Step 02: Computing Relevant Snow Metrics on a Subset of SNOTEL Stations

Keith Jennings

Getting started

First, we need to load our packages like we did in our previous notebook.

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.3.0
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(snotelr)
library(cowplot)
```

```
##
## Attaching package: 'cowplot'
##
## The following object is masked from 'package:lubridate':
##
##     stamp
```

```r
theme_set(theme_cowplot())
library(knitr)
library(Kendall)
library(trend)
```

Next, we will `source` the functions I've created to derive water year information and compute snow metrics. These files can be found in `analysis/functions`.

```r
source("functions/snow_metrics.R")
source("functions/meteo_metrics.R")
source("functions/water_year.R")
```

You can see these functions now in your Environment pane. All the `source` function does is run the R scripts that contain the functions I've built. (We'll return to the actual functions later.)

Defining our subset

Useful metadata

In our work we will want to use all or most of the long-term SNOTEL records, but here we'll start with a subset. To start building this subset, we'll need a couple data sources. One is the SNOTEL metadata we can grab from `snotelr`:

```
snotel_info <- snotel_info()
snotel_info %>%
  head()
```

```
##   network state            site_name
## 1    SNTL    AK     elmendorf field
## 2    SNTL    UT           elk ridge
## 3    SNTL    AK              hoonah
## 4    SNTL    AK pilgrim hot springs
## 5    SNTL    AK          seven mile
## 6    SNTL    CA          lost lakes
##                                   description      start        end
## 1               Outlet Ship Creek (190204010404) 2024-10-01 2025-06-17
## 2                Cottonwood Creek (140802010402) 2024-10-01 2025-06-17
## 3 Port Fredrick-Frontal Icy Strait (190102110906) 2023-10-01 2025-06-17
## 4    Paystreak Creek-Pilgrim River (190501050702) 2024-07-01 2025-06-17
## 5                Outlet Ray River (190804040306) 2024-09-01 2025-06-17
## 6     Upper West Fork Carson River (160502010301) 2024-09-01 2025-06-17
##   latitude longitude elev        county site_id
## 1    61.25   -149.82   52     Anchorage    1332
## 2    37.82   -109.77 2603      San Juan    1323
## 3    58.12   -135.41  463 Hoonah-angoon    1318
## 4    65.09   -164.92    6          Nome    1327
## 5    65.94   -149.86  201 Yukon-koyukuk    1330
## 6    38.65   -119.95 2633        Alpine    1331
```

The other is the HARBOR dataset from Scott Peckham:

```
harbor_url <- "https://raw.githubusercontent.com/peckhams/nextgen_basin_repo/refs/heads/main/__Collated/
basin_info <- read_tsv(harbor_url)
```

```
## Rows: 30717 Columns: 51
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (42): Site_ID, NWS_Loc_ID, GOES_ID, RFC, WFO/CWA, HSA, HUC, Site_Name, S...
## dbl  (9): Lon, Lat, Area, Minlon, Maxlon, Minlat, Maxlat, Closest_Site_Dist,...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
basin_info %>%
  head()
```

```
## # A tibble: 6 x 51
##   Site_ID  NWS_Loc_ID GOES_ID  RFC     ‘WFO/CWA‘ HSA   HUC   Site_Name Site_Type
##   <chr>    <chr>      <chr>    <chr>   <chr>     <chr> <chr> <chr>     <chr>
## 1 01010000 NINM1      17C662A6 NERFC   CAR       CAR   0101~ St. John~ Stream
## 2 01010070 BBRM1      DDB36490 NERFC   CAR       CAR   0101~ Big Blac~ Stream
## 3 01010100 -          -        -       -         -     0101~ Shields ~ Stream
## 4 01010500 DICM1      DE2AB664 NERFC   CAR       CAR   0101~ St. John~ Stream
## 5 01011000 ALLM1      DDB3A18E NERFC   CAR       CAR   0101~ Allagash~ Stream
## 6 01011500 STFB3      45A46214 unknown CAR       CAR   <NA>  St. Fran~ Stream
## # i 42 more variables: Stage_Data <chr>, PEDTS_Obs <chr>, State_Code <chr>,
## #   Country_Code <chr>, Lon <dbl>, Lat <dbl>, Elev <chr>, Elev_Units <chr>,
## #   Area <dbl>, Area_Units <chr>, Horiz_Datum <chr>, Vert_Datum <chr>,
## #   Minlon <dbl>, Maxlon <dbl>, Minlat <dbl>, Maxlat <dbl>, Long_Name <chr>,
## #   Closest_Site_ID <chr>, Closest_Site_Dist <dbl>, Site_URL <chr>,
## #   HUC_URL <chr>, NWS_URL <chr>, Status_as_FPS <chr>, Start_Date <chr>,
## #   End_Date <chr>, Eco_Region <chr>, HLR_Code_Outlet <dbl>, ...
```

We discussed the SNOTEL file in our previous notebook. The HARBOR (Harmonized Attributes of River Basins in One Repo) dataset is an exhaustive accounting of sometimes overlapping basin data sources from USGS NWIS, CAMELS, NWS River Forecast Centers, etc.

At first glance these two datasets have nothing in common, but buried in the `description` column of `snotel_info` is a HUC (hydrologic unit code) ID that we can match to the `HUC` column in `basin_info`. We have to do a bit of string manipulation first.

```
snotel_info <- snotel_info %>%
  mutate(HUC = stringr::str_extract(string = description,
                                    pattern = "(?<=\\().*(?=\\))"))
```

Now we that we've extracted the HUC ID from in between the parentheses, we can join the two datasets.

```
all_info <- left_join(snotel_info,
                 basin_info,
                 by = "HUC")
```

```
## Warning in left_join(snotel_info, basin_info, by = "HUC"): Detected an unexpected many-to-many relati
## i Row 1 of ‘x‘ matches multiple rows in ‘y‘.
## i Row 27096 of ‘y‘ matches multiple rows in ‘x‘.
## i If a many-to-many relationship is expected, set ‘relationship =
##   "many-to-many"‘ to silence this warning.
```

The warning above indicates there are some rows in `snotel_info` that have multiple matches in `basin_info` and vice versa. This occurs when there are multiple SNOTEL stations in a given HUC or when a SNOTEL station finds itself in multiple nested basins.

Using metadata to select SNOTEL stations

The combined dataframe includes multiple columns we can use to split the data. A few examples:

- All SNOTEL stations in a USGS GAGES II Reference basin
  - 94 matches

- SNOTEL stations above 1500 m in Oregon
  - 54 matches
- SNOTEL stations in the WestMtns ecoregion within a CAMELS basin with a snow-dom hydrograph type
  - 1 match
- And so on…

For now, we'll start with a relatively small SNOTEL and USGS Gages II Reference subset with at least 40 yrs of data.

```
subset_info <- all_info %>%
  filter(year(start) <= 1985 & year(end) >= 2024) %>%
  filter(Is_GAGES2_Ref == "Y")
```

There are some duplicates in the subset, so we'll filter to just the highest elevation basins (assuming they're more representative of the SNOTEL-observed snow conditions).

```
subset_info <- subset_info %>%
  group_by(site_id) %>%
  slice_max(order_by = Elev, with_ties = FALSE) %>%
  ungroup()
```

Now we have our subset of 33 SNOTEL stations.

SNOTEL data

Accessing station data

Now we'll identify the `site_id` for each station in our subset, put it in a vector, and download the data with `snotelr`.

```
sites <- subset_info %>% pull(site_id)
```

NOTE: If you don't want to wait while snotelr downloads the dataset, skip ahead to the commented-out cell that says `df <- readRDS("../data/snotel_camels_subset.RDS")`, uncomment it, and run it.

