CKD Case Definition 1: ESRD with transplant

```
library(tidyverse)
library(bigrquery)

# This query represents dataset "kidney_transplant" for domain
"condition" and was generated for All of Us Controlled Tier Dataset v7
dataset_93801899_condition_sql <- paste("
    SELECT
        c_occurrence.person_id,
        c_occurrence.condition_concept_id,
        c_standard_concept.concept_name as standard_concept_name,
        c_standard_concept.concept_code as standard_concept_code,
        c_standard_concept.vocabulary_id as standard_vocabulary
    FROM
        ( SELECT
            *
        FROM
            `condition_occurrence` c_occurrence
        WHERE
            (
                condition_concept_id IN (SELECT
                    DISTINCT c.concept_id
                FROM
                    `cb_criteria` c
                JOIN
                    (SELECT
                        CAST(cr.id as string) AS id
                    FROM
                        `cb_criteria` cr
                    WHERE
                        concept_id IN (199991, 42539502)
                        AND full_text LIKE '%_rank1]%'      ) a
                        ON (c.path LIKE CONCAT('%.', a.id, '.%')
                        OR c.path LIKE CONCAT('%.', a.id)
                        OR c.path LIKE CONCAT(a.id, '.%')
                        OR c.path = a.id)
                WHERE
                    is_standard = 1
                    AND is_selectable = 1)
            )
            AND (
                c_occurrence.PERSON_ID IN (SELECT
                    distinct person_id
                FROM
                    `cb_search_person` cb_search_person
                WHERE
```

```
                    cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_whole_genome_variant = 1 )
                    AND cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_ehr_data = 1 ) )
            )) c_occurrence
    LEFT JOIN
        `concept` c_standard_concept
           ON c_occurrence.condition_concept_id =
c_standard_concept.concept_id", sep="")

# Formulate a Cloud Storage destination path for the data exported
from BigQuery.
# NOTE: By default data exported multiple times on the same day will
overwrite older copies.
#       But data exported on a different days will write to a new
location so that historical
#       copies can be kept as the dataset definition is changed.
condition_93801899_path <- file.path(
  Sys.getenv("WORKSPACE_BUCKET"),
  "bq_exports",
  Sys.getenv("OWNER_EMAIL"),
  strftime(lubridate::now(), "%Y%m%d"),  # Comment out this line if
you want the export to always overwrite.
  "condition_93801899",
  "condition_93801899_*.csv")
message(str_glue('The data will be written to
{condition_93801899_path}. Use this path when reading ',
                 'the data into your notebooks in the future.'))

# Perform the query and export the dataset to Cloud Storage as CSV
files.
# NOTE: You only need to run `bq_table_save` once. After that, you can
#       just read data from the CSVs in Cloud Storage.
bq_table_save(
  bq_dataset_query(Sys.getenv("WORKSPACE_CDR"),
dataset_93801899_condition_sql, billing =
Sys.getenv("GOOGLE_PROJECT")),
  condition_93801899_path,
  destination_format = "CSV")


# Read the data directly from Cloud Storage into memory.
```

```
# NOTE: Alternatively you can `gsutil -m cp {condition_93801899_path}`
to copy these files
#       to the Jupyter disk.
read_bq_export_from_workspace_bucket <- function(export_path) {
  col_types <- cols(standard_concept_name = col_character(),
standard_concept_code = col_character(), standard_vocabulary =
col_character())
  bind_rows(
    map(system2('gsutil', args = c('ls', export_path), stdout = TRUE,
stderr = TRUE),
        function(csv) {
          message(str_glue('Loading {csv}.'))
          chunk <- read_csv(pipe(str_glue('gsutil cat {csv}')),
col_types = col_types, show_col_types = FALSE)
          if (is.null(col_types)) {
            col_types <- spec(chunk)
          }
          chunk
        }))
}
kidney_transplant_condition_df <-
read_bq_export_from_workspace_bucket(condition_93801899_path)

unique(kidney_transplant_condition_df$standard_concept_name)

# This query represents dataset "kidney_transplant" for domain
"procedure" and was generated for All of Us Controlled Tier Dataset v7
dataset_93801899_procedure_sql <- paste("
    SELECT
        procedure.person_id,
        procedure.procedure_concept_id,
        p_standard_concept.concept_name as standard_concept_name,
        p_standard_concept.concept_code as standard_concept_code,
        p_standard_concept.vocabulary_id as standard_vocabulary
    FROM
        ( SELECT
            *
        FROM
            `procedure_occurrence` procedure
        WHERE
            (
                procedure_concept_id IN (SELECT
                    DISTINCT c.concept_id
                FROM
                    `cb_criteria` c
                JOIN
                    (SELECT
                        CAST(cr.id as string) AS id
                    FROM
```

```
                        `cb_criteria` cr
                    WHERE
                        concept_id IN (4322471)
                        AND full_text LIKE '%_rank1]%'      ) a
                        ON (c.path LIKE CONCAT('%.', a.id, '.%')
                        OR c.path LIKE CONCAT('%.', a.id)
                        OR c.path LIKE CONCAT(a.id, '.%')
                        OR c.path = a.id)
                WHERE
                    is_standard = 1
                    AND is_selectable = 1)
            )
            AND (
                procedure.PERSON_ID IN (SELECT
                    distinct person_id
                FROM
                    `cb_search_person` cb_search_person
                WHERE
                    cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_whole_genome_variant = 1 )
                    AND cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_ehr_data = 1 ) )
            )) procedure
    LEFT JOIN
        `concept` p_standard_concept
            ON procedure.procedure_concept_id =
p_standard_concept.concept_id", sep="")

# Formulate a Cloud Storage destination path for the data exported
from BigQuery.
# NOTE: By default data exported multiple times on the same day will
overwrite older copies.
#       But data exported on a different days will write to a new
location so that historical
#       copies can be kept as the dataset definition is changed.
procedure_93801899_path <- file.path(
  Sys.getenv("WORKSPACE_BUCKET"),
  "bq_exports",
  Sys.getenv("OWNER_EMAIL"),
  strftime(lubridate::now(), "%Y%m%d"),  # Comment out this line if
you want the export to always overwrite.
```

```
  "procedure_93801899",
  "procedure_93801899_*.csv")
message(str_glue('The data will be written to
{procedure_93801899_path}. Use this path when reading ',
                 'the data into your notebooks in the future.'))

# Perform the query and export the dataset to Cloud Storage as CSV
files.
# NOTE: You only need to run `bq_table_save` once. After that, you can
#       just read data from the CSVs in Cloud Storage.
bq_table_save(
  bq_dataset_query(Sys.getenv("WORKSPACE_CDR"),
dataset_93801899_procedure_sql, billing =
Sys.getenv("GOOGLE_PROJECT")),
  procedure_93801899_path,
  destination_format = "CSV")


# Read the data directly from Cloud Storage into memory.
# NOTE: Alternatively you can `gsutil -m cp {procedure_93801899_path}`
to copy these files
#       to the Jupyter disk.
read_bq_export_from_workspace_bucket <- function(export_path) {
  col_types <- cols(standard_concept_name = col_character(),
standard_concept_code = col_character(), standard_vocabulary =
col_character())
  bind_rows(
    map(system2('gsutil', args = c('ls', export_path), stdout = TRUE,
stderr = TRUE),
        function(csv) {
          message(str_glue('Loading {csv}.'))
          chunk <- read_csv(pipe(str_glue('gsutil cat {csv}')),
col_types = col_types, show_col_types = FALSE)
          if (is.null(col_types)) {
            col_types <- spec(chunk)
          }
          chunk
        }))
}
kidney_transplant_procedure_df <-
read_bq_export_from_workspace_bucket(procedure_93801899_path)

unique(kidney_transplant_procedure_df$standard_concept_name)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.5.2     ✔ tibble    3.2.1
✔ lubridate 1.9.4     ✔ tidyr     1.3.1
```

```
✔ purrr     1.0.4
── Conflicts ─────────────────────────────────────────
tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force
all conflicts to become errors
The data will be written to gs://fc-secure-672eeb92-4859-4ed9-9f59-
d4349f3534a0/bq_exports/micah_hysong@researchallofus.org/20250825/
condition_93801899/condition_93801899_*.csv. Use this path when
reading the data into your notebooks in the future.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/condition_93801899/
condition_93801899_000000000000.csv.


