This is code to regenerate the data pull from NHANES, but this time from 1999-2018 so we get a large dataset. Additionally, and more importantly, we need to filter only adults (over the age of 18).

1. Install libraries (bless those who have come before me to make these libraries!). What these libraries do

```
requests downloads XPT files

pandas loads XPT and wrangles data

tqdm gives a progress bar

functools.reduce helps with clean merging
```

In [99]:

```python
# If needed: !pipinstall libraries for first time

import os, time, requests
import pandas as pd
from functools import reduce
from tqdm import tqdm

pd.set_option("display.max_columns", 200)
```

2.Configure cycles and file naming

The problem is that NHANES changed names slightly over time. What this chunk does is defines the cycle suffixes and which files to download per cycle.

Diet totals: early cycles use DRXTOT/DRXTOT*B (no Day split); later cycles use DR1TOT** (we'll use Day 1 only to simplify the dataset).

The rest (DEMO/BMX/BPX/HDL/TRIGLY/GLU/BPQ) follow the usual suffix letters.

In [100…

```python
# Configure cycles and file naming


CYCLE_LETTER = {
    1999: "",    # 1999–2000
    2001: "B",   # 2001–2002
    2003: "C",   # 2003–2004
    2005: "D",
    2007: "E",
    2009: "F",
    2011: "G",
    2013: "H",
    2015: "I",
    2017: "J",   # 2017–2018
}
```

```python
def lbl(base: str, year: int) -> str:
    suf = CYCLE_LETTER[year]
    return f"{base}_{suf}" if suf else base

def files_for_cycle(year: int):
    # Dietary totals (yayhoo used Day 1 only; early cycles are DRXTOT)
    if year in (1999, 2001):
        diet_labels = ["DRXTOT" if year == 1999 else f"DRXTOT_{CYCLE_LETTER[year]}"]
    else:
        diet_labels = [f"DR1TOT_{CYCLE_LETTER[year]}"]

    # HDL: early cycles use LAB13/L13_B/L13_C, then HDL_*
    if year == 1999:
        hdl_label = "LAB13"
    elif year == 2001:
        hdl_label = "L13_B"
    elif year == 2003:
        hdl_label = "L13_C"
    else:
        hdl_label = lbl("HDL", year)

    # TRIGLY: early cycles use LAB13AM/L13AM_B/L13AM_C, then TRIGLY_*
    if year == 1999:
        trig_label = "LAB13AM"
    elif year == 2001:
        trig_label = "L13AM_B"
    elif year == 2003:
        trig_label = "L13AM_C"
    else:
        trig_label = lbl("TRIGLY", year)

    # GLUCOSE: early cycles use LAB10AM/L10AM_B/L10AM_C, then GLU_*
    if year == 1999:
        glu_label = "LAB10AM"      # 1999–2000
    elif year == 2001:
        glu_label = "L10AM_B"      # 2001–2002
    elif year == 2003:
        glu_label = "L10AM_C"      # 2003–2004
    else:
        glu_label = lbl("GLU", year)   # 2005–2018: GLU_D..GLU_J

    return dict(
        DEMO = lbl("DEMO", year),
        BMX  = lbl("BMX", year),
        BPX  = lbl("BPX", year),
        DIET = diet_labels,      # list of one
        HDL  = hdl_label,
        TRIG = trig_label,
        GLU  = glu_label,
        BPQ  = lbl("BPQ", year),
    )

ALL_CYCLES = sorted(CYCLE_LETTER.keys())
ALL_CYCLES

#Sniff check
# Expect to see {'GLU': 'LAB10AM'} in 1999, 'L10AM_B' in 2001, 'L10AM_C' in 2003, and
from collections import OrderedDict
preview = OrderedDict()
```

```
for yr in sorted(CYCLE_LETTER):
    f = files_for_cycle(yr)
    preview[(yr, 'HDL')]  = f['HDL']
    preview[(yr, 'TRIG')] = f['TRIG']
    preview[(yr, 'GLU')]  = f['GLU']
preview
```

