College of Engineering & Applied Sciences

CSPB 3022

Introduction To Data Science With Probability And Statistics

Exam Notes

Taylor Larrechea

2024

Exam 1 Notes

Data Frames

A data frame in Pandas is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns). It's akin to a spreadsheet or SQL table and is one of the most commonly used Pandas data structures.

You can create a data frame from various sources, such as:

- Lists
- Dictionaries
- NumPy arrays
- CSV files
- SQL databases

The structure of data frames can be summed up by:

- **Rows**: Each row represents a single observation or record.
- **Columns**: Each column represents a variable or feature. Columns can be of different data types (integer, string, float, etc.).
- **Index**: This is the 'key' for rows, similar to an index in a database. It's an immutable array, allowing fast access to data.

Some operations that can be used with data frames are:

- **Data Manipulation**: Adding, deleting, and modifying both rows and columns.
- **Filtering**: Selecting a subset of rows or columns based on some criteria.
- **Sorting and Grouping**: Organizing data based on values in certain columns.
- **Merging and Joining**: Combining multiple data frames.
- **Handling Missing Data**: Identifying and imputing missing values.

Data Frame Operation Examples

Here are some examples of data frame manipulation in pandas:

Creation

```python
import pandas as pd

# Creating a data frame from a dictionary
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'Los Angeles', 'Chicago']}
df = pd.DataFrame(data)

```

Adding A Column

```python
# Adding a new column
df['Salary'] = [70000, 80000, 90000]

```

Deleting A Column

```python
# Deleting a column
df.drop('Age', axis=1, inplace=True)

```

Filtering Data

```python
# Filtering rows where Salary is greater than 75000
high_earners = df[df['Salary'] > 75000]

```

Sorting Data

```python
# Sorting data by Salary in descending order
df_sorted = df.sort_values(by='Salary', ascending=False)

```

Merging Data Frames

```python
# Creating another data frame
additional_data = pd.DataFrame({'Name': ['Alice', 'Bob'], 'Experience': [5, 10]})

# Merging data frames
merged_df = pd.merge(df, additional_data, on='Name', how='left')

```

Handling Missing Data

```python
# Filling missing values with zero
df_filled = df.fillna(0)

```

Reading Data

```python
# Reading data from a CSV file
df_from_csv = pd.read_csv('data.csv')

# Writing data to a CSV file
df.to_csv('output.csv', index=False)

```

Combinatorics

Combinatorics is a branch of mathematics dealing with the study of countable, discrete structures and their properties. It's particularly important in computer science, where understanding how to count and arrange objects is crucial for algorithm design, data structure optimization, and problem-solving. Here's a summary of the key concepts in combinatorics:

- **Counting Principles**
  - **The Rule of Sum**: If there are A ways to do something and B ways to do another thing, and these two things cannot happen at the same time, then there are $A + B$ ways to choose one of these actions.
  - **The Rule of Product**: If there are A ways to do something and B ways to do another thing after that, then there are $A \cdot B$ ways to perform both actions.

- **Permutations**
  - Permutations are the arrangements of objects in a specific order.
  - The number of permutations of n distinct objects is $n!$ (n factorial), which is the product of all positive integers up to n.
  - For arranging r objects out of n available objects, the formula is

$$\boxed{{}^{n}P_r = \frac{n!}{(n-r)!}.}$$

- **Combinations**
  - Combinations refer to the selection of objects without regard to the order.
  - The number of ways to choose r objects from n different objects is given by

$$\boxed{\binom{n}{r} = \frac{n!}{r!(n-r)!}.}$$

  - Combinations are used when the order doesn't matter.

- **Binomial Theorem**
  - It provides a formula for the expansion of powers of a binomial (sum of two terms).
  - The Binomial Theorem states that:

$$\boxed{(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} \cdot a^{n-k} \cdot b^k}$$

    * This means the expansion is a sum of terms, where the exponents of a start at n and decrease to 0, while the exponents of b start at 0 and increase to n. The coefficients of each term are the corresponding binomial coefficients.
  - The coefficients of the terms in the expansion are the binomial coefficients, which can be calculated using combinations.