```
time_start = Sys.time()
df <- snotel_download(sites, internal = T)
```

```
## Downloading site: hewinta , with id: 521

## Downloading site: hole-in-rock , with id: 528

## Downloading site: hams fork , with id: 509

## Downloading site: echo peak , with id: 463
```

```
## Downloading site: rubicon #2 , with id: 724

## Downloading site: lamoille #3 , with id: 570

## Downloading site: munson ridge , with id: 950

## Downloading site: ward creek #3 , with id: 848

## Downloading site: summit ranch , with id: 802

## Downloading site: mt hood test site , with id: 651

## Downloading site: lasal mountain , with id: 572

## Downloading site: corral pass , with id: 418

## Downloading site: olallie meadows , with id: 672

## Downloading site: hagans meadow , with id: 508

## Downloading site: heavenly valley , with id: 518

## Downloading site: independence lake , with id: 541

## Downloading site: joe wright , with id: 551

## Downloading site: vail mountain , with id: 842

## Downloading site: franklin basin , with id: 484

## Downloading site: mud ridge , with id: 655

## Downloading site: north fork , with id: 666

## Downloading site: daniels-strawberry , with id: 435

## Downloading site: pickle keg , with id: 691

## Downloading site: steel creek park , with id: 790

## Downloading site: vernon creek , with id: 844

## Downloading site: dome lake , with id: 451

## Downloading site: little warm , with id: 585

## Downloading site: shell creek , with id: 751
```

```
## Downloading site: spring creek divide , with id: 779

## Downloading site: many glacier , with id: 613

## Downloading site: bear creek , with id: 321

## Downloading site: west yellowstone , with id: 924

## Downloading site: northeast entrance , with id: 670
```

```
time_end = Sys.time()
time_end - time_start
```

```
## Time difference of 2.98921 mins
```

First, we'll select just the columns we need.

```
# Downselect to just the columns we want
# Rename var columns to include units
df <- df %>%
  select(site_id, date,
         swe_mm = snow_water_equivalent,
         snow_depth_mm = snow_depth,
         ppt_mm = precipitation,
         tair_av_degC = temperature_mean)
```

Then we'll add date information:

```
# Add additional date information
df <- df %>%
  mutate(date = ymd(date),
         wyear = wateryear(date),
         dowy = day_of_wateryear(date))
```

Then we'll save it as an RDS file (I've commented this part out because it doesn't need to re-run).

Note: I saved the file for two reasons: 1) in case we have bandwidth issues and 2) to analyze again in the next notebook.

```
# saveRDS(object = df,
#         file = "../data/snotel_camels_subset.RDS")
```

We can uncomment out the following if we need to import the saved data.

```
# df <- readRDS("../data/snotel_camels_subset.RDS")
```

Pre-processing the data

We want to only include years in our analysis with a certain percentage of valid SWE observations. We're going to make that threshold 100% here (SNOTEL SWE has a relatively robust QC protocol), but you can choose other values.

```r
# Calculate the percentage of valid SWE observations per water year
site_summary_by_wyear <- df %>%
  group_by(site_id, wyear) %>%
  summarize(n_expected = ifelse(any(wyear %% 4 == 0),
                                  366,
                                  365),
            n_obs = sum(!is.na(swe_mm)),
            pct_valid = (n_obs / n_expected) * 100) %>%
  ungroup()
```

```
## `summarise()` has grouped output by 'site_id'. You can override using the
## `.groups` argument.
```

```r
# Provide a threshold of valid obs that we'll consider to be a complete water year
pct_valid_thresh = 100

# Identify sites and water years that meet our threshold
valid_sites_wyears <- site_summary_by_wyear %>%
  filter(pct_valid >= pct_valid_thresh) %>%
  select(site_id, wyear)

# Filter using inner join to only sites and water years in our valid data frame
df_filter <-
  inner_join(df, valid_sites_wyears,
             by = c("site_id", "wyear"))
```

Calculate metrics

Now we'll use our SNOTEL data subset to compute various metrics:

- Maximum snow water equivalent (SWE)
- Maximum SWE day of water year (DOWY)
- Snow cover duration
- Snow-on day
- Snow-off day
- Melt season length
- Snowmelt rate
- Snow seasonality metric (SSM)
- April 1 SWE
- Snowmelt before max SWE
- Snowmelt before max SWE percent of total snowmelt
- Snowmelt before max SWE to max SWE ratio
- Snowmelt center of mass DOWY
- Peak SWE to annual precipitation ratio
- And several meteorological data metrics

```r
metrics <- df_filter %>%
  group_by(site_id, wyear) %>%
  summarize(max_swe_mm = maxSWE(swe_mm),
            max_swe_dowy = maxSWE_DOWY(swe_mm, dowy),
```

```r
           scd_days = scd(swe_mm),
           snow_on_day = firstSnow(swe_mm, dowy),
           snow_off_day = lastSnow(swe_mm, dowy, max_swe_dowy),
           melt_season_days = meltSeason(max_swe_dowy, snow_off_day),
           melt_rate_mm_d = meltRate(melt_season_days, max_swe_mm),
           ssm = snowSeasonality(swe_mm),
           swe_apr1_mm = april1SWE(swe_mm, dowy, wyear),
           pre_max_swe_melt_mm = preMaxMelt(swe_mm, dowy, max_swe_dowy),
           pre_max_swe_melt_total_melt_pct = preMaxMeltPctTotalMelt(pre_max_swe_melt_mm, swe_mm),
           pre_max_swe_melt_max_swe_ratio = preMaxMeltMaxSWERatio(pre_max_swe_melt_mm, max_swe_mm),
           melt_com_day = meltCoM(swe_mm, dowy),
           swe_to_ppt_ratio = sweToPptRatio(max_swe_mm, ppt_mm),
           fall_ppt_mm = seasonalPrecip(ppt_mm, date, SEASON = "fall"),
           winter_ppt_mm = seasonalPrecip(ppt_mm, date, SEASON = "winter"),
           spring_ppt_mm = seasonalPrecip(ppt_mm, date, SEASON = "spring"),
           summer_ppt_mm = seasonalPrecip(ppt_mm, date, SEASON = "summer"),
           annual_ppt_mm = totalPrecip(ppt_mm),
           fall_tair_degC = seasonalTemp(tair_av_degC, date, SEASON = "fall"),
           winter_tair_degC = seasonalTemp(tair_av_degC, date, SEASON = "winter"),
           spring_tair_degC = seasonalTemp(tair_av_degC, date, SEASON = "spring"),
           summer_tair_degC = seasonalTemp(tair_av_degC, date, SEASON = "summer"),
           annual_tair_degC = meanTemp(tair_av_degC))
```

```
## `summarise()` has grouped output by 'site_id'. You can override using the
## `.groups` argument.
```

