Statistics in Data Science

overview

1. **Introduction to Statistics in Data Science**
2. **Distributions**
3. **Point Estimates**
4. **Distribution Estimators: MoM, MLE, KDE**
5. **Statistical Hypothesis Testing**
6. **Correlation**
7. **Practical Examples**

Point Estimates

- Suppose you are a teacher with 1,000 students and you want to estimate the average test score for a mid-term exam.
  - You randomly select 50 students and find that their average score is 85.
  - The point estimate for the average test score of all 1,000 students is 85.
- Imagine you are a real estate agent trying to estimate the average price of houses in a particular region.
  - You randomly sample 100 houses and calculate an average selling price of $250,000.
  - The point estimate for the average selling price of houses in that region is $250,000.
- In a country, there's a big election coming up and a polling agency wants to estimate the proportion of voters who support Candidate A.
  - They survey 1,200 people, and 720 say they support Candidate A.
  - The point estimate for the proportion of voters who support Candidate A in the entire population is 720120012007200 or 0.6 (or 60%).

Point Estimates

- Point estimation involves using a single value (point estimate) to approximate a population parameter. The most common point estimate is the sample mean, which is used to estimate the population mean.

Point Estimates

The sample mean is usually not exactly the same as the population mean.

- This difference can be caused by many factors including poor survey design, biased sampling methods and the randomness inherent to drawing a sample from a population.

- That's why it's often useful to calculate confidence intervals along with point estimates to understand the range in which the true parameter likely falls.

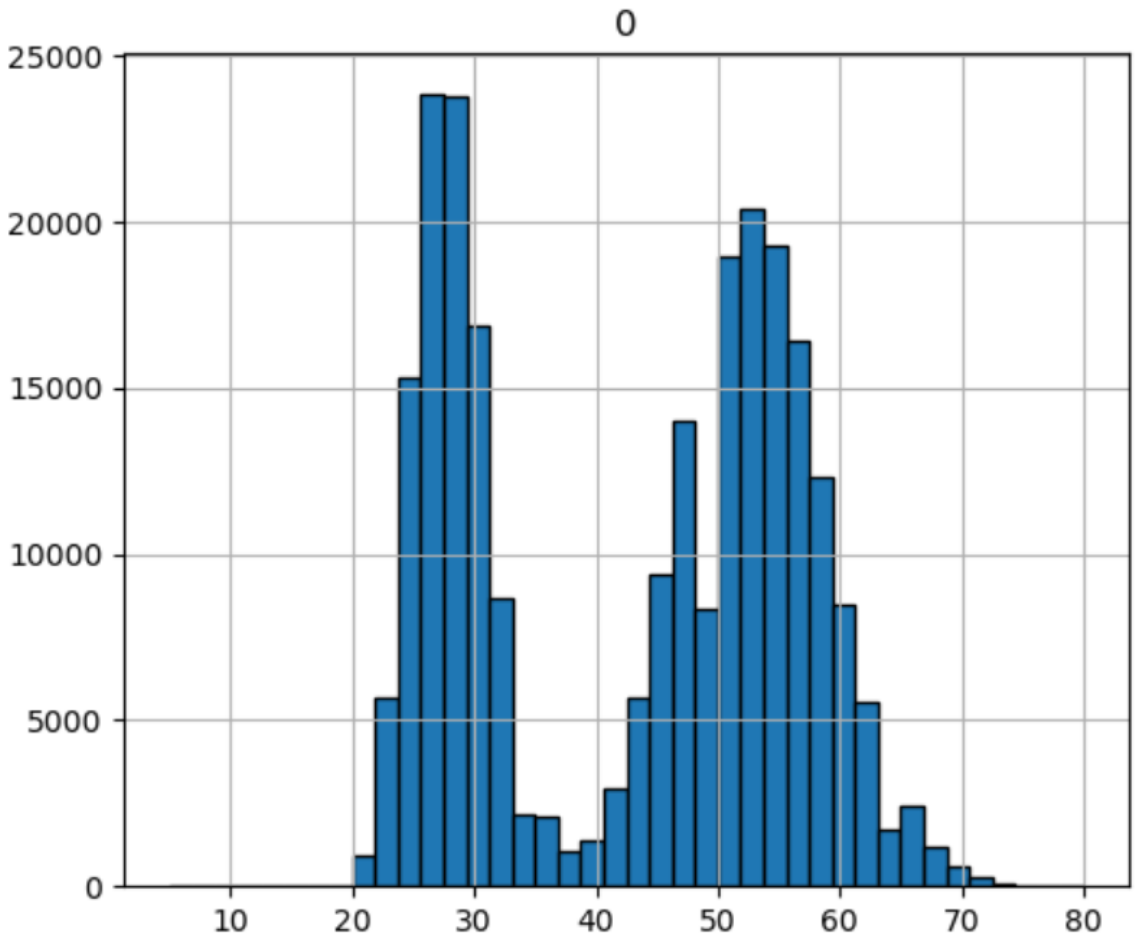
```
np.random.seed(10)
population_ages1 = stats.poisson.rvs(loc=18, mu=35, size=150000)
population_ages2 = stats.poisson.rvs(loc=18, mu=10, size=100000)
population_ages = np.concatenate((population_ages1, population_ages2))
```