Out[100]:

```
OrderedDict([((1999, 'HDL'), 'LAB13'),
             ((1999, 'TRIG'), 'LAB13AM'),
             ((1999, 'GLU'), 'LAB10AM'),
             ((2001, 'HDL'), 'L13_B'),
             ((2001, 'TRIG'), 'L13AM_B'),
             ((2001, 'GLU'), 'L10AM_B'),
             ((2003, 'HDL'), 'L13_C'),
             ((2003, 'TRIG'), 'L13AM_C'),
             ((2003, 'GLU'), 'L10AM_C'),
             ((2005, 'HDL'), 'HDL_D'),
             ((2005, 'TRIG'), 'TRIGLY_D'),
             ((2005, 'GLU'), 'GLU_D'),
             ((2007, 'HDL'), 'HDL_E'),
             ((2007, 'TRIG'), 'TRIGLY_E'),
             ((2007, 'GLU'), 'GLU_E'),
             ((2009, 'HDL'), 'HDL_F'),
             ((2009, 'TRIG'), 'TRIGLY_F'),
             ((2009, 'GLU'), 'GLU_F'),
             ((2011, 'HDL'), 'HDL_G'),
             ((2011, 'TRIG'), 'TRIGLY_G'),
             ((2011, 'GLU'), 'GLU_G'),
             ((2013, 'HDL'), 'HDL_H'),
             ((2013, 'TRIG'), 'TRIGLY_H'),
             ((2013, 'GLU'), 'GLU_H'),
             ((2015, 'HDL'), 'HDL_I'),
             ((2015, 'TRIG'), 'TRIGLY_I'),
             ((2015, 'GLU'), 'GLU_I'),
             ((2017, 'HDL'), 'HDL_J'),
             ((2017, 'TRIG'), 'TRIGLY_J'),
             ((2017, 'GLU'), 'GLU_J')])
```

3 Download helper and scrape all files

What this chunk does is downloads each XPT file. It will create a local folder and cache files there. This takes a little while :)

In [101…

```
# 3) Download helper and fetch all files

BASE_URL = "https://wwwn.cdc.gov/Nchs/Data/Nhanes/Public/{year}/DataFiles/{label}.xpt"
XPT_DIR = "nhanes_xpt_1999_2018"
os.makedirs(XPT_DIR, exist_ok=True)

def fetch_xpt(year: int, label: str, retries=3, backoff=2.0) -> str:
    """Download a single XPT to disk; return local path."""
    url = BASE_URL.format(year=year, label=label)
    dest = os.path.join(XPT_DIR, f"{label}.xpt")
    if os.path.exists(dest) and os.path.getsize(dest) > 1024:
        return dest
    for attempt in range(1, retries+1):
        r = requests.get(url, timeout=60)
        if r.ok and len(r.content) > 1024:
```

```
            with open(dest, "wb") as f:
                f.write(r.content)
            return dest
        time.sleep(backoff * attempt)
    raise RuntimeError(f"Failed to fetch {label} ({year}) at {url}")

# Download everything we'll need
download_manifest = []
for yr in tqdm(ALL_CYCLES, desc="Downloading XPTs by cycle"):
    files = files_for_cycle(yr)
    for key in ["DEMO","BMX","BPX","HDL","TRIG","GLU","BPQ"]:
        label = files[key]
        path = fetch_xpt(yr, label)
        download_manifest.append(dict(year=yr, comp=key, label=label, path=path))
    # Diet is a list (size 1)
    for label in files["DIET"]:
        path = fetch_xpt(yr, label)
        download_manifest.append(dict(year=yr, comp="DIET", label=label, path=path))

len(download_manifest)
```

```
Downloading XPTs by cycle: 100%|
| 10/10 [00:00<00:00, 924.26it/s]
```

Out[101]:

```
80
```

4. Load XPT: DataFrames (per cycle), select only needed columns

What this does is reads each component for a cycle.

For diet, keeps Day-1 totals (or maps early DRXT → *DR1T*).

For HDL/GLU/TRIG, tries a small set of candidate column names and standardizes to:

HDL → LBDHDD

Glucose → LBXGLU

Triglycerides → LBXTR

Adds cycle_begin_year and stores the per-cycle wide frame.