```r
# Export data
saveRDS(object = metrics,
        file = "../data/snotel_camels_subset_metrics.RDS")
```

We can take a quick look at these tabular data.

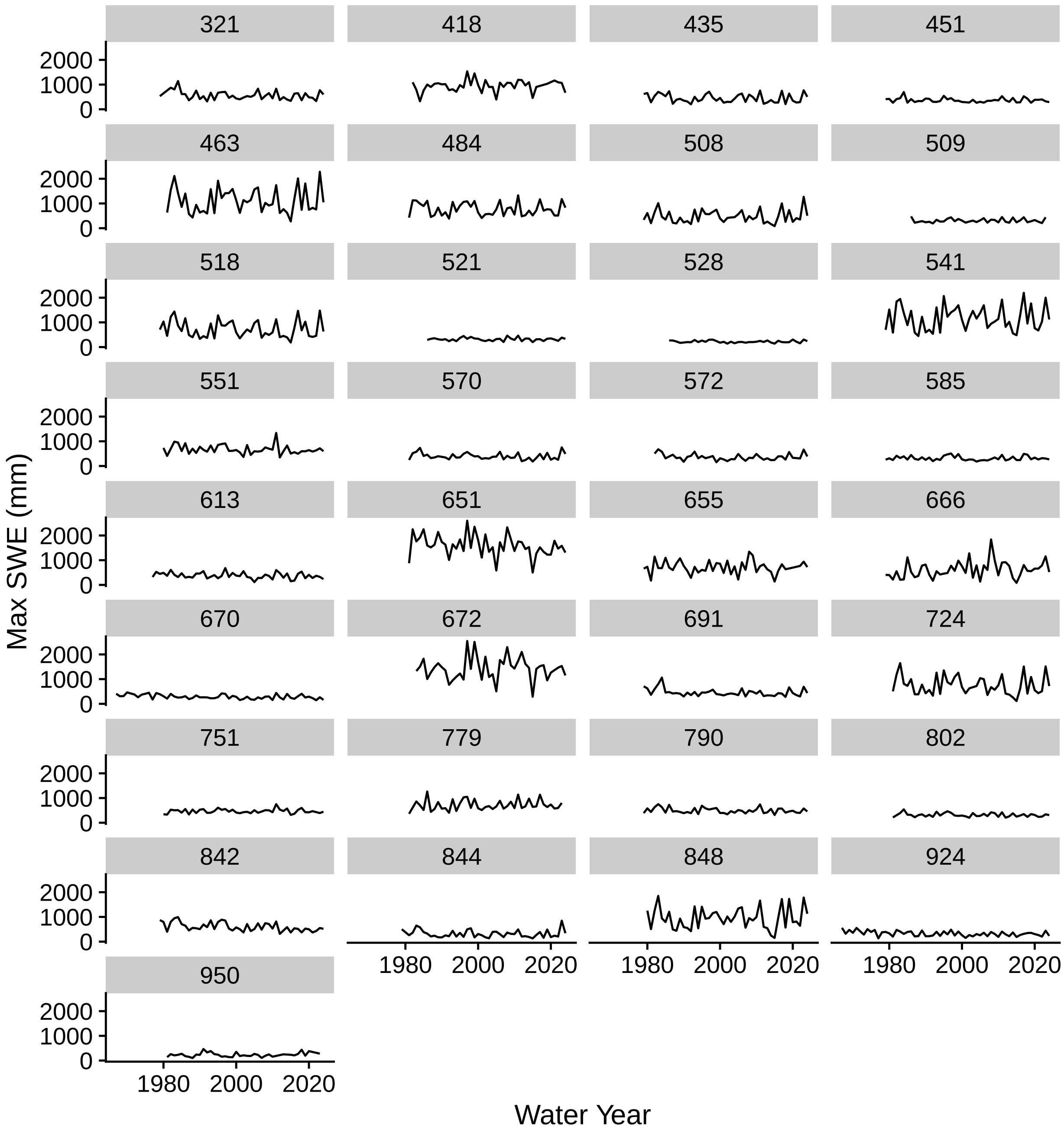
```r
metrics %>%
  head()
```

```
## # A tibble: 6 x 26
## # Groups:   site_id [1]
##   site_id wyear max_swe_mm max_swe_dowy scd_days snow_on_day snow_off_day
##     <dbl> <dbl>      <dbl>        <dbl>    <int>       <dbl>        <dbl>
## 1     321  1979       531.          188      186          60          246
## 2     321  1982       874.          194      244          12          259
## 3     321  1983       803.          232      238           1          262
## 4     321  1984      1143           220      223          49          272
## 5     321  1985       617.          182      213          24          237
## 6     321  1986       612.          176      244           7          245
## # i 19 more variables: melt_season_days <dbl>, melt_rate_mm_d <dbl>, ssm <dbl>,
## #   swe_apr1_mm <dbl>, pre_max_swe_melt_mm <dbl>,
## #   pre_max_swe_melt_total_melt_pct <dbl>,
## #   pre_max_swe_melt_max_swe_ratio <dbl>, melt_com_day <dbl>,
## #   swe_to_ppt_ratio <dbl>, fall_ppt_mm <dbl>, winter_ppt_mm <dbl>,
## #   spring_ppt_mm <dbl>, summer_ppt_mm <dbl>, annual_ppt_mm <dbl>,
## #   fall_tair_degC <dbl>, winter_tair_degC <dbl>, spring_tair_degC <dbl>, ...
```

We can also plot some of the outcomes.

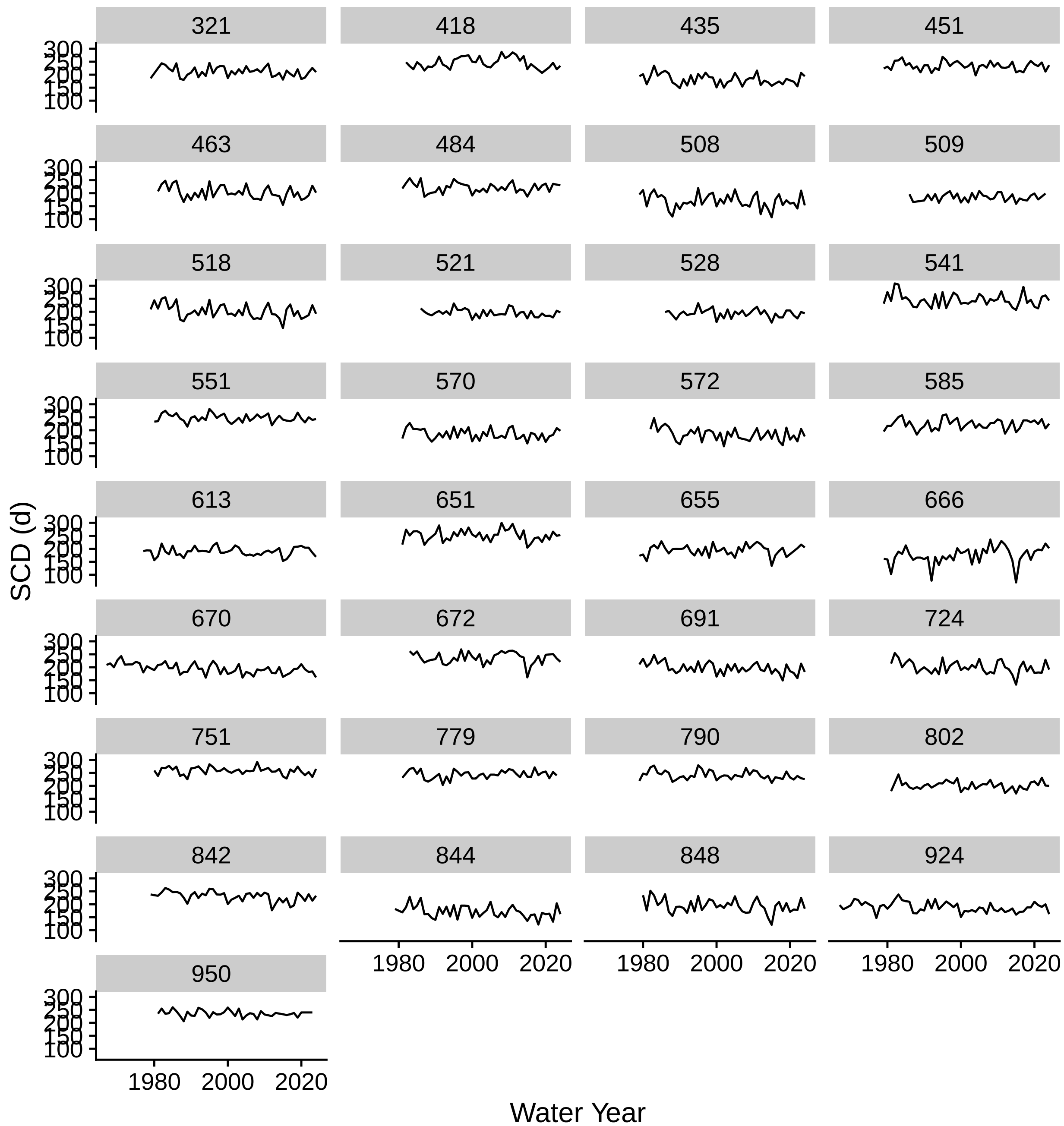
Maximum SWE