population_ages1 = stats.poisson.rvs(loc=18, mu=35, size=150000):
This line generates a sample of 150,000 ages using the Poisson distribution with a mean (**mu**) of 35. The **loc=18** parameter shifts the distribution by 18 units, so the ages produced will be 18 + whatever the Poisson distribution generates. As a result, the ages are somewhat centered around 53 (18 + 35).

```
pd.DataFrame(population_ages).hist(range=(5,80), bins=40, ec='black', figsize=(6,5))
```

population_ages2 = stats.poisson.rvs(loc=18, mu=10, size=100000):
Similarly, this line generates a sample of 100,000 ages using the Poisson distribution with a mean of 10, and shifted by 18 units. The ages from this line are somewhat centered around 28 (18 + 10).

```
array([[<AxesSubplot:title={'center':'0'}>]], dtype=object)
```



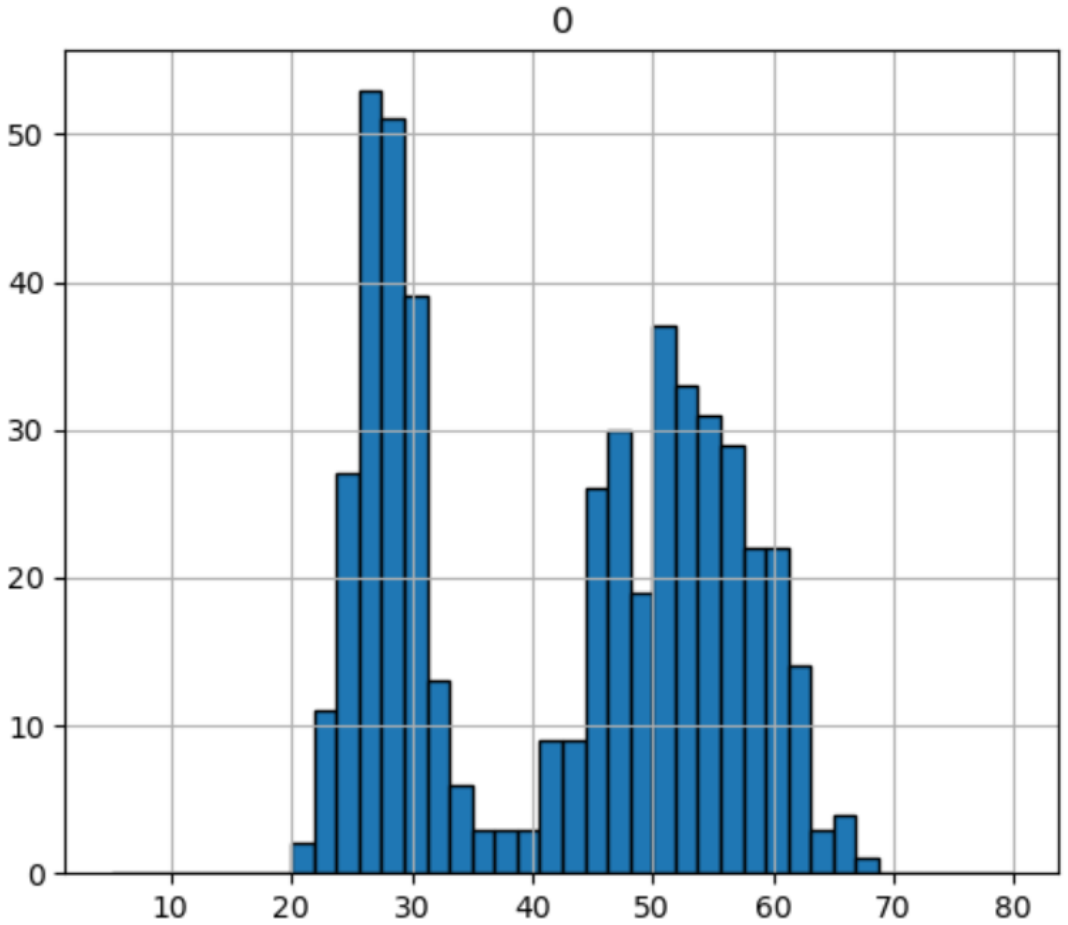

```
population_ages.mean()
```

```
43.002372
```

- Sample population:

```
np.random.seed(6)
sample_ages = np.random.choice(a= population_ages,
                               size=500)            # Sample 1000 values

pd.DataFrame(sample_ages).hist(range=(5,80), bins=40, ec='black', figsize=(6,5))
```

```
array([[<AxesSubplot:title={'center':'0'}>]], dtype=object)
```




```
print ( sample_ages.mean() )                    # Show sample mean
```

```
42.388
```

Difference between the actual and sample:

```
population_ages.mean() - sample_ages.mean()   # Check difference between means
```

```
0.614372000000003
```

we can get a fairly accurate estimate of a large population by sampling a relatively small subset of individuals.

- **Sampling Distributions and The Central Limit Theorem**
  - Many statistical procedures assume that data follows a **normal distribution.**
  - Unfortunately, **real world data is often not normally distributed**, and the distribution of a sample tends to mirror the distribution of the population.

- The **central limit theorem - CLT** is one of the most important results of probability theory and serves as the foundation of many methods of statistical analysis.

- At a high level the theorem states, **the distribution of many sample means, known as a sampling distribution, will be normally distributed.**

- https://www.youtube.com/watch?v=4YLtvNeRIrg

Confidence Intervals

- **Interval Estimation (Confidence Intervals):**
- Interval estimation provides a range (confidence interval) within which we believe the population parameter is likely to fall. Confidence intervals are constructed around point estimates to quantify the uncertainty associated with estimation.
- For example, a 95% confidence interval for the population mean provides a range of values within which we are 95% confident that the true population mean lies.

Confidence Intervals

- We can calculate a confidence interval by taking a point estimate and then adding and subtracting a margin of error to create a range.

$$Confidence\ Interval\ (CI) = \bar{x}(sample\ mean) \pm margin\ of\ error$$

  - Margin of error is based on your desired confidence level, the spread of the data and the size of your sample.

  - The way you calculate the margin of error depends on whether you know the standard deviation of the population or not.

Confidence Intervals

If you know the standard deviation of the population, the margin of error is equal to:

$$z * \frac{\sigma}{\sqrt{n}}$$

Where

- σ (sigma) is the population standard deviation,
- n is sample size, and
- z is a number known as the z-critical value.

- The z-critical value is the number of standard deviations you'd have to go from the mean of the normal distribution to capture the proportion of the data associated with the desired confidence level.
- For instance, we know that roughly 95% of the data in a normal distribution lies within 2 standard deviations of the mean, so we could use 2 as the z-critical value for a 95% confidence interval.

Let's calculate a 95% confidence for our mean point estimate:

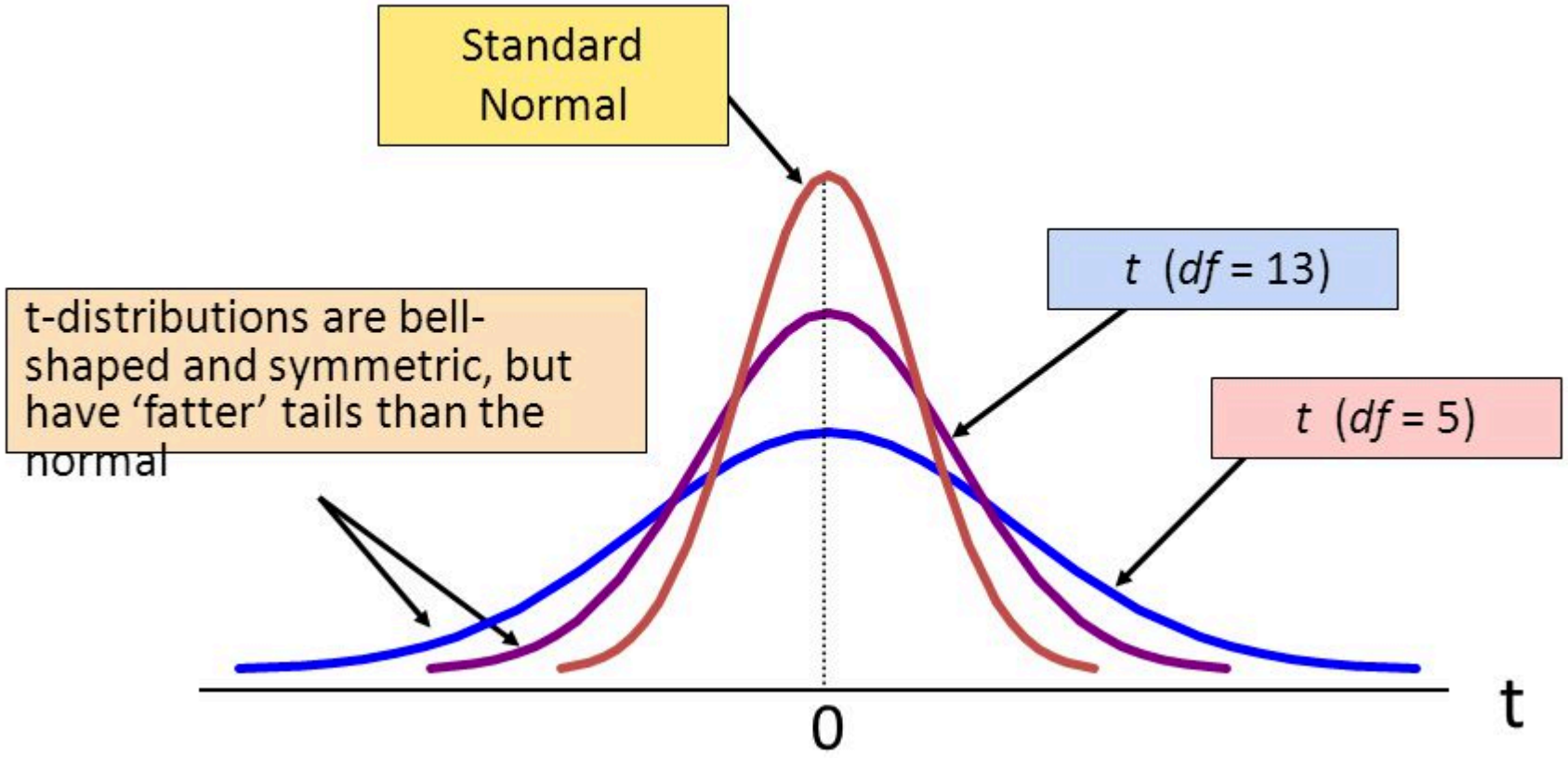
```
np.random.seed(10)

sample_size = 1000
sample = np.random.choice(a= population_ages, size = sample_size)
sample_mean = sample.mean()

z_critical = stats.norm.ppf(q = 0.975)  # Get the z-critical value*

print("z-critical value:")              # Check the z-critical value
print(z_critical)

pop_stdev = population_ages.std()  # Get the population standard deviation

margin_of_error = z_critical * (pop_stdev/math.sqrt(sample_size))

confidence_interval = (sample_mean - margin_of_error,
                       sample_mean + margin_of_error)

print("Sample Mean:")
print(sample_mean)

print("Confidence interval:")
print(confidence_interval)
```

```
z-critical value:
1.959963984540054
Sample Mean:
42.523
Confidence interval:
(41.70306406882683, 43.34293593117317)
```

We use stats.norm.ppf(q = 0.975) to get the desired z-critical value instead of q = 0.95 because the distribution has two tails.

Confidence Intervals

- If you don't know the standard deviation of the population, *you have to use the standard deviation of your sample* as a stand in when creating confidence intervals.
  - Since the sample standard deviation may not match the population parameter the interval will have more error when you don't know the population standard deviation.
  - To account for this error, we use what's known as a **t-critical value instead of the z-critical value**.
  - The **t-critical value** is drawn from what's known as a t-distribution



- The t-distribution is available in scipy.stats with the nickname "t" so we can get t-critical values with stats.t.ppf().

```
np.random.seed(10)

sample_size = 25
sample = np.random.choice(a= population_ages, size = sample_size)
sample_mean = sample.mean()

t_critical = stats.t.ppf(q = 0.975, df=24)  # Get the t-critical value*

print("t-critical value:")                  # Check the t-critical value
print(t_critical)

sample_stdev = sample.std()    # Get the sample standard deviation

sigma = sample_stdev/math.sqrt(sample_size)  # Standard deviation estimate
margin_of_error = t_critical * sigma

confidence_interval = (sample_mean - margin_of_error,
                       sample_mean + margin_of_error)

print("Confidence interval:")
print(confidence_interval)
```

```
t-critical value:
2.0638985616280205
Confidence interval:
(37.75711273701061, 48.0028872629894)
```

Note: when using the t-distribution, you have to supply the degrees of freedom (df)

For this type of test,
the df = sample size - 1.

If you have a large sample size, the t-distribution approaches the normal distribution.

Confidence Intervals

We can also make a confidence interval for a point estimate of a population proportion. In this case, the margin of error equals:

$$z * \sqrt{\frac{p(1-p)}{n}}$$

Where,

- z is the z-critical value for our confidence level,
- p is the point estimate of the population proportion and
- n is the sample size.

- Let's calculate a 95% confidence interval for Hispanics according to the sample proportion we calculated earlier (0.192):

```
z_critical = stats.norm.ppf(0.975)      # Record z-critical value

p = 0.192                               # Point estimate of proportion

n = 1000                                # Sample size

margin_of_error = z_critical * math.sqrt((p*(1-p))/n)

confidence_interval = (p - margin_of_error,  # Calculate the the interval
                       p + margin_of_error)

confidence_interval
```

```
(0.16758794241348748, 0.21641205758651252)
```

```
stats.norm.interval(alpha = 0.95,    # Confidence level
                    loc =  0.192,    # Point estimate of proportion
                    scale = math.sqrt((p*(1-p))/n))  # Scaling factor
```

```
(0.16758794241348748, 0.21641205758651252)
```

Covariance VS. Correlation

- Covariance and correlation describe how two random variables are related.
- Covariance measures the extent to which the relationship between two variables is linear.
- Correlation uses information about the variance of X and Y to normalize this metric.

