1. Importing Pandas

python
CopyEdit

```
import pandas as pd
```

2. Creating a DataFrame

python
CopyEdit

```
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'Los Angeles', 'Chicago']}

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

3. Reading & Writing Data

python
CopyEdit

```
df = pd.read_csv('file.csv')  # Read CSV file
df.to_csv('output.csv', index=False)  # Save DataFrame to CSV

df = pd.read_excel('file.xlsx')  # Read Excel file
df.to_excel('output.xlsx', index=False)  # Save to Excel

df = pd.read_json('file.json')  # Read JSON file
df.to_json('output.json')  # Save DataFrame to JSON
```

4. Viewing Data

python
CopyEdit

```
df.head()  # First 5 rows
df.tail()  # Last 5 rows
df.info()  # Summary of DataFrame
df.describe()  # Summary statistics
df.shape  # (rows, columns)
df.columns  # List of column names
df.dtypes  # Data types of each column
```

5. Selecting Data

python
CopyEdit

```
df['Name']  # Select a single column
df[['Name', 'Age']]  # Select multiple columns
df.iloc[0]  # Select first row
df.iloc[0:2]  # Select first two rows
df.loc[df['Age'] > 25]  # Filter rows where Age > 25
```

6. Adding & Removing Columns

python
CopyEdit

```
df['Salary'] = [50000, 60000, 70000]  # Add new column
df.drop('Salary', axis=1, inplace=True)  # Remove column
```

7. Handling Missing Data

python
CopyEdit

```
df.dropna()  # Remove rows with NaN values
df.fillna(value=0)  # Fill NaN values with 0
df.isnull().sum()  # Count missing values per column
```

8. Sorting & Grouping

python
CopyEdit

```
df.sort_values(by='Age', ascending=False)  # Sort by Age (descending)
df.groupby('City').mean()  # Group by City and get mean values
```

9. Merging & Joining Data

python
CopyEdit

```
df1 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie']})
df2 = pd.DataFrame({'ID': [1, 2, 3], 'Salary': [50000, 60000, 70000]})
merged_df = pd.merge(df1, df2, on='ID')  # Merge on ID column
```

10. Converting Data Types

python
CopyEdit

```
df['Age'] = df['Age'].astype(float)  # Convert Age to float
```

11. Applying Functions

python
CopyEdit

```
df['Age'] = df['Age'].apply(lambda x: x + 1)  # Increase Age by 1
```

Would you like more advanced pandas operations? 🚀

4o

You said:

df.drop('Salary', axis=1, inplace=True)

explain

ChatGPT said:

The command:

python

CopyEdit

```
df.drop('Salary', axis=1, inplace=True)
```

Breaking it Down:

1. **`df.drop()`** → This function is used to remove rows or columns from a DataFrame.
2. **`'Salary'`** → Specifies the column to be dropped.
3. **`axis=1`** → Specifies that we are dropping a column (not a row).
    - `axis=0` → Drops a row.
    - `axis=1` → Drops a column.
4. **`inplace=True`** → This modifies the original DataFrame directly instead of creating a new one.
    - If `inplace=False` (default), it returns a new DataFrame without modifying `df`.

Example:

Before Dropping

python

CopyEdit

```
import pandas as pd

# Creating a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'Salary': [50000, 60000, 70000]}

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

Output:

markdown

CopyEdit

```
     Name  Age  Salary
0  Alice   25  50000
1    Bob   30  60000
```

```
2 Charlie   35   70000
```

Dropping the "Salary" Column

python
CopyEdit

```
df.drop('Salary', axis=1, inplace=True)
print(df)
```

Output:

markdown
CopyEdit

```
     Name   Age
0   Alice    25
1     Bob    30
2 Charlie    35
```

Now, the `"Salary"` column has been removed from the DataFrame.

Key Takeaways:

- `axis=1` is used for dropping columns, while `axis=0` is for dropping rows.
- `inplace=True` directly modifies `df`, whereas `inplace=False` (default) returns a new DataFrame without modifying the original.

Let me know if you need further clarification! 😊

You said:

numpy

ChatGPT said:

NumPy Basics

NumPy (Numerical Python) is a powerful library for numerical computing in Python. It provides support for arrays, mathematical operations, and functions optimized for performance.