 [1] "Chronic rejection of renal transplant"
 [2] "Renal transplant rejection"
 [3] "Delayed renal graft function"
 [4] "Acute rejection of renal transplant - grade I"
 [5] "Transplant renal artery stenosis"
 [6] "Recurrent post-transplant renal disease"
 [7] "Failed renal transplant"
 [8] "Acute rejection of renal transplant"
 [9] "Transplant glomerulopathy"
[10] "Transplanted kidney present"
[11] "Disorder of transplanted kidney"

The data will be written to gs://fc-secure-672eeb92-4859-4ed9-9f59-
d4349f3534a0/bq_exports/micah_hysong@researchallofus.org/20250825/
procedure_93801899/procedure_93801899_*.csv. Use this path when
reading the data into your notebooks in the future.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/procedure_93801899/
procedure_93801899_000000000000.csv.


[1] "Transplant of kidney"          "Autotransplantation of kidney"
```

```
kidney_transplant <-
unique(c(kidney_transplant_condition_df$person_id,
kidney_transplant_procedure_df$person_id))
case_definition_1 <-
unique(c(kidney_transplant_condition_df$person_id,
kidney_transplant_procedure_df$person_id))
length(case_definition_1)
```

```
[1] 2253
```

CKD Case Definition 2: ESRD on dialysis

```
library(tidyverse)
library(bigrquery)

# This query represents dataset "dialysis" for domain "condition" and
was generated for All of Us Controlled Tier Dataset v7
dataset_04318565_condition_sql <- paste("
    SELECT
        c_occurrence.person_id,
        c_occurrence.condition_concept_id,
        c_standard_concept.concept_name as standard_concept_name,
        c_standard_concept.concept_code as standard_concept_code,
        c_standard_concept.vocabulary_id as standard_vocabulary,
        c_occurrence.condition_start_datetime
    FROM
        ( SELECT
            *
        FROM
            `condition_occurrence` c_occurrence
        WHERE
            (
                condition_concept_id IN (SELECT
                    DISTINCT c.concept_id
                FROM
                    `cb_criteria` c
                JOIN
                    (SELECT
                        CAST(cr.id as string) AS id
                    FROM
                        `cb_criteria` cr
                    WHERE
                        concept_id IN (4027133, 43021247)
                        AND full_text LIKE '%_rank1]%'      ) a
                        ON (c.path LIKE CONCAT('%.', a.id, '.%')
                        OR c.path LIKE CONCAT('%.', a.id)
                        OR c.path LIKE CONCAT(a.id, '.%')
                        OR c.path = a.id)
                WHERE
                    is_standard = 1
                    AND is_selectable = 1)
            )
            AND (
                c_occurrence.PERSON_ID IN (SELECT
                    distinct person_id
                FROM
```

```
                    `cb_search_person` cb_search_person
                WHERE
                    cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_whole_genome_variant = 1 )
                    AND cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_ehr_data = 1 ) )
            )) c_occurrence
    LEFT JOIN
        `concept` c_standard_concept
            ON c_occurrence.condition_concept_id =
c_standard_concept.concept_id", sep="")

# Formulate a Cloud Storage destination path for the data exported
from BigQuery.
# NOTE: By default data exported multiple times on the same day will
overwrite older copies.
#       But data exported on a different days will write to a new
location so that historical
#       copies can be kept as the dataset definition is changed.
condition_04318565_path <- file.path(
  Sys.getenv("WORKSPACE_BUCKET"),
  "bq_exports",
  Sys.getenv("OWNER_EMAIL"),
  strftime(lubridate::now(), "%Y%m%d"),  # Comment out this line if
you want the export to always overwrite.
  "condition_04318565",
  "condition_04318565_*.csv")
message(str_glue('The data will be written to
{condition_04318565_path}. Use this path when reading ',
                 'the data into your notebooks in the future.'))

# Perform the query and export the dataset to Cloud Storage as CSV
files.
# NOTE: You only need to run `bq_table_save` once. After that, you can
#       just read data from the CSVs in Cloud Storage.
bq_table_save(
  bq_dataset_query(Sys.getenv("WORKSPACE_CDR"),
dataset_04318565_condition_sql, billing =
Sys.getenv("GOOGLE_PROJECT")),
  condition_04318565_path,
  destination_format = "CSV")
```

```
# Read the data directly from Cloud Storage into memory.
# NOTE: Alternatively you can `gsutil -m cp {condition_04318565_path}`
to copy these files
#       to the Jupyter disk.
read_bq_export_from_workspace_bucket <- function(export_path) {
  col_types <- cols(standard_concept_name = col_character(),
standard_concept_code = col_character(), standard_vocabulary =
col_character())
  bind_rows(
    map(system2('gsutil', args = c('ls', export_path), stdout = TRUE,
stderr = TRUE),
        function(csv) {
          message(str_glue('Loading {csv}.'))
          chunk <- read_csv(pipe(str_glue('gsutil cat {csv}')),
col_types = col_types, show_col_types = FALSE)
          if (is.null(col_types)) {
            col_types <- spec(chunk)
          }
          chunk
        }))
}
dialysis_condition_df <-
read_bq_export_from_workspace_bucket(condition_04318565_path)
unique(dialysis_condition_df$standard_concept_name)

# This query represents dataset "dialysis" for domain "observation"
and was generated for All of Us Controlled Tier Dataset v7
dataset_04318565_observation_sql <- paste("
    SELECT
        observation.person_id,
        observation.observation_concept_id,
        o_standard_concept.concept_name as standard_concept_name,
        o_standard_concept.concept_code as standard_concept_code,
        o_standard_concept.vocabulary_id as standard_vocabulary,
        observation.observation_datetime
    FROM
        ( SELECT
            *
        FROM
            `observation` observation
        WHERE
            (
                observation_concept_id IN (4019967, 4059475, 4090651,
4301680, 46270032)
            )
            AND (
                observation.PERSON_ID IN (SELECT
                    distinct person_id
```

```
                FROM
                    `cb_search_person` cb_search_person
                WHERE
                    cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_whole_genome_variant = 1 )
                    AND cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_ehr_data = 1 ) )
            )) observation
    LEFT JOIN
        `concept` o_standard_concept
            ON observation.observation_concept_id =
o_standard_concept.concept_id", sep="")

# Formulate a Cloud Storage destination path for the data exported
from BigQuery.
# NOTE: By default data exported multiple times on the same day will
overwrite older copies.
