```
from pathlib import Path
import pandas as pd
import numpy as np
from IPython.display import display
```

```
# Data import
DATA_DIR = Path("/content")
OUT_DIR = Path("./w2_out")
OUT_DIR.mkdir(parents=True, exist_ok=True)
```

```
# raw dataset Overlook
def try_infer_date_column(df):
    """Guess a date column from common names or by parsing"""
    candidates = ["date", "Date", "DATE", "observation_date", "Ex-Date", "ex_date"]
    for c in candidates:
        if c in df.columns:
            return c
    # fallback: try parsing each column
    for c in df.columns:
        try:
            parsed = pd.to_datetime(df[c].head(30), errors="coerce")
            if parsed.notna().mean() > 0.6:
                return c
        except Exception:
            pass
    return None

csv_files = sorted([p for p in DATA_DIR.glob("*.csv")])
if not csv_files:
    print("[WARN] No CSV files found in /content. Please upload your data files.")
else:
    overview_rows = []
    for path in csv_files:
        df = pd.read_csv(path)

        # basic info
        nrows, ncols = df.shape
        dcol = try_infer_date_column(df)

        # check date range
        date_min, date_max = None, None
        if dcol is not None:
            dt = pd.to_datetime(df[dcol], errors="coerce")
            if dt.notna().any():
                date_min = str(dt.min().date())
                date_max = str(dt.max().date())

        num_cols = df.select_dtypes(include=[np.number]).shape[1]
        non_num_cols = ncols - num_cols

        # save a small sample for reference
        df.head(10).to_csv(OUT_DIR / f"sample__{path.stem}.csv", index=False)

        overview_rows.append({
            "filename": path.name,
            "rows": nrows,
            "cols": ncols,
            "date_col": dcol,
            "date_min": date_min,
            "date_max": date_max,
            "numeric_cols": num_cols,
            "non_numeric_cols": non_num_cols,
            "columns_preview": ", ".join(map(str, df.columns[:6]))
        })

        print(f"[CHECK] {path.name}: shape={df.shape}, "
              f"date_col={dcol}, date_range=({date_min}, {date_max})")

    # build overview table
    overview = pd.DataFrame(overview_rows).sort_values("filename").reset_index(drop=True)
    overview.to_csv(OUT_DIR / "files_overview.csv", index=False)

    # show as a table in notebook
```

```
display(overview.head())
print("\nOverview saved to:", OUT_DIR / "files_overview.csv")
```

```
[CHECK] bxp_dividends.csv: shape=(55, 3), date_col=Ex-Date, date_range=(2014-12-29, 2025-06-30)
[CHECK] bxp_prices.csv: shape=(247, 6), date_col=Date, date_range=(2005-02-28, 2025-08-31)
[CHECK] dgs10_m_20250826.csv: shape=(764, 3), date_col=date, date_range=(1962-01-31, 2025-08-31)
[CHECK] elme_dividends.csv: shape=(55, 3), date_col=Ex-Date, date_range=(2015-01-01, 2025-09-17)
[CHECK] elme_prices.csv: shape=(247, 6), date_col=Date, date_range=(2005-02-28, 2025-08-31)
[CHECK] eqr_dividends.csv: shape=(55, 3), date_col=Ex-Date, date_range=(2014-12-23, 2025-06-24)
[CHECK] eqr_prices.csv: shape=(385, 6), date_col=Date, date_range=(1993-08-31, 2025-08-31)
[CHECK] la_cpi_Washington.csv: shape=(1327, 2), date_col=observation_date, date_range=(1914-12-01, 2025-06-01)
[CHECK] la_cpi_boston.csv: shape=(1328, 2), date_col=observation_date, date_range=(1914-12-01, 2025-07-01)