1. Importing NumPy

python

CopyEdit

```
import numpy as np
```

2. Creating Arrays

a) Creating a NumPy Array

python

CopyEdit

```
arr = np.array([1, 2, 3, 4, 5])  # 1D array
print(arr)
```

Output:

csharp

CopyEdit

```
[1 2 3 4 5]
```

b) Creating a 2D Array

python

CopyEdit

```
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2d)
```

Output:

lua

CopyEdit

```
[[1 2 3]
 [4 5 6]]
```

3. Array Attributes

python

CopyEdit

```
print(arr.shape)  # (5,) -> Shape of the array
print(arr2d.shape)  # (2, 3) -> 2 rows, 3 columns
print(arr.dtype)  # int32 (or int64)
print(arr.size)  # 5 -> Total elements
```

4. Creating Special Arrays

a) Zeros and Ones

python
CopyEdit

```
zeros = np.zeros((3, 3))  # 3x3 matrix of zeros
ones = np.ones((2, 4))  # 2x4 matrix of ones
```

b) Random Arrays

python
CopyEdit

```
rand_arr = np.random.rand(3, 3)  # 3x3 matrix with random values between 0 and 1
rand_int = np.random.randint(1, 10, (2, 3))  # Random integers from 1 to 9 in a 2x3 matrix
```

c) Identity Matrix

python
CopyEdit

```
identity = np.eye(3)  # 3x3 Identity Matrix
```

5. Indexing and Slicing

python
CopyEdit

```
arr = np.array([10, 20, 30, 40, 50])

print(arr[2])  # 30 (Indexing)
print(arr[1:4])  # [20 30 40] (Slicing)
```

2D Indexing

python
CopyEdit

```
arr2d = np.array([[1, 2, 3], [4, 5, 6]])

print(arr2d[1, 2])  # 6 (Row 1, Column 2)
print(arr2d[:, 1])  # [2 5] (All rows, column 1)
```

6. Mathematical Operations

python
CopyEdit

```
arr = np.array([1, 2, 3, 4])
```

```
print(arr + 5)  # [6 7 8 9]
print(arr * 2)  # [2 4 6 8]
print(arr ** 2)  # [1 4 9 16]
```

Element-wise Operations

python

CopyEdit

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

print(arr1 + arr2)  # [5 7 9]
print(arr1 * arr2)  # [4 10 18]
```

7. Aggregations

python

CopyEdit

```
arr = np.array([10, 20, 30, 40])

print(np.sum(arr))  # 100
print(np.mean(arr))  # 25.0
print(np.max(arr))  # 40
print(np.min(arr))  # 10
print(np.std(arr))  # Standard deviation
```

8. Reshaping and Transposing

python

CopyEdit

```
arr = np.array([[1, 2, 3], [4, 5, 6]])

reshaped = arr.reshape((3, 2))  # Change shape to (3,2)
transposed = arr.T  # Transpose (swap rows and columns)
```

9. Stacking Arrays

python

CopyEdit

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

vert_stack = np.vstack((arr1, arr2))  # Vertical stacking
hor_stack = np.hstack((arr1, arr2))  # Horizontal stacking
```

10. Filtering & Conditional Selection

python

CopyEdit

```
arr = np.array([10, 20, 30, 40])

print(arr[arr > 20])  # [30 40]
```

Summary

Operation	NumPy Command
Create array	`np.array([1,2,3])`
Zeros array	`np.zeros((3,3))`
Ones array	`np.ones((2,2))`
Random numbers	`np.random.rand(3,3)`
Identity matrix	`np.eye(3)`
Reshape array	`arr.reshape((3,2))`
Transpose	`arr.T`
Sum of elements	`np.sum(arr)`
Mean	`np.mean(arr)`
Maximum value	`np.max(arr)`
Element-wise addition	`arr1 + arr2`
Conditional selection	`arr[arr > 10]`

Would you like an explanation for a specific topic? 🚀

4o

You said:

scipy

ChatGPT said:

SciPy Basics

SciPy (Scientific Python) is a library built on NumPy that provides functions for scientific and technical computing, including linear algebra, optimization, integration, interpolation, and statistics.