#       But data exported on a different days will write to a new
location so that historical
#       copies can be kept as the dataset definition is changed.
observation_04318565_path <- file.path(
  Sys.getenv("WORKSPACE_BUCKET"),
  "bq_exports",
  Sys.getenv("OWNER_EMAIL"),
  strftime(lubridate::now(), "%Y%m%d"),  # Comment out this line if
you want the export to always overwrite.
  "observation_04318565",
  "observation_04318565_*.csv")
message(str_glue('The data will be written to
{observation_04318565_path}. Use this path when reading ',
                 'the data into your notebooks in the future.'))

# Perform the query and export the dataset to Cloud Storage as CSV
files.
# NOTE: You only need to run `bq_table_save` once. After that, you can
#       just read data from the CSVs in Cloud Storage.
bq_table_save(
  bq_dataset_query(Sys.getenv("WORKSPACE_CDR"),
dataset_04318565_observation_sql, billing =
Sys.getenv("GOOGLE_PROJECT")),
  observation_04318565_path,
```

```
  destination_format = "CSV")


# Read the data directly from Cloud Storage into memory.
# NOTE: Alternatively you can `gsutil -m cp
{observation_04318565_path}` to copy these files
#       to the Jupyter disk.
read_bq_export_from_workspace_bucket <- function(export_path) {
  col_types <- cols(standard_concept_name = col_character(),
standard_concept_code = col_character(), standard_vocabulary =
col_character())
  bind_rows(
    map(system2('gsutil', args = c('ls', export_path), stdout = TRUE,
stderr = TRUE),
        function(csv) {
          message(str_glue('Loading {csv}.'))
          chunk <- read_csv(pipe(str_glue('gsutil cat {csv}')),
col_types = col_types, show_col_types = FALSE)
          if (is.null(col_types)) {
            col_types <- spec(chunk)
          }
          chunk
        }))
}
dialysis_observation_df <-
read_bq_export_from_workspace_bucket(observation_04318565_path)
unique(dialysis_observation_df$standard_concept_name)

# This query represents dataset "dialysis" for domain "procedure" and
was generated for All of Us Controlled Tier Dataset v7
dataset_04318565_procedure_sql <- paste("
    SELECT
        procedure.person_id,
        procedure.procedure_concept_id,
        p_standard_concept.concept_name as standard_concept_name,
        p_standard_concept.concept_code as standard_concept_code,
        p_standard_concept.vocabulary_id as standard_vocabulary,
        procedure.procedure_datetime
    FROM
        ( SELECT
            *
        FROM
            `procedure_occurrence` procedure
        WHERE
            (
                procedure_concept_id IN (SELECT
                    DISTINCT c.concept_id
                FROM
                    `cb_criteria` c
                JOIN
```

```
                    (SELECT
                        CAST(cr.id as string) AS id
                    FROM
                        `cb_criteria` cr
                    WHERE
                        concept_id IN (4032243)
                        AND full_text LIKE '%_rank1]%'      ) a
                        ON (c.path LIKE CONCAT('%.', a.id, '.%')
                        OR c.path LIKE CONCAT('%.', a.id)
                        OR c.path LIKE CONCAT(a.id, '.%')
                        OR c.path = a.id)
                WHERE
                    is_standard = 1
                    AND is_selectable = 1)
            )
            AND (
                procedure.PERSON_ID IN (SELECT
                    distinct person_id
                FROM
                    `cb_search_person` cb_search_person
                WHERE
                    cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_whole_genome_variant = 1 )
                    AND cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_ehr_data = 1 ) )
            )) procedure
    LEFT JOIN
        `concept` p_standard_concept
            ON procedure.procedure_concept_id =
p_standard_concept.concept_id", sep="")

# Formulate a Cloud Storage destination path for the data exported
from BigQuery.
# NOTE: By default data exported multiple times on the same day will
overwrite older copies.
#       But data exported on a different days will write to a new
location so that historical
#       copies can be kept as the dataset definition is changed.
procedure_04318565_path <- file.path(
  Sys.getenv("WORKSPACE_BUCKET"),
  "bq_exports",
```

```
  Sys.getenv("OWNER_EMAIL"),
  strftime(lubridate::now(), "%Y%m%d"),  # Comment out this line if
you want the export to always overwrite.
  "procedure_04318565",
  "procedure_04318565_*.csv")
message(str_glue('The data will be written to
{procedure_04318565_path}. Use this path when reading ',
                 'the data into your notebooks in the future.'))

# Perform the query and export the dataset to Cloud Storage as CSV
files.
# NOTE: You only need to run `bq_table_save` once. After that, you can
#       just read data from the CSVs in Cloud Storage.
bq_table_save(
  bq_dataset_query(Sys.getenv("WORKSPACE_CDR"),
dataset_04318565_procedure_sql, billing =
Sys.getenv("GOOGLE_PROJECT")),
  procedure_04318565_path,
  destination_format = "CSV")


# Read the data directly from Cloud Storage into memory.
# NOTE: Alternatively you can `gsutil -m cp {procedure_04318565_path}`
to copy these files
#       to the Jupyter disk.
read_bq_export_from_workspace_bucket <- function(export_path) {
  col_types <- cols(standard_concept_name = col_character(),
standard_concept_code = col_character(), standard_vocabulary =
col_character())
  bind_rows(
    map(system2('gsutil', args = c('ls', export_path), stdout = TRUE,
stderr = TRUE),
        function(csv) {
          message(str_glue('Loading {csv}.'))
          chunk <- read_csv(pipe(str_glue('gsutil cat {csv}')),
col_types = col_types, show_col_types = FALSE)
          if (is.null(col_types)) {
            col_types <- spec(chunk)
          }
          chunk
        }))
}
dialysis_procedure_df <-
read_bq_export_from_workspace_bucket(procedure_04318565_path)
unique(dialysis_procedure_df$standard_concept_name)
```

```
The data will be written to gs://fc-secure-672eeb92-4859-4ed9-9f59-
d4349f3534a0/bq_exports/micah_hysong@researchallofus.org/20250825/
condition_04318565/condition_04318565_*.csv. Use this path when
reading the data into your notebooks in the future.
```

```
Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/condition_04318565/
condition_04318565_000000000000.csv.


 [1] "Mechanical complication of dialysis catheter"
 [2] "Infection of hemodialysis arteriovenous fistula"
 [3] "Complication of dialysis"
 [4] "Complication of renal dialysis"
 [5] "Leakage of peritoneal dialysis catheter"
 [6] "Complication associated with dialysis catheter"
 [7] "Infection associated with peritoneal dialysis catheter"
 [8] "Peritoneal dialysis-associated peritonitis"
 [9] "Migration of peritoneal dialysis catheter"
[10] "Mechanical complication of peritoneal dialysis catheter"
[11] "Peritoneal dialysis catheter exit site infection"
[12] "Malfunction of peritoneal dialysis catheter"

The data will be written to gs://fc-secure-672eeb92-4859-4ed9-9f59-
d4349f3534a0/bq_exports/micah_hysong@researchallofus.org/20250825/
observation_04318565/observation_04318565_*.csv. Use this path when
reading the data into your notebooks in the future.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/observation_04318565/
observation_04318565_000000000000.csv.


