

Statistical Foundations for Data Science

Prepared by Dr. Sumaya Alkubaisi

FNT633
Machine Learning in Finance

Lecture 4



Learning Goals

By the end of this lecture, you will be able to:

- Distinguish descriptive vs inferential statistics.
- Compute and interpret common summary statistics.
- Use hypothesis tests and confidence intervals.
- Visualize distributions, relationships, and outliers.
- Apply stats in finance using Python.

What Is Statistics?

- Statistics is the science of collecting, organizing, analyzing, and interpreting data to make informed decisions under uncertainty.

In finance:

It helps quantify risk, estimate returns, and validate whether observed market patterns are real or random.

Two main branches:

- Descriptive statistics: Summarize and describe data (mean, variance, charts).
- Inferential statistics: Make predictions or decisions about a larger population from a sample (confidence intervals, hypothesis tests).

Why statistics matter in finance

Quantify Risk and Return

Use mean, variance, and standard deviation to measure expected profit and volatility.
Helps investors compare assets and decide the best risk-return trade-off.
Tools: covariance, correlation, portfolio diversification.

Test Trading Hypotheses

Apply hypothesis testing (t-test, ANOVA) to check if a trading rule or strategy truly improves returns.
Distinguishes real patterns from random market noise.

Validate Models and Measure Uncertainty

Use confidence intervals, R^2 , and residual analysis to evaluate how reliable models are.
Quantifies uncertainty and ensures model predictions are statistically sound.

Descriptive vs Inferential Statistics

Descriptive Statistics:

- Used to summarize and describe the main features of a dataset.
- Focus on what the data shows directly.
- Includes measures like mean, median, mode, variance, standard deviation, and visual tools such as histograms and boxplots.

Example: “The average loan amount in our dataset is \$25,000.”

Inferential Statistics:

- Used to make conclusions or predictions about a larger population based on a sample.
- Involves probability theory and sampling to estimate or test hypotheses.
- Includes confidence intervals, t-tests, and p-values.

Example: “Based on a sample of loans, we estimate the average default rate in the entire bank portfolio.”

Python Demo: Example dataset — create and inspect

#All missing or invalid values were cleaned in Lectures 2 & 3.

```
import pandas as pd
```

```
data = {  
    "age": [25, 40, 30, 35, 28],  
    "salary": [50000, 60000, 55000, 52000, 45000],  
    "loan_amount": [20000, 25000, 22000, 18000, 30000],  
    "default": ["No", "Yes", "No", "No", "Yes"]  
}
```

```
df = pd.DataFrame(data)
```

```
df.info()
```

```
df.describe().round(2)
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 5 entries, 0 to 4  
Data columns (total 4 columns):  
#   Column      Non-Null Count  Dtype    
---  ---        
0    age         5 non-null     int64    
1    salary      5 non-null     int64    
2    loan_amount  5 non-null     int64    
3    default     5 non-null     object   
dtypes: int64(3), object(1)  
memory usage: 292.0+ bytes
```

	age	salary	loan_amount
count	5.00	5.00	5.00
mean	31.60	52400.00	23000.00
std	5.94	5594.64	4690.42
min	25.00	45000.00	18000.00
25%	28.00	50000.00	20000.00
50%	30.00	52000.00	22000.00
75%	35.00	55000.00	25000.00
max	40.00	60000.00	30000.00

Python Demo: Measures of central tendency (mean, median, mode)

tells us what a “normal” or “average” observation looks like (The expected return).

```
print("Mean salary:", df['salary'].mean())  
print("Median salary:", df['salary'].median())  
print("Mode default:", df['default'].mode().tolist())
```

```
➞ Mean salary: 52400.0  
   Median salary: 52000.0  
   Mode default: ['No']
```

`.mode()` in pandas can return more than one value if several values occur with the same highest frequency.

`.mode().tolist()` simply converts that Series into a normal Python list so it's easier to read or print.

Python Demo: Measures of spread (variance, std, range, IQR)

#How far apart the data points are from the middle or from each other

```
print("Variance:", df['salary'].var())
print("Std dev:", df['salary'].std())
print("Range:", df['salary'].max() - df['salary'].min())
Q1, Q3 = df['salary'].quantile([0.25, 0.75])
print("IQR:", Q3 - Q1)
```

```
➞ Variance: 31300000.0
   Std dev: 5594.640292279746
   Range: 15000
   IQR: 5000.0
```

Standard Deviation (≈ 5595): On average, salaries differ by about 5,595 from the mean.

A large standard deviation $\rightarrow$ salaries are widely spread/high variation (High risk / volatility).

A small standard deviation $\rightarrow$ salaries are close to the average/ low variation (Stable / low risk).

Range (15,000): Difference between the highest and lowest salary — gives total spread.

IQR (5,000): The middle 50% of salaries lie within a 5,000 range — helps ignore outliers and shows central spread.

Python Demo: Shape of distribution (skewness)

```
from scipy.stats import skew  
print("Skewness:", skew(df['salary']))
```

```
Skewness: 0.0597605174034246
```

- Skewness measures how symmetrical or asymmetrical a distribution is.
- Positive skew (> 0) → longer right tail → more high outliers (some customers have very large values, e.g., big loans or salaries).
- Negative skew (< 0) → longer left tail → more low outliers (many small loans or low returns).
- Zero skew → data are roughly symmetrical (balanced around the mean).

