**Pandas → Colab, copy‑paste friendly (basic → advanced) + Matplotlib**

Each block is a separate Colab cell. Paste/run top-to-bottom. Everything is self-contained—no external files needed.

**0) Setup & imports**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

np.random.seed(42)

pd.set\_option("display.max\_rows", 10)

pd.set\_option("display.width", 120)

**1) Create Series & DataFrame**

s = pd.Series([10, 20, 30], index=["a","b","c"])

print("Series:\n", s)

df = pd.DataFrame({

"id": np.arange(1, 11),

"category": np.random.choice(["A","B","C"], size=10),

"subcat": np.random.choice(["X","Y"], size=10),

"value": np.random.randn(10).round(3)\*10 + 50,

"count": np.random.randint(1, 100, size=10),

"date": pd.date\_range("2024-01-01", periods=10, freq="D")

})

df.head()

**2) Inspect data quickly**

display(df.head())

display(df.tail(3))

display(df.sample(3, random\_state=0))

display(df.info())

display(df.describe(numeric\_only=True))

**3) Selecting data: columns, rows, masks**

# Columns

display(df["value"].head()) # Series

display(df[["category","value"]].head()) # DataFrame

# Integer-location vs label-location

display(df.iloc[0:3, :]) # first 3 rows, all cols

display(df.loc[0:3, ["id","value"]]) # rows by labels (inclusive end), selected cols

# Boolean masking

mask = (df["value"] > 50) & (df["count"] < 50)

display(df[mask])

# Query API (readable for complex conditions)

display(df.query("value > 50 and count < 50"))

**4) Assigning & creating new columns**

df2 = df.copy()

df2["value\_z"] = (df2["value"] - df2["value"].mean())/df2["value"].std()

df2["is\_big"] = (df2["value"] > df2["value"].median())

df2 = df2.assign(value\_sq = df2["value"]\*\*2)

display(df2.head())

# With .where to keep shape

df2["value\_pos"] = df2["value"].where(df2["value"] > 0, other=np.nan)

df2.head()

**5) Missing data basics**

df3 = df2.copy()

df3.loc[[2,5], "subcat"] = np.nan

display(df3.isna().sum())

df3["subcat\_filled"] = df3["subcat"].fillna("MISSING")

df3["value\_filled"] = df3["value"].fillna(df3["value"].median())

df3.head()

**6) String & datetime accessors**

# .str

df4 = df3.copy()

df4["cat\_lower"] = df4["category"].str.lower()

df4["cat\_flag"] = df4["category"].str.contains("A")

display(df4[["category","cat\_lower","cat\_flag"]].head())

# .dt

df4["year"] = df4["date"].dt.year

df4["dow"] = df4["date"].dt.day\_name()

df4[["date","year","dow"]].head()

**7) Sorting, ranking, deduping**

display(df4.sort\_values(["value","count"], ascending=[False, True]).head())

display(df4.nlargest(3, "value"))

display(df4.nsmallest(3, "count"))

# Duplicates example

dup = pd.concat([df4.iloc[:3], df4.iloc[:3], df4.iloc[3:6]], ignore\_index=True)

display(dup.duplicated().head(10))

display(dup.drop\_duplicates().head(10))

**8) GroupBy: agg, transform, filter**

g = df4.groupby(["category","subcat\_filled"], dropna=False)

agg = g.agg(

n=("id","count"),

mean\_value=("value","mean"),

sum\_count=("count","sum")

).reset\_index()

display(agg)

# Named aggregation + multiple funcs

agg2 = df4.groupby("category").agg(

value\_mean=("value","mean"),

value\_std=("value","std"),

count\_sum=("count","sum")

)

display(agg2)

# transform keeps shape (broadcasts group statistic)

df4["value\_centered\_by\_cat"] = df4["value"] - df4.groupby("category")["value"].transform("mean")

df4[["category","value","value\_centered\_by\_cat"]].head(10)

**9) Pivot tables & crosstabs**

pt = pd.pivot\_table(

df4,

index="category",

columns="subcat\_filled",

values="value",

aggfunc="mean",

margins=True

)

display(pt)

ct = pd.crosstab(df4["category"], df4["subcat\_filled"], margins=True, normalize="index")

display(ct.round(3))

**10) Reshaping: melt, stack/unstack, wide ↔ long**

wide = df4.pivot(index="id", columns="subcat\_filled", values="value")

display(wide.head())

long = wide.reset\_index().melt(id\_vars="id", var\_name="subcat", value\_name="value")

display(long.head())

# MultiIndex stack/unstack

stacked = wide.stack() # -> long Series with MultiIndex (id, subcat)

unstacked = stacked.unstack()

display(stacked.head())

display(unstacked.head())

**11) Joins / merges (one-to-one, one-to-many)**

# Build small lookup tables

cats = pd.DataFrame({

"category": ["A","B","C"],

"cat\_desc": ["Alpha", "Beta", "Gamma"]

})

subcats = pd.DataFrame({

"subcat\_filled": ["X","Y","MISSING"],

"sub\_desc": ["Ex", "Why", "Unknown"]

})

merged = df4.merge(cats, on="category", how="left").merge(subcats, on="subcat\_filled", how="left")

display(merged.head())

# Index-based join

lookup = cats.set\_index("category")

display(df4.join(lookup, on="category").head())

**12) Datetime index, resampling, rolling windows**

# Make a time series (daily) with noise

ts = pd.DataFrame({

"date": pd.date\_range("2024-01-01", periods=120, freq="D"),

"y": np.sin(np.linspace(0, 6\*np.pi, 120)) + np.random.randn(120)\*0.2

})

ts = ts.set\_index("date")

display(ts.head())

# Resample monthly mean

display(ts.resample("M").mean())

# Rolling mean (7-day)

ts["y\_roll7"] = ts["y"].rolling(7, min\_periods=1).mean()

ts.head(12)

**13) Categorical & memory tips**

df5 = df4.copy()

df5["category\_cat"] = df5["category"].astype("category")

df5["subcat\_cat"] = df5["subcat\_filled"].astype("category")

display(df5.dtypes)

# Convert numerics safely

df5["value\_num"] = pd.to\_numeric(df5["value"], errors="coerce")

df5.info()

**14) Apply, map, replace, np.select**

# map / replace for label encoding

map\_dict = {"A": 0, "B": 1, "C": 2}

df6 = df5.copy()

df6["cat\_code"] = df6["category"].map(map\_dict)

# apply row-wise (use sparingly; vectorize when possible)

df6["score"] = df6.apply(lambda r: r["value"]\*0.6 + r["count"]\*0.4, axis=1)

# np.select for multi-conditions

conds = [

df6["value"] >= df6["value"].quantile(0.66),

df6["value"] >= df6["value"].quantile(0.33)

]

choices = ["high", "mid"]

df6["value\_band"] = np.select(conds, choices, default="low")

df6[["category","value","count","value\_band","score"]].head(10)

**15) Avoid chained assignment (the SettingWithCopy gotcha)**

# BAD (may warn): df[df.value > 50]["flag"] = True

# GOOD:

df7 = df6.copy()

mask = df7["value"] > 50

df7.loc[mask, "flag"] = True

df7.loc[~mask, "flag"] = False

df7.head()

**16) Concatenate & append patterns**

part1 = df6.iloc[:5].copy()

part2 = df6.iloc[5:].copy()

combined = pd.concat([part1, part2], ignore\_index=True)

display(combined.equals(df6.reset\_index(drop=True)))

# Row-wise add with keys

combo\_keys = pd.concat({"top": part1, "bottom": part2})

combo\_keys.head(7)

**17) I/O basics: CSV & Parquet (in-memory demo)**

# Write CSV

df6.to\_csv("demo.csv", index=False)

# Read CSV

df\_csv = pd.read\_csv("demo.csv")

display(df\_csv.head())

# Parquet (fast + typed). In Colab, pyarrow is available.

df6.to\_parquet("demo.parquet", index=False)

df\_parq = pd.read\_parquet("demo.parquet")

display(df\_parq.head())

**18) Pair with Matplotlib (quick plots)**

# 1) Histogram of value

plt.figure()

plt.hist(df6["value"], bins=10)

plt.title("Histogram of value")

plt.xlabel("value"); plt.ylabel("freq")

plt.show()

# 2) Boxplot by category

plt.figure()

df6.boxplot(column="value", by="category")

plt.suptitle("")

plt.title("Value by Category")

plt.xlabel("category"); plt.ylabel("value")

plt.show()

# 3) Scatter: value vs count colored by category

plt.figure()

for cat, sub in df6.groupby("category"):

plt.scatter(sub["count"], sub["value"], label=cat)

plt.legend()

plt.title("Count vs Value by Category")

plt.xlabel("count"); plt.ylabel("value")

plt.show()

# 4) Time series + rolling mean

plt.figure()

ts["y"].plot(label="y")

ts["y\_roll7"].plot(label="rolling 7d")

plt.legend()

plt.title("Time Series with 7-day Rolling Mean")

plt.xlabel("date"); plt.ylabel("y")

plt.show()

**19) Bonus: Pivot heatmap (Matplotlib)**

heat = pd.pivot\_table(df6, index="category", columns="subcat\_filled", values="value", aggfunc="mean")

plt.figure()

plt.imshow(heat, aspect="auto")

plt.colorbar(label="mean value")

plt.yticks(range(len(heat.index)), heat.index)

plt.xticks(range(len(heat.columns)), heat.columns, rotation=45)

plt.title("Mean Value Heatmap by Category/Subcat")

plt.tight\_layout()

plt.show()