Pandas vs Tidyverse Cheatsheet (Complete with Solutions)

This guide compares common data wrangling operations in **pandas (Python)** and **tidyverse (R)** with detailed examples and **solved exercises**.

1. Reading and Inspecting Data

Task	pandas (Python)	tidyverse (R)
Read CSV	`pd.read_csv("file.csv")`	`read_csv("file.csv")`
Read Excel	`pd.read_excel("file.xlsx")`	`readxl::read_excel("file.xlsx")`
Quick look	`df.head(), df.info(), df.describe()`	`head(df), glimpse(df), summary(df)`

Exercise (Solved): Load `sales.csv`, show first 10, dtypes, summary; set missing to 0 on import.
Pandas

```python
import pandas as pd

df = pd.read_csv("sales.csv", na_values=["", "NA", "NaN"])
df = df.fillna(0)

print(df.head(10))
print(df.dtypes)
print(df.describe(include="all"))
```

Tidyverse

```r
library(readr); library(dplyr)

df <- read_csv("sales.csv", na = c("", "NA", "NaN")) %>%
  mutate(across(where(is.numeric), ~replace_na(.x, 0)))

head(df, 10)
glimpse(df)
summary(df)
```

2. Selecting Columns

Method	pandas	tidyverse
Select by name	`df[["col1","col2"]]`	`select(col1, col2)`
Range	`df.loc[:, "col1":"col3"]`	`select(col1:col3)`
By position	`df.iloc[:, 0:3]`	`select(1:3)`
Drop	`df.drop(["col1"], axis=1)`	`select(-col1)`
By pattern	`df.filter(regex="^col")`	`select(starts_with("col"))`

Exercise (Solved): Keep `address*`, drop names containing `temp`, then 1st 3 + last col.
Pandas

```python
addr = df.filter(regex="^address")
df_no_temp = df.drop(columns=df.filter(regex="temp").columns)
subset = pd.concat([df.iloc[:, 0:3], df.iloc[:, -1]], axis=1)
```

Tidyverse

```r
library(dplyr)
out <- df %>%
  select(starts_with("address")) %>%
  select(-contains("temp")) %>%
  select(1:3, last_col())
```

3. Filtering Rows

Method	pandas	tidyverse
Single	`df[df["age"]>30]`	`filter(df, age>30)`
Multiple	`(df["a"]>1)&(df["b"]=="X")`	`filter(df, a>1, b=="X")`
Query	`df.query("a>1 & b=='X'")`	—
By position	`df.iloc[0:10]`	`slice(1:10)`

Exercise (Solved): Complex filter + top 20 by salary.
Pandas

```
f = ((df["age"]>35) & (df["salary"]>50000)) | \
    ((df["department"]=="Engineering") & (df["years_experience"]>=3))
top20 = df.loc[f].sort_values("salary", ascending=False).head(20)
```

Tidyverse

```
top20 <- df %>%
  filter((age > 35 & salary > 50000) |
         (department == "Engineering" & years_experience >= 3)) %>%
  arrange(desc(salary)) %>%
  slice_head(n = 20)
```

4. Sorting/Arranging

Operation	pandas	tidyverse
Basic sort	`df.sort_values("col")`	`arrange(col)`
Desc	`df.sort_values("col",False)`	`arrange(desc(col))`
Multi	`df.sort_values(["c1","c2"] )`	`arrange(c1, c2)`
Mixed	`ascending=[True,False]`	`arrange(c1, desc(c2))`

Exercise (Solved): Sort by category↑, price↓, rating↓; top 5 per category.
Pandas

```
sorted_df = df.sort_values(["category","price","rating"],
                           ascending=[True, False, False])
top5_each = sorted_df.groupby("category", group_keys=False).head(5)
```

Tidyverse

```
library(dplyr)
top5_each <- df %>%
  arrange(category, desc(price), desc(rating)) %>%
  group_by(category) %>%
  slice_head(n = 5) %>%
  ungroup()
```

5. Handling Missing Values

Operation	pandas	tidyverse
Check	`df.isna().sum()`	`colSums(is.na(df))`
Drop	`df.dropna()`	`drop_na(df)`
Fill value	`df["c"].fillna(0)`	`mutate(c = replace_na(c, 0))`
FFill/BFill	`df["c"].ffill()/bfill()`	`tidyr::fill(df, c, .direction="down")`

Exercise (Solved): Time series: fill sales with prev day; price with category mean; drop rows where both quantity and price missing.