In [102…

```
# 4) Load XPTs and select only the columns we need (with robust HDL/GLU/TRIG mapping)

def read_xpt(path: str) -> pd.DataFrame:
    return pd.read_sas(path, format="xport", encoding="latin-1")

# exact column lists we want
KEEP_DEMO = ["SEQN", "RIDAGEYR", "RIAGENDR"]
KEEP_BMX  = ["SEQN", "BMXWT", "BMXHT", "BMXBMI", "BMXWAIST"]
KEEP_BPX  = ["SEQN", "BPXSY1", "BPXDI1"]
KEEP_BPQ  = ["SEQN", "BPQ050A", "BPQ100D"]

# Diet Day1 target names (yayhoo used Day 1)
KEEP_DIET_DAY1 = [
    "SEQN", "DR1TKCAL", "DR1TPROT", "DR1TSUGR", "DR1TCARB",
    "DR1TTFAT", "DR1TSFAT", "DR1TMFAT", "DR1TPFAT"
```

```
]

# Early-cycle (1999/2001) DRXTOT fallback (same semantics as Day 1 totals)
# include alt sugar names sometimes seen in metadata
KEEP_DIET_DRXT = [
    "SEQN",
    "DRXTKCAL", "DRXTPROT",
    "DRXTSUGR", "DRXTSUGR1", "DRXTSUGR2",   # sugar variants
    "DRXTCARB",
    "DRXTTFAT", "DRXTSFAT", "DRXTMFAT", "DRXTPFAT"
]

# Candidate name maps for early lab files → standardized names
CANDIDATE_MAPS = {
    "HDL":  [("LBDHDD","LBDHDD"), ("LBDHDL","LBDHDD"), ("LBXHDD","LBDHDD"), ("LBXHDL",
    "GLU":  [("LBXGLU","LBXGLU"), ("LBXGLU1","LBXGLU")],  # rare alt
    "TRIG": [("LBXTR","LBXTR"), ("LBDTRSI","LBXTR")],     # extend if needed
}

def pick_and_standardize(df: pd.DataFrame, candidates) -> tuple[pd.DataFrame, str|None
    """
    From df, pick the first present source column from candidates [(src,std),...],
    rename it to std, and return (SEQN + std subset, used_source_name).
    """
    for src, std in candidates:
        if src in df.columns:
            sub = df[["SEQN", src]].rename(columns={src: std})
            return sub, src
    # None found → return SEQN-only; we'll merge and keep NaN for the lab
    return df[["SEQN"]].copy(), None

cycle_frames = []
used_log = []  # optional: to inspect which source columns were used

for yr in tqdm(ALL_CYCLES, desc="Reading & selecting columns"):
    files = files_for_cycle(yr)

    # Read each component once, then subset to the fields we need
    demo_raw = read_xpt(fetch_xpt(yr, files["DEMO"]))
    bmx_raw  = read_xpt(fetch_xpt(yr, files["BMX"]))
    bpx_raw  = read_xpt(fetch_xpt(yr, files["BPX"]))
    bpq_raw  = read_xpt(fetch_xpt(yr, files["BPQ"]))

    demo = demo_raw[[c for c in KEEP_DEMO if c in demo_raw.columns]]
    bmx  = bmx_raw [[c for c in KEEP_BMX  if c in bmx_raw.columns]]