Covariance

It is calculated as

$$cov_{x,y} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

Where,

- x = the independent variable
- y = the dependent variable
- N = number of data points in the sample
- $\bar{x}$ = the mean of the independent variable x
- $\bar{y}$ = the mean of the dependent variable y

When the two variables are identical, *covariance is same as variance.*

Covariance

- The sign of the covariance shows the trend in the linear relationship between the variables.
  - A positive sign indicates that the variables are directly related.

    i.e. when one increases the other one also increases.
  - A negative sign indicates that the variables are inversely related, so that when one increases the other decreases.

- Numpy.cov()

1. **For two datasets `X` and `Y`:**

```python
import numpy as np

# Sample data
X = np.random.rand(50)
Y = 2 * X + np.random.normal(0, 0.1, 50)

# Calculate covariance matrix
cov_matrix = np.cov(X, Y)

# Extract the covariance value between X and Y
cov_value = cov_matrix[0, 1]
print(cov_value)
```

The covariance matrix produced for `X` and `Y` is:

```lua
[[Var(X)   Cov(X,Y)]
 [Cov(X,Y) Var(Y)]]
```

For a single dataset (e.g., finding the variance of `X`):

The variance of a dataset is simply the covariance of the dataset with itself.

```python
# Calculate variance of X
var_X = np.cov(X)[0, 0]
print(var_X)
```

Correlation

•Correlation is a statistical measure that describes the extent to which two variables change together. If one variable tends to go up when the other goes up, there is a positive correlation. If one variable tends to go down when the other goes up, there is a negative correlation. If there's little or no predictable relationship in the changes of the two variables, they are said to have no or zero correlation.

•The value of correlation coefficient is always between -1 and 1.

•Once we've normalized the metric to the -1 to 1 scale, we can make meaningful statements and compare correlations.

Correlation

To normalize Covariance, consider

$$\rho_{corr} = \frac{Cov(x,y)}{\sqrt{Cov(x,x)}\sqrt{Cov(y,y)}}$$

$$= \frac{Cov(x,y)}{\sigma(x)\sigma(y)}$$

- where ρ is the correlation coefficient of two series x and y.

Just like covariance, a positive coefficient indicates that the variables are directly related and a negative coefficient indicates that the variables are inversely related.

Correlation

- In Python, using libraries like numpy or pandas, you can easily calculate correlation coefficients.
  - numpy.corrcoef()
  - With pandas, if you have a DataFrame, df.corr().

- The Pearson correlation measures only linear relationships. If there's a non-linear relationship, correlation might be close to zero even if there's a strong relationship between variables.

- Outliers can have a significant impact on the correlation coefficient.

It's always a good idea to visualize your data to ensure that a reported correlation is not driven by just a few outlier points.

- Point estimation
  - Sampling distribution & CLT

- Confidence Intervals
  - z-critical value & t-critical value
  - Covariance & correlation

In summary, estimation in statistics is the process of making educated guesses or predictions about population parameters using sample data.

It involves calculating point estimates, constructing confidence intervals, and assessing the quality and reliability of the estimates.

Estimation is a fundamental concept in statistical analysis, hypothesis testing, and decision-making in various fields, including science, business, and social sciences.

In-class exercise: statistical analysis with the **Iris dataset**

1.Brief Overview of the Iris Dataset:

```
from sklearn.datasets import load_iris
data = load_iris()
X = data['data']  # Features: Sepal length, sepal width, petal length, petal width
y = data['target']  # Target variable: Iris species
```

2. Select pairs of variables (e.g., sepal length and sepal width, or petal length and petal width) and analyze their relationships.

3. Calculate the covariance matrix for the selected variable pairs and discuss what the covariance values indicate. For example, does a positive covariance between sepal length and sepal width mean there's a relationship between the two?

4.Compute the correlation matrix to standardize the strength of the relationships. Please interpret whether the variables are strongly correlated (close to 1 or -1) or weakly correlated (close to 0).

5. Visualization: Create scatter plots for the selected variable pairs using matplotlib or seaborn. Make sure to include correlation coefficients on the scatter plots to aid interpretation.