Pandas

```
df = df.sort_values("date")
df["sales"] = df["sales"].ffill()
df["price"] = df.groupby("category")["price"] \
                .transform(lambda s: s.fillna(s.mean()))
df = df.dropna(subset=["quantity","price"], how="all")
```

Tidyverse

```
library(dplyr); library(tidyr)
df <- df %>%
  arrange(date) %>%
  fill(sales, .direction = "down") %>%
  group_by(category) %>%
  mutate(price = if_else(is.na(price), mean(price, na.rm=TRUE), price)) %>%
  ungroup() %>%
  filter(!(is.na(quantity) & is.na(price)))
```

6. Creating New Columns

Operation	pandas	tidyverse
Calc	`df["bmi"]=df["w"]/df["h"]**2`	`mutate(bmi=w/h^2)`
If-else	`np.where(cond,a,b)`	`if_else(cond,a,b)`
Case	`np.select([...],[...])`	`case_when(...)`
Row-wise	`df.apply(f,axis=1)`	`rowwise()%>% mutate(...)`

Exercise (Solved): Risk score + DTI and flag.

Pandas

```
import numpy as np
high = (df["income"] < 30000) | (df["age"] > 65)
medium = (df["income"] < 50000) & (df["age"].between(25,65, inclusive="both"))
df["risk"] = np.select([high, medium], ["High","Medium"], default="Low")

df["dti"] = df["debt"] / df["income"].replace(0, np.nan)
df["high_dti"] = df["dti"] > 0.4
```

Tidyverse

```
library(dplyr)
df <- df %>%
  mutate(
    risk = case_when(
      income < 30000 | age > 65 ~ "High",
      income < 50000 & between(age, 25, 65) ~ "Medium",
      TRUE ~ "Low"
    ),
    dti = debt / if_else(income == 0, NA_real_, income),
    high_dti = dti > 0.4
  )
```

7. Renaming Columns

Operation	pandas	tidyverse

Single	`df.rename(columns={"old":"new"})`	`rename(new = old)`
Multiple	`df.rename(columns={...})`	`rename(...)`
All	`df.columns = [... ]`	`set_names(c(...))`

Exercise (Solved): Standardize names to snake_case; unify variants (e.g., firstName/first_name/fname).

Pandas

```
import re
def to_snake(s):
    s = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', s)
    s = re.sub('([a-z0-9])([A-Z])', r'\1_\2', s)
    s = s.replace(".", "_").replace("-", "_")
    s = re.sub(r'__+', '_', s)
    return s.lower()

df = df.rename(columns=lambda c: to_snake(c))
df = df.rename(columns={"fname":"first_name", "firstname":"first_name"})
```

Tidyverse

```
library(janitor); library(dplyr)
df <- df %>%
  janitor::clean_names() %>%
  rename(first_name = fname, first_name = firstname)
```

8. Grouping and Aggregation

Task	pandas	tidyverse
Mean per group	`groupby("dept")["salary"].mean()`	`group_by(dept)%>% summarise(mean(salary))`
Multi stats	`agg(["mean","sum"])`	`summarise(across(salary, list(mean=mean,sum=sum)))`
Rank in group	`groupby("id")["x"].rank()`	`group_by(id)%>% mutate(r=dense_rank(x))`

Exercise (Solved): For region/quarter: total sales, AOV, unique customers, and rank region by total within quarter.

Pandas

```
agg = (df.groupby(["region","quarter"])
         .agg(total_sales=("sales","sum"),
              orders=("order_id","nunique"),
              unique_customers=("customer_id","nunique"),
              avg_order_value=("sales","mean"))
         .reset_index())
agg["rank_in_qtr"] = agg.groupby("quarter")["total_sales"] \
                        .rank(method="dense", ascending=False).astype(int)
```

Tidyverse

```
library(dplyr)
agg <- df %>%
  group_by(region, quarter) %>%
  summarise(
    total_sales = sum(sales, na.rm=TRUE),
    orders = n_distinct(order_id),
    unique_customers = n_distinct(customer_id),
    avg_order_value = mean(sales, na.rm=TRUE),
    .groups = "drop"
  ) %>%
```

```r
  group_by(quarter) %>%
  mutate(rank_in_qtr = dense_rank(desc(total_sales))) %>%
  ungroup()
```

9. Joins

Type	pandas	tidyverse
Inner	`pd.merge(df1,df2,on="id")`	`inner_join(df1,df2,by="id")`
Left	`pd.merge(df1,df2,on="id",how="left")`	`left_join(df1,df2,by="id")`
Right	`how="right"`	`right_join(...)`
Full	`how="outer"`	`full_join(...)`
Semi/Anti	—	`semi_join()/anti_join()`

Exercise (Solved): Join customers→orders→products; customers never ordered; products never sold; revenue per customer category.