In finance:

Skewness helps identify direction of risk —

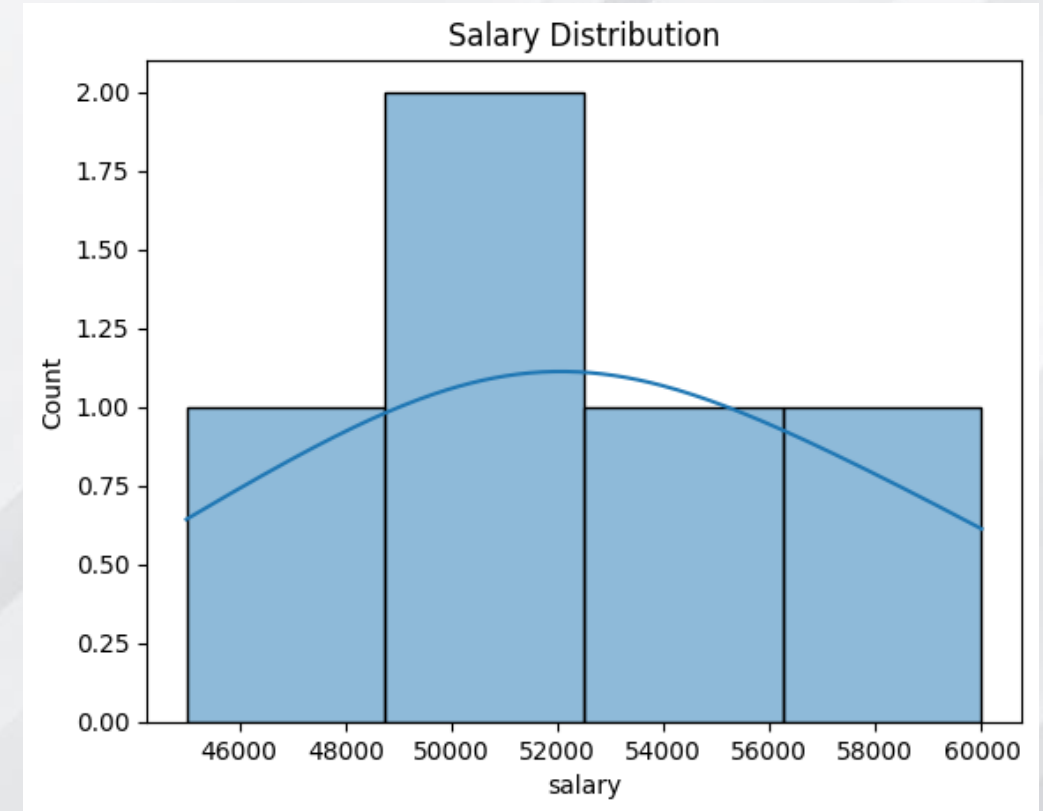
Positive skew = small frequent losses, few large gains.

Negative skew = small frequent gains, few large losses → riskier pattern.

Python Demo: Histogram + KDE

```
sns.histplot(df['salary'], kde=True)  
plt.title("Salary Distribution")  
plt.show()
```

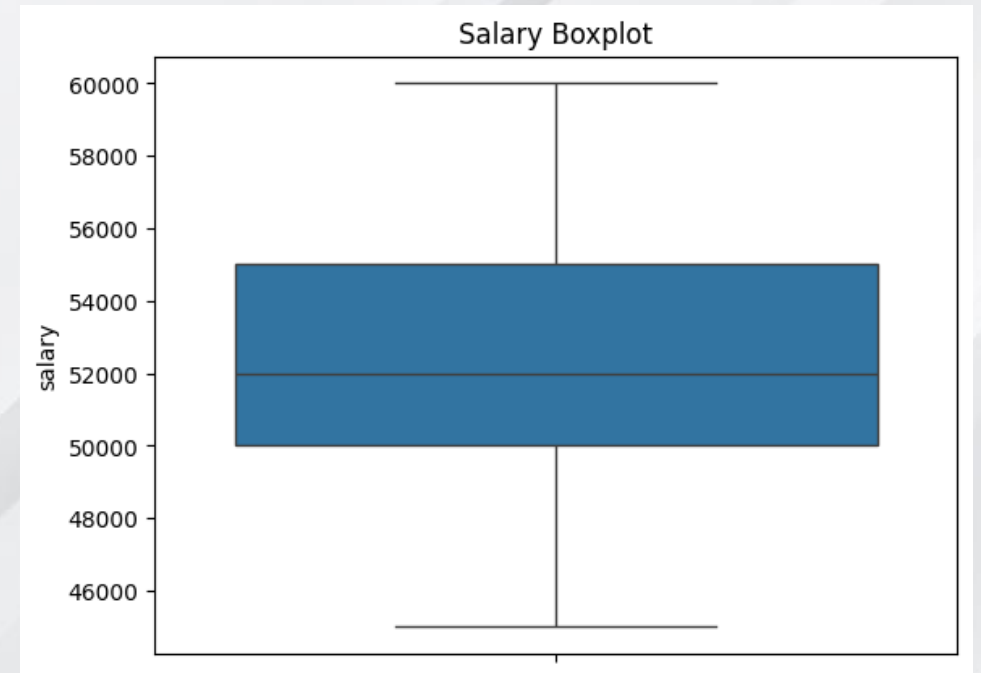
- The salaries are fairly evenly distributed around the mean (~52,400)
- KDE (Kernel Density Estimate) is a smooth curve that estimates the probability density of the data.
- It helps see the shape of the distribution more clearly than just the bars, e.g., peaks, symmetry, or multiple clusters.