1. Installing SciPy

If you haven't installed SciPy yet, you can install it using:

bash

CopyEdit

```
pip install scipy
```

2. Importing SciPy

python

CopyEdit

```
import scipy as sp
import numpy as np  # SciPy is built on NumPy
```

3. Modules in SciPy

SciPy has several submodules for different types of scientific computations:

Submodule	Purpose
`scipy.linalg`	Linear algebra functions (matrix operations)
`scipy.optimize`	Optimization (finding minima/maxima)
`scipy.integrate`	Integration (solving integrals)
`scipy.stats`	Statistics and probability
`scipy.spatial`	Spatial computations (distance calculations)
`scipy.interpolate`	Interpolation (estimating values between points)
`scipy.fft`	Fast Fourier Transform (signal processing)
`scipy.ndimage`	Image processing

4. Linear Algebra (`scipy.linalg`)

This module provides matrix operations and solving systems of linear equations.

a) Determinant of a Matrix

python

CopyEdit

```
from scipy.linalg import det
```

```
A = np.array([[1, 2], [3, 4]])
print(det(A))  # Determinant of matrix A
```

b) Inverse of a Matrix

python
CopyEdit

```
from scipy.linalg import inv

A_inv = inv(A)  # Compute inverse
print(A_inv)
```

c) Solving a System of Linear Equations

For the system:

A
x
=
b
Ax=b
python
CopyEdit

```
from scipy.linalg import solve

A = np.array([[3, 2], [1, 4]])
b = np.array([5, 6])
x = solve(A, b)
print(x)  # Solution of Ax = b
```

5. Optimization (`scipy.optimize`)

Used for finding minima, maxima, and root finding.

a) Finding the Minimum of a Function

Let's minimize the function

f
(
x
)
=
x
2
+
3
x
+
5

f(x)=x2+3x+5.

python
CopyEdit

```
from scipy.optimize import minimize

def f(x):
    return x**2 + 3*x + 5

result = minimize(f, x0=0)  # x0 is the initial guess
print(result.x)  # Optimal value of x
```

b) Finding the Root of an Equation

Solving

f
(
x
)
=
x
2
−
4
=
0
f(x)=x2−4=0.

python
CopyEdit

```
from scipy.optimize import root

def equation(x):
    return x**2 - 4

sol = root(equation, x0=1)  # x0 is the initial guess
print(sol.x)  # Root of equation
```

6. Integration (`scipy.integrate`)

Used for computing definite integrals.

a) Definite Integral

Integrating

f
(

x
)
=
x
2
f(x)=x2 from 0 to 2.

python
CopyEdit
```
from scipy.integrate import quad

def f(x):
    return x**2

integral, error = quad(f, 0, 2)
print(integral)  # Output: 2.67
```

7. Interpolation (`scipy.interpolate`)

Interpolation helps estimate values between given data points.

python
CopyEdit
```
from scipy.interpolate import interp1d

x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

f = interp1d(x, y, kind='linear')  # Linear interpolation
print(f(3.5))  # Predict value at x = 3.5
```

8. Statistics (`scipy.stats`)

Used for probability distributions, mean, median, and statistical tests.

a) Mean, Median, and Mode

python
CopyEdit
```
from scipy import stats

data = [1, 2, 2, 3, 4, 5, 6]

print(stats.mode(data))  # Mode
print(np.mean(data))  # Mean
print(np.median(data))  # Median
```

b) Normal Distribution

python
CopyEdit

```
from scipy.stats import norm

# Probability Density Function (PDF) at x=0
print(norm.pdf(0, loc=0, scale=1))  # Mean=0, Std=1
```

9. Distance Calculation (`scipy.spatial.distance`)

a) Euclidean Distance

python
CopyEdit

```
from scipy.spatial.distance import euclidean

point1 = np.array([1, 2])
point2 = np.array([4, 6])

dist = euclidean(point1, point2)
print(dist)  # Output: 5.0
```

b) Cosine Similarity

python
CopyEdit

```
from scipy.spatial.distance import cosine

vec1 = np.array([1, 0, -1])
vec2 = np.array([-1, 0, 1])

similarity = 1 - cosine(vec1, vec2)
print(similarity)  # Output: -1 (opposite vectors)
```

10. Fourier Transform (`scipy.fft`)

Used for signal processing and frequency analysis.

python
CopyEdit

```
from scipy.fft import fft

signal = np.array([1, 2, 3, 4])
transformed = fft(signal)
print(transformed)
```