Pandas

```python
cust_orders = pd.merge(customers, orders, on="customer_id", how="left")
full = pd.merge(cust_orders, products, on="product_id", how="left")

# Customers with no orders (anti-join)
never_ordered = customers[~customers["customer_id"].isin(orders["customer_id"])
    ]

# Products never sold
sold_products = orders["product_id"].dropna().unique()
never_sold = products[~products["product_id"].isin(sold_products)]

# Revenue per customer category
full["revenue"] = full["quantity"] * full["price"]
rev_by_cat = (full.groupby("customer_category")["revenue"]
                  .sum().reset_index())
```

Tidyverse

```r
library(dplyr)
full <- customers %>%
  left_join(orders, by="customer_id") %>%
  left_join(products, by="product_id") %>%
  mutate(revenue = quantity * price)

never_ordered <- customers %>%
  anti_join(orders, by="customer_id")

never_sold <- products %>%
  anti_join(orders, by="product_id")

rev_by_cat <- full %>%
  group_by(customer_category) %>%
  summarise(revenue = sum(revenue, na.rm=TRUE), .groups="drop")
```

10. Reshaping Data

Operation	pandas	tidyverse
Wide→Long	`df.melt(...)`	`pivot_longer(...)`
Long→Wide	`df.pivot(...)`	`pivot_wider(...)`

Exercise (Solved): Wide (Q1..Q4) → long; compute YoY growth by region/quarter; pivot back to show growth by quarter and region.

Pandas

```
# columns: region, year, Q1, Q2, Q3, Q4
long = df.melt(id_vars=["region","year"],
               value_vars=["Q1","Q2","Q3","Q4"],
               var_name="quarter", value_name="sales")
long = long.sort_values(["region","quarter","year"])
long["yoy_growth"] = long.groupby(["region","quarter"])["sales"].pct_change()

growth = (long.pivot_table(index="region", columns="quarter",
                           values="yoy_growth", aggfunc="last")
               .reset_index())
```

Tidyverse

```
library(dplyr); library(tidyr)
long <- df %>%
  pivot_longer(cols = c(Q1,Q2,Q3,Q4),
               names_to = "quarter", values_to = "sales") %>%
  arrange(region, quarter, year) %>%
  group_by(region, quarter) %>%
  mutate(yoy_growth = (sales - lag(sales)) / lag(sales)) %>%
  ungroup()

growth <- long %>%
  group_by(region, quarter) %>%
  summarise(yoy_growth = dplyr::last(yoy_growth), .groups="drop") %>%
  pivot_wider(names_from = quarter, values_from = yoy_growth)
```

11. String Operations

Operation	pandas	tidyverse
Contains	`str.contains("a")`	`str_detect(col,"a")`
Replace	`str.replace("a","b")`	`str_replace(col,"a","b")`
Split	`str.split("-",expand=True)`	`separate(col, into=..., sep="-")`

Exercise (Solved): Clean addresses: extract ZIP, standardize state abbrev, split names, flag apartments.

Pandas

```
import re
# ZIP (US 5 digits)
df["zip"] = df["address"].str.extract(r'(\b\d{5}\b)')
state_map = {"Calif.":"CA", "California":"CA", "Tx":"TX"}
df["state"] = df["state"].replace(state_map).str.upper()

names = df["full_name"].str.split(r"\s+", n=1, expand=True)
df["first_name"] = names[0]; df["last_name"] = names[1]

df["has_apartment"] = df["address"].str.contains(
    r'\b(apt|apartment|unit|#)\b', case=False, na=False)
```

Tidyverse

```
library(dplyr); library(stringr); library(tidyr)
state_map <- c("Calif."="CA","California"="CA","Tx"="TX")

df <- df %>%
```

```
  mutate(
    zip = str_extract(address, "\\b\\d{5}\\b"),
    state = toupper(recode(state, !!!state_map)),
    has_apartment = str_detect(address, "\\b(apt|apartment|unit|#)\\b")
  ) %>%
  separate(full_name, into=c("first_name","last_name"), sep="\\s+", extra="
     merge", fill="right")
```

12. Duplicates and Sampling

Operation	pandas	tidyverse
Drop dups	`drop_duplicates()`	`distinct()`
Sample	`sample(n=100)`	`slice_sample(n=100)`
Top-n	`nlargest(10,"col")`	`slice_max(col,n=10)`

Exercise (Solved): Remove duplicates by email+phone, sample 20%, top 10% by purchase value.