[1] "Dependence on renal dialysis"       "Non-compliance with renal
dialysis"
[3] "Dialysis finding"                   "Dialysis care"

The data will be written to gs://fc-secure-672eeb92-4859-4ed9-9f59-
d4349f3534a0/bq_exports/micah_hysong@researchallofus.org/20250825/
procedure_04318565/procedure_04318565_*.csv. Use this path when
reading the data into your notebooks in the future.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/procedure_04318565/
procedure_04318565_000000000000.csv.


[1] "Extracorporeal membrane oxygenation"
[2] "Peritoneal dialysis"
[3] "Peritoneal dialysis catheter maintenance"
[4] "Dialysis procedure"
[5] "Ultrafiltration"
```

```
[6] "Hemodialysis"
[7] "Continuous venovenous hemodialysis"
[8] "Renal dialysis"
```

```
names(dialysis_condition_df) <- c("person_id", "concept_id",
"standard_concept_name", "standard_concept_code",
                                  "standard_vocaabulary", "date_time")
names(dialysis_observation_df) <- c("person_id", "concept_id",
"standard_concept_name", "standard_concept_code",
                                  "standard_vocaabulary", "date_time")
names(dialysis_procedure_df) <- c("person_id", "concept_id",
"standard_concept_name", "standard_concept_code",
                                  "standard_vocaabulary", "date_time")
dialysis_1 <- rbind(dialysis_condition_df, dialysis_observation_df)
dialysis <- rbind(dialysis_1, dialysis_procedure_df)
```

```
library(tidyverse)
library(bigrquery)

# This query represents dataset "acute_kidney" for domain "condition"
and was generated for All of Us Controlled Tier Dataset v7
dataset_99489694_condition_sql <- paste("
    SELECT
        c_occurrence.person_id,
        c_occurrence.condition_concept_id,
        c_standard_concept.concept_name as standard_concept_name,
        c_standard_concept.concept_code as standard_concept_code,
        c_standard_concept.vocabulary_id as standard_vocabulary,
        c_occurrence.condition_start_datetime
    FROM
        ( SELECT
            *
        FROM
            `condition_occurrence` c_occurrence
        WHERE
            (
                condition_concept_id IN (SELECT
                    DISTINCT c.concept_id
                FROM
                    `cb_criteria` c
                JOIN
                    (SELECT
                        CAST(cr.id as string) AS id
                    FROM
                        `cb_criteria` cr
                    WHERE
                        concept_id IN (197320)
                        AND full_text LIKE '%_rank1]%'      ) a
                        ON (c.path LIKE CONCAT('%.', a.id, '.%')
                        OR c.path LIKE CONCAT('%.', a.id)
```

```
                        OR c.path LIKE CONCAT(a.id, '.%')
                        OR c.path = a.id)
                WHERE
                    is_standard = 1
                    AND is_selectable = 1)
            )
            AND (
                c_occurrence.PERSON_ID IN (SELECT
                    distinct person_id
                FROM
                    `cb_search_person` cb_search_person
                WHERE
                    cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_whole_genome_variant = 1 )
                    AND cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_ehr_data = 1 ) )
            )) c_occurrence
    LEFT JOIN
        `concept` c_standard_concept
            ON c_occurrence.condition_concept_id =
c_standard_concept.concept_id", sep="")

# Formulate a Cloud Storage destination path for the data exported
from BigQuery.
# NOTE: By default data exported multiple times on the same day will
overwrite older copies.
#       But data exported on a different days will write to a new
location so that historical
#       copies can be kept as the dataset definition is changed.
condition_99489694_path <- file.path(
  Sys.getenv("WORKSPACE_BUCKET"),
  "bq_exports",
  Sys.getenv("OWNER_EMAIL"),
  strftime(lubridate::now(), "%Y%m%d"),  # Comment out this line if
you want the export to always overwrite.
  "condition_99489694",
  "condition_99489694_*.csv")
message(str_glue('The data will be written to
{condition_99489694_path}. Use this path when reading ',
                 'the data into your notebooks in the future.'))
```

```r
# Perform the query and export the dataset to Cloud Storage as CSV
files.
# NOTE: You only need to run `bq_table_save` once. After that, you can
#       just read data from the CSVs in Cloud Storage.
bq_table_save(
  bq_dataset_query(Sys.getenv("WORKSPACE_CDR"),
dataset_99489694_condition_sql, billing =
Sys.getenv("GOOGLE_PROJECT")),
  condition_99489694_path,
  destination_format = "CSV")


# Read the data directly from Cloud Storage into memory.
# NOTE: Alternatively you can `gsutil -m cp {condition_99489694_path}`
to copy these files
#       to the Jupyter disk.
read_bq_export_from_workspace_bucket <- function(export_path) {
  col_types <- cols(standard_concept_name = col_character(),
standard_concept_code = col_character(), standard_vocabulary =
col_character())
  bind_rows(
    map(system2('gsutil', args = c('ls', export_path), stdout = TRUE,
stderr = TRUE),
        function(csv) {
          message(str_glue('Loading {csv}.'))
          chunk <- read_csv(pipe(str_glue('gsutil cat {csv}')),
col_types = col_types, show_col_types = FALSE)
          if (is.null(col_types)) {
            col_types <- spec(chunk)
          }
          chunk
        }))
}
acute_kidney_df <-
read_bq_export_from_workspace_bucket(condition_99489694_path)
unique(acute_kidney_df$standard_concept_name)
```

```
The data will be written to gs://fc-secure-672eeb92-4859-4ed9-9f59-
d4349f3534a0/bq_exports/micah_hysong@researchallofus.org/20250825/
condition_99489694/condition_99489694_*.csv. Use this path when
reading the data into your notebooks in the future.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/condition_99489694/
condition_99489694_000000000000.csv.