    bpx  = bpx_raw [[c for c in KEEP_BPX  if c in bpx_raw.columns]]
    bpq  = bpq_raw [[c for c in KEEP_BPQ  if c in bpq_raw.columns]]

    # Diet: DR1TOT_* preferred; else map DRXTOT(_B) to DR1T* names
    diet_label = files["DIET"][0]
    diet_df = read_xpt(fetch_xpt(yr, diet_label))
    has_day1 = any(col in diet_df.columns for col in KEEP_DIET_DAY1 if col != "SEQN")
    has_drxt = any(col in diet_df.columns for col in KEEP_DIET_DRXT if col != "SEQN")

    if has_day1:
        diet = diet_df[[c for c in KEEP_DIET_DAY1 if c in diet_df.columns]]
        used_log.append((yr, "DIET", diet_label, "DR1T*"))
    elif has_drxt:
        diet = diet_df[[c for c in KEEP_DIET_DRXT if c in diet_df.columns]].rename(col
            "DRXTKCAL":"DR1TKCAL",
```

```
            "DRXTPROT":"DR1TPROT",
            "DRXTSUGR":"DR1TSUGR",
            "DRXTSUGR1":"DR1TSUGR",
            "DRXTSUGR2":"DR1TSUGR",
            "DRXTCARB":"DR1TCARB",
            "DRXTTFAT":"DR1TTFAT",
            "DRXTSFAT":"DR1TSFAT",
            "DRXTMFAT":"DR1TMFAT",
            "DRXTPFAT":"DR1TPFAT"
        })
        used_log.append((yr, "DIET", diet_label, "DRXT*→DR1T*"))
    else:
        raise KeyError(f"Diet columns not found for year {yr} in {diet_label}")

    # Labs: pick whichever source name exists and standardize to LBDHDD/LBXGLU/LBXTR
    hdl_raw  = read_xpt(fetch_xpt(yr, files["HDL"]))
    hdl, hdl_src = pick_and_standardize(hdl_raw, CANDIDATE_MAPS["HDL"])
    used_log.append((yr, "HDL",  files["HDL"],  hdl_src or "NOT_FOUND"))

    glu_raw  = read_xpt(fetch_xpt(yr, files["GLU"]))
    glu, glu_src = pick_and_standardize(glu_raw, CANDIDATE_MAPS["GLU"])
    used_log.append((yr, "GLU",  files["GLU"],  glu_src or "NOT_FOUND"))

    trig_raw = read_xpt(fetch_xpt(yr, files["TRIG"]))
    trig, trig_src = pick_and_standardize(trig_raw, CANDIDATE_MAPS["TRIG"])
    used_log.append((yr, "TRIG", files["TRIG"], trig_src or "NOT_FOUND"))

    # Merge per cycle (left joins, DEMO first, like yayhoo)
    pieces = [demo, bmx, bpx, diet, glu, hdl, trig, bpq]
    cyc = reduce(lambda L, R: pd.merge(L, R, on="SEQN", how="left"), pieces)
    cyc["cycle_begin_year"] = yr
    cycle_frames.append(cyc)

# Optional: inspect which lab source columns were used per cycle
# for row in used_log: print(row)

len(cycle_frames)
```

```
Reading & selecting columns: 100%|
| 10/10 [00:04<00:00,  2.22it/s]
```