Pandas

```
dedup = df.drop_duplicates(subset=["email","phone"])

sample20 = dedup.sample(frac=0.2, random_state=42)

total = (orders.groupby("customer_id")["purchase_value"].sum()
           .reset_index(name="total_purchase"))
q90 = total["total_purchase"].quantile(0.90)
top10pct = total[total["total_purchase"] >= q90]
```

Tidyverse

```
library(dplyr)
dedup <- df %>% distinct(email, phone, .keep_all = TRUE)

sample20 <- dedup %>% slice_sample(prop = 0.2)

total <- orders %>%
  group_by(customer_id) %>%
  summarise(total_purchase = sum(purchase_value, na.rm=TRUE), .groups="drop")

q90 <- quantile(total$total_purchase, 0.90, na.rm=TRUE)
top10pct <- total %>% filter(total_purchase >= q90)
```

13. Concatenating/Binding

Operation	pandas	tidyverse
Rows	`pd.concat([df1,df2])`	`bind_rows(df1,df2)`
Cols	`pd.concat([df1,df2],axis=1)`	`bind_cols(df1,df2)`

Exercise (Solved): Combine monthly files by region, add source id, compute YTD by region and category.

Pandas

```
import glob, os
files = glob.glob("data/region_*_2025-*.csv")
dfs = []
for f in files:
    d = pd.read_csv(f)
    d["source"] = os.path.basename(f)
    dfs.append(d)
master = pd.concat(dfs, ignore_index=True)
```

```
ytd = (master.groupby(["region","product_category"])["sales"]
             .sum().reset_index(name="ytd_sales"))
```

Tidyverse

```
library(readr); library(dplyr); library(purrr); library(stringr)

files <- Sys.glob("data/region_*_2025-*.csv")
master <- files %>%
  set_names(basename) %>%
  map_dfr(~ read_csv(.x), .id = "source")

ytd <- master %>%
  group_by(region, product_category) %>%
  summarise(ytd_sales = sum(sales, na.rm=TRUE), .groups="drop")
```

14. Date/Time Operations

Operation	pandas	tidyverse
Parse	`pd.to_datetime(...)`	`lubridate::ymd(...)`
Parts	`dt.year / .month`	`year(), month()`
Diff	`(d2-d1).dt.days`	`as.numeric(d2-d1)`

Exercise (Solved): Lifetime (first→last), quarters, inactive ¿90 days.
Pandas

```
import pandas as pd
orders["date"] = pd.to_datetime(orders["date"])
cust = (orders.groupby("customer_id")["date"]
              .agg(first="min", last="max", recent="max").reset_index())
cust["lifetime_days"] = (cust["last"] - cust["first"]).dt.days
cust["quarter"] = cust["last"].dt.to_period("Q").astype(str)

ref = pd.Timestamp.today().normalize()
inactive = cust[(ref - cust["recent"]).dt.days > 90]
```

Tidyverse

```
library(dplyr); library(lubridate)
orders <- orders %>% mutate(date = ymd(date))

cust <- orders %>%
  group_by(customer_id) %>%
  summarise(first = min(date),
            last  = max(date),
            recent = max(date), .groups="drop") %>%
  mutate(lifetime_days = as.numeric(last - first),
         quarter = paste0(year(last), "␣Q", quarter(last)))

ref <- Sys.Date()
inactive <- cust %>% filter(as.numeric(ref - recent) > 90)
```

15. Applying Custom Functions

Method	pandas	tidyverse
Column-wise	`apply/transform`	`mutate/map_*`
Row-wise	`df.apply(f,axis=1)`	`rowwise()`