 [1] "Acute kidney injury due to circulatory failure"
 [2] "Crush syndrome"
```

```
 [3] "Hepatorenal syndrome due to a procedure"
 [4] "Acute kidney injury due to sepsis"
 [5] "Post-delivery acute renal failure with postnatal problem"
 [6] "Acute renal failure due to acute cortical necrosis"
 [7] "Postpartum acute renal failure"
 [8] "Acute renal failure syndrome"
 [9] "Acute renal failure caused by contrast agent"
[10] "Acute renal papillary necrosis with renal failure"
[11] "Hepatorenal syndrome"
[12] "Acute-on-chronic renal failure"
[13] "Acute nontraumatic kidney injury"
[14] "Acute kidney failure stage 3"
```

```
#Remove cases of dialysis that occur within one month of Acute Kidney
injury
library(lubridate)
# Calculate the date range for filtering
dialysis$date_time <- as.POSIXct(dialysis$date_time)
acute_kidney_df$condition_start_datetime <-
as.POSIXct(acute_kidney_df$condition_start_datetime)

dialysis <- dialysis %>%
  mutate(
    lower_bound = date_time - months(1),
    upper_bound = date_time + months(1)
  )



# Perform a cross join and filter
rows_to_remove <- acute_kidney_df %>%
  inner_join(dialysis, by = "person_id", relationship = "many-to-
many") %>%
  filter(condition_start_datetime >= lower_bound &
condition_start_datetime <= upper_bound) %>%
  select(person_id, date_time) %>%
  distinct()

# Remove the identified rows from the dialysis dataframe
dialysis_cleaned <- dialysis %>%
  anti_join(rows_to_remove, by = c("person_id", "date_time"))

dialysis_cleaned <- dialysis_cleaned %>% distinct(person_id, .keep_all
= TRUE)

nrow(dialysis)
nrow(dialysis_cleaned)
```

```
[1] 70158
```

```
[1] 2777
```

```
dialysis<-dialysis_cleaned
case_definition_2<-dialysis_cleaned
```

CKD Case Definition 3: NKF CKD

```
library(tidyverse)
library(bigrquery)

# This query represents dataset "Creatinine" for domain "measurement"
and was generated for All of Us Controlled Tier Dataset v7
dataset_60313913_measurement_sql <- paste("
    SELECT
        measurement.person_id,
        measurement.measurement_concept_id,
        m_standard_concept.concept_name as standard_concept_name,
        m_standard_concept.concept_code as standard_concept_code,
        m_standard_concept.vocabulary_id as standard_vocabulary,
        measurement.measurement_datetime,
        measurement.measurement_type_concept_id,
        m_type.concept_name as measurement_type_concept_name,
        measurement.operator_concept_id,
        m_operator.concept_name as operator_concept_name,
        measurement.value_as_number,
        measurement.value_as_concept_id,
        m_value.concept_name as value_as_concept_name,
        measurement.unit_concept_id,
        m_unit.concept_name as unit_concept_name,
        measurement.range_low,
        measurement.range_high,
        measurement.visit_occurrence_id,
        m_visit.concept_name as visit_occurrence_concept_name,
        measurement.measurement_source_value,
        measurement.measurement_source_concept_id,
        m_source_concept.concept_name as source_concept_name,
        m_source_concept.concept_code as source_concept_code,
        m_source_concept.vocabulary_id as source_vocabulary,
        measurement.unit_source_value,
        measurement.value_source_value
    FROM
        ( SELECT
            *
        FROM
            `measurement` measurement
        WHERE
            (
                measurement_concept_id IN (
                    SELECT
                        DISTINCT c.concept_id
```

```
                    FROM
                        `cb_criteria` c
                    JOIN
                        (
                            SELECT
                                CAST(cr.id as string) AS id
                            FROM
                                `cb_criteria` cr
                            WHERE
                                concept_id IN (
                                    37029387
                                )
                                AND full_text LIKE '%_rank1]%'
                        ) a
                            ON (
                                c.path LIKE CONCAT('%.',
                            a.id,
                            '.%')
                            OR c.path LIKE CONCAT('%.',
                            a.id)
                            OR c.path LIKE CONCAT(a.id,
                            '.%')
                            OR c.path = a.id)
                        WHERE
                            is_standard = 1
                            AND is_selectable = 1
                        )
                )
                AND (
                    measurement.PERSON_ID IN (
                        SELECT
                            distinct person_id
                        FROM
                            `cb_search_person` cb_search_person
                        WHERE
                            cb_search_person.person_id IN (
                                SELECT
                                    person_id
                                FROM
                                    `cb_search_person` p
                                WHERE
                                    has_whole_genome_variant = 1
                            )
                            AND cb_search_person.person_id IN (
                                SELECT
                                    person_id
                                FROM
                                    `cb_search_person` p
                                WHERE
```

```
                                    has_ehr_data = 1
                            )
                        )
                    )
            ) measurement
        LEFT JOIN
            `concept` m_standard_concept
                ON measurement.measurement_concept_id =
m_standard_concept.concept_id
        LEFT JOIN
            `concept` m_type
                ON measurement.measurement_type_concept_id =
m_type.concept_id
        LEFT JOIN
            `concept` m_operator
                ON measurement.operator_concept_id =
m_operator.concept_id
        LEFT JOIN
            `concept` m_value
                ON measurement.value_as_concept_id =
m_value.concept_id
        LEFT JOIN
            `concept` m_unit
                ON measurement.unit_concept_id = m_unit.concept_id
        LEFT JOIn
            `visit_occurrence` v
                ON measurement.visit_occurrence_id =
v.visit_occurrence_id
        LEFT JOIN
            `concept` m_visit
                ON v.visit_concept_id = m_visit.concept_id
        LEFT JOIN
            `concept` m_source_concept
                ON measurement.measurement_source_concept_id =
m_source_concept.concept_id", sep="")

# Formulate a Cloud Storage destination path for the data exported
from BigQuery.
# NOTE: By default data exported multiple times on the same day will
overwrite older copies.
#       But data exported on a different days will write to a new
location so that historical
#       copies can be kept as the dataset definition is changed.
measurement_60313913_path <- file.path(
  Sys.getenv("WORKSPACE_BUCKET"),
  "bq_exports",
  Sys.getenv("OWNER_EMAIL"),
  strftime(lubridate::now(), "%Y%m%d"),  # Comment out this line if
you want the export to always overwrite.
```

```
  "measurement_60313913",
  "measurement_60313913_*.csv")
message(str_glue('The data will be written to
{measurement_60313913_path}. Use this path when reading ',
                 'the data into your notebooks in the future.'))

# Perform the query and export the dataset to Cloud Storage as CSV
files.
# NOTE: You only need to run `bq_table_save` once. After that, you can
#       just read data from the CSVs in Cloud Storage.
bq_table_save(
  bq_dataset_query(Sys.getenv("WORKSPACE_CDR"),
dataset_60313913_measurement_sql, billing =
Sys.getenv("GOOGLE_PROJECT")),
  measurement_60313913_path,
  destination_format = "CSV")

# Read the data directly from Cloud Storage into memory.
# NOTE: Alternatively you can `gsutil -m cp
{measurement_60313913_path}` to copy these files
#       to the Jupyter disk.
read_bq_export_from_workspace_bucket <- function(export_path) {
  col_types <- cols(standard_concept_name = col_character(),
standard_concept_code = col_character(), standard_vocabulary =
col_character(), measurement_type_concept_name = col_character(),
operator_concept_name = col_character(), value_as_concept_name =
col_character(), unit_concept_name = col_character(),
visit_occurrence_concept_name = col_character(),
measurement_source_value = col_character(), source_concept_name =
col_character(), source_concept_code = col_character(),
source_vocabulary = col_character(), unit_source_value =
col_character(), value_source_value = col_character())
  bind_rows(
    map(system2('gsutil', args = c('ls', export_path), stdout = TRUE,
stderr = TRUE),
        function(csv) {
          message(str_glue('Loading {csv}.'))
          chunk <- read_csv(pipe(str_glue('gsutil cat {csv}')),
col_types = col_types, show_col_types = FALSE)
          if (is.null(col_types)) {
            col_types <- spec(chunk)
          }
          chunk
        }))
}
measurement_df <-
read_bq_export_from_workspace_bucket(measurement_60313913_path)

measurement_df <- measurement_df %>%
                filter(unit_source_value == 258797006 |
```

```
unit_source_value == "mg/dL") |>
                      filter(standard_concept_name == "Creatinine
[Mass/volume] in Serum or Plasma")

unique(measurement_df$standard_concept_name)
```

```
The data will be written to gs://fc-secure-672eeb92-4859-4ed9-9f59-
d4349f3534a0/bq_exports/micah_hysong@researchallofus.org/20250825/
measurement_60313913/measurement_60313913_*.csv. Use this path when
reading the data into your notebooks in the future.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/measurement_60313913/
measurement_60313913_000000000000.csv.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/measurement_60313913/
measurement_60313913_000000000001.csv.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/measurement_60313913/
measurement_60313913_000000000002.csv.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/measurement_60313913/
measurement_60313913_000000000003.csv.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/measurement_60313913/
measurement_60313913_000000000004.csv.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/measurement_60313913/
measurement_60313913_000000000005.csv.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/measurement_60313913/
measurement_60313913_000000000006.csv.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/measurement_60313913/
measurement_60313913_000000000007.csv.