- **Binomial Distribution**
  - A binomial distribution is a discrete probability distribution that models the number of successes in a fixed number of independent Bernoulli trials. A Bernoulli trial is an experiment with exactly two possible outcomes, typically termed "success" and "failure".
  - In the context of the binomial distribution, $P(x = k)$ denotes the probability of getting exactly k successes in n trials. The formula for this is

$$\boxed{P(x = k) = \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k}.}$$

    * $\binom{n}{k}$ (read as "n choose k") is the binomial coefficient, representing the number of ways to choose k successes from n trials.
    * p is the probability of success on an individual trial.
    * $1-p$ is the probability of failure (since the trials are binary, the sum of the probabilities of success and failure is 1).
    * p^k is the probability of having k successes.
    * $(1-p)^{n-k}$ is the probability of having $n-k$ failures.

groupby

The `groupby` method is used to split data into groups based on some criteria, apply a function to each group independently, and then combine the results into a data structure. The process is often summarized as split-apply-combine.

The way `groupby` works is by the following:

- **Split**: The data is divided into groups based on one or more keys. This is done by mapping a function over the index or columns of the DataFrame.
- **Apply**: A function is applied to each group independently. This function could be an aggregation (computing a summary statistic), transformation (standardizing data within a group), or filtration (removing data based on group properties).
- **Combine**: The results of the function application are combined into a new data structure.

The basic syntax for how `groupby` works is as follows:

```
df.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=NoDefault.
no_default, observed=False, dropna=True)

```

- **by**: Specifies the grouping criteria. Can be a function, column name, or list of column names.
- **axis**: Determines whether to group by rows (0) or columns (1).
- **level**: If the axis is a MultiIndex (hierarchical), groups by a particular level or levels.
- **as_index**: For aggregated output, returns object with group labels as the index. Setting it to False will return group labels in the columns.
- **sort**: Sorts group keys. By default, it's set to True.

groupby Example

Here is a simple example of utilizing `groupby` in Pandas:

```python
import pandas as pd

# Example DataFrame
data = {
    'Date': ['2023-01-01', '2023-01-01', '2023-01-02', '2023-01-02', '2023-01-03'],
    'Product': ['A', 'B', 'A', 'A', 'B'],
    'Sales': [100, 200, 150, 100, 250]
}
df = pd.DataFrame(data)

# Group by 'Product' and sum up the sales
grouped_df = df.groupby('Product')['Sales'].sum()

print(grouped_df)

```

agg

The `agg` method in Pandas, when used with groupby, is a versatile tool for performing multiple aggregation operations on your grouped data. It allows for more flexibility than just applying a single aggregate function like `sum` or `mean` directly. With `agg`, you can apply different aggregation functions to your data simultaneously, and even specify custom functions to suit your analysis needs.

After grouping your DataFrame with the `groupby` method, you can use `agg` to specify one or more aggregation operations to apply to the grouped data. `agg` can take a variety of inputs:

- A single aggregation function as a string (e.g. `'sum'`, `'mean'`).

- A list of functions (e.g., `['sum', 'mean', 'max']`), applying each function to each column of each group.
- A dictionary where keys are column names and values are functions or lists of functions, allowing different aggregation for different columns.

agg Example

Expanding on the previous example that was used with `groupby`, the example below encapsulates how to use the `agg` function with `groupby`:

```python
import pandas as pd

# Example DataFrame
data = {
    'Date': ['2023-01-01', '2023-01-01', '2023-01-02', '2023-01-02', '2023-01-03'],
    'Product': ['A', 'B', 'A', 'A', 'B'],
    'Sales': [100, 200, 150, 100, 250]
}
df = pd.DataFrame(data)

# Group by 'Product' and apply multiple aggregation functions to 'Sales'
grouped_df = df.groupby('Product')['Sales'].agg(['sum', 'mean', 'max'])

print(grouped_df)

```

This would output the total, average, and maximum sales for each product.

More advanced usage can be applied to the `agg` function, below is an example with the previous data frame in the last example of this advanced usage:

```python
grouped_df = df.groupby('Product').agg({
    'Sales': ['sum', 'mean'],
    'Date': ['min', 'max']
})

```

This performs a sum and mean aggregation on the `Sales` and finds the minimum and maximum `Date` for each `Product`.

Joining DataFrames

Joining data frames is a fundamental operation in data analysis, allowing you to combine data from different sources based on common identifiers. Pandas offers several methods for joining DataFrames, akin to SQL joins, including `merge`, `join`, and `concat`. Understanding these methods and when to use each is key to effective data manipulation.