Group apply	`groupby().apply(f)`	`group_modify()`

Exercise (Solved): RFM scoring + segment ranking.
Pandas

```
import numpy as np, pandas as pd
orders["date"] = pd.to_datetime(orders["date"])
snapshot = orders["date"].max()

rfm = (orders.groupby("customer_id")
         .agg(Recency=("date", lambda s: (snapshot - s.max()).days),
              Frequency=("order_id","nunique"),
              Monetary=("amount","sum"))
         .reset_index())

rfm["R_Score"] = pd.qcut(rfm["Recency"], 5, labels=[5,4,3,2,1]).astype(int)
rfm["F_Score"] = pd.qcut(rfm["Frequency"].rank(method="first"), 5, labels
    =[1,2,3,4,5]).astype(int)
rfm["M_Score"] = pd.qcut(rfm["Monetary"].rank(method="first"), 5, labels
    =[1,2,3,4,5]).astype(int)
rfm["RFM_Score"] = rfm[["R_Score","F_Score","M_Score"]].sum(axis=1)
rfm["segment"] = np.where(rfm["RFM_Score"]>=12, "Gold",
                   np.where(rfm["RFM_Score"]>=9, "Silver","Bronze"))

rfm = rfm.sort_values(["segment","RFM_Score"], ascending=[True,False])
```

Tidyverse

```
library(dplyr)
snapshot <- max(orders$date)

rfm <- orders %>%
  group_by(customer_id) %>%
  summarise(
    Recency = as.numeric(snapshot - max(date)),
    Frequency = n_distinct(order_id),
    Monetary = sum(amount, na.rm=TRUE), .groups="drop"
  ) %>%
  mutate(
    R_Score = ntile(desc(Recency), 5),  # smaller Recency => higher score
    F_Score = ntile(Frequency, 5),
    M_Score = ntile(Monetary, 5),
    RFM_Score = R_Score + F_Score + M_Score,
    segment = case_when(
      RFM_Score >= 12 ~ "Gold",
      RFM_Score >= 9  ~ "Silver",
      TRUE ~ "Bronze"
    )
  ) %>%
  arrange(segment, desc(RFM_Score))
```

16. Data Type Conversions

Operation	pandas	tidyverse
To numeric	`pd.to_numeric(...,errors="coerce")`	`as.numeric(...)`
To datetime	`pd.to_datetime(...,errors="coerce")`	`lubridate::parse_date_time()`
Categorical	`astype("category")`	`factor()`

Exercise (Solved): Fix numeric-as-strings; parse mixed date formats; optimize with categories.
Pandas

```
num_cols = ["qty","price","amount"]
for c in num_cols:
    df[c] = pd.to_numeric(df[c], errors="coerce")

df["date"] = pd.to_datetime(df["date"], errors="coerce", format=None)
mask = df["date"].isna()
df.loc[mask, "date"] = pd.to_datetime(df.loc[mask, "date_str_alt"], errors="
    coerce")

low_card = [c for c in df.columns if df[c].dtype=="object" and df[c].nunique()
    < 0.2*len(df)]
for c in low_card:
    df[c] = df[c].astype("category")
```

Tidyverse

```
library(dplyr); library(lubridate)
df <- df %>%
  mutate(across(c(qty, price, amount), ~as.numeric(as.character(.x))))

df <- df %>%
  mutate(date = suppressWarnings(parse_date_time(date, orders = c("ymd","dmy","
      mdy"))))

low_card <- names(df)[sapply(df, \(x) is.character(x) && dplyr::n_distinct(x) <
    0.2*nrow(df))]
df[low_card] <- lapply(df[low_card], factor)
```

17. Index Operations

Operation	pandas	tidyverse
Reset	`reset_index()`	`tibble::rownames_to_column()`
Set	`set_index("col")`	`tibble::column_to_rownames("col")`

Exercise (Solved): Create MultiIndex (date, product_id), reset for analysis, then restore and resample.

Pandas

```
sales["date"] = pd.to_datetime(sales["date"])
mi = sales.set_index(["date","product_id"]).sort_index()

flat = mi.reset_index()  # analysis

# restore and resample to monthly per product
monthly = (flat.set_index(["date","product_id"])
               .groupby(level="product_id")
               .apply(lambda g: g.reset_index(level=1, drop=True)
                                 .resample("M")["units"].sum()))
```

Tidyverse

```
# Tidyverse doesn't use row indexes the same way; keep explicit cols
library(dplyr); library(lubridate)
sales <- sales %>% mutate(date = ymd(date)) %>% arrange(date, product_id)

monthly <- sales %>%
  group_by(product_id, month = floor_date(date, "month")) %>%
  summarise(units = sum(units, na.rm=TRUE), .groups="drop")
```

18. Categorical Data

Operation	pandas	tidyverse
Create	`pd.Categorical(...)`	`factor(...)`
Order	`Categorical(..., ordered=True)`	`factor(..., ordered=TRUE)`
Recode	`cat.rename_categories`	`forcats::fct_recode`

Exercise (Solved): Ordered factors (Poor¡Fair¡Good¡Excellent), reorder product categories by avg rating, recode inconsistent names.