Out[102]:

```
10
```

5. Concatenate all cycles into one wide dataset

What this chunk does is stacks all per-cycle frames into one gigantic DataFrame. Each row is one SEQN per cycle.

In [103…

```
# Concatenate all cycles

data_wide = pd.concat(cycle_frames, ignore_index=True, sort=False)

print("Shape:", data_wide.shape)
data_wide.head(3)
data_wide.to_csv('originaldump.csv', index=False)
```

```
Shape: (101316, 23)
```

6. Final rename to yayhoo headers + recodes (run on data_wide)

What this chunk does:

Takes the merged wide dataset data_wide (which should include DEMO, BMX, BPX, DIET Day-1, GLU, HDL, TRIG, BPQ).

Renames columns to match the yayhoo notebook exactly.

Recodes gender (1→"male", 2→"female") and meds_hbp/meds_chol (1→"TRUE", 2→"FALSE").

Reports any missing columns so you know what (if anything) didn't merge.

In [104…

```
# Final rename to yayhoo headers + recodes (must run on data_wide)

import pandas as pd

# Safety check: ensure the merged frame exists
assert 'data_wide' in globals(), "data_wide not found — run Chunk 5 first."

# Map original NHANES names → yayhoo names
rename_map = {
    # Demographics / exam / BP
    "RIDAGEYR": "age",
    "RIAGENDR": "gender",
    "BPXSY1":   "bp_sys",
    "BPXDI1":   "bp_di",
    "BMXWT":    "weight",
    "BMXHT":    "height",
    "BMXBMI":   "bmi",
    "BMXWAIST": "waist",

    # Diet (Day 1 totals)
    # (For 1999/2001 we normalized DRXT*→DR1T* earlier in Chunk 4)
    "DR1TKCAL": "kcal",
    "DR1TPROT": "protein",
    "DR1TSUGR": "sugar",
    "DR1TCARB": "carb",
    "DR1TTFAT": "fat_total",
    "DR1TSFAT": "fat_sat",
    "DR1TMFAT": "fat_mon",
    "DR1TPFAT": "fat_poly",

    # Labs
    "LBDHDD":   "hdl",
    "LBXGLU":   "glucose",
    "LBXTR":    "triglycerides",

    # Questionnaire (meds)
    "BPQ050A":  "meds_hbp",
    "BPQ100D":  "meds_chol",
}

# Create the renamed frame
data_yayhoo = data_wide.rename(columns=rename_map).copy()
```

```python
# Recode gender: 1→"male", 2→"female"
if "gender" in data_yayhoo.columns:
    data_yayhoo["gender"] = data_yayhoo["gender"].map({1: "male", 2: "female"}).astype

# Recode meds: 1→"TRUE", 2→"FALSE"
for mcol in ["meds_hbp", "meds_chol"]:
    if mcol in data_yayhoo.columns:
        data_yayhoo[mcol] = data_yayhoo[mcol].map({1: "TRUE", 2: "FALSE"})

# Keep only the yayhoo set (plus SEQN and cycle_begin_year)
final_order = [
    "SEQN", "cycle_begin_year",
    "age", "gender",
    "bp_sys", "bp_di",
    "weight", "height", "bmi", "waist",
    "kcal", "protein", "sugar", "carb", "fat_total", "fat_sat", "fat_mon", "fat_poly",
    "hdl", "glucose", "triglycerides",
    "meds_hbp", "meds_chol",
]
data_yayhoo = data_yayhoo[[c for c in final_order if c in data_yayhoo.columns]]

# Quick report so you can verify at a glance
expected = set(final_order)
present  = set(data_yayhoo.columns)
missing  = sorted(list(expected - present))
print("Final columns:", list(data_yayhoo.columns))
print("Missing expected columns:", missing if missing else "None")

# Peek
data_yayhoo.head(5)

data_yayhoo.to_csv('renamed.csv', index=False)
```

```
Final columns: ['SEQN', 'cycle_begin_year', 'age', 'gender', 'bp_sys', 'bp_di', 'weig
ht', 'height', 'bmi', 'waist', 'kcal', 'protein', 'sugar', 'carb', 'fat_total', 'fat_
sat', 'fat_mon', 'fat_poly', 'hdl', 'glucose', 'triglycerides', 'meds_hbp', 'meds_cho
l']
Missing expected columns: None
```

In [105…

```python
### Sniff Check
# replace 'data_yayhoo' with your current final dataset name
sample_df = data_yayhoo.sample(n=50, random_state=42)  # reproducible random sample
sample_path = "sample.csv"

sample_df.to_csv(sample_path, index=False)
print(f"Random sample of 50 rows saved to: {sample_path}")
data_yayhoo
```

```
Random sample of 50 rows saved to: sample.csv
```

Out[105]:

	SEQN	cycle_begin_year	age	gender	bp_sys	bp_di	weight	height	bmi	waist	kca
0	1.0	1999	2.0	female	NaN	NaN	12.5	91.6	14.90	45.7	1358.8
1	2.0	1999	77.0	male	106.0	58.0	75.4	174.0	24.90	98.0	2463.0
2	3.0	1999	10.0	female	110.0	60.0	32.9	136.6	17.63	64.7	1517.6
3	4.0	1999	1.0	male	NaN	NaN	13.3	NaN	NaN	NaN	1474.9
4	5.0	1999	49.0	male	122.0	82.0	92.5	178.3	29.10	99.9	2658.1
...	...	...	...	...	...	...	...	...	...	...	.
101311	102952.0	2017	70.0	female	136.0	74.0	49.0	156.5	20.00	82.2	1080.0
101312	102953.0	2017	42.0	male	124.0	76.0	97.4	164.9	35.80	114.8	3072.0
101313	102954.0	2017	41.0	female	116.0	66.0	69.1	162.6	26.10	86.4	1757.0
101314	102955.0	2017	14.0	female	114.0	62.0	111.9	156.6	45.60	113.5	1256.0
101315	102956.0	2017	38.0	male	150.0	98.0	111.5	175.8	36.10	122.0	1591.0