1. **merge**: The `merge` function is the most versatile method for joining two DataFrames. It allows you to perform inner, outer, left, and right joins by specifying how you want the DataFrames to be merged.

   - **Syntax**: `pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False)`
   - **Parameters**:
     - `left, right`: The DataFrames you want to join.
     - `how`: Specifies the type of join (`'left', 'right', 'outer', 'inner'`).
     - `on`: The column(s) to join on. Must be found in both DataFrames.
     - `left_on, right_on`: Columns from the left and right DataFrames to use as keys if they have different names.
     - `left_index, right_index`: If True, use the index (row labels) from the left or right DataFrame as its join key(s).

merge Example

Below is a simple example of how `merge` works by joining DataFrames:

```python
import pandas as pd

# Example DataFrames
df1 = pd.DataFrame({'key': ['A', 'B', 'C', 'D'], 'value': range(4)})
df2 = pd.DataFrame({'key': ['B', 'D', 'D', 'E'], 'value': range(4, 8)})

# Inner join on 'key'
inner_joined = pd.merge(df1, df2, on='key', how='inner')

# Outer join on 'key'
outer_joined = pd.merge(df1, df2, on='key', how='outer')

```

2. **join**: The `join` method is a convenient method for combining DataFrames based on their indexes or on a key column. It's a simpler interface than `merge` for index-based joining.

   - **Syntax**: `DataFrame.join(other, on=None, how='left', lsuffix='', rsuffix='')`
   - **Parameters**:
     - `other`: The DataFrame to join with.
     - `on`: The column or index level names to join on in the `other` DataFrame. Must be found in both the calling DataFrame and `other`.
     - `how`: Type of join (`'left'`, `'right'`, `'outer'`, `'inner'`).
     - `lsuffix, rsuffix`: Suffixes to apply to overlapping column names in the DataFrames.

join Example

Below is a simple example of how `join` works by joining DataFrames:

```python
# Assuming df1 and df2 from the previous example
# Join df2 to df1 using the index of df1 and the 'key' column of df2
joined = df1.join(df2.set_index('key'), on='key', how='left', lsuffix='_df1', rsuffix='_df2')

```

3. **concat**: The `concat` function is used for concatenating DataFrames along a particular axis (row-wise or column-wise). It's useful for stacking DataFrames vertically or horizontally.

   - **Syntax**: `pd.concat(objs, axis=0, join='outer')`
   - **Parameters**:
     - `objs`: A sequence of DataFrames to concatenate.
     - `axis`: The axis to concatenate along (0 for rows, 1 for columns).
     - `join`: How to handle indexes on other axis (`'outer'`, `'inner'`).

concat Example

Below is a simple example of how `concat` works by joining DataFrames:

```python
# Vertical concatenation
vertical_concat = pd.concat([df1, df2])

# Horizontal concatenation, assuming same indexes
horizontal_concat = pd.concat([df1, df2], axis=1)

```

Measures Of Central Tendency

There are core ways we can measure information about data, below are some pertinent ways that we measure central tendency in data science:

1. **Mean** - The mean, often referred to as the average, is calculated by summing all the values in the data set and then dividing by the number of values. It's a measure that is sensitive to outliers, meaning that a single very high or very low value can significantly affect the mean.

   The formula for calculating the mean of a data set is:

   $$\boxed{\text{Mean} = \frac{\sum_{i=1}^{n} x_i}{n}}$$

   where x_i represents each value in the dat set and n is the total number of values.

2. **Median** - The median is the middle value in a data set when the values are arranged in ascending or descending order. If there's an even number of observations, the median is the average of the two middle numbers. Unlike the mean, the median is not affected by outliers, making it a better measure of central tendency for skewed distributions.

   - **Finding The Median**:
     - Arrange the data in ascending order.
     - If the number of observations (n) is odd, the median is the value at position $\frac{n+1}{2}$.
     - If n is even, the median is the average of the values at positions $\frac{n}{2}$ and $\frac{n}{2}+1$.

3. **Mode** - The mode is the value that appears most frequently in a data set. A data set may have one mode (unimodal), more than one mode (bimodal or multimodal), or no mode at all if all values occur with the same frequency. The mode is particularly useful for categorical data where we want to identify the most common category.

   - **Key Characteristics**:
     - The mode is the only measure of central tendency that can be used with nominal data (data categorized without a natural order).
     - A data set may have multiple modes, making it a good measure for understanding variability in data.