Python Demo: Boxplot for outliers

```
sns.boxplot(y=df['salary'])  
plt.title("Salary Boxplot")  
plt.show()
```

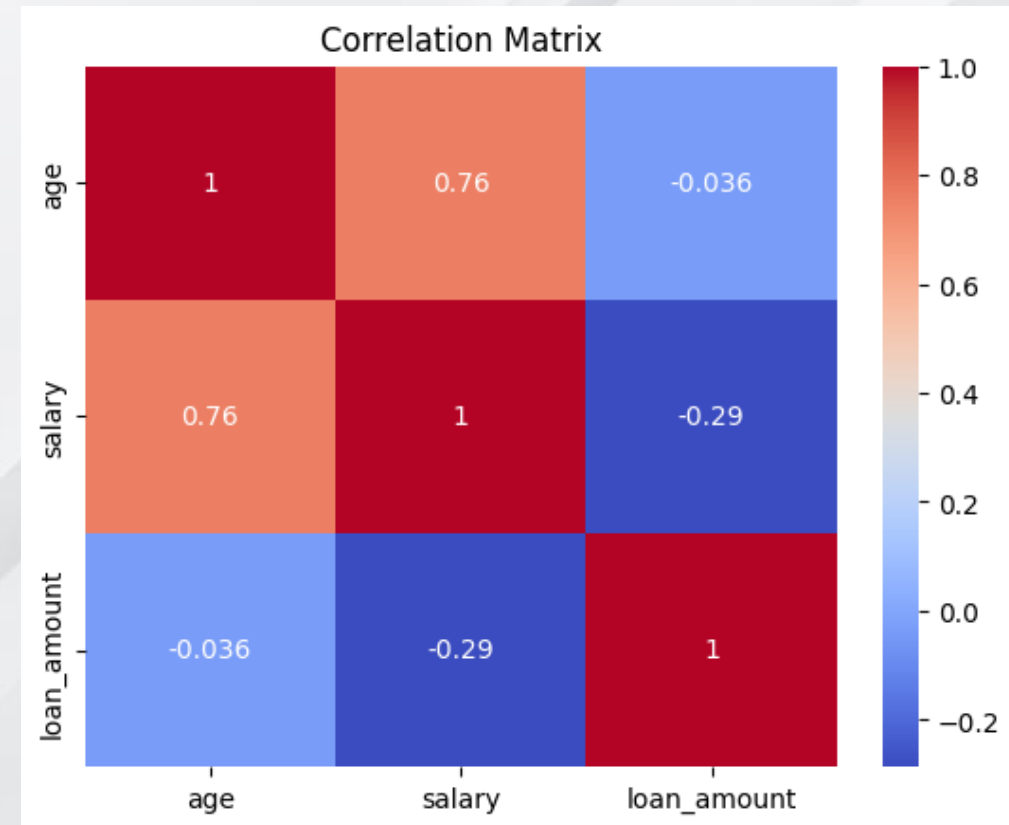
- Shows: Median, Q1–Q3 (interquartile range), whiskers, and outliers.
- Median salary (~52,000) is central; salaries outside whiskers are rare/outliers. Helps detect extreme borrowers.



Python Demo: Correlation (Pearson)

```
corr = df[['age', 'salary', 'loan_amount']].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```

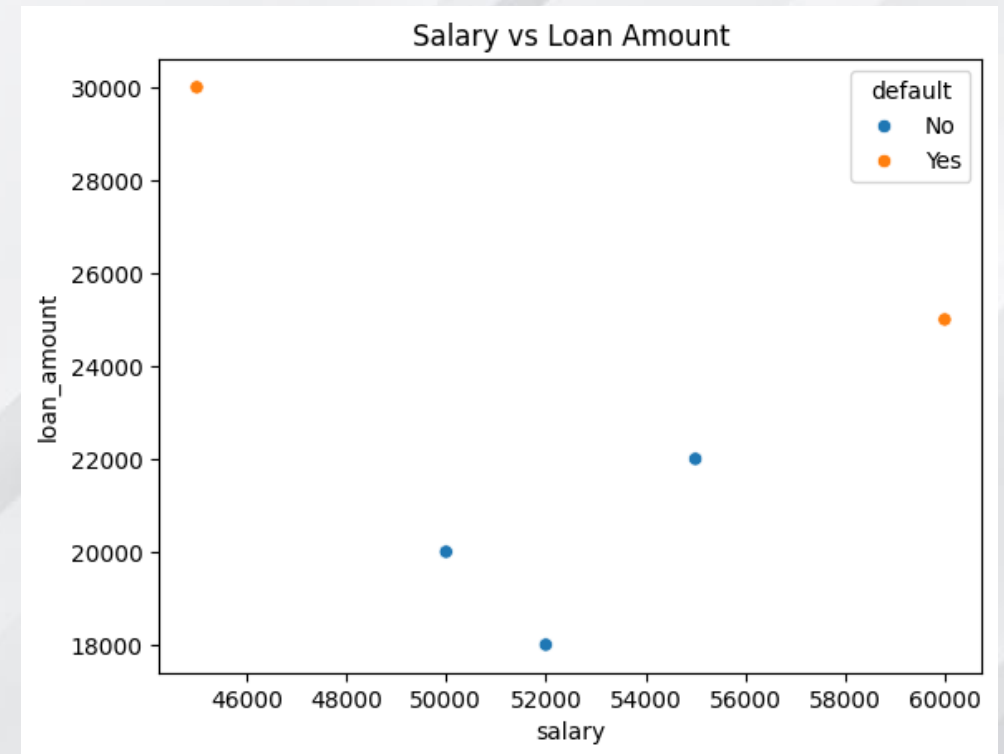
- Shows: Pairwise linear relationships between numeric variables (age, salary, loan_amount).
- Positive correlation → as salary increases, loan amount tends to increase slightly.
- Helps identify relationships for risk assessment.



Python Demo: Scatter plot & linear relationship