```

```
Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/measurement_60313913/
measurement_60313913_000000000008.csv.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/measurement_60313913/
measurement_60313913_000000000009.csv.


[1] "Creatinine [Mass/volume] in Serum or Plasma"
```

```r
# This snippet assumes that you run setup first

# This code copies a file from your Google Bucket into a dataframe

# replace 'test.csv' with the name of the file in your google bucket
(don't delete the quotation marks)
name_of_file_in_bucket <- 'Demographic_and_ancestry_covariates.csv'

################################################################################
##
##
################## DON'T CHANGE FROM HERE
################################
##
################################################################################
##

# Get the bucket name
my_bucket <- Sys.getenv('WORKSPACE_BUCKET')

# Copy the file from current workspace to the bucket
system(paste0("gsutil cp ", my_bucket, "/data/",
name_of_file_in_bucket, " ."), intern=T)

# Load the file into a dataframe
demographics  <- read_csv(name_of_file_in_bucket)

character(0)

Rows: 162193 Columns: 28
── Column specification
————————————————————————————————————————————————————
Delimiter: ","
chr  (8): SexGender, income, education, where_born, military,
healthcare, di...
dbl  (9): person_id, race_unknown, age_today, LGBTQIA, ehr_length,
relative_...
```

```
lgl  (8): AIAN, Asian, Black, Mid, Multiple, PI, White, His
date (3): date_of_birth, min_date, max_date

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.
```

```r
SexGender <- data.frame(person_id = demographics$person_id, SexGender
= demographics$SexGender, DOB = demographics$date_of_birth)
merged_df <- left_join(measurement_df, SexGender, by = "person_id")

# Use snippet 'add_age_to_demographics' to calculate the age of people
in your demographics.
# It assumes the 'Setup' snippet has been executed.
# It also assumes that you got your demographics dataframe from
Dataset Builder

# Note: This snippet calculates current age and does not take into
account whether the person is already dead


## -----[ CHANGE THE DATAFRAME NAME(S) `YOUR_DATASET_NAME_person_df`
TO MATCH YOURS FROM DATASET BUILDER] -----
merged_df <- merged_df %>%
                mutate_if(is.list, as.character) %>%
                mutate(
                    age = year(merged_df$measurement_datetime) -
year(merged_df$DOB) -
                              ifelse(month(merged_df$measurement_datetime)
< month(merged_df$DOB), 1, 0)
                )

merged_df <- merged_df %>%
  mutate(eGFR_2021 = case_when(SexGender == "Cis_male" ~ 142 *
(pmin(value_as_number/0.9, 1)^-0.302) * (pmax(value_as_number/0.9,
1)^-1.200) * (0.9938^as.numeric(age)), # Male
                               SexGender == "Cis_female" ~ 142 *
(pmin(value_as_number/0.7, 1)^-0.241) * (pmax(value_as_number/0.7,
1)^-1.200) * (0.9938^as.numeric(age)) * 1.012, # Female
                               SexGender == "Non-cis" ~ 142 *
(pmin(value_as_number/0.8, 1)^-0.2715) * (pmax(value_as_number/0.8,
1)^-1.200) * (0.9938^as.numeric(age)) * 1.006))

merged_df2 <- merged_df |> select(1:6, 30)
nrow(merged_df2)
```

```
[1] 4214898
```

```r
merged_df2<-drop_na(merged_df2)
```

```r
most_recent_measurements <- merged_df2 %>%
  group_by(person_id) %>%
  slice_max(order_by = measurement_datetime, n = 1)

library(tidyverse)
library(bigrquery)

# This query represents dataset "egfr_disruptors" for domain
"condition" and was generated for All of Us Controlled Tier Dataset v7
dataset_66443155_condition_sql <- paste("
    SELECT
        c_occurrence.person_id,
        c_occurrence.condition_concept_id,
        c_standard_concept.concept_name as standard_concept_name,
        c_standard_concept.concept_code as standard_concept_code,
        c_standard_concept.vocabulary_id as standard_vocabulary,
        c_occurrence.condition_start_datetime
    FROM
        ( SELECT
            *
        FROM
            `condition_occurrence` c_occurrence
        WHERE
            (
                condition_concept_id IN (SELECT
                    DISTINCT c.concept_id
                FROM
                    `cb_criteria` c
                JOIN
                    (SELECT
                        CAST(cr.id as string) AS id
                    FROM
                        `cb_criteria` cr
                    WHERE
                        concept_id IN (132797, 197320, 201965,
37311320, 45770903)
                        AND full_text LIKE '%_rank1]%'      ) a
                        ON (c.path LIKE CONCAT('%.', a.id, '.%')
                        OR c.path LIKE CONCAT('%.', a.id)
                        OR c.path LIKE CONCAT(a.id, '.%')
                        OR c.path = a.id)
                WHERE
                    is_standard = 1
                    AND is_selectable = 1)
            )
            AND (
                c_occurrence.PERSON_ID IN (SELECT
                    distinct person_id
                FROM
                    `cb_search_person` cb_search_person
```

```
                WHERE
                    cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_whole_genome_variant = 1 )
                    AND cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_ehr_data = 1 ) )
            )) c_occurrence
    LEFT JOIN
        `concept` c_standard_concept
           ON c_occurrence.condition_concept_id =
c_standard_concept.concept_id", sep="")

# Formulate a Cloud Storage destination path for the data exported
from BigQuery.
# NOTE: By default data exported multiple times on the same day will
overwrite older copies.
#       But data exported on a different days will write to a new
location so that historical
#       copies can be kept as the dataset definition is changed.
condition_66443155_path <- file.path(
  Sys.getenv("WORKSPACE_BUCKET"),
  "bq_exports",
  Sys.getenv("OWNER_EMAIL"),
  strftime(lubridate::now(), "%Y%m%d"),  # Comment out this line if
you want the export to always overwrite.
  "condition_66443155",
  "condition_66443155_*.csv")
message(str_glue('The data will be written to
{condition_66443155_path}. Use this path when reading ',
                 'the data into your notebooks in the future.'))

# Perform the query and export the dataset to Cloud Storage as CSV
files.
# NOTE: You only need to run `bq_table_save` once. After that, you can
#       just read data from the CSVs in Cloud Storage.
bq_table_save(
  bq_dataset_query(Sys.getenv("WORKSPACE_CDR"),
dataset_66443155_condition_sql, billing =
Sys.getenv("GOOGLE_PROJECT")),
  condition_66443155_path,
  destination_format = "CSV")
```

```r
# Read the data directly from Cloud Storage into memory.
# NOTE: Alternatively you can `gsutil -m cp {condition_66443155_path}`
to copy these files
#       to the Jupyter disk.
read_bq_export_from_workspace_bucket <- function(export_path) {
  col_types <- cols(standard_concept_name = col_character(),
standard_concept_code = col_character(), standard_vocabulary =
col_character())
  bind_rows(
    map(system2('gsutil', args = c('ls', export_path), stdout = TRUE,
stderr = TRUE),
        function(csv) {
          message(str_glue('Loading {csv}.'))
          chunk <- read_csv(pipe(str_glue('gsutil cat {csv}')),
col_types = col_types, show_col_types = FALSE)
          if (is.null(col_types)) {
            col_types <- spec(chunk)
          }
          chunk
        }))
}
egfr_disruptors_df <-
read_bq_export_from_workspace_bucket(condition_66443155_path)

unique(egfr_disruptors_df$standard_concept_name)

#Remove inappropriate code
egfr_disruptors_df<-
egfr_disruptors_df[egfr_disruptors_df$standard_concept_code!
=140031000119103,]
```

```
The data will be written to gs://fc-secure-672eeb92-4859-4ed9-9f59-
d4349f3534a0/bq_exports/micah_hysong@researchallofus.org/20250825/
condition_66443155/condition_66443155_*.csv. Use this path when
reading the data into your notebooks in the future.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_h
ysong@researchallofus.org/20250825/condition_66443155/
condition_66443155_000000000000.csv.


 [1] "Severe sepsis"

 [2] "Sepsis due to Serratia"

 [3] "Gonococcemia"

 [4] "Sepsis due to Escherichia coli"
```

```
 [5] "Acute meningococcemia"
 [6] "Traumatic shock"
 [7] "Sepsis due to Streptococcus"
 [8] "Sepsis due to Enterobacter"
 [9] "Acute kidney injury due to sepsis"
[10] "Septic shock"
[11] "Sepsis due to incomplete miscarriage"
[12] "Sepsis due to coagulase negative Staphylococcus"
[13] "Postpartum acute renal failure"
[14] "Sepsis due to anaerobic bacteria"
[15] "Hypovolemic shock"
[16] "Obstetric shock with postnatal problem"
[17] "Sepsis due to Bacillus anthracis"
[18] "Shock due to anesthesia"
[19] "Obstetric shock - delivered"
[20] "Sepsis due to methicillin resistant Staphylococcus aureus"
[21] "Sepsis due to Candida"
[22] "Puerperal pelvic sepsis"
[23] "Acute kidney failure stage 3"
[24] "Sepsis following molar AND/OR ectopic pregnancy"
[25] "Sepsis due to Gram negative bacteria"
[26] "Sepsis due to methicillin-sensitive Staphylococcus aureus"
[27] "Sepsis due to Salmonella"
[28] "Puerperal sepsis"
[29] "Pyemia"
[30] "Crush syndrome"
```

```
[31] "Sepsis due to Actinomyces"

[32] "Hemorrhagic shock"

[33] "Acute nontraumatic kidney injury"

[34] "Obstetric shock - delivered with postnatal problem"

[35] "Gram positive sepsis"

[36] "Hypovolemia"

[37] "Acute renal failure due to acute cortical necrosis"

[38] "Sepsis due to Staphylococcus"

[39] "Sepsis caused by methicillin susceptible Staphylococcus aureus"

[40] "Sepsis due to Erysipelothrix"

[41] "Shock following molar AND/OR ectopic pregnancy"

[42] "Sepsis due to Pseudomonas"

[43] "Post-delivery acute renal failure with postnatal problem"

[44] "Postoperative hypovolemic shock"

[45] "Sepsis due to urinary tract infection"

[46] "Acute kidney injury due to circulatory failure"

[47] "Septicemic plague"

[48] "Sepsis due to Haemophilus influenzae"

[49] "Sepsis due to Staphylococcus aureus"

[50] "Sepsis caused by Enterococcus"

[51] "Cardiogenic shock"

[52] "Sepsis due to Listeria monocytogenes"

[53] "Acute renal papillary necrosis with renal failure"

[54] "Acute-on-chronic renal failure"

[55] "Sepsis due to disease caused by Severe acute respiratory
syndrome coronavirus 2"
[56] "Hepatorenal syndrome due to a procedure"
```

```
[57] "Hepatorenal syndrome"

[58] "Postoperative shock"

[59] "Sepsis due to Streptococcus pyogenes"

[60] "Sepsis due to Streptococcus pneumoniae"

[61] "Miscarriage with sepsis"

[62] "Acute renal failure syndrome"

[63] "Sepsis without septic shock"

[64] "Sepsis due to Streptococcus group D"

[65] "Sepsis due to Streptococcus agalactiae"

[66] "Prerenal renal failure"

[67] "Anaphylactic shock"

[68] "Acute renal failure caused by contrast agent"

[69] "Sepsis"

[70] "Postoperative septic shock"

[71] "Toxic shock syndrome"

[72] "Puerperal sepsis with postnatal complication"

[73] "Bacterial sepsis"

[74] "Shock during AND/OR following labor AND/OR delivery"

[75] "Streptococcal toxic shock syndrome"

[76] "Shock"

[77] "Sepsis without acute organ dysfunction"

[78] "Meningococcemia"
```

```
#Remove eGFR that co-occur with disruptors

# Calculate the date range for filtering
most_recent_measurements$measurement_datetime <-
as.POSIXct(most_recent_measurements$measurement_datetime)
egfr_disruptors_df$condition_start_datetime <-
as.POSIXct(egfr_disruptors_df$condition_start_datetime)
```

```
most_recent_measurements <- most_recent_measurements %>%
  mutate(
    lower_bound = measurement_datetime - months(1),
    upper_bound = measurement_datetime + months(1)
  )

# Perform a cross join and filter
rows_to_remove <- egfr_disruptors_df %>%
  inner_join(most_recent_measurements, by = "person_id", relationship
= "many-to-many") %>%
  filter(condition_start_datetime >= lower_bound &
condition_start_datetime <= upper_bound) %>%
  select(person_id, measurement_datetime) %>%
  distinct()

most_recent_measurements_cleaned <- most_recent_measurements %>%
  anti_join(rows_to_remove, by = c("person_id",
"measurement_datetime"))

potential_cases<-
most_recent_measurements_cleaned[most_recent_measurements_cleaned$eGFR
_2021<60, ]
nrow(potential_cases)
```

```
[1] 12644
```

```
potential_cases_list<-setdiff(potential_cases$person_id,
dialysis$person_id)
potential_cases_list<-setdiff(potential_cases_list, kidney_transplant)
potential_cases <- potential_cases[potential_cases$person_id %in%
potential_cases_list, ]
nrow(potential_cases)
```

```
[1] 11355
```

Now check if they have another low eGFR 3 months prior

```
# Find the most recent datetime for each person_id
most_recent_measurements_list <- merged_df2 %>%
  group_by(person_id) %>%
  summarize(most_recent_datetime = max(measurement_datetime, na.rm =
TRUE), .groups = 'drop')

# Exclude rows with the most recent measurement_datetime for each
person_id
other_measurements <- merged_df2 %>%
  left_join(most_recent_measurements_list, by = "person_id") %>%
  filter(measurement_datetime != most_recent_datetime) %>%
  select(-most_recent_datetime)
```

```r
#Filter down to only our potential cases
other_measurements <- other_measurements[other_measurements$person_id
%in% potential_cases_list, ]

most_recent_measurements <- most_recent_measurements %>%
  mutate(
    three_months_prior = measurement_datetime - months(3),
  )

other_measurements <- other_measurements |> select(1,6,7)
names(other_measurements) <- c("person_id", "previous_datetime",
"previous_eGFR")

# Perform a cross join and filter
cases_definition3a <- other_measurements %>%
  inner_join(most_recent_measurements, by = "person_id", relationship
= "many-to-many") %>%
  filter(previous_datetime <= three_months_prior) %>%
  filter(previous_eGFR < 60) %>%
  select(person_id) %>%
  distinct()

nrow(cases_definition3a)
```

```
[1] 9400
```

Now check for EHR code

```r
library(tidyverse)
library(bigrquery)

# This query represents dataset "kidney_disease" for domain
"condition" and was generated for All of Us Controlled Tier Dataset v7
dataset_82100016_condition_sql <- paste("
    SELECT
        c_occurrence.person_id,
        c_occurrence.condition_concept_id,
        c_standard_concept.concept_name as standard_concept_name,
        c_standard_concept.concept_code as standard_concept_code,
        c_standard_concept.vocabulary_id as standard_vocabulary
    FROM
        ( SELECT
            *
        FROM
            `condition_occurrence` c_occurrence
        WHERE
            (
                condition_concept_id IN (SELECT
                    DISTINCT c.concept_id
                FROM
```

```
                    `cb_criteria` c
                JOIN
                    (SELECT
                        CAST(cr.id as string) AS id
                    FROM
                        `cb_criteria` cr
                    WHERE
                        concept_id IN (198124, 46271022)
                        AND full_text LIKE '%_rank1]%'      ) a
                        ON (c.path LIKE CONCAT('%.', a.id, '.%')
                        OR c.path LIKE CONCAT('%.', a.id)
                        OR c.path LIKE CONCAT(a.id, '.%')
                        OR c.path = a.id)
                WHERE
                    is_standard = 1
                    AND is_selectable = 1)
            )
            AND (
                c_occurrence.PERSON_ID IN (SELECT
                    distinct person_id
                FROM
                    `cb_search_person` cb_search_person
                WHERE
                    cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_whole_genome_variant = 1 )
                    AND cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `cb_search_person` p
                    WHERE
                        has_ehr_data = 1 ) )
            )) c_occurrence
    LEFT JOIN
        `concept` c_standard_concept
            ON c_occurrence.condition_concept_id =
c_standard_concept.concept_id", sep="")

# Formulate a Cloud Storage destination path for the data exported
from BigQuery.
# NOTE: By default data exported multiple times on the same day will
overwrite older copies.
#       But data exported on a different days will write to a new
location so that historical
#       copies can be kept as the dataset definition is changed.
condition_82100016_path <- file.path(
```

```
  Sys.getenv("WORKSPACE_BUCKET"),
  "bq_exports",
  Sys.getenv("OWNER_EMAIL"),
  strftime(lubridate::now(), "%Y%m%d"),  # Comment out this line if
you want the export to always overwrite.
  "condition_82100016",
  "condition_82100016_*.csv")
message(str_glue('The data will be written to
{condition_82100016_path}. Use this path when reading ',
                 'the data into your notebooks in the future.'))

# Perform the query and export the dataset to Cloud Storage as CSV
files.
# NOTE: You only need to run `bq_table_save` once. After that, you can
#       just read data from the CSVs in Cloud Storage.
bq_table_save(
  bq_dataset_query(Sys.getenv("WORKSPACE_CDR"),
dataset_82100016_condition_sql, billing =
Sys.getenv("GOOGLE_PROJECT")),
  condition_82100016_path,
  destination_format = "CSV")


# Read the data directly from Cloud Storage into memory.
# NOTE: Alternatively you can `gsutil -m cp {condition_82100016_path}`
to copy these files
#       to the Jupyter disk.
read_bq_export_from_workspace_bucket <- function(export_path) {
  col_types <- cols(standard_concept_name = col_character(),
standard_concept_code = col_character(), standard_vocabulary =
col_character())
  bind_rows(
    map(system2('gsutil', args = c('ls', export_path), stdout = TRUE,
stderr = TRUE),
        function(csv) {
          message(str_glue('Loading {csv}.'))
          chunk <- read_csv(pipe(str_glue('gsutil cat {csv}')),
col_types = col_types, show_col_types = FALSE)
          if (is.null(col_types)) {
            col_types <- spec(chunk)
          }
          chunk
        }))
}
kidney_disease_df <-
read_bq_export_from_workspace_bucket(condition_82100016_path)
```

```
The data will be written to gs://fc-secure-672eeb92-4859-4ed9-9f59-
d4349f3534a0/bq_exports/micah_hysong@researchallofus.org/20250825/
condition_82100016/condition_82100016_*.csv. Use this path when
```

```
reading the data into your notebooks in the future.

Loading
gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/bq_exports/micah_hysong@researchallofus.org/20250825/condition_82100016/
condition_82100016_000000000000.csv.
```

```
cases_definition3b<-intersect(potential_cases$person_id,
kidney_disease_df$person_id)
length(cases_definition3b)
```

```
[1] 8232
```

```
cases<-unique(c(case_definition_1, case_definition_2$person_id,
cases_definition3a$person_id, cases_definition3b ))
length(cases)
```

```
[1] 14259
```

Controls

```
most_recent_measure_not_ckd<-
most_recent_measurements_cleaned[most_recent_measurements_cleaned$eGFR
_2021>=90,]
nrow(most_recent_measure_not_ckd)
```

```
[1] 49984
```

```
controls0<-setdiff(most_recent_measure_not_ckd$person_id,
kidney_disease_df$person_id)
controls<-setdiff(controls0, cases)
length(controls0)
```

```
[1] 39650
```

```
df_cases <- data.frame(
  person_id = cases,
  status = 1
)

df_controls <- data.frame(
  person_id = controls,
  status = 0
)

final_df <- rbind(df_cases, df_controls)
nrow(final_df)
n_distinct(final_df$person_id)
```

```
[1] 53862
```

```
[1] 53862
```

```
# This snippet assumes that you run setup first

# This code saves your dataframe into a csv file in a "data" folder in
Google Bucket

# Replace df with THE NAME OF YOUR DATAFRAME
my_dataframe <- final_df

# Replace 'test.csv' with THE NAME of the file you're going to store
in the bucket (don't delete the quotation marks)
destination_filename <- 'eMERGE_CKD_case_control_df.csv'

################################################################################
##
##
################## DON'T CHANGE FROM HERE
################################
##
################################################################################
##

# store the dataframe in current workspace
write_excel_csv(my_dataframe, destination_filename)

# Get the bucket name
my_bucket <- Sys.getenv('WORKSPACE_BUCKET')

# Copy the file from current workspace to the bucket
system(paste0("gsutil cp ./", destination_filename, " ", my_bucket,
"/data/"), intern=T)

# Check if file is in the bucket
system(paste0("gsutil ls ", my_bucket, "/data/*.csv"), intern=T)
```

```
character(0)
```

```
[1]
"gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/data/Demographic_
and_ancestry_covariates.csv"
[2]
"gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/data/all_demograp
hics.csv"
[3]
"gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/data/eMERGE_CAD_c
ase_control_df.csv"
[4]
"gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/data/eMERGE_CKD_c
```

```
ase_control_df.csv"
[5]
"gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/data/eMERGE_afib_
case_control_df.csv"
[6]
"gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/data/eMERGE_asthm
a_case_control_df.csv"
[7]
"gs://fc-secure-672eeb92-4859-4ed9-9f59-d4349f3534a0/data/eMERGE_breas
t_cancer_case_control_df.csv"
```