[CHECK] la_cpi_chicago.csv: shape=(1328, 2), date_col=observation_date, date_range=(1914-12-01, 2025-07-01)
[CHECK] la_cpi_los_angeles.csv: shape=(494, 2), date_col=date, date_range=(1980-01-01, 2017-12-01)
[CHECK] la_cpi_miami.csv: shape=(572, 2), date_col=observation_date, date_range=(1977-11-01, 2025-06-01)
[CHECK] la_cpi_ny.csv: shape=(1328, 2), date_col=observation_date, date_range=(1914-12-01, 2025-07-01)
[CHECK] rexr_dividends.csv: shape=(55, 3), date_col=Ex-Date, date_range=(2014-12-29, 2025-09-30)
[CHECK] rexr_prices.csv: shape=(146, 6), date_col=Date, date_range=(2013-07-31, 2025-08-31)
[CHECK] slg_dividends.csv: shape=(98, 3), date_col=Ex-Date, date_range=(2014-12-30, 2025-08-29)
[CHECK] slg_prices.csv: shape=(247, 6), date_col=Date, date_range=(2005-02-28, 2025-08-31)
[CHECK] trno_dividends.csv: shape=(55, 3), date_col=Ex-Date, date_range=(2014-12-29, 2025-09-29)
[CHECK] trno_prices.csv: shape=(187, 6), date_col=Date, date_range=(2010-02-28, 2025-08-31)
[CHECK] unemployment_msa_m_20250826.csv: shape=(2550, 4), date_col=date, date_range=(1990-01-01, 2025-06-01)
```

	filename	rows	cols	date_col	date_min	date_max	numeric_cols	non_numeric_cols	columns_previ
0	bxp_dividends.csv	55	3	Ex-Date	2014-12-29	2025-06-30	1	2	Ex-Date, Pay Da Amou
1	bxp_prices.csv	247	6	Date	2005-02-28	2025-08-31	5	1	Date, Open, Hig Low, Clo Volur
2	dgs10_m_20250826.csv	764	3	date	1962-01-31	2025-08-31	1	2	date, series_ val
3	elme_dividends.csv	55	3	Ex-Date	2015-01-01	2025-09-17	1	2	Ex-Date, Pay Da Amou
4	elme_prices.csv	247	6	Date	2005-02-28	2025-08-31	5	1	Date, Open, Hig Low, Clo Volur

```
Overview saved to: w2_out/files_overview.csv
```

```
# Data basic standardization and simple engineer features
#covert data to monthly
def to_month_end(s):
    """Convert date to month-end date"""
    dt = pd.to_datetime(s, errors="coerce")
    return (dt + pd.offsets.MonthEnd(0)).dt.normalize()

#calculate the fluctuation of month to month
def level_diff(s, periods=1):
    """Month-to-month difference"""
    return s.diff(periods=periods)

#take the last price in a month
def mon_agg_last(x):
    """Get the last available value in a month"""
    return x.dropna().iloc[-1] if x.notna().any() else np.nan

#convert the stock price to monthly
def month_end_close(price_df, date_col="Date", close_col="Close"):
    tmp = price_df.copy()
    tmp[date_col] = to_month_end(tmp[date_col])
    tmp[close_col] = pd.to_numeric(tmp[close_col], errors="coerce")
    out = (tmp.groupby(date_col, as_index=False)[close_col]
                .agg(mon_agg_last)
                .rename(columns={date_col: "date", close_col: "adj_price"}))
    return out
#claculate monthly dividents
def monthly_sum_by_exdate(div_df, date_col="Ex-Date", amt_col="Amount"):
    tmp = div_df.copy()
    tmp[date_col] = to_month_end(tmp[date_col])
    tmp[amt_col] = pd.to_numeric(tmp[amt_col], errors="coerce")
```

```
        out = (tmp.groupby(date_col, as_index=False)[amt_col]
                    .sum()
                    .rename(columns={date_col: "date", amt_col: "dividend"}))
        return out
# divident of a year
def compute_ttm_dividend(div_monthly):
        s = div_monthly.sort_values("date")["dividend"].fillna(0.0)
        return s.rolling(window=12, min_periods=1).sum()
```

```
# map the company and city
COMPANY_TO_METRO = {
        "BXP": "Boston",
        "SLG": "New York",
        "ELME": "Washington",
        "EQR": "Chicago",