Summary of SciPy Functions

Module	Function	Description
`scipy.linalg`	`det(A)`	Compute determinant of a
	`inv(A)`	Compute inverse of a matrix
	`solve(A, b)`	Solve linear system A x = b Ax=b
`scipy.optimize`	`minimize(f, x0)`	Find minimum of function
	`root(f, x0)`	Find root of function
`scipy.integrate`	`quad(f, a, b)`	Compute integral ∫ a b f (x) d x ∫ab f(x)dx
`scipy.interpolate`	`interp1d(x, y, kind)`	Interpolate between data points
`scipy.stats`	`stats.mode(data)`	Compute mode
	`norm.pdf(x, loc, scale)`	Normal distribution PDF
`scipy.spatial.distance`	`euclidean(p1, p2)`	Compute Euclidean distance
	`cosine(vec1, vec2)`	Compute Cosine similarity
`scipy.fft`	`fft(signal)`	Compute Fourier transform

Conclusion

SciPy extends NumPy for advanced mathematical, statistical, and scientific computations. It is widely used in: ✅ Machine Learning
✅ Data Science
✅ Signal Processing
✅ Optimization Problems

Let me know if you need further explanation! 🚀

4o

You said:

descriptive analytics all cmds
ChatGPT said:

Descriptive Analytics in Python

Descriptive analytics helps summarize and describe the key characteristics of data using statistical measures. In Python, we primarily use **NumPy, Pandas, and SciPy** to perform descriptive analytics.

1. Importing Required Libraries

`python`
CopyEdit

```
import numpy as np
import pandas as pd
from scipy import stats
```

2. Loading Data

`python`
CopyEdit

```
# Creating a sample DataFrame
data = {
    'Age': [25, 30, 35, 40, 45, 50, 55, 60, 65, 70],
    'Salary': [40000, 50000, 60000, 70000, 80000, 90000,
100000, 110000, 120000, 130000]
}

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

3. Summary Statistics

a) Basic Summary

`python`
CopyEdit

```
print(df.describe())  # Summary of numerical columns
```

Output:

`matlab`
CopyEdit

```
             Age        Salary
count  10.00000     10.000000
mean   47.50000  85000.000000
```

```
std    15.13825  25000.000000
min    25.00000  40000.000000
25%    36.25000  62500.000000
50%    47.50000  85000.000000
75%    58.75000  107500.000000
max    70.00000  130000.000000
```

4. Central Tendency

a) Mean (Average)

python

CopyEdit

```
print(df.mean())  # Column-wise mean
```

Or for a specific column:

python

CopyEdit

```
print(df['Age'].mean())
```

b) Median

python

CopyEdit

```
print(df.median())  # Column-wise median
```

Or:

python

CopyEdit

```
print(df['Salary'].median())
```

c) Mode

python

CopyEdit

```
print(df.mode())  # Column-wise mode
```

Or:

python

CopyEdit

```
print(df['Age'].mode())
```

5. Dispersion Measures

a) Standard Deviation

python

CopyEdit

```
print(df.std())  # Column-wise standard deviation
```

b) Variance

python
CopyEdit

```
print(df.var())  # Column-wise variance
```

c) Range (Max - Min)

python
CopyEdit

```
print(df.max() - df.min())  # Range for each column
```

d) Interquartile Range (IQR)

python
CopyEdit

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

6. Shape of Data Distribution

a) Skewness (Measures Symmetry)

python
CopyEdit

```
print(df.skew())  # Skewness of each column
```

- **Skewness > 0** → Right-skewed (positively skewed)
- **Skewness < 0** → Left-skewed (negatively skewed)
- **Skewness ≈ 0** → Normally distributed

b) Kurtosis (Measures Peakedness)

python
CopyEdit

```
print(df.kurt())  # Kurtosis of each column
```

- **Kurtosis > 3** → Leptokurtic (Sharp peak)
- **Kurtosis < 3** → Platykurtic (Flat peak)
- **Kurtosis = 3** → Mesokurtic (Normal peak)

7. Correlation & Covariance

a) Correlation (Relationship between variables)

python
CopyEdit

```
print(df.corr())  # Correlation matrix
```

- **Values close to 1** → Strong positive correlation
- **Values close to -1** → Strong negative correlation
- **Values close to 0** → No correlation

b) Covariance (How two variables vary together)

python
CopyEdit

```
print(df.cov())  # Covariance matrix
```

8. Percentiles

python
CopyEdit

```
print(df.quantile([0.25, 0.50, 0.75]))  # 25th, 50th (median), 75th percentile
```

9. Counting Unique Values & Frequency

a) Count Non-Null Values

python
CopyEdit

```
print(df.count())  # Number of non-null values per column
```

b) Count Unique Values

python
CopyEdit

```
print(df['Age'].nunique())  # Number of unique values in 'Age'
```

c) Frequency of Each Value

python
CopyEdit

```
print(df['Age'].value_counts())  # Count occurrences of each value
```

10. Outlier Detection

a) Using Z-Score

python
CopyEdit

```
z_scores = np.abs(stats.zscore(df))
outliers = df[(z_scores > 3).any(axis=1)]  # Outliers beyond 3 standard deviations
print(outliers)
```

b) Using IQR Method

python
CopyEdit

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

outliers = df[(df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))].dropna()
print(outliers)
```

Summary of Descriptive Analytics Commands

Measure	Command
Summary Statistics	`df.describe()`
Mean	`df.mean()`
Median	`df.median()`
Mode	`df.mode()`
Standard Deviation	`df.std()`
Variance	`df.var()`
Range	`df.max() - df.min()`
Interquartile Range (IQR)	`df.quantile(0.75) - df.quantile(0.25)`
Skewness	`df.skew()`
Kurtosis	`df.kurt()`
Correlation Matrix	`df.corr()`
Covariance Matrix	`df.cov()`
Percentiles	`df.quantile([0.25, 0.50, 0.75])`
Unique Values Count	`df['column'].nunique()`
Value Counts	`df['column'].value_counts()`
Outlier Detection (Z-Score)	`stats.zscore(df)`
Outlier Detection (IQR)	`df[(df < (Q1 - 1.5 * IQR))

Conclusion

Descriptive analytics is essential for **data exploration, outlier detection, and data summarization**. It helps in **understanding the dataset before performing further analysis or modeling**.

Would you like me to explain any specific part in more detail? 😊🚀

4o

You said:

stats model

ChatGPT said:

Statsmodels in Python

Statsmodels is a Python library for statistical modeling, hypothesis testing, and data exploration. It provides tools for **descriptive statistics, regression analysis, hypothesis testing, and time series analysis**.

1. Installing Statsmodels

bash

CopyEdit

```
pip install statsmodels
```

2. Importing Required Libraries

python

CopyEdit

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

3. Descriptive Statistics

Statsmodels provides detailed statistical summaries.

python

CopyEdit

```
from statsmodels.stats.descriptivestats import describe

# Sample data
data = {'Age': [25, 30, 35, 40, 45, 50, 55, 60, 65, 70],
        'Salary': [40000, 50000, 60000, 70000, 80000, 90000,
100000, 110000, 120000, 130000]}
```

```
df = pd.DataFrame(data)

# Generate summary statistics
summary = describe(df)
print(summary)
```

4. Linear Regression

a) Ordinary Least Squares (OLS)

python
CopyEdit

```
# Defining dependent and independent variables
X = df['Age']
y = df['Salary']

X = sm.add_constant(X)  # Adds an intercept term

# Fitting the model
model = sm.OLS(y, X).fit()

# Model summary
print(model.summary())
```

🔷 **Interpreting OLS Output:**

- **R-squared**: Goodness of fit (closer to 1 is better).
- **P-value**: If $p < 0.05$, the independent variable significantly affects the dependent variable.
- **Coefficients**: The estimated effect of each independent variable.

b) Regression Using Formulas (`smf.ols`)

python
CopyEdit

```
model = smf.ols('Salary ~ Age', data=df).fit()
print(model.summary())
```

5. Hypothesis Testing

a) t-test (One Sample)

Tests if the mean of a sample differs significantly from a population mean.

python
CopyEdit

```
from statsmodels.stats.weightstats import ttest_1samp
```

```
t_stat, p_value, df = ttest_1samp(df['Salary'], 75000)  #
Test if mean Salary = 75000
print(f"T-statistic: {t_stat}, P-value: {p_value}")
```

b) Two-Sample t-test

python
CopyEdit

```
from statsmodels.stats.weightstats import ttest_ind

group1 = [50, 55, 60, 65, 70]  # Sample 1
group2 = [40, 45, 50, 55, 60]  # Sample 2

t_stat, p_value, df = ttest_ind(group1, group2)
print(f"T-statistic: {t_stat}, P-value: {p_value}")
```