101316 rows × 23 columns

7. Write to file.

In [106…

```
data_yayhoo.to_csv("Nutrition Data from NHANES.csv", index=False)
```

8. Retain only adults. I need to see what was left so I did a summary.

In [107…

```
#Drop rows missing age and keep only adults
n_before = len(data_yayhoo)
data_yayhoo_adults = data_yayhoo.dropna(subset=["age"])
data_yayhoo_adults = data_yayhoo_adults[data_yayhoo_adults["age"] >= 18].copy()
n_after = len(data_yayhoo_adults)

print(f"Filtered dataset: {n_after:,} of {n_before:,} rows retained ({n_after/n_before

# Optional: quick age summary
print(data_yayhoo_adults["age"].describe())

# Save to CSV for downstream analysis
os.makedirs("nhanes_outputs", exist_ok=True)
out_path = "nhanes_1999_2018_yayhoo_adults.csv"
data_yayhoo_adults.to_csv(out_path, index=False)
print(f"Saved adult-only dataset to: {out_path}")
```

```
Filtered dataset: 59,204 of 101,316 rows retained (58.4%).
count    59204.000000
mean        47.728431
std         19.498301
min         18.000000
25%         31.000000
50%         47.000000
75%         64.000000
max         85.000000
Name: age, dtype: float64
Saved adult-only dataset to: nhanes_1999_2018_yayhoo_adults.csv
```

9. Now I want to keep all the observations I can. So I need to impute the things that are possible to impute.

I went back to NHANES on how data appeared missing. The problem is that in some cases (fasting glucose for example) the measurements were only collected in a subsample. The Dietary Recall part is hard and NHANES recommends not imputing. So we will remove observations with missing fasting and missing dietary recall varaibles. Then we will impute the anthro and BP data.

Missingness Strategy Summary

Category	Variables	Strategy	Justification
Structurally missing (fasting subset)	`triglycerides`, `glucose`, `hdl`	❌ *Do not impute.* Restrict analysis to fasting participants or leave missing.	These are only measured for the fasting subsample; missing is *by design*, not random.
Dietary recall nonresponse	`kcal`, `protein`, `carb`, `fat_total`, `fat_sat`, `fat_mon`, `fat_poly`, `sugar`	⚠️ *Do not impute.*	NHANES assigns "missing" when no 24-hour recall interview exists — this is *not random* and reflects survey logistics.
Anthropometrics + BP (safe numeric impute)	`weight`, `height`, `bmi`, `waist`, `bp_sys`, `bp_di`	✅ *Impute.*	These are continuous, well-correlated, and missing due to random attendance or measurement failure.

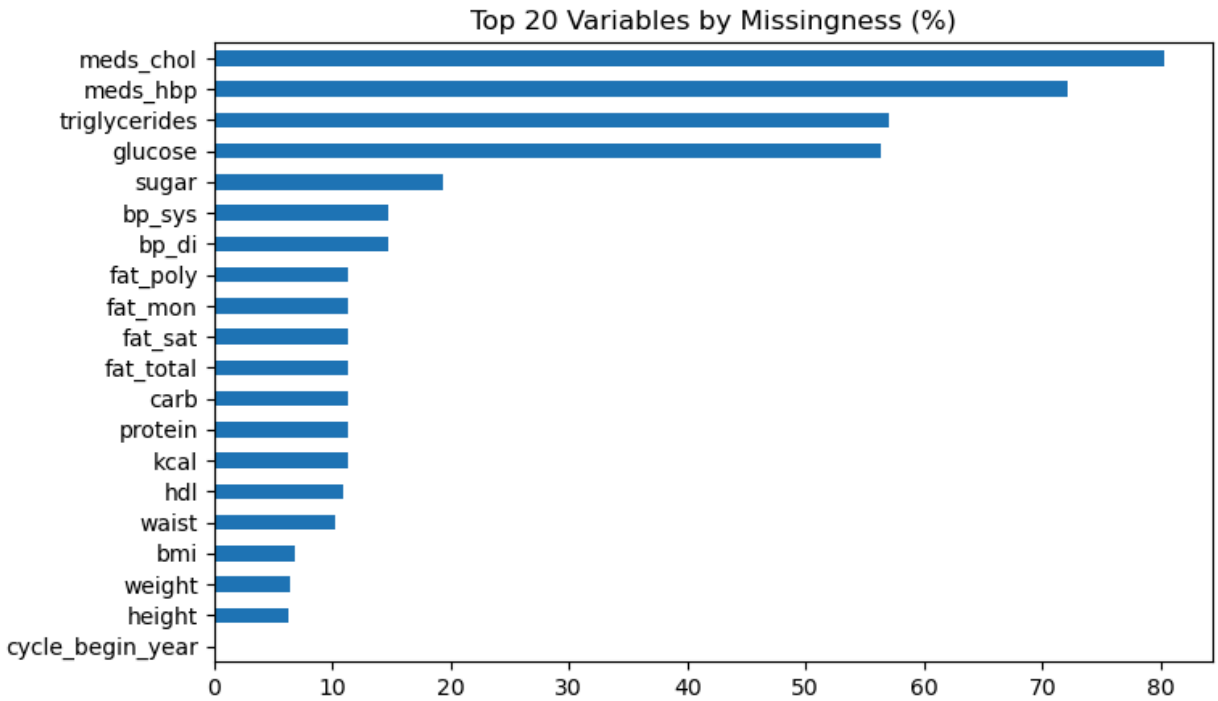
This chunk examines the dataset and checks for how much I have missing (Jake LaPorte style :))