Pandas

```
order = ["Poor","Fair","Good","Excellent"]
df["satisfaction"] = pd.Categorical(df["satisfaction"], categories=order,
    ordered=True)

avg = df.groupby("product_category")["rating"].mean()
ordered_cats = avg.sort_values().index
df["product_category"] = pd.Categorical(df["product_category"], categories=
    ordered_cats, ordered=True)

df["product_category"] = df["product_category"].replace({"elec":"Electronics",
    "Elec":"Electronics"})
```

Tidyverse

```
library(dplyr); library(forcats)
df <- df %>%
  mutate(
    satisfaction = factor(satisfaction, levels=c("Poor","Fair","Good","
        Excellent"), ordered=TRUE),
    product_category = fct_recode(product_category, Electronics="elec",
        Electronics="Elec")
  )

cat_order <- df %>% group_by(product_category) %>%
  summarise(avg = mean(rating, na.rm=TRUE), .groups="drop") %>%
  arrange(avg) %>% pull(product_category)

df$product_category <- fct_relevel(df$product_category, cat_order)
```

19. Value Counts and Frequency Tables

Operation	pandas	tidyverse
Counts	`value_counts()`	`count(col, sort=TRUE)`
Proportions	`value_counts(normalize=True)`	`count(col)%>% mutate(prop=n/sum(n))`
Crosstab	`pd.crosstab(c1,c2)`	`janitor::tabyl(c1,c2)`

Exercise (Solved): Purchase freq by segment; payment vs customer type; popular product pairs.

Pandas

```
freq = customers["segment"].value_counts(normalize=False).reset_index(name="n")

cros = pd.crosstab(orders["payment_method"], orders["customer_type"])

# product pairs within an order
from itertools import combinations
pairs = (orders.groupby("order_id")["product_id"].apply(lambda s: list(set(s)))
    )
rows = []
for prods in pairs:
```

```
    for a,b in combinations(sorted(prods), 2):
        rows.append((a,b))
pairs_df = pd.DataFrame(rows, columns=["prod_a","prod_b"])
popular_pairs = pairs_df.value_counts().reset_index(name="count").sort_values("
    count", ascending=False)
```

Tidyverse

```
library(dplyr); library(tidyr); library(purrr); library(janitor)
freq <- customers %>% count(segment, sort = TRUE)

cros <- janitor::tabyl(orders$payment_method, orders$customer_type)

pairs <- orders %>%
  group_by(order_id) %>%
  summarise(items = list(unique(product_id)), .groups="drop") %>%
  mutate(pairs = map(items, ~ t(combn(sort(.x), 2)) %>% as.data.frame())) %>%
  select(pairs) %>%
  unnest(pairs) %>%
  rename(prod_a = V1, prod_b = V2) %>%
  count(prod_a, prod_b, sort = TRUE)
```

20. Row/Column Summary Operations

Operation	pandas	tidyverse
Row sums	`df.sum(axis=1)`	`rowSums(across(where(is.numeric)))`
Row means	`df.mean(axis=1)`	`rowMeans(across(where(is.numeric)))`

Exercise (Solved): Portfolio value, diversification count, incomplete accounts.

Pandas

```
hold_cols = ["asset_a","asset_b","asset_c","asset_d"]
df["portfolio_value"] = df[hold_cols].sum(axis=1)
df["diversification"] = (df[hold_cols] != 0).sum(axis=1)
incomplete = df[df[hold_cols].isna().any(axis=1)]
```

Tidyverse

```
library(dplyr)
hold_cols <- c("asset_a","asset_b","asset_c","asset_d")
df <- df %>%
  mutate(
    portfolio_value = rowSums(across(all_of(hold_cols)), na.rm=TRUE),
    diversification = rowSums(across(all_of(hold_cols), ~.x != 0))
  )
incomplete <- df %>% filter(if_any(all_of(hold_cols), is.na))
```

21. Advanced Text Operations

Operation	pandas	tidyverse
Count patt.	`str.count("pat")`	`str_count(col,"pat")`
Find all	`str.findall("pat")`	`str_extract_all(col,"pat")`

Exercise (Solved): Feedback: count sentiment words, extract emails/phones, normalize product mentions, length categories.