6. ANOVA (Analysis of Variance)

Used to compare means of more than two groups.

python
CopyEdit

```
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Sample data
data = pd.DataFrame({'Group': ['A', 'A', 'B', 'B', 'C', 'C'],
                     'Score': [80, 85, 78, 82, 90, 92]})

# Fit ANOVA model
anova_model = ols('Score ~ Group', data=data).fit()
anova_table = sm.stats.anova_lm(anova_model, typ=2)

print(anova_table)
```

7. Logistic Regression

For binary classification problems.

python
CopyEdit

```
from statsmodels.api import Logit

# Sample binary classification data
data = {'Age': [22, 25, 30, 35, 40, 45, 50, 55, 60, 65],
        'Purchased': [0, 0, 0, 1, 1, 1, 1, 1, 1, 1]}  # 1 = Purchased, 0 = Not Purchased

df = pd.DataFrame(data)

# Independent and dependent variables
X = sm.add_constant(df['Age'])
y = df['Purchased']

# Fit logistic regression
logit_model = sm.Logit(y, X).fit()
print(logit_model.summary())
```

8. Time Series Analysis

a) Augmented Dickey-Fuller (ADF) Test

Used to check if a time series is stationary.

python
CopyEdit

```
from statsmodels.tsa.stattools import adfuller

time_series_data = [10, 12, 14, 13, 15, 18, 20, 22, 24, 25]
adf_test = adfuller(time_series_data)

print(f"ADF Statistic: {adf_test[0]}")
print(f"P-value: {adf_test[1]}")
```

If P-value < 0.05, the time series is **stationary**.

b) Autoregressive Integrated Moving Average (ARIMA)

python
CopyEdit

```
from statsmodels.tsa.arima.model import ARIMA

# Creating an ARIMA model
model = ARIMA(time_series_data, order=(1,1,1))  # (p,d,q)
arima_result = model.fit()

print(arima_result.summary())
```

9. Summary of Important Commands

Task	Command

Descriptive Statistics	`describe(df)`
Linear Regression (OLS)	`sm.OLS(y, X).fit()`
t-test (One Sample)	`ttest_1samp(df['col'], value)`
t-test (Two Sample)	`ttest_ind(group1, group2)`
ANOVA	`sm.stats.anova_lm(ols_model, typ=2)`
Logistic Regression	`sm.Logit(y, X).fit()`
ADF Test (Stationarity Check)	`adfuller(time_series_data)`
ARIMA Model	`ARIMA(time_series_data, order=(p,d,q)).fit()`

Conclusion

Statsmodels is a powerful library for: ✅ **Statistical Analysis**
✅ **Regression & Hypothesis Testing**
✅ **Time Series Analysis**

Let me know if you need a deeper explanation on any topic! 🚀

You said:

group by dunction to_csv function
ChatGPT said:

1. `groupby()` Function in Pandas

The `groupby()` function in Pandas is used to **group data** based on one or more columns and apply aggregate functions.

Example Dataset

python
CopyEdit

```
import pandas as pd

# Creating a sample DataFrame
data = {
    'Department': ['HR', 'IT', 'IT', 'HR', 'Finance',
'Finance', 'IT', 'HR'],
    'Employee': ['Alice', 'Bob', 'Charlie', 'David', 'Eve',
'Frank', 'Grace', 'Hank'],
    'Salary': [50000, 60000, 70000, 55000, 65000, 75000,
72000, 53000]
}

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

Basic `groupby()` Usage

1. Group by a Single Column and Aggregate

python
CopyEdit

```
grouped = df.groupby('Department')['Salary'].mean()
print(grouped)
```

🔷 This calculates the **average salary** for each department.

2. Group by Multiple Columns

python
CopyEdit

```
grouped = df.groupby(['Department', 'Employee'])
['Salary'].sum()
print(grouped)
```

🔷 Groups by **both Department and Employee**.

3. Applying Multiple Aggregate Functions

python
CopyEdit

```
grouped = df.groupby('Department')['Salary'].agg(['mean',
'max', 'min', 'count'])
print(grouped)
```

🔷 Computes **mean, max, min, and count** for each department.

4. Filtering Groups

python
CopyEdit

```
grouped = df.groupby('Department').filter(lambda x:
x['Salary'].mean() > 60000)
print(grouped)
```

🔷 Keeps only departments where **average salary > 60000**.

2. `to_csv()` Function

The `to_csv()` function is used to **save DataFrames to CSV files**.

Saving Data to a CSV File

python
CopyEdit

```
df.to_csv('employees.csv', index=False)
```

🔷 Saves `df` to a file named **employees.csv** without the index column.

Saving Only Specific Columns

python
CopyEdit

```
df[['Employee', 'Salary']].to_csv('salaries.csv',
index=False)
```

🔷 Saves only **Employee** and **Salary** columns.

Saving Without Header

python
CopyEdit

```
df.to_csv('no_header.csv', index=False, header=False)
```

🔷 Saves the file **without column headers**.

multivariate ananlysis :pca python code for diabetes dataset

ChatGPT said:

Here's a Python code snippet to perform Principal Component Analysis (PCA) on a diabetes dataset using `scikit-learn`. The code assumes you have the diabetes dataset in CSV format. If you are using the popular **PIMA Indians Diabetes Dataset**, you can download it from Kaggle or UCI Machine Learning Repository.

Steps:

1. Load the dataset.
2. Standardize the data.
3. Apply PCA.
4. Visualize explained variance.

PCA Implementation on Diabetes Dataset

python
CopyEdit

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv("diabetes.csv")  # Replace with actual path if needed

# Extract features (assuming the last column is the target variable)
X = df.iloc[:, :-1]
y = df.iloc[:, -1]   # Target (Diabetes: 1 or 0)

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=2)  # Reduce to 2 components for visualization
X_pca = pca.fit_transform(X_scaled)

# Explained variance ratio
explained_variance = pca.explained_variance_ratio_
print("Explained Variance Ratio:", explained_variance)

# Scatter plot of first two principal components
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='coolwarm', edgecolor='k', alpha=0.7)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA of Diabetes Dataset")
plt.colorbar(label="Diabetes (1=Yes, 0=No)")
plt.show()
```

Key Points:

- **Standardization:** PCA requires the data to be standardized for better performance.
- **Variance Explained:** `pca.explained_variance_ratio_` shows the proportion of variance retained.
- **Visualization:** Scatter plot helps to see how well PCA separates diabetic vs. non-diabetic cases.

Would you like me to modify this to include more components or another visualization? 🚀

4o

You said:

naive bayes

ChatGPT said:

Here's a **Naïve Bayes** classifier implementation for the **Diabetes dataset** using `sklearn`'s `GaussianNB`. The model will classify whether a person has diabetes (1) or not (0) based on the dataset.

Naïve Bayes Classifier for Diabetes Prediction

python
CopyEdit

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

# Load the dataset
df = pd.read_csv("diabetes.csv")  # Replace with actual path

# Extract features and target
X = df.iloc[:, :-1]  # All columns except last (features)
y = df.iloc[:, -1]   # Last column (target variable)
```

```
# Split into training and testing sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize features (Naïve Bayes can work without this,
but scaling improves performance)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the Naïve Bayes model
nb = GaussianNB()
nb.fit(X_train, y_train)

# Predictions
y_pred = nb.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print("\nClassification Report:\n",
classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test,
y_pred))
```

Key Points:

- **Gaussian Naïve Bayes** is used because the dataset features are continuous.
- **Standardization** improves model performance.
- **Evaluation Metrics:** Accuracy, precision, recall, F1-score, and confusion matrix are printed.

Would you like me to try **MultinomialNB** or **BernoulliNB** instead? 🚀

4o

You said:

svm

ChatGPT said:

Here's an **SVM (Support Vector Machine) Classifier** implementation for the **Diabetes dataset** using `scikit-learn`. The model will classify whether a person has diabetes (1) or not (0).

SVM Classifier for Diabetes Prediction

python
CopyEdit

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

# Load the dataset
df = pd.read_csv("diabetes.csv")  # Replace with actual path

# Extract features and target
X = df.iloc[:, :-1]  # All columns except last (features)
y = df.iloc[:, -1]   # Last column (target variable)

# Split into training and testing sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize features (SVM is sensitive to feature scales)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the SVM model with RBF kernel
svm = SVC(kernel='rbf', C=1.0, gamma='scale',
random_state=42)
svm.fit(X_train, y_train)

# Predictions
y_pred = svm.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print("\nClassification Report:\n",
classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test,
y_pred))
```