```r
ggplot(metrics, aes(wyear, max_swe_mm)) +
  geom_line() +
  facet_wrap(~as.factor(site_id), ncol = 4) +
  labs(x = "Water Year", y = "Max SWE (mm)")
```



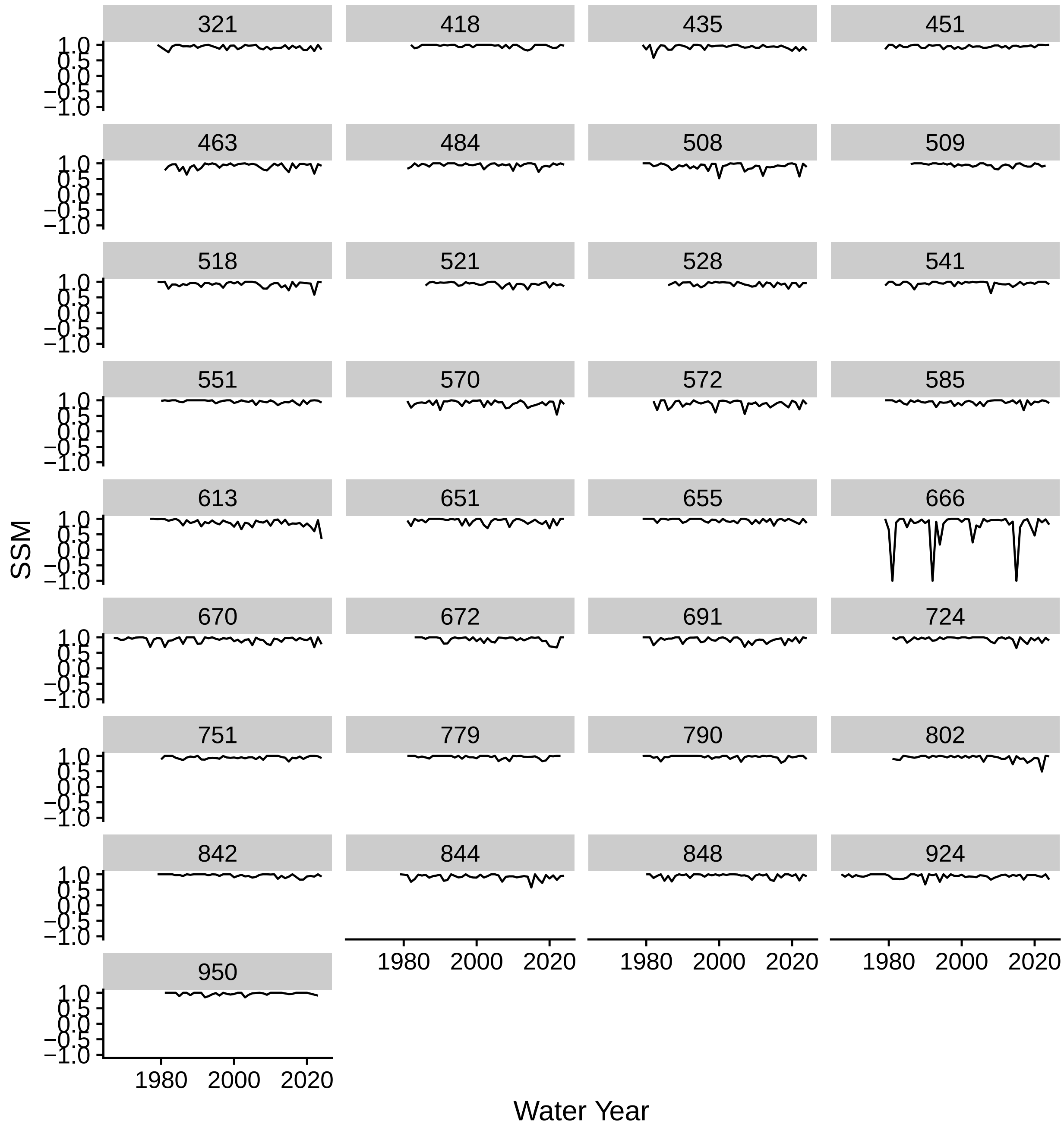
Snow cover duration (SCD)

```r
ggplot(metrics, aes(wyear, scd_days)) +
  geom_line() +
  facet_wrap(~as.factor(site_id), ncol = 4) +
  labs(x = "Water Year", y = "SCD (d)")
```



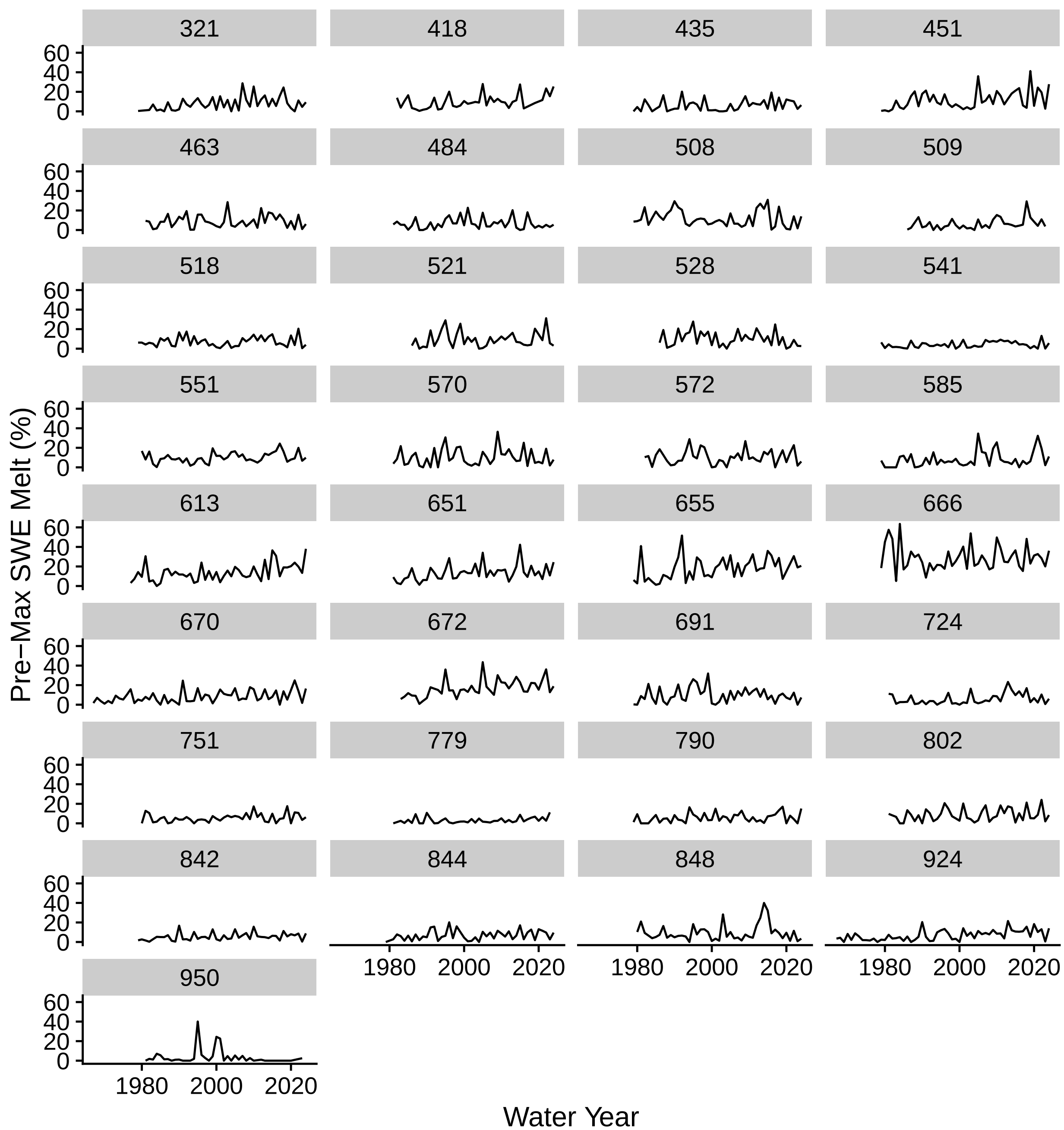
Snow seasonality metric (SSM)