```
sns.scatterplot(x='salary', y='loan_amount', hue='default', data=df)  
plt.title("Salary vs Loan Amount")  
plt.show()
```

- Shows: Relationship between two numeric variables (e.g., salary vs loan_amount), colored by default status.
- Higher salaries tend to take larger loans;



Sampling

- Large datasets can be slow to inspect. Sampling lets you look at a few rows to understand structure, check for errors, or spot trends.
- Quickly explore data, test methods, or estimate statistics without using the full dataset.
- Sampling lets us quickly estimate portfolio risk, average loan amounts, or default rates without analyzing the entire dataset.
- **Syntax:** `df.sample(n=k, random_state=...)`
pick **k** rows randomly.

Example:

`sample = df.sample(n=4, random_state=0)` → randomly selects 4 borrowers for inspection.

Inferential statistics overview

Goal: Make conclusions about a population from a sample.

Typical flow:

- State H_0 (null hypothesis) — e.g., mean salary = 50,000.
- Compute test statistic (t-value).
- Compute p-value → probability of observing data if H_0 is true.
- Decide → reject H_0 if $p < 0.05$ (significant).

What are H_0 and H_1 ?

H_0 (null hypothesis): There is no difference or effect — the situation is normal (benchmark holds)

H_1 (alternative hypothesis): There is a difference or effect — something has changed (benchmark changes).

Example: Coin Flip Example

You suspect a coin might be biased (not fair).

H_0 (Null Hypothesis): The coin is fair $\rightarrow$ Probability of heads = 0.5

H_1 (Alternative Hypothesis): The coin is not fair $\rightarrow$ Probability of heads $\neq$ 0.5

You flip the coin 20 times and get 17 heads

That's much more than expected by chance.

$\rightarrow$ This gives evidence against H_0 , so we reject H_0 and say the coin is likely biased.

One-Sample t-test (Compare to a benchmark)

Goal: Check if your sample mean differs from a known or expected value.

Example:

An investor believes a stock's average daily return = 1%.

You collect 30 days of returns and find an average of 1.4%.

H_0 : Mean return = 1%

H_1 : Mean return $\neq$ 1%

- If the p-value < 0.05 , we reject H_0 and conclude that the stock's true average return is significantly different from 1% — meaning the stock is not performing as expected.
 - If $p \geq 0.05$, we fail to reject H_0 , meaning the sample doesn't provide enough evidence to say the average return differs from 1%.
- ❖ Use when comparing one sample to a benchmark (expected mean).

Two-Sample t-test (Compare two groups)

Goal: Check if two groups have different means.

Example:

Compare loan amounts of customers who defaulted vs. didn't default.

H_0 : Mean loan (default) = Mean loan (no default)

H_1 : Means are different

- If $p < 0.05$, $\rightarrow$ then we reject H_0 , the loan amounts differ significantly between the two groups.
- If $p \geq 0.05$, we fail to reject H_0 , meaning there's no significant difference between the two groups' means.
- ❖ Use this when comparing two independent samples.
- Paired t-test (`ttest_rel`) — Use when measurements are paired (before/after for same subjects).
- Syntax: `stats.ttest_rel(a,b)` — paired t-test.

Python Demo: one-sample t-test

```
from scipy import stats
```

```
# test if mean stock return differs from 1%  
returns = [0.8, 1.2, 1.0, 1.5, 0.7, 1.3, 0.9]  
t_stat, p_value = stats.ttest_1samp(returns, 1.0)
```

```
print("t:", t_stat)  
print("p:", p_value)
```

- returns → our sample data.
- 1.0 → the population mean under H_0 (the expected value).
- t_stat → the test statistic value.
- p_value → probability that such data would occur if H_0 were true.
- If p_value < 0.05 → Reject H_0 (the mean is significantly different).

Python Demo: two-sample t-test (comparing the means of two groups)

