interest], Q1 - outlier\_threshold, Q3 + outlier\_threshold)

# Print the resulting DataFrame after outlier handling

print(df\_filtered)

print(df\_transformed)

In this updated code, the **numpy.clip()** function is used to clip the values of the **variable\_of\_interest** column within the range defined by Q1 - **outlier\_threshold** and Q3 + **outlier\_threshold**. This replaces the original values that fall outside this range with the corresponding boundary values.

Logistic Regression is a statistical algorithm used for binary classification problems, where the goal is to predict the probability of an instance belonging to a particular class. Despite its name, logistic regression is primarily used for classification tasks rather than regression tasks.

In logistic regression, the dependent variable is categorical and represents the binary outcome (e.g., Yes/No, True/False, 0/1). The algorithm models the relationship between the dependent variable and one or more independent variables (also known as features or predictors) using a logistic function or sigmoid function.

A confusion matrix is a table that is used to evaluate the performance of a classification model. It provides a detailed breakdown of the model's predictions and the actual classes of the data.

A confusion matrix is typically organized into four quadrants:

1. True Positive (TP): This quadrant represents the cases where the model predicted the positive class correctly. The actual class is positive, and the model correctly identified it as positive.
2. True Negative (TN): This quadrant represents the cases where the model predicted the negative class correctly. The actual class is negative, and the model correctly identified it as negative.
3. False Positive (FP): This quadrant represents the cases where the model predicted the positive class incorrectly. The actual class is negative, but the model wrongly identified it as positive. This is also known as a Type I error.
4. False Negative (FN): This quadrant represents the cases where the model predicted the negative class incorrectly. The actual class is positive, but the model wrongly identified it as negative. This is also known as a Type II error.

A confusion matrix allows you to assess the performance of a classification model by calculating various metrics such as accuracy, precision, recall, and F1 score.

Predicted Negative Predicted Positive

Actual Negative TN FP

Actual Positive FN TP

Using the values in the confusion matrix, you can calculate several evaluation metrics:

* Accuracy: (TP + TN) / (TP + TN + FP + FN)
* Precision: TP / (TP + FP)
* Recall (also known as Sensitivity or True Positive Rate): TP / (TP + FN)
* Specificity (True Negative Rate): TN / (TN + FP)
* F1 Score: 2 \* (Precision \* Recall) / (Precision + Recall)

The confusion matrix provides a comprehensive view of the performance of a classification model, highlighting both correct and incorrect predictions for each class. It helps in understanding the model's strengths and weaknesses and can guide further improvements or adjustments to the model.

In the Titanic dataset, the "Embarked" column indicates the port of embarkation for each passenger. It represents the location where the passenger boarded the Titanic. The "Embarked" column can have three different values:

* "C" stands for Cherbourg
* "Q" stands for Queenstown
* "S" stands for Southampton

These values indicate the three ports from which passengers boarded the Titanic.

The "Embarked" column is a categorical feature and can be used as an input for analysis or modeling purposes. It may be relevant in understanding potential correlations or patterns related to the passengers' port of embarkation and their survival status or other characteristics in the dataset.

**sns.countplot** is a function provided by the seaborn library in Python. It is used to plot the count of observations in each category of a categorical variable. This plot is helpful for visualizing the distribution of categorical data and comparing the frequencies of different categories.

Here's an example of how to use **sns.countplot** to plot the count of observations in the "Embarked" column of the Titanic dataset:

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load the Titanic dataset from a CSV file

data = pd.read\_csv('titanic.csv')

# Plot the count of observations in the "Embarked" column

sns.countplot(x='Embarked', data=data)

# Set plot labels and title

plt.xlabel('Port of Embarkation')

plt.ylabel('Count')

plt.title('Count of Passengers by Port of Embarkation')

# Display the plot

plt.show()

libraries: pandas, seaborn, and matplotlib.pyplot. We load the Titanic dataset into a pandas DataFrame named **data**.

We then use **sns.countplot** to create the count plot. We pass the column name "Embarked" to the **x** parameter and the DataFrame **data** to the **data** parameter.

After creating the plot, we set the labels for the x-axis and y-axis using **plt.xlabel** and **plt.ylabel**, respectively. We also set a title for the plot using **plt.title**.

Finally, we display the plot using **plt.show()**.

Make sure to adjust the code based on the column name and dataset you are working with.

**catplot** is the appropriate function to use for creating categorical plots in seaborn, including bar plots.

Stemming and lemmatization are both techniques used in natural language processing (NLP) for reducing words to their base or root form. However, they differ in their approaches and the results they produce.

Stemming:

* Stemming is a process of reducing words to their base or root form by removing affixes or suffixes.
* It applies simple and heuristic rules to cut off the end of words, often resulting in a word stem that may not be a valid word itself.
* The purpose of stemming is to reduce words to a common form so that variations of the same word can be treated as the same token.
* Stemming is usually faster than lemmatization and can be implemented with simple rule-based algorithms.
* Example: The word "running" is stemmed to "run", "walked" is stemmed to "walk".

Lemmatization:

* Lemmatization, on the other hand, involves determining the base or dictionary form of a word, known as the lemma.
* It uses more advanced linguistic rules and morphological analysis to convert words to their canonical form.
* The lemma is a valid word that represents the dictionary form of a word, maintaining the correct meaning.
* Lemmatization takes into account the part of speech (POS) of a word and performs more accurate transformations.
* Example: The word "running" is lemmatized to "run", "walked" is lemmatized to "walk".

In summary, the main differences between stemming and lemmatization are:

* Stemming aims to reduce words to a base form by removing suffixes, while lemmatization transforms words to their canonical form or lemma.
* Stemming can result in word stems that may not be actual words, while lemmatization produces valid words.
* Stemming is usually faster and more computationally efficient, while lemmatization is more accurate and linguistically informed.

The choice between stemming and lemmatization depends on the specific task and the desired trade-off between speed and accuracy. Stemming is often preferred for applications where speed is crucial and a loose approximation of the base form is sufficient. Lemmatization is typically chosen when the accuracy of word transformation and maintaining valid word forms is important, such as in language understanding or text generation tasks.

Term Frequency (TF): TF measures the frequency of a term within a document. It is calculated by dividing the number of times a term appears in a document by the total number of terms in the document. TF assigns higher weights to terms that occur more frequently in a document.

Inverse Document Frequency (IDF): IDF measures the rarity or importance of a term across a collection of documents. It is calculated by taking the logarithm of the total number of documents divided by the number of documents containing the term. IDF assigns higher weights to terms that occur less frequently in the entire document collection, indicating their significance.

The TF-IDF score is obtained by multiplying the TF and IDF values together. The higher the TF-IDF score for a term in a document, the more important or relevant that term is to that document.

pip install nltk

nltk.download('punkt')

from nltk.corpus import stopwords

nltk.download('stopwords')

from nltk import pos\_tag

nltk.download('averaged\_perceptron\_tagger')

from nltk import PorterStemmer

from nltk import WordNetLemmatizer

nltk.download('wordnet')

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorize=TfidfVectorizer()

tf\_idf\_vectors=vectorize.fit\_transform(tokenzied\_text)

tf\_idf\_matrix=tf\_idf\_vectors.toarray()

print(tf\_idf\_matrix)

from sklearn.preprocessing import LabelEncoder

from sklearn.naive\_bayes import GaussianNB