```r
ggplot(metrics, aes(wyear, ssm)) +
  geom_line() +
  facet_wrap(~as.factor(site_id), ncol = 4) +
  labs(x = "Water Year", y = "SSM")
```



Pre-max SWE snowmelt as percent of total snowmelt

```r
ggplot(metrics, aes(wyear, pre_max_swe_melt_total_melt_pct)) +
  geom_line() +
  facet_wrap(~as.factor(site_id), ncol = 4) +
  labs(x = "Water Year", y = "Pre-Max SWE Melt (%)")
```



However, it is hard to tell what, if anything, is happening at our sites over time. So what we'll do next is compute some trends.

Compute trends

First we'll make a function to make a "long" version of our `metrics` dataframe and then compute various trend stats, such as the Mann-Kendall p-value and Sen's slope.

```
analyze_snow_trends <- function(DF) {
  # Pivot to make long dataframe
  df_long <- DF %>%
    pivot_longer(
      cols = -c(site_id, wyear), # could add column names as argument to make function generalizable
      names_to = "metric",
      values_to = "value"
    ) %>%
    filter(!is.na(value)) %>%    # remove NAs
    filter(!is.infinite(value))  # remove Infs

  # Take df_long and compute trend values per site_id and metric
  df_long %>%
    group_by(site_id, metric) %>%
    summarise(
      n_years = n(),
      mann_kendall = list(MannKendall(value)),
      sens_slope = list(sens.slope(value)),
      .groups = "drop"
    ) %>%
    mutate(
      mk_tau = map_dbl(mann_kendall, ~ .x$tau),
      mk_p = map_dbl(mann_kendall, ~ .x$sl),
      sen_slope = map_dbl(sens_slope, ~ .x$estimates),
      sen_p = map_dbl(sens_slope, ~ .x$p.value)
    ) %>%
    select(site_id, metric, n_years, mk_tau, mk_p, sen_slope, sen_p)
}
```

Now we'll apply this function to `metrics`.

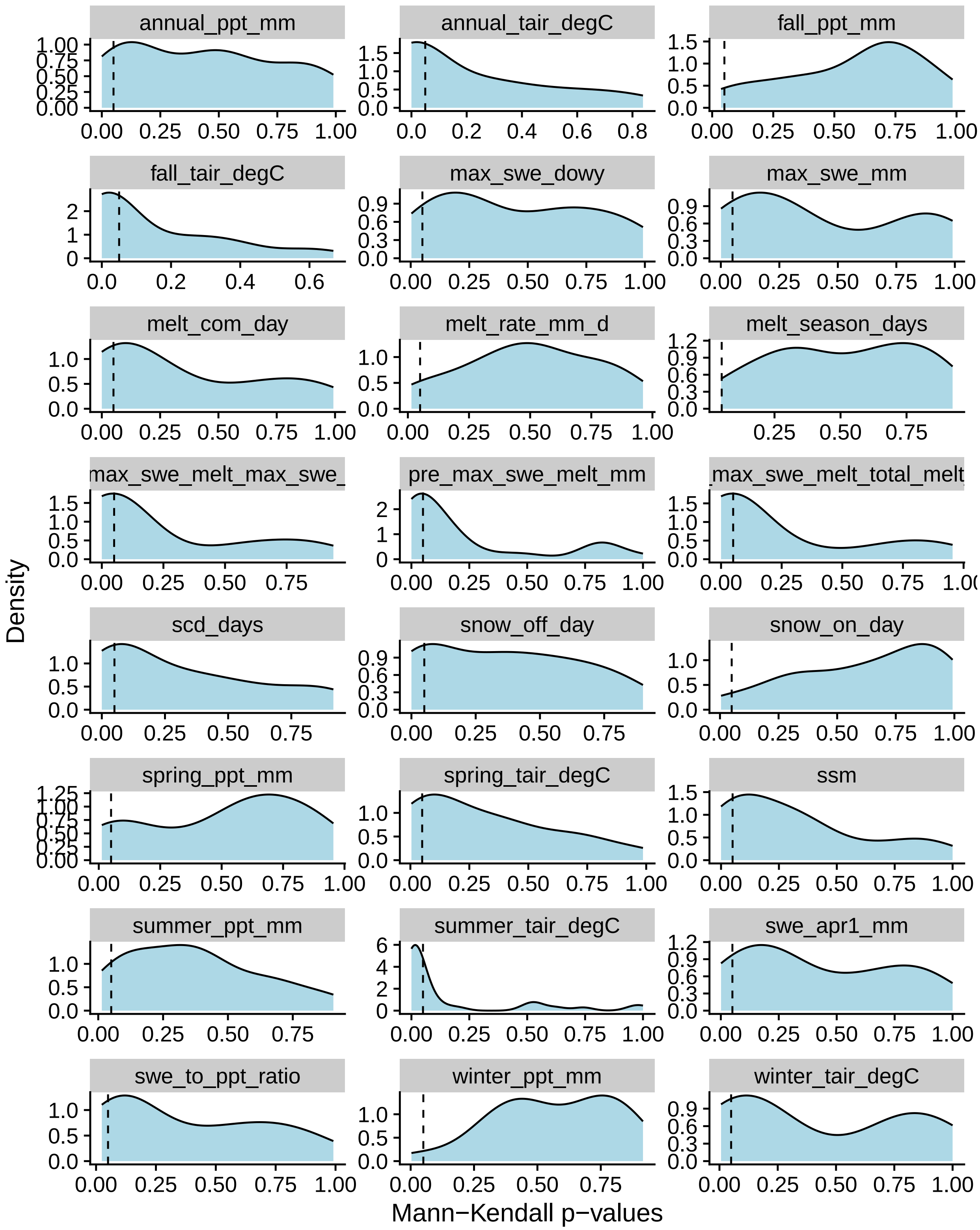
```
trends <- analyze_snow_trends(metrics)
trends %>%
  head()
```