```
from scipy import stats
```

```
# compare stock returns from two different companies
```

```
company_A = [1.2, 1.0, 1.3, 1.1, 0.9, 1.4]
```

```
company_B = [0.8, 0.9, 1.0, 0.7, 0.9, 1.1]
```

```
t_stat, p_value = stats.ttest_ind(company_A, company_B)
```

```
print("t:", t_stat)
```

```
print("p:", p_value)
```

H_0 : The two companies have the same average return.

H_1 : The two companies have different average returns.

If $p_value < 0.05 \rightarrow$ Reject $H_0 \rightarrow$ their average returns are significantly different.

Confidence Intervals (t distribution)

- A confidence interval tells us where the real average is likely to be.

Example with our loan data:

- We calculate the average loan amount from our sample.
- The 95% confidence interval might be, say, \$18,000 to \$30,000.
- This means: “We are pretty sure the true average loan for all customers is somewhere in this range.”
- It’s basically a range around the sample mean that accounts for uncertainty.
- So instead of saying “the average loan is \$22,000,” we say “it’s probably between \$18,000 and \$30,000.”

Python Demo: Confidence Intervals (t distribution)

```
# after taking a sample from our loan dataset
```

```
mean_loan = df['loan_amount'].mean()
```

```
se_loan = df['loan_amount'].std(ddof=1) / np.sqrt(len(df))
```

```
ci_loan = stats.t.interval(0.95, df=len(df)-1, loc=mean_loan, scale=se_loan)
```

```
mean_loan, ci_loan
```

- `se_loan = df['loan_amount'].std(ddof=1) / np.sqrt(len(df))`: Computes the standard error (SE):
 - SE measures how much the sample mean might vary if we took different samples.

```
ci_loan = stats.t.interval(0.95, df=len(df)-1, loc=mean_loan, scale=se_loan):
```

Calculates the 95% confidence interval using the t-distribution.

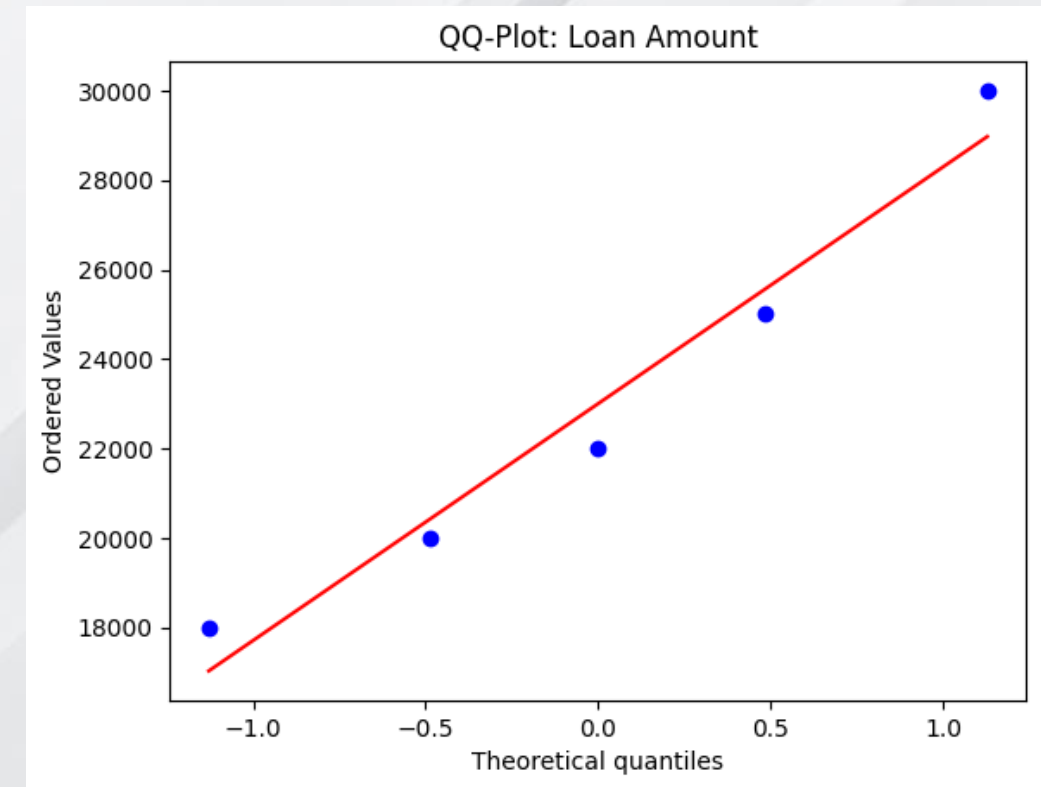
Nonparametric idea: ranks & medians

- When data are not normal (for example, some people have very large loans or salaries that make the average misleading), we cannot use normal tests like the t-test.
- So instead, we use nonparametric methods such as:
 1. Median → the middle value, not affected by extreme numbers.
 2. Rank-based tests (like the Mann-Whitney test) → compare two groups using their ranks (order), not the raw values.

Python Demo: Probability plots (normality check)

```
stats.probplot(df['loan_amount'], dist="norm", plot=plt)
plt.title("QQ-Plot: Loan Amount")
plt.show()
```

- `stats.probplot(x, dist, plot=plt)`: visually check if data follows a normal distribution.
 - In the visual, points close to the line → roughly normal distribution (Loan amount are approximately normally distributed).
 - Normality assumption matters for t-tests, confidence intervals, and risk models.



Python Demo: Nonparametric idea: ranks & medians

#compares two independent groups to see if one tends to have higher values than the other — without assuming normal distribution.