Pandas

```
pos_words = ["good","great","excellent","love"]
neg_words = ["bad","poor","terrible","hate"]

df["pos_count"] = df["feedback"].str.lower().apply(
    lambda s: sum(s.count(w) for w in pos_words))
df["neg_count"] = df["feedback"].str.lower().apply(
    lambda s: sum(s.count(w) for w in neg_words))

df["emails"] = df["feedback"].str.findall(r'[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[
    A-Za-z]{2,}')
df["phones"] = df["feedback"].str.findall(r'\+?\d[\d\s\-]{7,}\d')

# Standardize product names (e.g., iPhone 12 -> IPHONE12)
df["feedback_std"] = df["feedback"].str.replace(r'iPhone\s*12','IPHONE12',
    regex=True)

df["text_len"] = df["feedback"].str.len()
df["len_bucket"] = pd.cut(df["text_len"], bins=[0,50,150,1e9],
                          labels=["short","medium","long"])
```

Tidyverse

```
library(dplyr); library(stringr); library(purrr)
pos_words <- c("good","great","excellent","love")
neg_words <- c("bad","poor","terrible","hate")

count_words <- function(txt, words) sum(map_int(words, ~ str_count(str_to_lower
    (txt), .x)))
df <- df %>%
  mutate(
    pos_count = map_int(feedback, ~ count_words(.x, pos_words)),
    neg_count = map_int(feedback, ~ count_words(.x, neg_words)),
    emails = str_extract_all(feedback, "[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-
        Za-z]{2,}"),
    phones = str_extract_all(feedback, "\\+?\\d[\\d\\s\\-]{7,}\\d"),
    feedback_std = str_replace_all(feedback, regex("iPhone\\s*12", ignore_case=
        TRUE), "IPHONE12"),
    text_len = str_length(feedback),
    len_bucket = cut(text_len, breaks=c(0,50,150,Inf), labels=c("short","medium
        ","long"))
  )
```

22. Conditional Operations

Operation	pandas	tidyverse
Mask/Where	`mask()/where()`	`if_else()/case_when()`
Clip	`clip(lower,upper)`	`pmax/pmin`

Exercise (Solved): Cap at 99th pct; negative revenues to 0; mask junior salaries.
Pandas

```
p99 = df["value"].quantile(0.99)
df["value_capped"] = df["value"].clip(upper=p99)

df["revenue"] = df["revenue"].clip(lower=0)

df["salary_masked"] = df["salary"].where(df["level"]!="Junior", other=pd.NA)
```

Tidyverse

```r
library(dplyr)
p99 <- quantile(df$value, 0.99, na.rm=TRUE)
df <- df %>%
  mutate(
    value_capped = pmin(value, p99),
    revenue = pmax(revenue, 0),
    salary_masked = if_else(level == "Junior", NA_real_, salary)
  )
```

23. Exporting Data

Format	pandas	tidyverse
CSV	`to_csv("out.csv", index=False)`	`readr::write_csv(df,"out.csv")`
Excel	`to_excel("out.xlsx", index=False)`	`writexl::write_xlsx(df,"out.xlsx")`
Parquet	`to_parquet("out.parquet")`	`arrow::write_parquet(df,"out.parquet")`

Exercise (Solved): Export CSV; Excel by department (sheets); Parquet.
Pandas

```python
df.to_csv("processed.csv", index=False)

with pd.ExcelWriter("by_department.xlsx") as xw:
    for dept, sub in df.groupby("department"):
        sub.to_excel(xw, sheet_name=str(dept)[:31], index=False)

df.to_parquet("processed.parquet")
```

Tidyverse

```r
library(readr); library(writexl); library(dplyr); library(arrow)

write_csv(df, "processed.csv")

sheets <- df %>% group_split(department) %>%
  setNames(df %>% distinct(department) %>% pull(department))
write_xlsx(sheets, "by_department.xlsx")

arrow::write_parquet(df, "processed.parquet")
```