```
## # A tibble: 6 x 7
##   site_id metric           n_years  mk_tau     mk_p sen_slope    sen_p
##     <dbl> <chr>              <int>   <dbl>    <dbl>     <dbl>    <dbl>
## 1     321 annual_ppt_mm         44  0.0127 0.911       0.180  0.911
## 2     321 annual_tair_degC      36  0.435  0.000200    0.0505 0.000200
## 3     321 fall_ppt_mm           44 -0.0359 0.739      -0.492  0.739
## 4     321 fall_tair_degC        35  0.408  0.000589    0.0723 0.000589
## 5     321 max_swe_dowy          44 -0.0554 0.606      -0.118  0.606
## 6     321 max_swe_mm            44 -0.146  0.166      -3.06   0.166
```

```
# Export trends data
saveRDS(object = trends,
        file = "../data/snotel_camels_subset_trends.RDS")
```

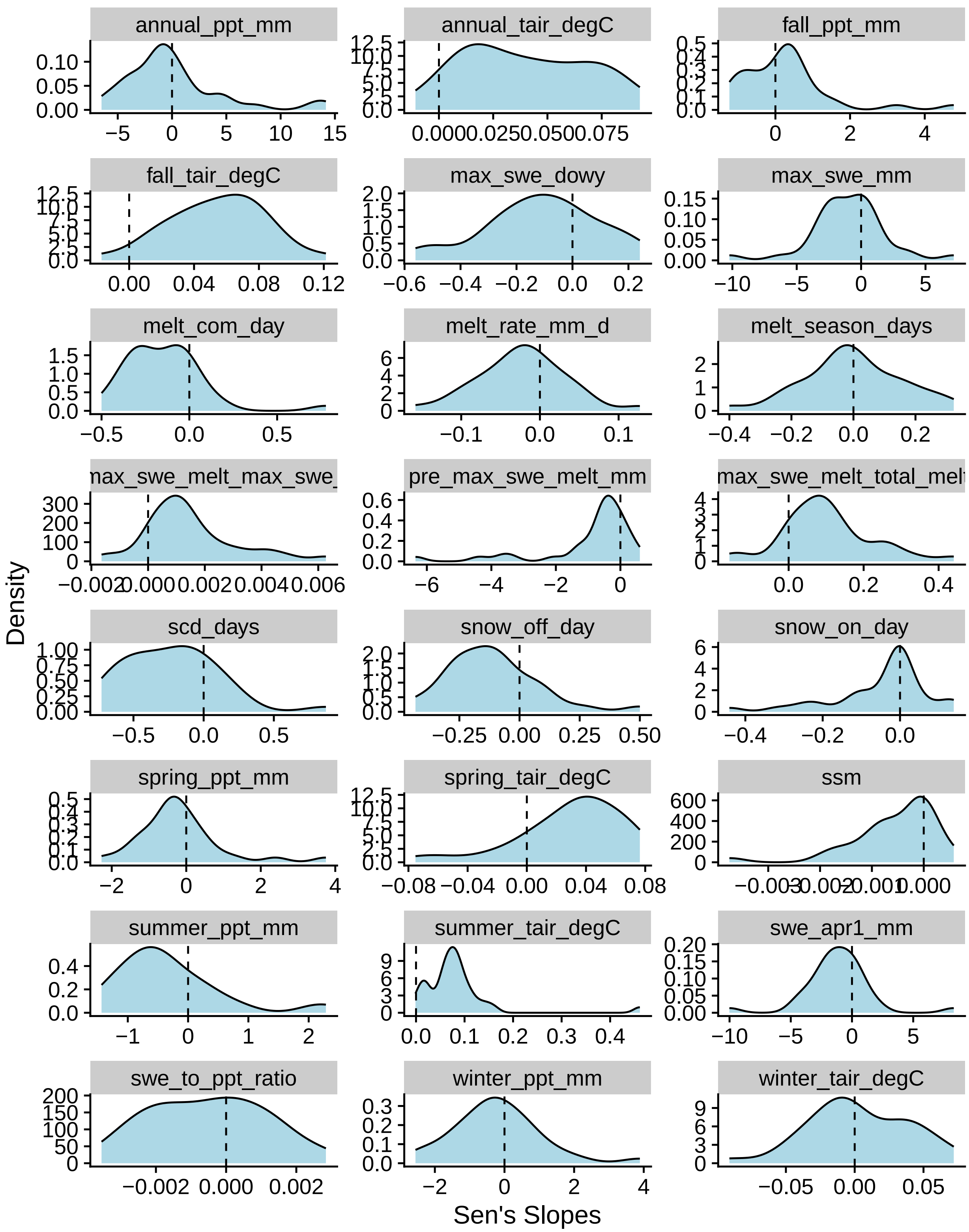
We can view distributions of the Mann-Kendall p-values and Sen's slopes.

```
ggplot(trends, aes(mk_p)) +
  geom_density(fill = "lightblue") +
  geom_vline(xintercept = 0.05, lty = "dashed") +
  facet_wrap(~metric, ncol = 3, scales = "free") +
  labs(x = "Mann-Kendall p-values", y = "Density")
```




```r
ggplot(trends, aes(sen_slope)) +
  geom_density(fill = "lightblue") +
  geom_vline(xintercept = 0, lty = "dashed") +
```

```r
facet_wrap(~metric, ncol = 3, scales = "free") +
labs(x = "Sen's Slopes", y = "Density")
```

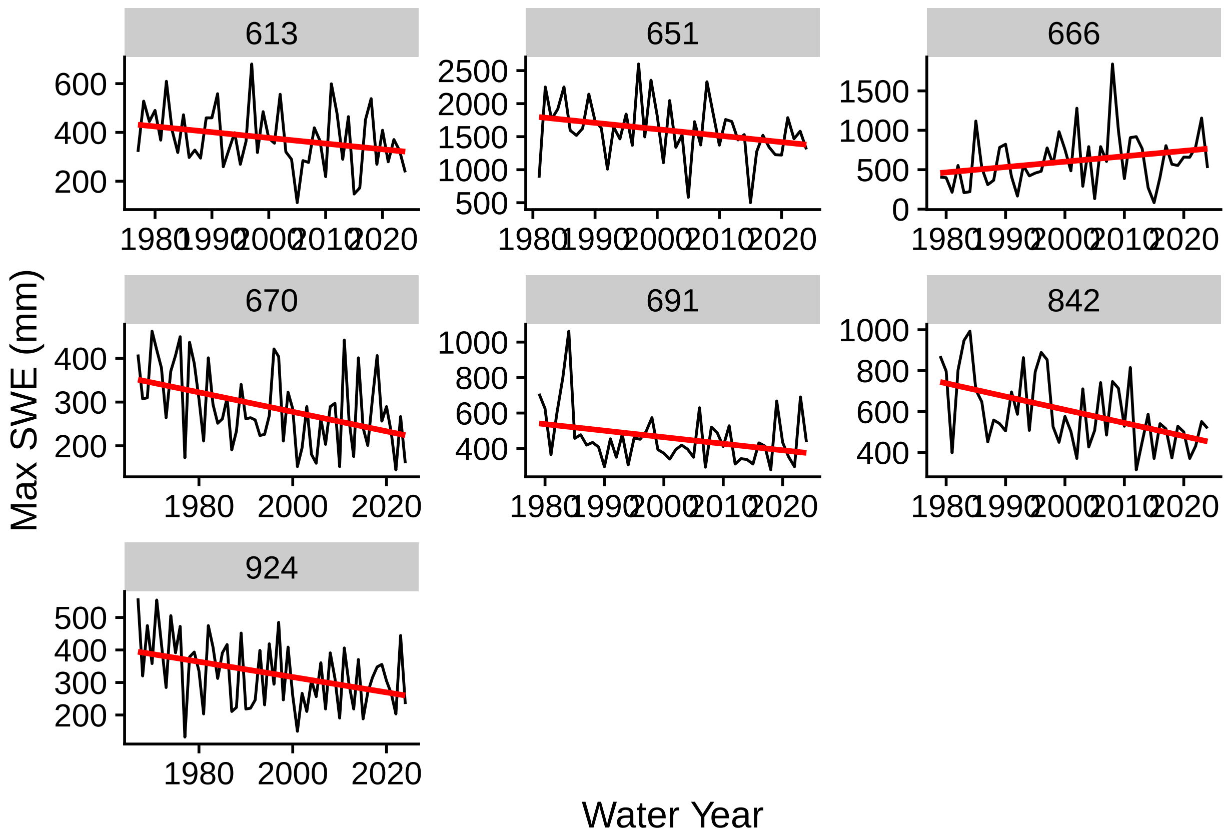


We can now re-examine some of the previous plots, looking at only sites with significant changes.

Maximum SWE with statistically siginificant trends

```r
p_thresh = 0.05
metrics %>%
  filter(site_id %in% filter(trends, metric == "max_swe_mm" & mk_p < 0.05)$site_id) %>%
  ggplot(aes(wyear, max_swe_mm)) +
  geom_line() +
  geom_smooth(method = "lm", se = F, color = "red") +
  facet_wrap(~as.factor(site_id), scales = "free") +
  labs(x = "Water Year", y = "Max SWE (mm)")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

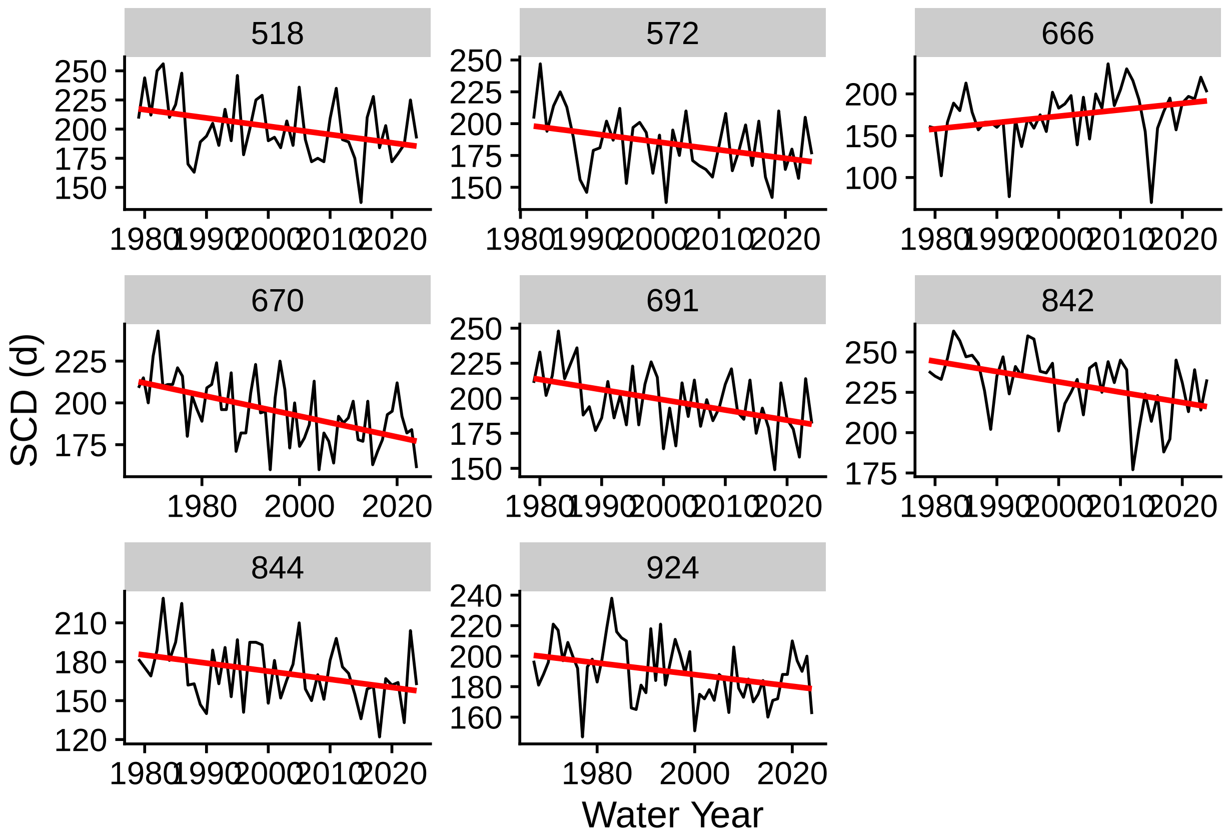


SCD with statistically siginificant trends

```r
metrics %>%
  filter(site_id %in% filter(trends, metric == "scd_days" & mk_p < 0.05)$site_id) %>%
  ggplot(aes(wyear, scd_days)) +
  geom_line() +
```

```r
  geom_smooth(method = "lm", se = F, color = "red") +
  facet_wrap(~as.factor(site_id), scales = "free") +
  labs(x = "Water Year", y = "SCD (d)")
```

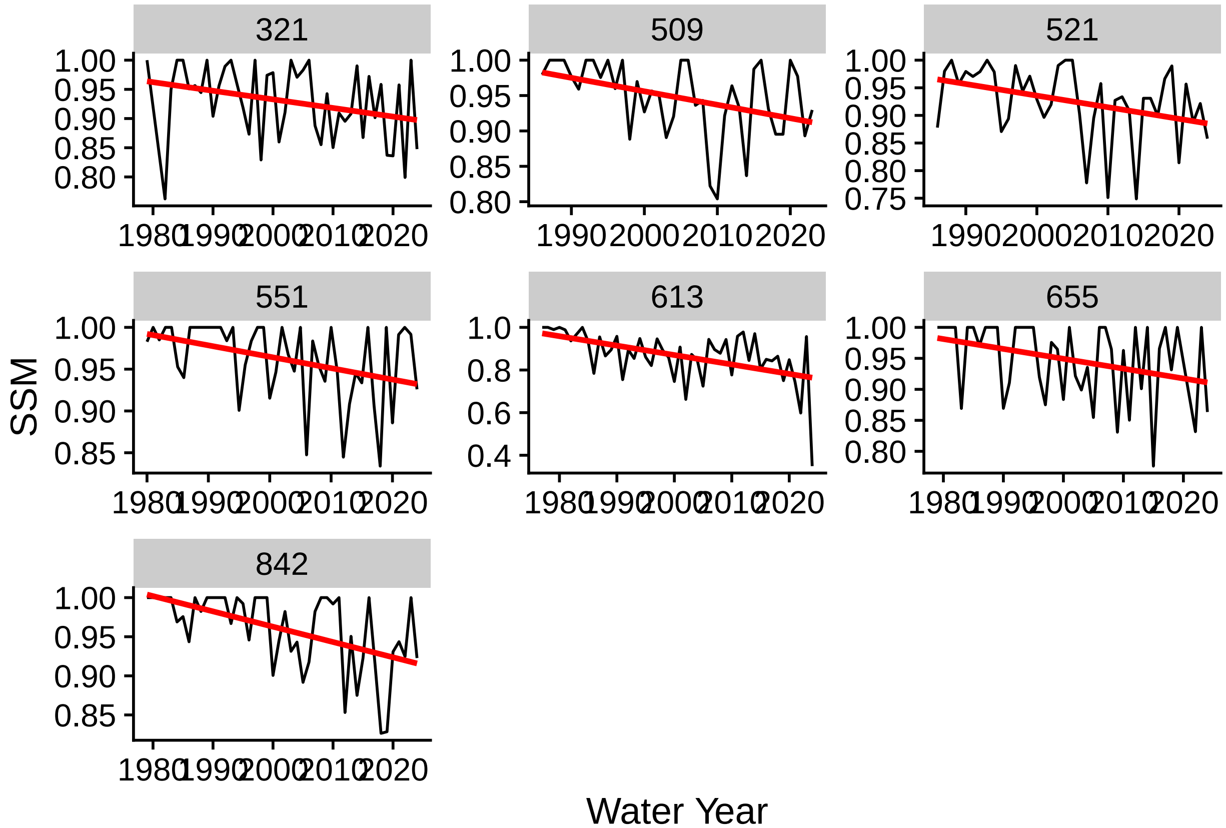
```
## `geom_smooth()` using formula = 'y ~ x'
```



Snow seasonality metric (SSM) with statistically siginificant trends

```r
metrics %>%
  filter(site_id %in% filter(trends, metric == "ssm" & mk_p < 0.05)$site_id) %>%
  ggplot(aes(wyear, ssm)) +
  geom_line() +
  geom_smooth(method = "lm", se = F, color = "red") +
  facet_wrap(~as.factor(site_id), scales = "free") +
  labs(x = "Water Year", y = "SSM")
```

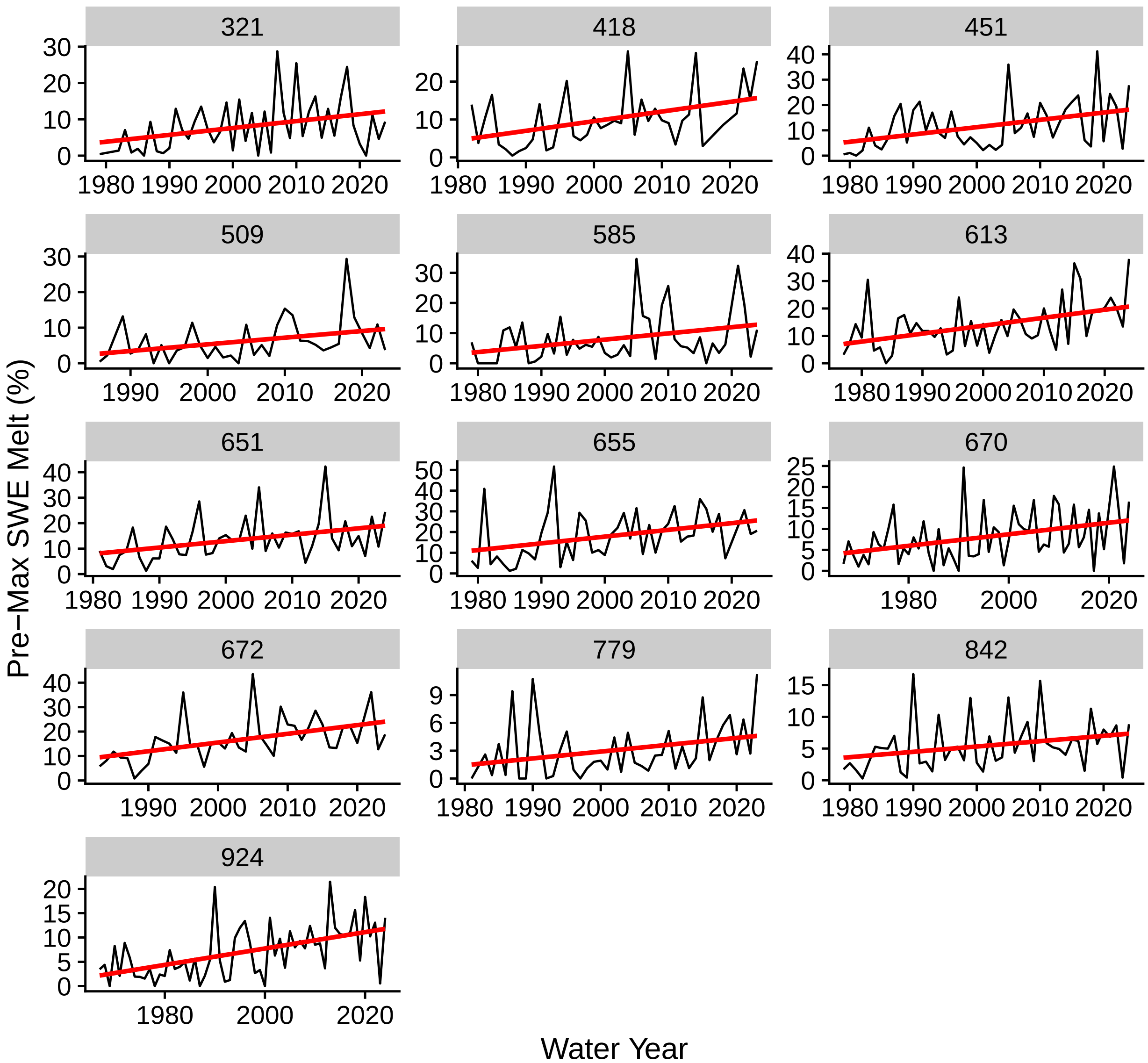
```
## `geom_smooth()` using formula = 'y ~ x'
```



Pre-max SWE snowmelt as percent of total snowmelt with statistically siginificant trends

```
metrics %>%
  filter(site_id %in% filter(trends, metric == "pre_max_swe_melt_total_melt_pct" & mk_p < 0.05)$site_id
  ggplot(aes(wyear, pre_max_swe_melt_total_melt_pct)) +
  geom_line() +
  geom_smooth(method = "lm", se = F, color = "red") +
  facet_wrap(~as.factor(site_id), scales = "free", ncol =3) +
  labs(x = "Water Year", y = "Pre-Max SWE Melt (%)")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



We can summarize the trends even further to see which ones have the most prevalent statistically significant results.

```
p_thresh = 0.05
trend_summary <- trends %>%
  group_by(metric) %>%
  summarize(mk_pct_sig = (sum(mk_p < p_thresh) / n()) * 100,
            sen_slope_av = mean(sen_slope),
            sen_slope_av_sig = mean(sen_slope[mk_p < p_thresh]))
trend_summary %>%
  arrange(-mk_pct_sig)
```

```
## # A tibble: 24 x 4
##    metric                               mk_pct_sig sen_slope_av sen_slope_av_sig
##    <chr>                                     <dbl>        <dbl>            <dbl>
##  1 summer_tair_degC                           60.6       0.0815            0.110
##  2 annual_tair_degC                           51.5       0.0397            0.0622
```

```
##  3 fall_tair_degC                        48.5      0.0542           0.0729
##  4 pre_max_swe_melt_max_swe_ratio        42.4      0.00147          0.00284
##  5 pre_max_swe_melt_mm                   39.4     -0.910           -1.94
##  6 pre_max_swe_melt_total_melt_pct       39.4      0.102            0.210
##  7 scd_days                              24.2     -0.224           -0.432
##  8 spring_tair_degC                      24.2      0.0296           0.0623
##  9 winter_tair_degC                      24.2      0.00300          0.0176
## 10 max_swe_mm                            21.2     -1.01            -2.78
## # i 14 more rows
```

Now that we've looked at these data, we'll explore the use of spatial SWE information in our work.