```
from scipy.stats import mannwhitneyu
```

```
group_yes = df[df['default']=='Yes']['loan_amount']
```

```
group_no = df[df['default']=='No']['loan_amount']
```

```
stat, p = mannwhitneyu(group_yes, group_no)
```

```
print(stat, p)
```

- If $p < 0.05$, the two groups have significantly different loan distributions.
- If $p \geq 0.05$, there's no significant difference between them.

Bootstrapping (concept)

- Bootstrapping means we take many random samples (with replacement) from our data to estimate how much results (like the mean or median) can vary.

Why we use it:

- Sometimes we don't know the true theoretical distribution, Bootstrapping lets us approximate it using only our sample.

Example:

Imagine we have historical data of loan returns or portfolio returns.

We can resample this data many times and calculate the average return each time.

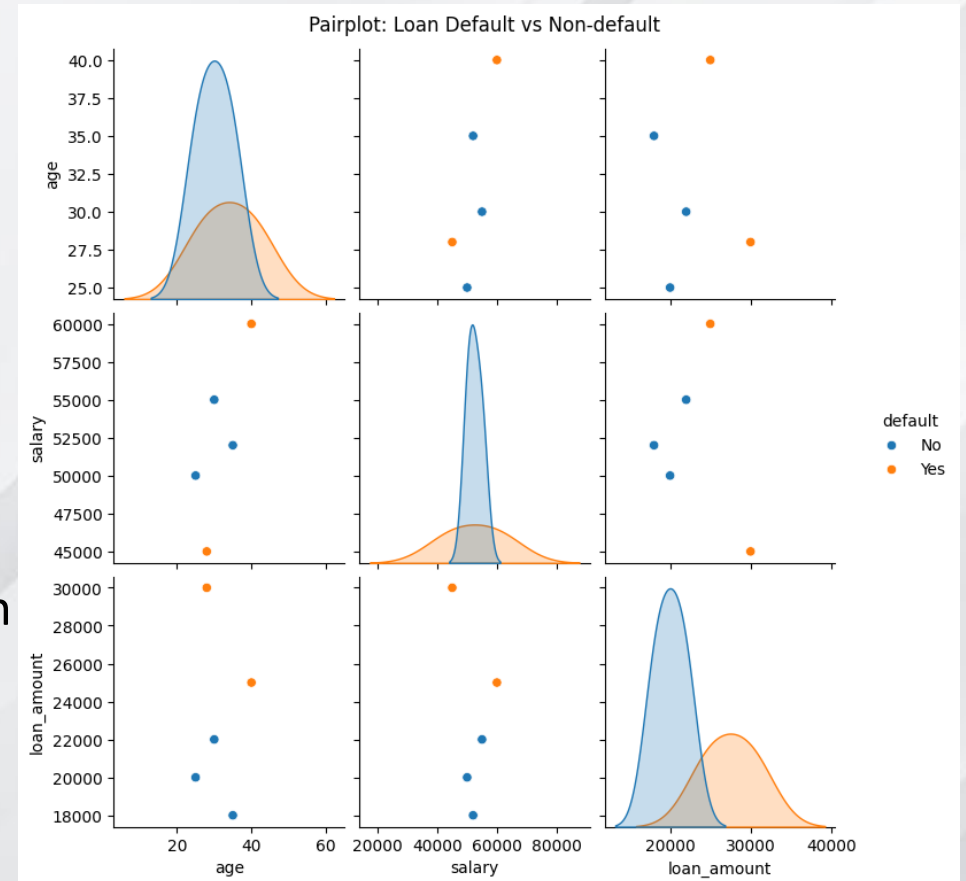
- From these averages, we can estimate a confidence interval — showing how uncertain or stable our expected return is.

Python Demo: Visual pairwise exploration: pairplot

#Shows relationships between all numeric variables in a dataset

```
sns.pairplot(df[['age','salary','loan_amount',  
'default']], hue='default')  
plt.suptitle("Pairplot: Loan Default vs  
Non-default", y=1.02)  
plt.show()
```

- If the red points (default = Yes) appear in areas with higher loan amounts and lower salaries, it may suggest borrowers with lower income and larger loans are more likely to default.
- The blue points (default = No) clustering in moderate salary and smaller loan regions show more stable borrowers.
- The diagonal histograms show each variable's distribution for both groups — for example, we might see that most defaults occur among customers with lower salaries.



Python Demo: Categorical Summaries: value_counts() and Barplot

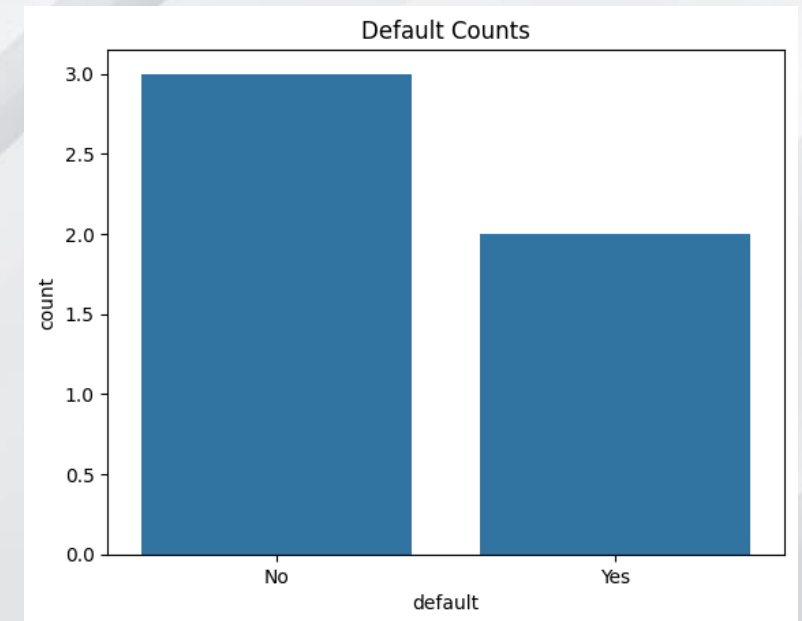
#quickly see how many customers defaulted vs. how many didn't

```
print(df['default'].value_counts())
```

```
sns.countplot(x='default', data=df)  
plt.title("Default Counts")  
plt.show()
```

- This tells us whether the dataset is balanced or not.
- If one bar is much taller (for example, many “No” and few “Yes”), it means the dataset is imbalanced — a common issue in loan prediction.
- Such imbalance affects how well a model can learn to detect risky (defaulting) customers.

```
default  
No      3  
Yes     2  
Name: count, dtype: int64
```



Python Demo: Grouped statistics and Detecting outliers using z-score (multi-column)

#Compare summary statistics across groups.

```
df.groupby('default')['loan_amount'].agg(['mean','median','count']).round(2)
```

Example: Are loans higher for defaulters vs non-defaulters? Helps detect **risk patterns**.

#Flag rows where any variable deviates >3 standard deviations.

```
z = np.abs(stats.zscore(df[['loan_amount','interest_rate']]))
```

```
outliers = (z > 3).any(axis=1)
```

```
df_out = df[outliers]
```

```
df_out
```

Example: Detect unusually large loans, extreme returns, or abnormal interest rates that may affect analysis or indicate fraud.

Rolling statistics

- We take the average of a few values at a time (for example, every 10 points)
- Then we move forward one step and take the next 10 values
- This helps us smooth out the ups and downs in the data.

Example:

Let's say you have the loan amounts for 10 customers:

If we use a rolling window = 3,

that means we calculate the average of 3 loans at a time:

First average = $(10,000 + 12,000 + 9,000) / 3 = 10,333$

Next average = $(12,000 + 9,000 + 11,000) / 3 = 10,666$

and so on...

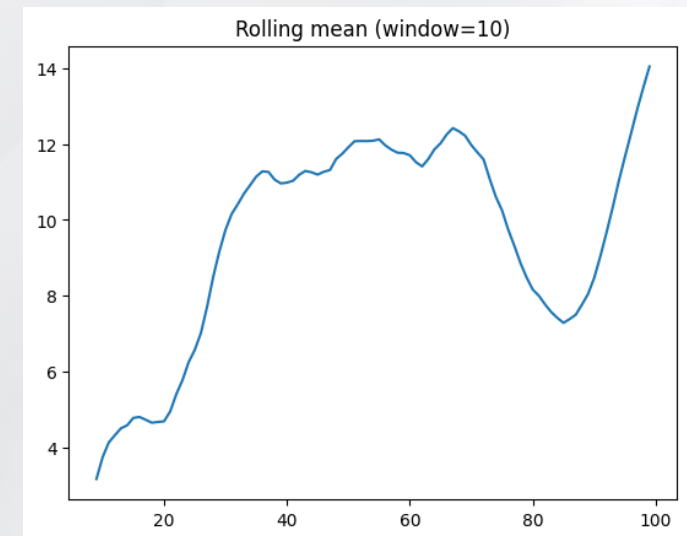
This way, the line of averages looks smoother — no sharp jumps.

Python Demo: Rolling statistics

#We use rolling mean to see trends —for example, average loan demand or stock #price over time —without getting distracted by small daily changes.

```
s = pd.Series(np.random.randn(100)).cumsum()  
s.rolling(window=10).mean().plot(title='Rolling mean (window=10)')  
plt.show()
```

- `pd.Series(np.random.randn(100)).cumsum()`: creates fake data that goes up and down (like changing loan values over time).
- `.rolling(window=10)`: takes 10 values at a time.
- `.mean()`: finds the average of those 10 values.
- `.plot()`: draws a smooth line.



Real-world finance use case: Portfolio Risk Prediction.

This Jupyter notebook will:

- Teach descriptive & inferential statistics
- Visualize relationships and distributions
- Prepare data for regression/classification of portfolio risk (low, medium, high)

Lab Activity: Statistical Foundations for Data Science

```
# -----
```

```
# Application: Portfolio Risk Prediction
```

```
# -----
```

```
# Importing essential libraries
```

```
import pandas as pd
```

```
import numpy as np
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

```
from scipy import stats
```