        "REXR": "Los Angeles",
        "TRNO": "Miami",
}

# the dividents and price of company
COMPANY_FILES = {
        "BXP": ("bxp_prices.csv", "bxp_dividends.csv"),
        "ELME": ("elme_prices.csv", "elme_dividends.csv"),
        "EQR": ("eqr_prices.csv", "eqr_dividends.csv"),
        "REXR": ("rexr_prices.csv", "rexr_dividends.csv"),
        "SLG": ("slg_prices.csv", "slg_dividends.csv"),
        "TRNO": ("trno_prices.csv", "trno_dividends.csv"),
}
```

```
#Data cleaning
#First aspect --- Firms' data (price and dividents)
company_tables = []
for tic, (price_file, div_file) in COMPANY_FILES.items():
    #read the raw data
        px_raw = pd.read_csv(DATA_DIR / price_file)
        dv_raw = pd.read_csv(DATA_DIR / div_file)
        print(f"[{tic}] raw shapes -> prices={px_raw.shape}, dividends={dv_raw.shape}")

        # git rid of same rows
        px = px_raw.drop_duplicates().copy()
        dv = dv_raw.drop_duplicates().copy()
        # standardize time
        px["Date"] = pd.to_datetime(px["Date"], errors="coerce")
        dv["Ex-Date"] = pd.to_datetime(dv["Ex-Date"], errors="coerce")
        #transfer numbers to folat and non numbers to NAN
        px["Close"] = pd.to_numeric(px["Close"], errors="coerce")
        dv["Amount"] = pd.to_numeric(dv["Amount"], errors="coerce")
        #Get rid of NANs
        px = px[px["Date"].notna() & px["Close"].notna()]
        dv = dv[dv["Ex-Date"].notna() & dv["Amount"].notna()]

        # monthly level
        px_m = month_end_close(px)
        dv_m = monthly_sum_by_exdate(dv)

        #take the same time period
        if not dv_m.empty and not px_m.empty:
          min_date = max(px_m["date"].min(), dv_m["date"].min())
          max_date = min(px_m["date"].max(), dv_m["date"].max())
          px_m = px_m[(px_m["date"] >= min_date) & (px_m["date"] <= max_date)]
          dv_m = dv_m[(dv_m["date"] >= min_date) & (dv_m["date"] <= max_date)]

        # merge and claculate TTM
        cur = (pd.merge(px_m, dv_m, on="date", how="left")
                    .sort_values("date")
                    .assign(dividend=lambda d: d["dividend"].fillna(0.0)))
        cur["dividend_ttm"] = compute_ttm_dividend(cur)
        cur["company"] = tic

        print(f"[{tic}] monthly rows={cur.shape[0]}, "
              f"range=({cur['date'].min().date()}, {cur['date'].max().date()})")

        company_tables.append(cur[["date","company","adj_price","dividend","dividend_ttm"]])
```

```
all_companies = pd.concat(company_tables, ignore_index=True)
all_companies.to_csv(OUT_DIR / "step2_company_monthly_all.csv", index=False)
print("[all_companies] shape:", all_companies.shape)
display(all_companies.head(20))
```

```
[BXP] raw shapes -> prices=(247, 6), dividends=(55, 3)
[BXP] monthly rows=127, range=(2014-12-31, 2025-06-30)
[ELME] raw shapes -> prices=(247, 6), dividends=(55, 3)
[ELME] monthly rows=128, range=(2015-01-31, 2025-08-31)
[EQR] raw shapes -> prices=(385, 6), dividends=(55, 3)
[EQR] monthly rows=127, range=(2014-12-31, 2025-06-30)
[REXR] raw shapes -> prices=(146, 6), dividends=(55, 3)
[REXR] monthly rows=129, range=(2014-12-31, 2025-08-31)
[SLG] raw shapes -> prices=(247, 6), dividends=(98, 3)
[SLG] monthly rows=129, range=(2014-12-31, 2025-08-31)
[TRNO] raw shapes -> prices=(187, 6), dividends=(55, 3)
[TRNO] monthly rows=129, range=(2014-12-31, 2025-08-31)
[all_companies] shape: (769, 5)
```

	date	company	adj_price	dividend	dividend_ttm
0	2014-12-31	BXP	100.7570	5.80	5.80
1	2015-01-31	BXP	108.6960	7.75	13.55
2	2015-02-28	BXP	107.6030	0.00	13.55
3	2015-03-31	BXP	110.5230	0.65	14.20
4	2015-04-30	BXP	104.0960	0.00	14.20
5	2015-05-31	BXP	102.2990	0.00	14.20
6	2015-06-30	BXP	95.7034	0.65	14.85
7	2015-07-31	BXP	97.5042	0.00	14.85
8	2015-08-31	BXP	89.6563	0.00	14.85
9	2015-09-30	BXP	94.1603	0.65	15.50
10	2015-10-31	BXP	100.0920	0.00	15.50
11	2015-11-30	BXP	99.3991	0.00	15.50
12	2015-12-31	BXP	102.9440	1.90	11.60
13	2016-01-31	BXP	93.8176	3.85	7.70
14	2016-02-29	BXP	92.1475	0.00	7.70
15	2016-03-31	BXP	103.1100	0.65	7.70
16	2016-04-30	BXP	104.5650	0.00	7.70
17	2016-05-31	BXP	101.8890	0.00	7.70
18	2016-06-30	BXP	107.5730	0.65	7.70
19	2016-07-31	BXP	115.9130	0.00	7.70

```
#Data cleaning
#Second aspect --- CPI and umemployment
#CPI
cpi_tables = []
for city, (fname, _, __) in CPI_FILES.items():
        path = DATA_DIR / fname
        if not path.exists():
                print(f"[WARN] Missing CPI file for {city}")
                continue

        df = pd.read_csv(path)
        if "date" in df.columns:
                dcol = "date"
        elif "DATE" in df.columns:
                dcol = "DATE"
        elif "observation_date" in df.columns:
                dcol = "observation_date"
        else:
                dcol = df.columns[0]
```

```
        vcol = "value" if "value" in df.columns else df.columns[-1]

        # change the names of data
        df = df[[dcol, vcol]].rename(columns={dcol:"date", vcol:"cpi"})
        # transfer date and number
        df["date"] = to_month_end(df["date"])
        df["cpi"]   = pd.to_numeric(df["cpi"], errors="coerce")
        # git rid of NAN
        df = df[df["date"].notna() & df["cpi"].notna()].drop_duplicates()
        # sequence by time
        df = df.sort_values("date")
        # month to month and year to year
        df["cpi_yoy"] = df["cpi"]/df["cpi"].shift(12) - 1
        df["cpi_mom"] = df["cpi"]/df["cpi"].shift(1)   - 1
        df["metro"]     = city

        cpi_tables.append(df[["date","metro","cpi","cpi_yoy","cpi_mom"]])

# combine all cpi to one frame
cpi_all = pd.concat(cpi_tables, ignore_index=True) if cpi_tables else pd.DataFrame()
cpi_all.to_csv(OUT_DIR / "step3_cpi_all.csv", index=False)

print("CPI total shape:", cpi_all.shape)
display(cpi_all.head(10))

#Unemployment
unemp_path = DATA_DIR / "unemployment_msa_m_20250826.csv"
if not unemp_path.exists():
        print("[WARN] Unemployment file missing")
        unemp = pd.DataFrame(columns=["metro","date","unemp","unemp_mom"])
else:
        df = pd.read_csv(unemp_path)

        # standardize date and time
        df["date"]   = to_month_end(df["date"])
        df["value"] = pd.to_numeric(df["value"], errors="coerce")
        df = df[df["date"].notna() & df["value"].notna()].drop_duplicates()

        # metro data to the city
        df["metro"] = None
        for city in set(COMPANY_TO_METRO.values()):
                df.loc[df["city"].str.contains(city, na=False), "metro"] = city
        df = df.dropna(subset=["metro"])

        # take average to month
        unemp = (df.groupby(["metro","date"], as_index=False)["value"]
                        .mean()
                        .rename(columns={"value":"unemp"}))
        # month to month and year to year data
        unemp = unemp.sort_values(["metro","date"])
        unemp["unemp_mom"] = unemp.groupby("metro")["unemp"].transform(level_diff)

unemp.to_csv(OUT_DIR / "step4_unemployment_all.csv", index=False)

print("Unemployment shape:", unemp.shape)
display(unemp.head(10))
```

```
CPI total shape: (4004, 5)
```

	date	metro	cpi	cpi_yoy	cpi_mom
0	1914-12-31	Boston	10.5	NaN	NaN
1	1915-12-31	Boston	10.7	NaN	0.019048
2	1916-12-31	Boston	12.1	NaN	0.130841
3	1917-12-31	Boston	14.2	NaN	0.173554
4	1918-12-31	Boston	17.3	NaN	0.218310
5	1919-06-30	Boston	17.4	NaN	0.005780
6	1919-12-31	Boston	19.5	NaN	0.120690
7	1920-06-30	Boston	21.5	NaN	0.102564
8	1920-12-31	Boston	20.2	NaN	-0.060465
9	1921-05-31	Boston	18.0	NaN	-0.108911

```
Unemployment shape: (2550, 4)
```

	metro	date	unemp	unemp_mom
0	Boston	1990-01-31	5.3	NaN
1	Boston	1990-02-28	5.2	-0.1
2	Boston	1990-03-31	5.3	0.1
3	Boston	1990-04-30	5.4	0.1
4	Boston	1990-05-31	5.5	0.1
5	Boston	1990-06-30	5.7	0.2
6	Boston	1990-07-31	5.9	0.2
7	Boston	1990-08-31	6.1	0.2
8	Boston	1990-09-30	6.3	0.2
9	Boston	1990-10-31	6.5	0.2

```
#Data cleaning
#Third aspect --- 10Y Treasury
ust10_path = DATA_DIR / "dgs10_m_20250826.csv"
if not ust10_path.exists():
        warnings.warn("10Y file missing: dgs10_m_20250826.csv")
        ust10 = pd.DataFrame(columns=["date","ten_year","ten_year_mom"])
else:
        ust10 = pd.read_csv(ust10_path)
        # sequence as the date
        ust10["date"] = to_month_end(ust10["date"])
        ust10 = ust10.sort_values("date")
        # calculate month to month change
        ust10["ten_year"] = pd.to_numeric(ust10["value"], errors="coerce")
        ust10["ten_year_mom"] = ust10["ten_year"].diff(periods=1)
        # only keep the standard rows
        ust10 = ust10[["date","ten_year","ten_year_mom"]]

# save the file
ust10.to_csv(OUT_DIR / "step5_ust10.csv", index=False)
print("[ust10] shape:", ust10.shape)
if not ust10.empty:
        print(f"[ust10] range=({ust10['date'].min().date()}, {ust10['date'].max().date()})")
display(ust10.tail(10))
```

```
[ust10] shape: (764, 3)
[ust10] range=(1962-01-31, 2025-08-31)
```

	date	ten_year	ten_year_mom
754	2024-11-30	4.355789	0.260335
755	2024-12-31	4.391429	0.035639
756	2025-01-31	4.629048	0.237619
757	2025-02-28	4.451053	-0.177995
758	2025-03-31	4.280476	-0.170576
759	2025-04-30	4.279048	-0.001429
760	2025-05-31	4.423810	0.144762
761	2025-06-30	4.383500	-0.040310
762	2025-07-31	4.391818	0.008318
763	2025-08-31	4.270625	-0.121193

```
#Final merge: all data sources into one table
#all companies
all_companies = pd.concat(company_tables, ignore_index=True)
#name of the city
all_companies["metro"] = all_companies["company"].map(COMPANY_TO_METRO)
#CPI
df_merged = pd.merge(all_companies, cpi_all, on=["date","metro"], how="left")
#unemployment rate
df_merged = pd.merge(df_merged, unemp, on=["date","metro"], how="left")
#10Y yield
df_merged = pd.merge(df_merged, ust10, on="date", how="left")
#save and display the result
df_merged.to_csv(OUT_DIR / "final_dataset.csv", index=False)
print("[final_dataset] shape:", df_merged.shape)
display(df_merged.head(20))
```

```
[final_dataset] shape: (769, 13)
```

	date	company	adj_price	dividend	dividend_ttm	metro	cpi	cpi_yoy	cpi_mom	unemp	unemp_mom	ten_
0	2014-12-31	BXP	100.7570	5.80	5.80	Boston	NaN	NaN	NaN	4.7	-0.1	2.20
1	2015-01-31	BXP	108.6960	7.75	13.55	Boston	254.556	0.018399	-0.006657	4.7	0.0	1.88
2	2015-02-28	BXP	107.6030	0.00	13.55	Boston	NaN	NaN	NaN	4.6	-0.1	1.97