In [108…

```
import pandas as pd

df = data_yayhoo_adults

# Percent missing per column
missing = df.isna().mean().sort_values(ascending=False).round(3) * 100
print("Percent missing by variable:\n", missing)

# Quick visualization (optional)
import matplotlib.pyplot as plt
missing.head(20).plot.barh(figsize=(8,5), title="Top 20 Variables by Missingness (%)")
```

```
plt.gca().invert_yaxis()
plt.show()
```

```
Percent missing by variable:
 meds_chol           80.3
meds_hbp            72.1
triglycerides       57.0
glucose             56.3
sugar               19.4
bp_sys              14.8
bp_di               14.8
fat_poly            11.3
fat_mon             11.3
fat_sat             11.3
fat_total           11.3
carb                11.3
protein             11.3
kcal                11.3
hdl                 10.9
waist               10.2
bmi                  6.8
weight               6.4
height               6.3
cycle_begin_year     0.0
gender               0.0
age                  0.0
SEQN                 0.0
dtype: float64
```



In [109…

```
# Create fasting + complete diet subset from saved CSV
# -------------------------------------------------------
# Input  : df from previous chunk
# Output : nhanes_1999_2018_yayhoo_fasting_diet_complete.csv
# -------------------------------------------------------



fasting_vars = ["glucose", "hdl", "triglycerides"]
```

```python
diet_vars = ["kcal", "protein", "carb", "fat_total",
             "fat_sat", "fat_mon", "fat_poly", "sugar"]

#  Build mask for complete fasting AND complete diet. A mask is am indicator function
# that tells pandas which rows (or columns) to keep, modify, or analyze.
complete_mask = df[fasting_vars + diet_vars].notna().all(axis=1)

# Apply filter
df_fasting_diet = df.loc[complete_mask].copy()

# Track before / after counts
n_total = len(df)
n_keep  = len(df_fasting_diet)
print(f"Total adults: {n_total:,}")
print(f"Complete fasting + diet records: {n_keep:,} ({n_keep/n_total:.1%})")

# Save reduced dataset
os.makedirs("nhanes_outputs", exist_ok=True)
out_path = "nhanes_1999_2018_yayhoo_fasting_diet_complete.csv"
df_fasting_diet.to_csv(out_path, index=False)
print(f"Saved complete fasting + diet subset to: {out_path}")

# Sniff Check
print("\nFirst 5 rows:")
display(df_fasting_diet.head())
```

```
Total adults: 59,204
Complete fasting + diet records: 21,849 (36.9%)
Saved complete fasting + diet subset to: nhanes_1999_2018_yayhoo_fasting_diet_complet
e.csv

First 5 rows:
```