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

Lab Activity: Statistical Foundations for Data Science

```
# Create a small synthetic dataset for portfolio risk analysis
```

```
np.random.seed(42)
```

```
n = 150
```

```
data = {
```

```
    "portfolio_id": range(1, n+1),
```

```
    "avg_return": np.random.normal(0.08, 0.03, n), # average annual return
```

```
    "volatility": np.random.normal(0.12, 0.04, n), # std dev of returns
```

```
    "sharpe_ratio": np.random.normal(0.8, 0.25, n), # risk-adjusted return
```

```
    "investment_amount": np.random.randint(50000, 500000, n), # in USD
```

```
}
```

```
df = pd.DataFrame(data)
```

```
# Create a categorical target: risk level
```

```
df["risk_level"] = pd.cut(df["volatility"],
```

```
    bins=[0, 0.1, 0.15, 0.25],
```

```
    labels=["Low", "Medium", "High"])
```

```
df.head()
```

Lab Activity: Statistical Foundations for Data Science

Basic Info & Summary Statistics

```
df.info()  
df.describe().round(3)
```

Detect Missing Values & Duplicates

```
print("Missing values:\n", df.isnull().sum())  
print("\nDuplicated rows:", df.duplicated().sum())
```

Descriptive Statistics — Central Tendency & Dispersion

```
print("Mean return:", df["avg_return"].mean().round(3))  
print("Median volatility:", df["volatility"].median().round(3))  
print("Std of Sharpe ratio:", df["sharpe_ratio"].std().round(3))  
print("Range of investment:", df["investment_amount"].max() - df["investment_amount"].min())
```

Lab Activity: Statistical Foundations for Data Science

Visualizing Distributions

```
fig, axes = plt.subplots(1, 2, figsize=(12,5))
sns.histplot(df["avg_return"], kde=True, ax=axes[0])
axes[0].set_title("Average Return Distribution")
sns.boxplot(y=df["volatility"], ax=axes[1])
axes[1].set_title("Volatility Boxplot")
plt.show()
```

Correlation Analysis

```
corr = df[["avg_return", "volatility", "sharpe_ratio", "investment_amount"]].corr()
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

Lab Activity: Statistical Foundations for Data Science

Inferential Statistics — One-Sample t-Test

Hypothesis: Is average portfolio return significantly greater than 0.07 (7%)?

```
t_stat, p_value = stats.ttest_1samp(df["avg_return"], 0.07)
```

```
print(f"T-statistic: {t_stat:.3f}, P-value: {p_value:.4f}")
```

Group Analysis — Mean Return per Risk Level

```
group_stats = df.groupby("risk_level")[["avg_return", "sharpe_ratio"]].mean().round(3)
```

```
group_stats
```

#Visualize Group Differences

```
sns.boxplot(x="risk_level", y="avg_return", data=df)
```

```
plt.title("Average Return by Risk Level")
```

```
plt.show()
```

```
sns.violinplot(x="risk_level", y="sharpe_ratio", data=df)
```

```
plt.title("Sharpe Ratio by Risk Level")
```

```
plt.show()
```

Lab Activity: Statistical Foundations for Data Science

```
# Two-Sample t-Test — Compare Two Risk Levels
```

```
# Checks if high-risk portfolios have significantly different average returns from low-risk ones.
```

```
low = df[df["risk_level"]=="Low"]["avg_return"]
```

```
high = df[df["risk_level"]=="High"]["avg_return"]
```

```
t_stat, p_value = stats.ttest_ind(low, high, equal_var=False)
```

```
print(f"T-statistic: {t_stat:.3f}, P-value: {p_value:.4f}")
```

```
# Detecting Outliers (Z-score method)
```

```
z_scores = np.abs(stats.zscore(df[["avg_return", "volatility", "sharpe_ratio"]]))
```

```
outliers = (z_scores > 3).any(axis=1)
```

```
print("Number of outliers:", outliers.sum())
```

```
df[outliers].head()
```

```
#Handling Outliers (optional removal)
```

```
df_clean = df[~outliers].reset_index(drop=True)
```

```
print("Cleaned dataset shape:", df_clean.shape)
```

Lab Activity: Statistical Foundations for Data Science

Scaling Numerical Features

```
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df_clean[["avg_return", "volatility", "sharpe_ratio",
"investment_amount"]])
df_scaled = pd.DataFrame(scaled, columns=["avg_return", "volatility", "sharpe_ratio",
"investment_amount"])
df_scaled.head()
```

Statistical Visualization Summary

```
sns.pairplot(df_clean[["avg_return", "volatility", "sharpe_ratio", "risk_level"]],
            hue="risk_level", diag_kind="kde", corner=True)
plt.suptitle("Pairwise Relationships among Portfolio Features", y=1.02)
plt.show()
```

Recap

- Descriptive Statistics: Learned how to summarize financial data using mean, median, standard deviation, range, and percentiles.
- Inferential Statistics: Applied hypothesis testing (t-tests) to draw conclusions about population data from samples.
- Data Visualization: Used histograms, boxplots, violin plots, heatmaps, and pairplots to visually explore data distributions and relationships.
- Outlier Detection: Identified unusual values using Z-scores and addressed them to ensure data quality.
- Portfolio Risk Application: Applied all techniques to a real-world finance problem: predicting risk levels in investment portfolios.

References

- Klaas, J. Machine Learning for Finance: Principles and Practice for Financial Insiders.
- López de Prado, M. Advances in Financial Machine Learning.
- SciPy, Seaborn, and scikit-learn documentation.