Central Tendency Example

Consider the simple data set: `2,3,3,5,7,10`

Mean

$$\text{Mean} = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{2+3+3+5+7+10}{6} = \frac{30}{6} = 5.$$

Median

The data set is even, so the median is the average of the middle values (3rd and 4th values):

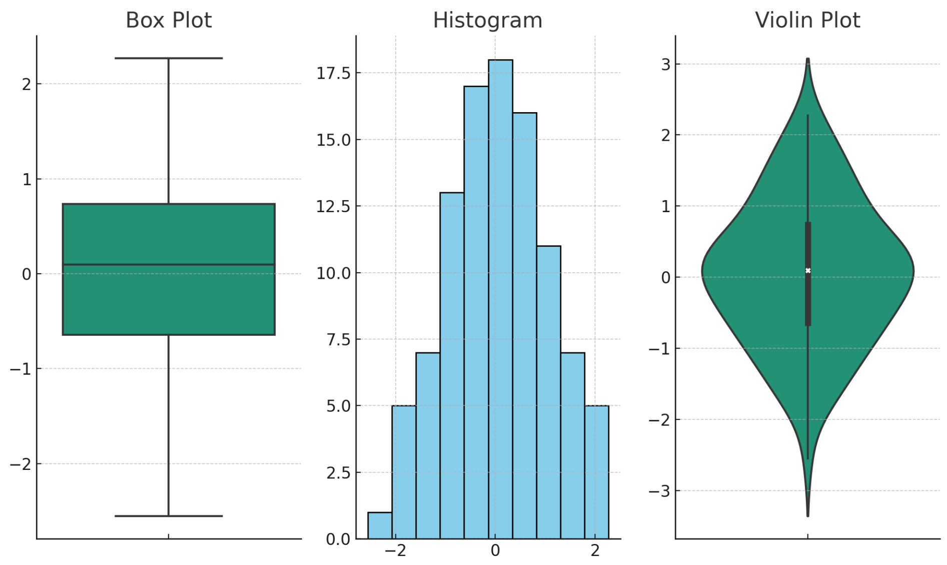
$$\text{Median} = \frac{3+5}{2} = \frac{8}{2} = 4.$$

Mode

The value `3` appears most frequently, so the mode of this data set is `3`.

Plot Interpretation

Plots are a vital concept in data science. Interpreting them is a vital skill and important for understanding data and how to draw conclusions. For this, we have created three types of plots that are common in data science. These plots can be seen below.



1. **Box Plot**: A box plot (or box-and-whisker plot) provides a visual summary of the key aspects of a distribution:
    - **Central line**: Represents the median of the data. In our box plot, the median is around 0, indicating the central tendency of the distribution.
    - **Box**: The edges of the box (the interquartile range, IQR) show the 25th percentile (Q1) and the 75th percentile (Q3) of the data. The length of the box indicates the spread of the central 50% of the data. A shorter box suggests less variability in the middle half of the data.
    - **Whiskers**: Extend from the box to show the range of the data. Typically, they extend to 1.5 * IQR beyond the quartiles, though this can vary. Points beyond the whiskers are considered outliers and are often plotted as individual points.
    - **Outliers**: Points outside the whiskers are plotted individually, indicating that they are unusual compared to the rest of the data. In our plot, there are a few outliers on both sides, indicating data points significantly higher or lower than the majority.
2. **Histogram**: A histogram displays the distribution of data by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin:
    - **Bins**: The x-axis represents bins or intervals of values. In our histogram, data is divided into 10 bins.
    - **Height of bars**: The y-axis represents the frequency of data points within each bin. The height of a bar shows how many data points fall into that range.
    - **Distribution shape**: The histogram helps identify the distribution shape of the data, whether it's normal, skewed, or bimodal, for example. Our histogram suggests a roughly normal distribution, as the bars form a bell-shaped curve around the center.
3. **Violin Plot**: A violin plot combines elements of box plots and density plots, providing a deeper understanding of the distribution:
    - **Outer shape**: Represents the kernel density estimation of the data, showing the distribution's density. Thicker sections of the violin plot indicate a higher density of data points, while thinner sections indicate lower density.
    - **Inner box plot**: Often included within the violin, showing the median (central dot) and the interquartile range (thick bar). The violin plot in our example shows a median around 0, similar to the box plot, with the bulk of data points concentrated around the center, indicating the central tendency and variability.

- **Width**: The width of the violin at different values indicates the density of the data at those values. In our plot, the widest part is around the median, suggesting the highest density of data points is near the center of the distribution.

IQR

The Interquartile Range (IQR) is a measure of statistical dispersion, or variability, that indicates the spread of the middle 50% of a dataset. It's calculated as the difference between the 75th percentile (Q3) and the 25th percentile (Q1) of the data. The IQR is used to build box plots, as we've seen, and is helpful for identifying outliers.

In essence, the IQR can be calculated with

$$\boxed{\text{IQR} = Q3 - Q1.}$$

where

- $Q1$ (the first quartile) is the median of the first half of the dataset (the 25th percentile).
- $Q3$ (the third quartile) is the median of the second half of the dataset (the 75th percentile).

IQR Strategy

The steps to calculate the IQR are as follows:

1. **Order the Data**: Arrange the data points in ascending order.
2. **Find the Median (Q2)**: This divides the dataset into two halves.
3. **Find Q1 and Q3**:
   - $Q1$ is the median of the data points to the left of the median of the dataset.
   - $Q3$ is the median of the data points to the right of the median.
4. **Calculate the IQR**: Subtract $Q1$ from $Q3$.

IQR is important because:

- **Resistant to Outliers**: Unlike range and standard deviation, the IQR focuses on the middle bulk of the data and is not affected by outliers. This makes it a more robust measure of spread for skewed distributions.
- **Identifying Outliers**: The IQR is used to define outliers. A common rule is that an outlier is any data point that lies more than 1.5 times the IQR above the third quartile (Q3) or below the first quartile (Q1).

IQR Example

Consider the sample data set `3,7,5,8,6,9`:

1. **Ordered Data Set**: `3,5,6,7,8,9`
2. **Median**: The median of this data set is $\frac{6+7}{2} = 6.5$
3. **Find Q1 and Q3**:
   - $Q1$ is the median of the lower 50%, `(3,5,6)`, and is 5
   - $Q3$ is the median of the upper 50%, `(7,8,9)`, and is 8
4. **Calculate IQR** $IQR = Q3 - Q1 = 8 - 5 = 3$

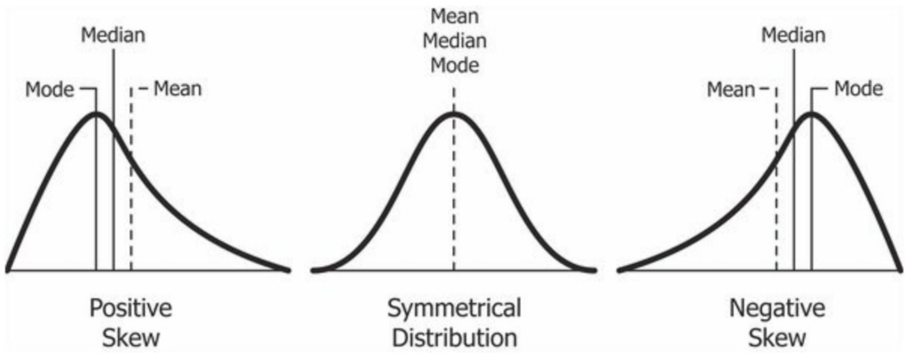
Skew

Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable. In simpler terms, it's a way to describe the direction and extent to which a distribution deviates from a normal distribution, where the mean, median, and mode are all the same.

The different types of skews are:

- **Positive Skew (Right-Skewed)**: The tail on the right side of the distribution is longer or fatter than the left side. In a positively skewed distribution, the mean is typically greater than the median, which is greater than the mode. This type of skew indicates that there are a number of exceptionally high values pulling the mean to the right.
- **Negative Skew (Left-Skewed)**: The tail on the left side of the distribution is longer or fatter than the right side. In a negatively skewed distribution, the mean is typically less than the median, which is less than the mode. This skew indicates that there are a number of exceptionally low values pulling the mean to the left.

When interpreting the central tendency from a skewed data set, the mean is located on the side of the skew. We can see the different central tendencies of each type of skew below.


Median
Mode
Mean
Positive Skew
Mean
Median
Mode
Symmetrical Distribution
Median
Mean
Mode
Negative Skew