	SEQN	cycle_begin_year	age	gender	bp_sys	bp_di	weight	height	bmi	waist	k
9965	9966.0	2001	39.0	male	128.0	8.000000e+01	91.7	174.2	30.22	100.8	305
9967	9968.0	2001	84.0	female	120.0	5.397605e-79	51.7	144.9	24.62	86.1	133
9968	9969.0	2001	51.0	female	120.0	7.200000e+01	58.0	161.4	22.26	74.0	179
9971	9972.0	2001	44.0	male	146.0	8.600000e+01	101.5	173.2	33.84	116.0	326
9972	9973.0	2001	63.0	female	100.0	7.400000e+01	64.9	156.2	26.60	94.2	197

10. I'm going to impute the anthopometrics and BP now. I added additional columns indicating when a value was imputed.

In [110…

```python
# Impute anthropometric and blood pressure variables (index-safe)
# ---------------------------------------------------------------
# Input : df_fasting_diet  (from previous chunk)
# Output: nhanes_outputs/nhanes_1999_2018_yayhoo_fasting_diet_imputed.csv

from sklearn.experimental import enable_iterative_imputer  # noqa
from sklearn.impute import IterativeImputer
import pandas as pd
import numpy as np
import os
```

```python
# Start from the correct source and make sure index is clean
df = df_fasting_diet.copy()

# Keep cycle/year & ID intact (just a guardrail)
assert {"SEQN","cycle_begin_year"}.issubset(df.columns), "SEQN/cycle_begin_year missir

# 1) Variables to impute and predictors
impute_vars = ["weight", "height", "bmi", "waist", "bp_sys", "bp_di"]
predictors  = ["age", "gender", "cycle_begin_year"] + impute_vars

missing_cols = [c for c in predictors if c not in df.columns]
if missing_cols:
    raise KeyError(f"Missing columns needed for imputation: {missing_cols}")

# Reset index so assignment is by position - had problems with this earlier
df = df.reset_index(drop=True)

# Build the matrix for imputation (encode gender)
sub = df[predictors].copy()
sub["gender"] = sub["gender"].map({"male": 0, "female": 1})

# If any stray NaNs in encoded gender, fill with mode to keep numeric
if sub["gender"].isna().any():
    mode_gender = 1 if (sub["gender"].mean(skipna=True) >= 0.5) else 0
    sub["gender"] = sub["gender"].fillna(mode_gender)

# Fit imputer with mice like package
imp = IterativeImputer(random_state=42, max_iter=30)
imputed = imp.fit_transform(sub)

# Rebuild imputed DataFrame
imputed_df = pd.DataFrame(imputed, columns=sub.columns, index=sub.index)
imputed_df["gender"] = imputed_df["gender"].round().map({0: "male", 1: "female"})

# Merge back by position (indexes now match)
df_imputed = df.copy()
for v in impute_vars:
    was_nan = df[v].isna()
    df_imputed[v] = imputed_df[v]              # index-aligned now
    df_imputed[f"imputed_{v}"] = was_nan       # True where we filled

# Sniff Check that there are no NaNs in the imputed set-there were once before :)
post_missing = df_imputed[impute_vars].isna().sum()
print("NaNs after imputation (should be 0):\n", post_missing)
assert int(post_missing.sum()) == 0, "Some imputed variables remain NaN."

#Save
os.makedirs("nhanes_outputs", exist_ok=True)
out_path = "nhanes_outputs/nhanes_1999_2018_yayhoo_fasting_diet_imputed.csv"
df_imputed.to_csv(out_path, index=False)
print(f"\nSaved imputed dataset to: {out_path}")

# Sniff Check: by-year row counts to confirm 2011+ are still present
print("\nRows by cycle (post-imputation):")
print(df_imputed.groupby("cycle_begin_year").size())
```

```
NaNs after imputation (should be 0):
 weight    0
height    0
bmi       0
waist     0
bp_sys    0
bp_di     0
dtype: int64

Saved imputed dataset to: nhanes_outputs/nhanes_1999_2018_yayhoo_fasting_diet_impute
d.csv

Rows by cycle (post-imputation):
cycle_begin_year
2001    2501
2003    2295
2005    2299
2007    2624
2009    2807
2011    2363
2013    2463
2015    2227
2017    2270
dtype: int64
```

In [ ]: