Setup

In [35]:
```
# import packages
import os
import subprocess
import numpy as np
import pandas as pd
import seaborn as sns
import base64
import matplotlib.pyplot as plt
from IPython.display import HTML
from datetime import timedelta
from scipy.stats import chi2_contingency
import datetime as dt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import fbeta_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import plot_roc_curve
from sklearn.metrics import auc
from sklearn.model_selection import StratifiedKFold
```

In [37]:
```
# function to create download links
def create_download_link( df, title = "Download CSV file", filename = "data.csv"
    csv = df.to_csv()
    b64 = base64.b64encode(csv.encode())
    payload = b64.decode()
    html = '<a download="{filename}" href="data:text/csv;base64,{payload}" targe
    html = html.format(payload=payload,title=title,filename=filename)
    return HTML(html)
```

Drug Exposures: Clopidogrel

In [39]:
```
# Codes
# prescription dispensed in pharmacy - 38000175
# inpatient adminstration - 38000180
# physician adminstered drug - 38000179
# prescription dispensed through mail orders - 38000176
```

In [41]:
```
# de section contains autogenerated code from All of Us concept sets
clopidogrel_3_sql = f"""
-- get drug exposures
with de as (
```

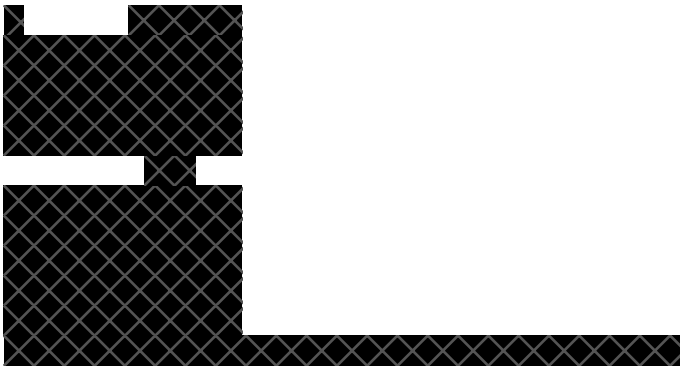
```
SELECT
    d_exposure.drug_concept_id,
    d_exposure.drug_exposure_start_date,
    d_exposure.drug_type_concept_id,
    d_exposure.VISIT_OCCURRENCE_ID as de_visit_occurrence,
    d_exposure.provider_id,
    d_exposure.days_supply,
    d_exposure.person_id,
    d_standard_concept.concept_name as drug_name
from
    ( SELECT
        *
    from
        `{os.environ["WORKSPACE_CDR"]}.drug_exposure` d_exposure
    WHERE
        (
            drug_concept_id in  (
                select
                    distinct ca.descendant_id
                from
                    `{os.environ["WORKSPACE_CDR"]}.cb_criteria_ancestor` ca
                join
                    (
                        select
                            distinct c.concept_id
                        from
                            `{os.environ["WORKSPACE_CDR"]}.cb_criteria` c
                        join
                            (
                                select
                                    cast(cr.id as string) as id
                                from
                                    `{os.environ["WORKSPACE_CDR"]}.cb_criter
                                where
                                    domain_id = 'DRUG'
                                    and is_standard = 1
                                    and concept_id in (
                                        1322184
                                    )
                                    and is_selectable = 1
                            ) a
                                on (
                                    c.path like concat('%.',
                                a.id,
                                '.%')
                                    or c.path like concat('%.',
                                    a.id))
                            where
                                domain_id = 'DRUG'
                                and is_standard = 1
                                and is_selectable = 1
                            ) b
                                on (
                                    ca.ancestor_id = b.concept_id
                                )
                        )
                    )
            ) d_exposure
    LEFT JOIN
        `{os.environ["WORKSPACE_CDR"]}.concept` d_route
           on d_exposure.ROUTE_CONCEPT_ID = d_route.CONCEPT_ID
```

```
        LEFT JOIN
            `{os.environ["WORKSPACE_CDR"]}.concept` d_type
               on d_exposure.drug_type_concept_id = d_type.CONCEPT_ID
        left join
            `{os.environ["WORKSPACE_CDR"]}.concept` d_standard_concept
               on d_exposure.DRUG_CONCEPT_ID = d_standard_concept.CONCEPT_ID
        LEFT JOIN
            `{os.environ["WORKSPACE_CDR"]}.concept` d_source_concept
               on d_exposure.DRUG_SOURCE_CONCEPT_ID = d_source_concept.CONCEPT_
        left join
            `{os.environ["WORKSPACE_CDR"]}.visit_occurrence` v
               on d_exposure.VISIT_OCCURRENCE_ID = v.VISIT_OCCURRENCE_ID
        LEFT JOIN
            `{os.environ["WORKSPACE_CDR"]}.concept` d_visit
               on v.VISIT_CONCEPT_ID = d_visit.CONCEPT_ID
),
drug_type_qualifier as (
    select * from de
    where drug_type_concept_id in (38000175, 38000180, 38000179, 38000176)
),
-- qualifier that there must be at least one visit occurrence in 365 days before
with_visit_before as (
    select de.person_id, drug_name, de.drug_exposure_start_date as index_date,
    max(vo.visit_start_date) as before_visit_start_date
    from drug_type_qualifier de
    join (
        select * from  `{os.environ["WORKSPACE_CDR"]}`.visit_occurrence
    ) vo
    on de.PERSON_ID = vo.person_id
    and vo.visit_start_date between DATE_SUB(de.drug_exposure_start_date, INTERV
    and DATE_SUB(de.drug_exposure_start_date, INTERVAL 1 DAY)
    group by person_id, drug_name, index_date
),
-- for all drug exposures that satisfy qualifier (visit in one year before), get
index_dates as (
    select person_id, index_date, drug_name, before_visit_start_date from
    (
        select *, ROW_NUMBER() over (PARTITION BY person_id order by index_date
        from with_visit_before
    )
    where time_order = 1
),
-- for each patient, get last drug exposure for clopidogrel
-- will be used to calculate follow_up_end_date
with_last_dispensed as (
    select person_id, max(drug_exposure_start_date) as last_dispensed from drug_
    group by person_id
),
-- for each patient and last dispensed date, get list of drug_names and max days
with_days_supply as (
    select a.person_id, STRING_AGG(a.drug_name) as drug_names, b.last_dispensed,
    join (
        select * from with_last_dispensed
    ) b
    on a.person_id = b.person_id and a.drug_exposure_start_date = b.last_dispens
    group by person_id, last_dispensed
),
-- calculate follow up period from last dispensed and days_supply
with_follow_up_period as (
    select *, DATE_ADD(last_dispensed, INTERVAL days_supply DAY) as follow_up_en
    from with_days_supply
```

```
),
-- add follow up period dataset to dataset of index dates
with_index_and_follow_up as (
    select a.person_id, a.drug_name, a.before_visit_start_date,
    a.index_date, b.last_dispensed, b.days_supply as last_days_supply, b.follow_
    date_diff(b.follow_up_end_date , a.index_date, DAY) as follow_up_period
    from index_dates a
    join (
        select * from with_follow_up_period
    ) b
    on a.person_id = b.person_id
),
-- get visit afterwards
-- qualifier that there must be at least one visit occurrence after index date d
with_visit_afterwards as (
    select a.person_id, a.drug_name, a.before_visit_start_date,
    a.index_date, a.last_dispensed, a.last_days_supply,
    a.follow_up_end_date, a.follow_up_period,
    min(vo.visit_start_date) as visit_after
    from with_index_and_follow_up a
    join (
        select * from  `{os.environ["WORKSPACE_CDR"]}`.visit_occurrence
    ) vo
    on a.person_id = vo.person_id
    and vo.visit_start_date between DATE_ADD(a.index_date, INTERVAL 1 DAY)
    and a.follow_up_end_date
    group by a.person_id, a.drug_name, a.before_visit_start_date,
    a.index_date, a.last_dispensed, a.follow_up_period, a.last_days_supply,
    a.follow_up_end_date

),
-- get all drug exposures of clopidogrel
with_all_dispensing_records as (
    select a.*, b.drug_exposure_start_date, b.days_supply from with_visit_afterw
    join (
        select * from drug_type_qualifier
    ) b
    on a.person_id = b.person_id and b.drug_exposure_start_date between a.index_
   -- order by person_id, index_date asc, drug_exposure_start_date asc
),
-- calculate days between previous dispensing record (using drug_exposure_start_
with_days_bw_records as (
    select *,
    case
        when previous_record is not null then DATE_DIFF(drug_exposure_start_date
        else 0
    end as time_since_previous_record
    from (
        select *,
        lag(drug_exposure_start_date) over
        (partition by person_id order by drug_exposure_start_date) as previous_r
        from with_all_dispensing_records
    )
),
-- get average days between drug exposures
with_avg_days_bw as (
    select person_id, drug_name, before_visit_start_date, index_date, last_dispe
    last_days_supply, follow_up_end_date, follow_up_period, visit_after,
    avg(time_since_previous_record) as avg_days_bw_records, count(*) as dispense
    from with_days_bw_records
    group by person_id, drug_name, before_visit_start_date, index_date, last_dis
```

```
    last_days_supply, follow_up_end_date, follow_up_period, visit_after
),
-- add person info concept ids
with_person_info as (
    select a.*, gender_concept_id, year_of_birth, month_of_birth, day_of_birth,
    race_concept_id, ethnicity_concept_id, location_id, provider_id, care_site_i
    person_source_value, gender_source_value, gender_source_concept_id, race_sou
    race_source_concept_id, ethnicity_source_value, ethnicity_source_concept_id,
    sex_at_birth_concept_id, sex_at_birth_source_concept_id,
    sex_at_birth_source_value,
     from with_avg_days_bw a
    inner join
    (
        select * from `{os.environ["WORKSPACE_CDR"]}`.person
    ) b
    on a.person_id = b.person_id
),
-- add concept names for person info
 with_concept_names as (
    select person.*,
    p_race_concept.concept_name as race,
    p_gender_concept.concept_name as gender,
    p_ethnicity_concept.concept_name as ethnicity,
    p_sex_at_birth_concept.concept_name as sex_at_birth
    from with_person_info person
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept p_race_concept
            on person.race_concept_id = p_race_concept.CONCEPT_ID
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept p_gender_concept
            on person.gender_concept_id = p_gender_concept.CONCEPT_ID
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept p_ethnicity_concept
            on person.ethnicity_concept_id = p_ethnicity_concept.CONCEPT_ID
    LEFT JOIN
         {os.environ["WORKSPACE_CDR"]}.concept p_sex_at_birth_concept
            on person.sex_at_birth_concept_id = p_sex_at_birth_concept.CONCEPT_I
)
select * from with_concept_names
"""
clopidogrel_3 = pd.read_gbq(clopidogrel_3_sql, dialect="standard")
```

In [43]:
```
clopidogrel_3['person_id']
```

Out[43]:


In [45]:
```
clopidogrel_3.head(3)
```

Out[45]:

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply

3 rows × 36 columns

In [47]:

```
# get care site ids for patients
care_sites_3_sql = f"""
with de as (
    SELECT
        d_exposure.drug_concept_id,
        d_exposure.drug_exposure_start_date,
        d_exposure.drug_type_concept_id,
        d_exposure.VISIT_OCCURRENCE_ID as de_visit_occurrence,
        d_exposure.provider_id,
        d_exposure.days_supply,
        d_exposure.person_id,
        d_type.concept_name as DRUG_TYPE_CONCEPT_NAME,
        d_standard_concept.concept_name as drug_name,
        v_ext.*
    from
        ( SELECT
            *
        from
            `{os.environ["WORKSPACE_CDR"]}.drug_exposure` d_exposure
        WHERE
            (
                drug_concept_id in  (
                    select
                        distinct ca.descendant_id
                    from
                        `{os.environ["WORKSPACE_CDR"]}.cb_criteria_ancestor` ca
                    join
                        (
                            select
                                distinct c.concept_id
                            from
                                `{os.environ["WORKSPACE_CDR"]}.cb_criteria` c
                            join
                                (
                                    select
                                        cast(cr.id as string) as id
                                    from
                                        `{os.environ["WORKSPACE_CDR"]}.cb_criter
                                    where
                                        domain_id = 'DRUG'
```

```
                                    and is_standard = 1
                                    and concept_id in (
                                        1322184
                                    )
                                    and is_selectable = 1
                                ) a
                                    on (
                                    c.path like concat('%.',
                                    a.id,
                                    '.%')
                                    or c.path like concat('%.',
                                    a.id))
                                where
                                    domain_id = 'DRUG'
                                    and is_standard = 1
                                    and is_selectable = 1
                                ) b
                                    on (
                                    ca.ancestor_id = b.concept_id
                                    )
                            )
                        )
                    ) d_exposure
            LEFT JOIN
                `{os.environ["WORKSPACE_CDR"]}.concept` d_route
                   on d_exposure.ROUTE_CONCEPT_ID = d_route.CONCEPT_ID
            LEFT JOIN
                `{os.environ["WORKSPACE_CDR"]}.concept` d_type
                   on d_exposure.drug_type_concept_id = d_type.CONCEPT_ID
            left join
                `{os.environ["WORKSPACE_CDR"]}.concept` d_standard_concept
                   on d_exposure.DRUG_CONCEPT_ID = d_standard_concept.CONCEPT_ID
            LEFT JOIN
                `{os.environ["WORKSPACE_CDR"]}.concept` d_source_concept
                   on d_exposure.DRUG_SOURCE_CONCEPT_ID = d_source_concept.CONCEPT_
            left join
                `{os.environ["WORKSPACE_CDR"]}.visit_occurrence` v
                   on d_exposure.VISIT_OCCURRENCE_ID = v.VISIT_OCCURRENCE_ID
            left join
                `{os.environ["WORKSPACE_CDR"]}.visit_occurrence_ext` v_ext
                   on d_exposure.VISIT_OCCURRENCE_ID = v_ext.VISIT_OCCURRENCE_ID
            LEFT JOIN
                `{os.environ["WORKSPACE_CDR"]}.concept` d_visit
                   on v.VISIT_CONCEPT_ID = d_visit.CONCEPT_ID
            LEFT JOIN
                `{os.environ["WORKSPACE_CDR"]}.person` d_person
                   on v.person_id = d_person.person_id
),
drug_type_qualifier as (
    select * from de
    where drug_type_concept_id in (38000175, 38000180, 38000179, 38000176)
),
group_by_person as (
    select person_id, src_id from drug_type_qualifier
    group by person_id, src_id
),
agg_care_sites as (
    select person_id, string_agg(src_id) as care_sites from group_by_person
    group by person_id
)
```

```
select * from agg_care_sites
"""
care_sites_3 = pd.read_gbq(care_sites_3_sql, dialect="standard")
```

In [49]:
```
care_sites_3
```

Out[49]:

	person_id	care_sites
	[redacted]	

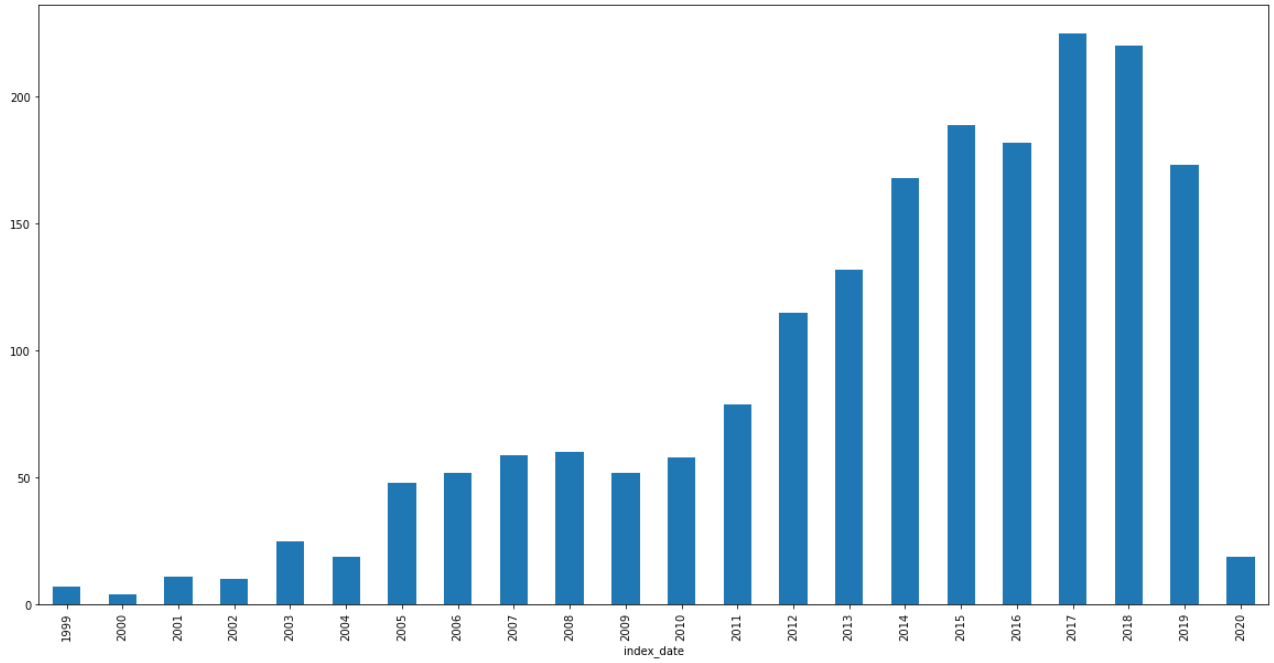
3939 rows × 2 columns

In [51]:
```
# join clopidogrel drug exposures with care sites for patients
clopidogrel_3 = pd.merge(clopidogrel_3, care_sites_3, on="person_id", how="left"
```

In [53]:
```
# download clopidogrel drug exposures
create_download_link(clopidogrel_3, 'clopidogrel 3 ', 'clopidogrel_3.csv')
```

Out[53]: clopidogrel 3

In [55]:
```
clopidogrel_3["index_date"].groupby([clopidogrel_3["index_date"].dt.year]).count
    kind="bar",
    figsize=(20,10)
)
plt.show()
```



Incident Cohort

Condition 1 Incidents

Gastrointestinal Bleeding

Description: Gastrointestinal bleeding events, defined by condition occurrences of Gastrointestinal hemorrhage GI bleeding conceptset occurring between start and end of inpatient or ER visit

In [57]:

```
# SQL for getting gastrointestinal hemorrhage conditions
# NOTE: source codes (ICD9 and ICD10) should be under condition_source_concept_i
# but there is an error in the database where it sometimes gets coded under cond
# my solution has been to pull both columns
# use condition_source by default, but if its coded under condition_status, use

cond1_codes = [ #SNOMED CODES
4138962, 4195231, 4147683, 4163865, 195584, 40482685, 28779, 4222896, 4296611, 2
4204555, 24973, 23808, 2002608, 198798, 4198381, 4209746, 4112183, 2108900, 2108
4101104, 443530, 197925, 4027663, 4291028]
insert_cond1_sql0 = ",".join([f"('{x}')" for x in cond1_codes])
gi_incident_sql = f"""
-- temp SQL table for condition codes for faster lookup using join
CREATE TEMP TABLE temp
(
  condition_codes STRING
);
INSERT INTO temp (
    condition_codes
)
VALUES {insert_cond1_sql0};

-- create official table with correct integer cast
CREATE TEMP TABLE search_codes (
```

```
    condition_codes INT64
);

INSERT INTO search_codes
SELECT CAST(condition_codes as INT64) as condition_codes FROM temp;

SELECT * FROM search_codes;

-- get all condition occurrences with condition_status and condition_source
with co as (
    select condition_concept.concept_name as condition_name,
    condition_status.concept_name as condition_status_name,
    condition_status.concept_code as condition_status_code,
    condition_status.vocabulary_id as condition_status_vocabulary,
    condition_source.concept_name as condition_source_name,
    condition_source.concept_code as condition_source_code,
    condition_source.vocabulary_id as condition_source_vocabulary,
    a.person_id as person_id,
    a.condition_concept_id,
    a.condition_start_date,
    a.condition_end_date,
    a.condition_type_concept_id,
    a.visit_occurrence_id,
    v.visit_occurrence_id as visit_occurrence_id_1,
    v.person_id as person_id_1,
    v.visit_concept_id,
    v.visit_start_date,
    v.visit_start_datetime,
    v.visit_end_date,
    v.visit_end_datetime,
    v.visit_type_concept_id,
    v.visit_source_value,
    v.visit_source_concept_id,
    v.admitting_source_concept_id,
    v.admitting_source_value,
    v.discharge_to_concept_id,
    v.discharge_to_source_value,
    v.preceding_visit_occurrence_id,
    a.condition_source_concept_id,
    a.condition_status_concept_id
    from {os.environ["WORKSPACE_CDR"]}.condition_occurrence a

    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept condition_concept
        on a.condition_concept_id = condition_concept.concept_id
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept condition_status
        on a.condition_status_concept_id = condition_status.concept_id
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept condition_source
        on a.condition_source_concept_id = condition_source.concept_id
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.visit_occurrence v
        on a.visit_occurrence_id = v.visit_occurrence_id
    where
    (condition_status_concept_id is not null or
    condition_source_concept_id is not null)
    and (condition_status_concept_id != 0 or
    condition_source_concept_id != 0)
    --limit 100
),
```

```
-- combine status and source columns, refer to notes at top of this section
combine_status_and_source as (
    select
    person_id,
    condition_name,
    condition_concept_id,
    condition_start_date,
    condition_end_date,
    condition_type_concept_id,
    visit_occurrence_id,

    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_concept_id
        else condition_source_concept_id
    END as source_concept_id,
    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_name
        else condition_source_name
    END as source_name,
    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_code
        else condition_source_code
    END as source_code,
    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_vocabulary
        else condition_source_vocabulary
    END as source_vocab,

    visit_occurrence_id_1,
    person_id_1,
    visit_concept_id,
    visit_start_date,
    visit_start_datetime,
    visit_end_date,
    visit_end_datetime,
    visit_type_concept_id,
    visit_source_value,
    visit_source_concept_id,
    admitting_source_concept_id,
    admitting_source_value,
    discharge_to_concept_id,
    discharge_to_source_value,
    preceding_visit_occurrence_id
from co
),
-- use inner join to get only conditions that we care about
target_conditions as (
    select distinct a.* from combine_status_and_source a
    inner join search_codes
    on a.condition_concept_id = search_codes.condition_codes
)
select * from target_conditions;
"""

"""
-- get earliest condition_occurrence per patient for incident coohort
```

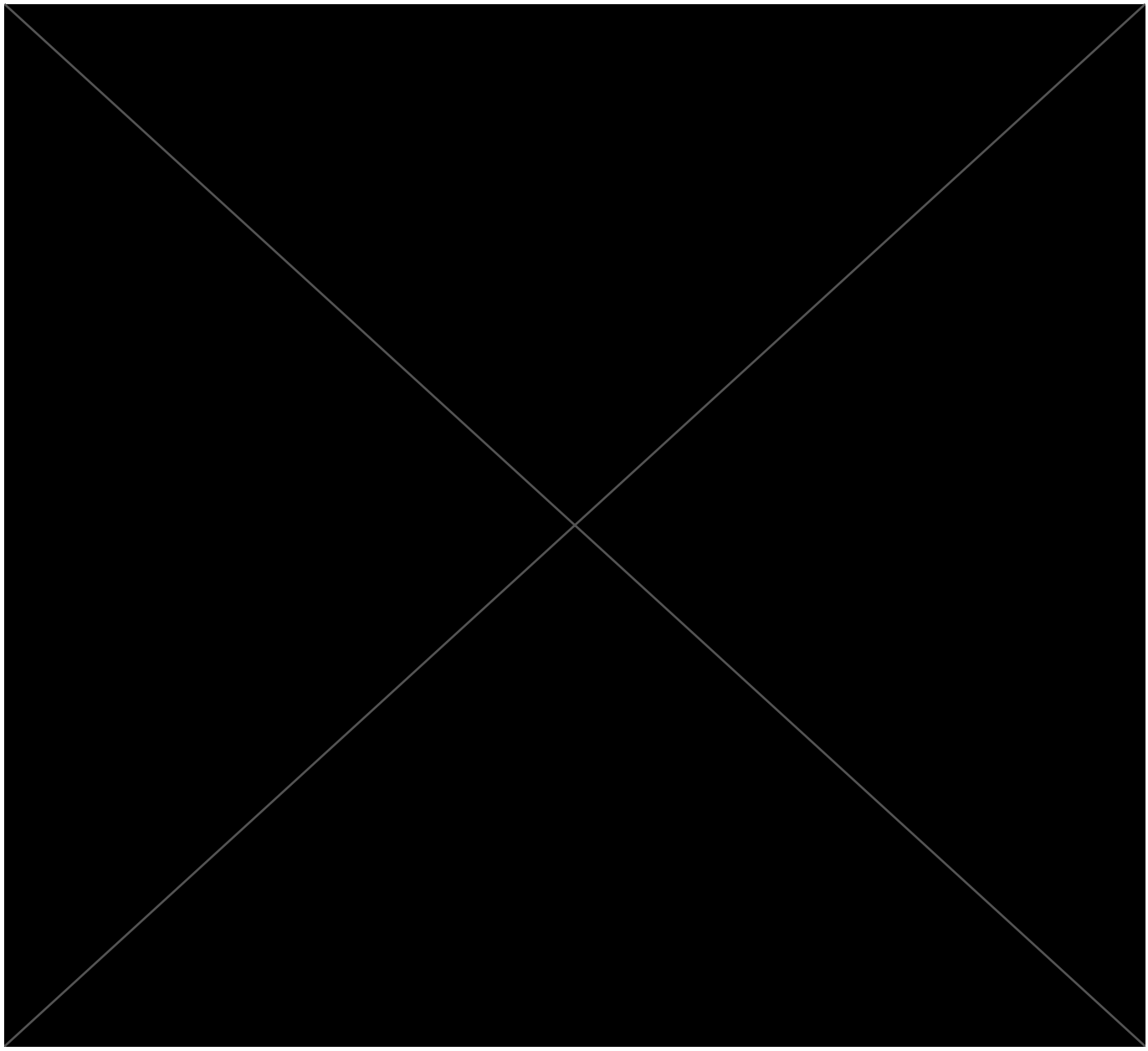
```
-- not using right now, because we want to keep all condition_occurrences
-- in case earliest one doesn't fall after drug exposure
incidence_cohort as (
    select person_id, condition_name, condition_concept_id, condition_start_date
    visit_occurrence_id, condition_type_concept_id, source_concept_id, source_na
    source_vocab from
        (
            select *, ROW_NUMBER() over (PARTITION BY person_id order by conditi
            from target_conditions
        )
        where time_order = 1
)
select * from target_conditions
"""
gi_cond1 = pd.read_gbq(gi_incident_sql, dialect="standard")
```

In [59]:

```
gi_cond1
```

Out[59]:

	person_id	condition_name	condition_concept_id	condition_start_date	condition_end_da



85902 rows × 26 columns

Condition 2 Incidents

Gastrointestinal Hemorrhage Description: Gastrointestinal hemorrhage by Siontis Circulation, 2018

In [61]:
```
# SQL for getting gastrointestinal hemorrhage conditions
# NOTE: source codes (ICD9 and ICD10) should be under condition_source_concept_i
# but there is an error in the database where it sometimes gets coded under cond
# my solution has been to pull both columns
# use condition_source by default, but if its coded under condition_status, use


cond2_codes = [46273183, 46270529, 46270145, 46270025, 46269911, 46269907, 46269
    45757654, 45757543, 4336230, 4294973, 4291649, 4289830, 4274491, 4247008, 42
    4231580, 4217947, 4211001, 4174044, 4169592, 4164920, 4112183, 4103703, 4046
    4031954, 4027729, 4006994, 443779, 443530, 442314, 442190, 441328, 441063, 4
    440756, 438796, 438468, 437326, 437323, 437027, 437021, 436729, 436148, 4358
    434402, 433515, 316457, 201885, 199855, 198801, 198798, 198467, 197925, 1970
    196442, 195845, 194986, 193795, 193250, 193249, 192671, 31335, 30770, 28779,
    26727, 26441, 24397, 24076, 23245, 23237, 22665]
insert_cond2_sql0 = ",".join([f"('{x}')" for x in cond2_codes])
gi_incident_sql = f"""
-- temp SQL table for condition codes for faster lookup using join
CREATE TEMP TABLE temp
(
  condition_codes STRING
);
INSERT INTO temp (
    condition_codes
)
VALUES {insert_cond2_sql0};

-- create official table with correct integer cast
CREATE TEMP TABLE search_codes (
    condition_codes INT64
);

INSERT INTO search_codes
SELECT CAST(condition_codes as INT64) as condition_codes FROM temp;

SELECT * FROM search_codes;

-- get all condition occurrences with condition_status and condition_source
with co as (
    select condition_concept.concept_name as condition_name,
    condition_status.concept_name as condition_status_name,
    condition_status.concept_code as condition_status_code,
    condition_status.vocabulary_id as condition_status_vocabulary,
    condition_source.concept_name as condition_source_name,
    condition_source.concept_code as condition_source_code,
    condition_source.vocabulary_id as condition_source_vocabulary,
    a.person_id as person_id,
    a.condition_concept_id,
    a.condition_start_date,
    a.condition_end_date,
```

```
    a.condition_type_concept_id,
    a.visit_occurrence_id,
    v.visit_occurrence_id as visit_occurrence_id_1,
    v.person_id as person_id_1,
    v.visit_concept_id,
    v.visit_start_date,
    v.visit_start_datetime,
    v.visit_end_date,
    v.visit_end_datetime,
    v.visit_type_concept_id,
    v.visit_source_value,
    v.visit_source_concept_id,
    v.admitting_source_concept_id,
    v.admitting_source_value,
    v.discharge_to_concept_id,
    v.discharge_to_source_value,
    v.preceding_visit_occurrence_id,
    a.condition_source_concept_id,
    a.condition_status_concept_id
    from {os.environ["WORKSPACE_CDR"]}.condition_occurrence a

    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept condition_concept
        on a.condition_concept_id = condition_concept.concept_id
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept condition_status
        on a.condition_status_concept_id = condition_status.concept_id
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept condition_source
        on a.condition_source_concept_id = condition_source.concept_id
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.visit_occurrence v
        on a.visit_occurrence_id = v.visit_occurrence_id
    where
    (condition_status_concept_id is not null or
    condition_source_concept_id is not null)
    and (condition_status_concept_id != 0 or
    condition_source_concept_id != 0)
    --limit 100
),

-- combine status and source columns, refer to notes at top of this section
combine_status_and_source as (
    select
    person_id,
    condition_name,
    condition_concept_id,
    condition_start_date,
    condition_end_date,
    condition_type_concept_id,
    visit_occurrence_id,

    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_concept_id
        else condition_source_concept_id
    END as source_concept_id,
    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_name
        else condition_source_name
```

```
    END as source_name,
    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_code
        else condition_source_code
    END as source_code,
    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_vocabulary
        else condition_source_vocabulary
    END as source_vocab,

    visit_occurrence_id_1,
    person_id_1,
    visit_concept_id,
    visit_start_date,
    visit_start_datetime,
    visit_end_date,
    visit_end_datetime,
    visit_type_concept_id,
    visit_source_value,
    visit_source_concept_id,
    admitting_source_concept_id,
    admitting_source_value,
    discharge_to_concept_id,
    discharge_to_source_value,
    preceding_visit_occurrence_id
from co
),
-- use inner join to get only conditions that we care about
target_conditions as (
    select distinct a.* from combine_status_and_source a
    inner join search_codes
    on a.condition_concept_id = search_codes.condition_codes
)
select * from target_conditions;
"""

"""
-- get earliest condition_occurrence per patient for incident coohort
-- not using right now, because we want to keep all condition_occurrences
-- in case earliest one doesn't fall after drug exposure
incidence_cohort as (
    select person_id, condition_name, condition_concept_id, condition_start_date
    visit_occurrence_id, condition_type_concept_id, source_concept_id, source_na
    source_vocab from
        (
            select *, ROW_NUMBER() over (PARTITION BY person_id order by conditi
            from target_conditions
        )
        where time_order = 1
)
select * from target_conditions
"""
gi_cond2 = pd.read_gbq(gi_incident_sql, dialect="standard")
```

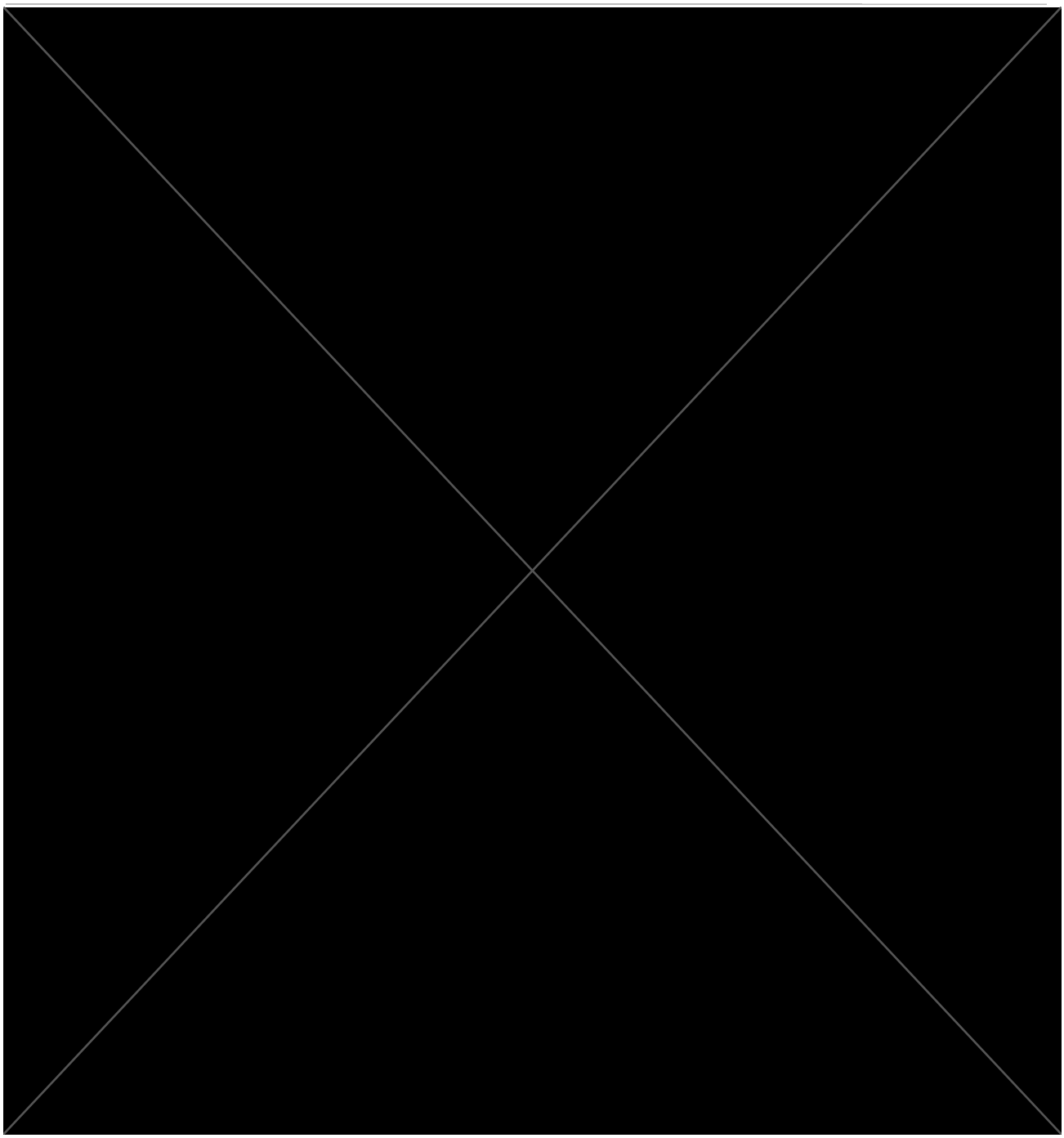
In [63]:

```
gi_cond2.where(gi_cond2['condition_concept_id'] == 192671.0).dropna()
```

Out[63]:

	person_id	condition_name	condition_concept_id	condition_start_date	condition_end_da

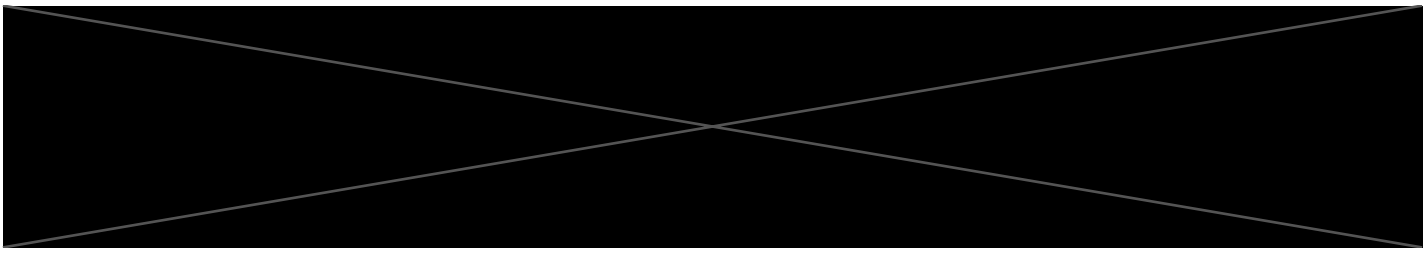
person_id	condition_name	condition_concept_id	condition_start_date	condition_end_da



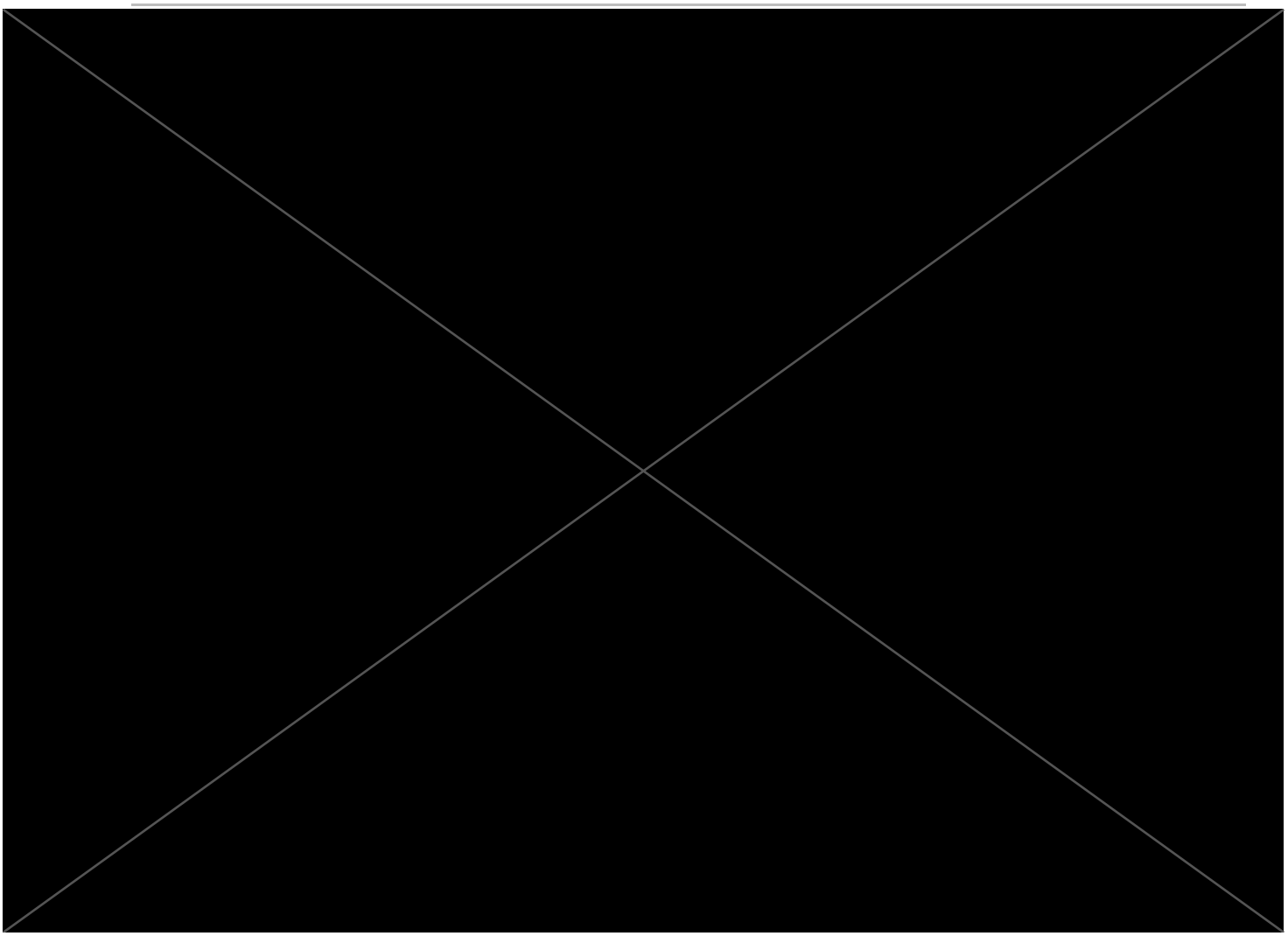
In [65]:
```
gi_cond1.where(gi_cond1['condition_concept_id'] == 192671.0).dropna()
```

Out[65]:

person_id	condition_name	condition_concept_id	condition_start_date	condition_end_da



person_id	condition_name	condition_concept_id	condition_start_date	condition_end_da



962 rows × 26 columns

Condition 3 Incidents

Intracranial Hemorrhage

Description: Hemorrhagic stroke events, defined by condition occurrences of intracranial bleed Hemorrhagic stroke conceptset occurring between start and end of inpatient or ER visit

In [67]:
```
# SQL for getting gastrointestinal hemorrhage conditions
# NOTE: source codes (ICD9 and ICD10) should be under condition_source_concept_i
# but there is an error in the database where it sometimes gets coded under cond
# my solution has been to pull both columns
# use condition_source by default, but if its coded under condition_status, use

cond3_codes = [ #SNOMED CODES
376713, 439847, 432923, 43530727, 4148906, 43530674, 42535425]
insert_cond3_sql0 = ",".join([f"('{x}')" for x in cond3_codes])
gi_incident_sql = f"""
-- temp SQL table for condition codes for faster lookup using join
CREATE TEMP TABLE temp
```

```
(
  condition_codes STRING
);
INSERT INTO temp (
    condition_codes
)
VALUES {insert_cond3_sql0};

-- create official table with correct integer cast
CREATE TEMP TABLE search_codes (
    condition_codes INT64
);

INSERT INTO search_codes
SELECT CAST(condition_codes as INT64) as condition_codes FROM temp;

SELECT * FROM search_codes;

-- get all condition occurrences with condition_status and condition_source
with co as (
    select condition_concept.concept_name as condition_name,
    condition_status.concept_name as condition_status_name,
    condition_status.concept_code as condition_status_code,
    condition_status.vocabulary_id as condition_status_vocabulary,
    condition_source.concept_name as condition_source_name,
    condition_source.concept_code as condition_source_code,
    condition_source.vocabulary_id as condition_source_vocabulary,
    a.person_id as person_id,
    a.condition_concept_id,
    a.condition_start_date,
    a.condition_end_date,
    a.condition_type_concept_id,
    a.visit_occurrence_id,
    v.visit_occurrence_id as visit_occurrence_id_1,
    v.person_id as person_id_1,
    v.visit_concept_id,
    v.visit_start_date,
    v.visit_start_datetime,
    v.visit_end_date,
    v.visit_end_datetime,
    v.visit_type_concept_id,
    v.visit_source_value,
    v.visit_source_concept_id,
    v.admitting_source_concept_id,
    v.admitting_source_value,
    v.discharge_to_concept_id,
    v.discharge_to_source_value,
    v.preceding_visit_occurrence_id,
    a.condition_source_concept_id,
    a.condition_status_concept_id
    from {os.environ["WORKSPACE_CDR"]}.condition_occurrence a

    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept condition_concept
        on a.condition_concept_id = condition_concept.concept_id
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept condition_status
        on a.condition_status_concept_id = condition_status.concept_id
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept condition_source
        on a.condition_source_concept_id = condition_source.concept_id
```

```
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.visit_occurrence v
        on a.visit_occurrence_id = v.visit_occurrence_id
    where
    (condition_status_concept_id is not null or
    condition_source_concept_id is not null)
    and (condition_status_concept_id != 0 or
    condition_source_concept_id != 0)
    --limit 100
),

-- combine status and source columns, refer to notes at top of this section
combine_status_and_source as (
    select
    person_id,
    condition_name,
    condition_concept_id,
    condition_start_date,
    condition_end_date,
    condition_type_concept_id,
    visit_occurrence_id,

    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_concept_id
        else condition_source_concept_id
    END as source_concept_id,
    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_name
        else condition_source_name
    END as source_name,
    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_code
        else condition_source_code
    END as source_code,
    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_vocabulary
        else condition_source_vocabulary
    END as source_vocab,

    visit_occurrence_id_1,
    person_id_1,
    visit_concept_id,
    visit_start_date,
    visit_start_datetime,
    visit_end_date,
    visit_end_datetime,
    visit_type_concept_id,
    visit_source_value,
    visit_source_concept_id,
    admitting_source_concept_id,
    admitting_source_value,
    discharge_to_concept_id,
    discharge_to_source_value,
    preceding_visit_occurrence_id
from co
),
-- use inner join to get only conditions that we care about
```

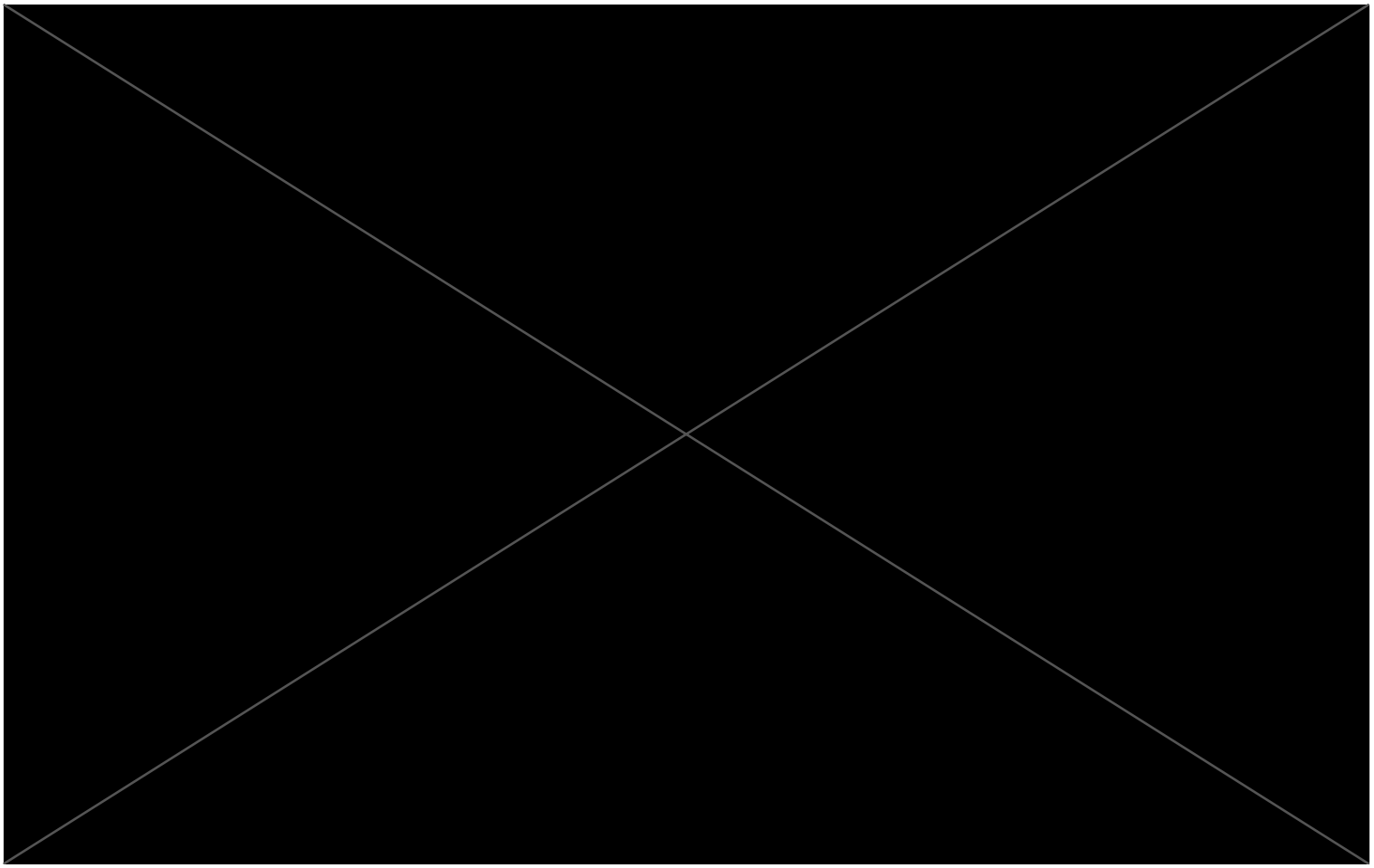
```
target_conditions as (
    select distinct a.* from combine_status_and_source a
    inner join search_codes
    on a.condition_concept_id = search_codes.condition_codes
)
select * from target_conditions;
"""

"""
-- get earliest condition_occurrence per patient for incident coohort
-- not using right now, because we want to keep all condition_occurrences
-- in case earliest one doesn't fall after drug exposure
incidence_cohort as (
    select person_id, condition_name, condition_concept_id, condition_start_date
    visit_occurrence_id, condition_type_concept_id, source_concept_id, source_na
    source_vocab from
        (
            select *, ROW_NUMBER() over (PARTITION BY person_id order by conditi
            from target_conditions
        )
        where time_order = 1
)
select * from target_conditions
"""
gi_cond3 = pd.read_gbq(gi_incident_sql, dialect="standard")
```

In [69]:

```
gi_cond3[gi_cond3["condition_concept_id"] == 439847]
```

Out[69]:

	person_id	condition_name	condition_concept_id	condition_start_date	condition_end_dat



	person_id	condition_name	condition_concept_id	condition_start_date	condition_end_dat
11781	1510929	Intracranial hemorrhage	439847	2018-11-25	2018-11-2

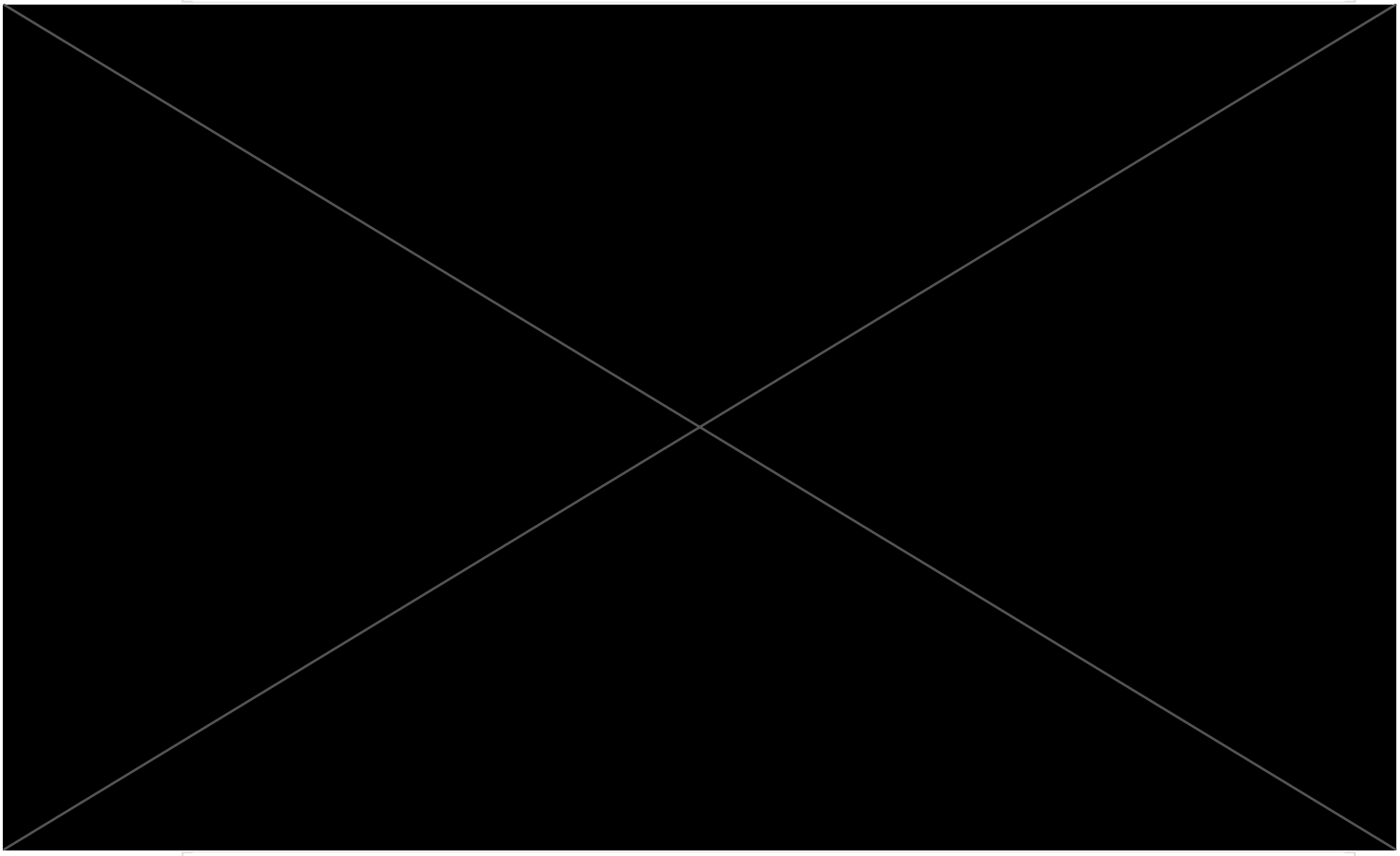
405 rows × 26 columns

Condition 4 Incidents

Intracranial Hemorrhage

Description: Hemorrhagic stroke events, defined by condition occurrences of intracranial bleed Hemorrhagic stroke conceptset occurring between start and end of inpatient or ER visit, with removal of concept_id = 4148906, 42535425, 43530674, 43530727

In [71]:
```
# SQL for getting gastrointestinal hemorrhage conditions
# NOTE: source codes (ICD9 and ICD10) should be under condition_source_concept_i
# but there is an error in the database where it sometimes gets coded under cond
# my solution has been to pull both columns
# use condition_source by default, but if its coded under condition_status, use


cond4_codes = [ #SNOMED CODES
376713, 439847, 432923]
insert_cond4_sql0 = ",".join([f"('{x}')" for x in cond4_codes])
gi_incident_sql = f"""
-- temp SQL table for condition codes for faster lookup using join
```



```
    a.condition_concept_id,
    a.condition_start_date,
    a.condition_end_date,
    a.condition_type_concept_id,
    a.visit_occurrence_id,
    v.visit_occurrence_id as visit_occurrence_id_1,
    v.person_id as person_id_1,
    v.visit_concept_id,
    v.visit_start_date,
    v.visit_start_datetime,
    v.visit_end_date,
    v.visit_end_datetime,
    v.visit_type_concept_id,
    v.visit_source_value,
    v.visit_source_concept_id,
    v.admitting_source_concept_id,
    v.admitting_source_value,
    v.discharge_to_concept_id,
    v.discharge_to_source_value,
    v.preceding_visit_occurrence_id,
    a.condition_source_concept_id,
    a.condition_status_concept_id
    from {os.environ["WORKSPACE_CDR"]}.condition_occurrence a

    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept condition_concept
        on a.condition_concept_id = condition_concept.concept_id
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept condition_status
        on a.condition_status_concept_id = condition_status.concept_id
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept condition_source
        on a.condition_source_concept_id = condition_source.concept_id
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.visit_occurrence v
        on a.visit_occurrence_id = v.visit_occurrence_id
    where
    (condition_status_concept_id is not null or
    condition_source_concept_id is not null)
    and (condition_status_concept_id != 0 or
    condition_source_concept_id != 0)
    --limit 100
),

-- combine status and source columns, refer to notes at top of this section
combine_status_and_source as (
    select
    person_id,
    condition_name,
    condition_concept_id,
    condition_start_date,
    condition_end_date,
    condition_type_concept_id,
    visit_occurrence_id,

    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_concept_id
        else condition_source_concept_id
    END as source_concept_id,
    CASE
```

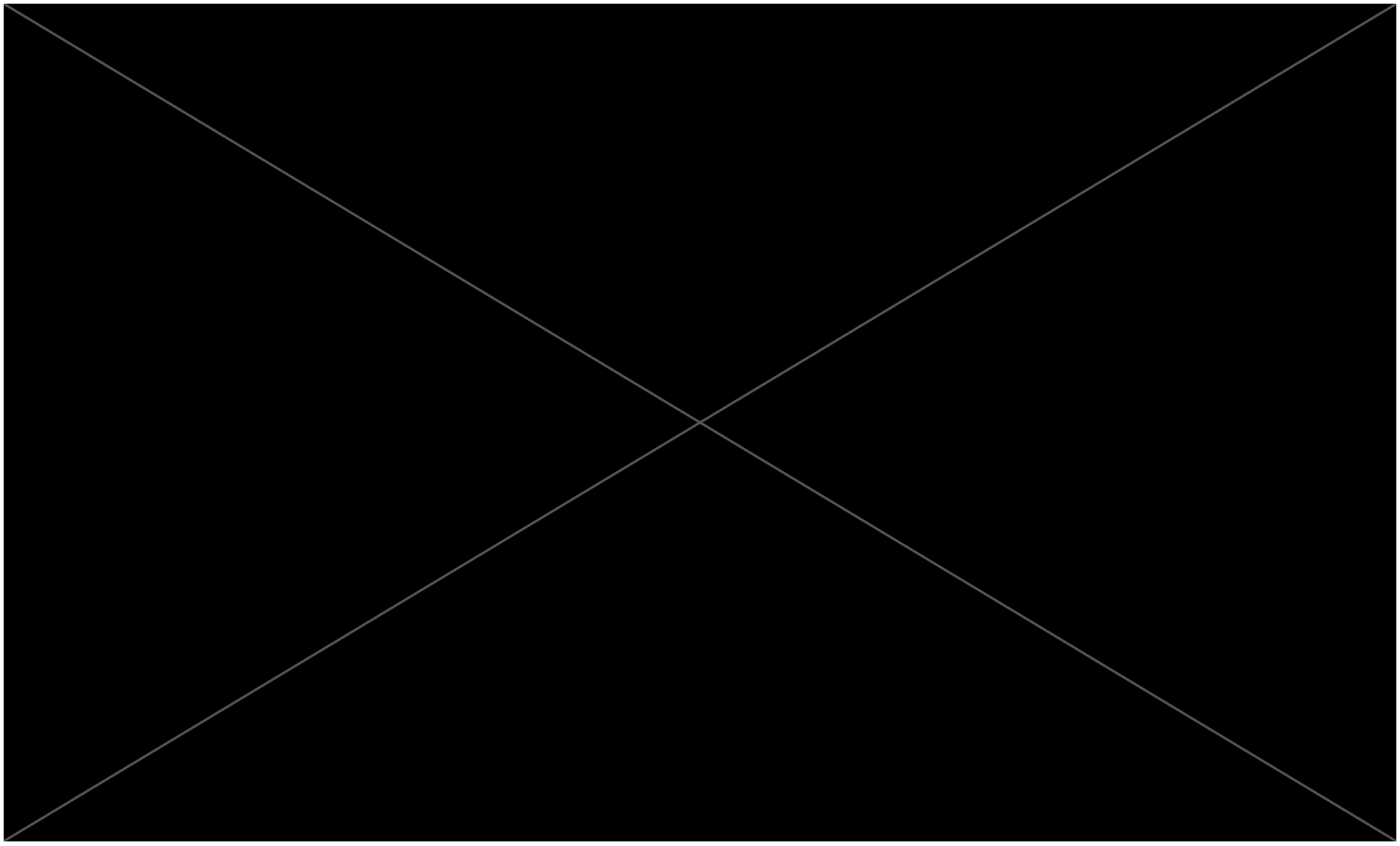
```
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_name
        else condition_source_name
    END as source_name,
    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_code
        else condition_source_code
    END as source_code,
    CASE
        when (condition_source_concept_id is null or condition_source_concept_id
        then condition_status_vocabulary
        else condition_source_vocabulary
    END as source_vocab,

    visit_occurrence_id_1,
    person_id_1,
    visit_concept_id,
    visit_start_date,
    visit_start_datetime,
    visit_end_date,
    visit_end_datetime,
    visit_type_concept_id,
    visit_source_value,
    visit_source_concept_id,
    admitting_source_concept_id,
    admitting_source_value,
    discharge_to_concept_id,
    discharge_to_source_value,
    preceding_visit_occurrence_id
from co
),
-- use inner join to get only conditions that we care about
target_conditions as (
    select distinct a.* from combine_status_and_source a
    inner join search_codes
    on a.condition_concept_id = search_codes.condition_codes
)
select * from target_conditions;
"""

"""
-- get earliest condition_occurrence per patient for incident coohort
-- not using right now, because we want to keep all condition_occurrences
-- in case earliest one doesn't fall after drug exposure
incidence_cohort as (
    select person_id, condition_name, condition_concept_id, condition_start_date
    visit_occurrence_id, condition_type_concept_id, source_concept_id, source_na
    source_vocab from
        (
            select *, ROW_NUMBER() over (PARTITION BY person_id order by conditi
            from target_conditions
        )
        where time_order = 1
)
select * from target_conditions
"""
gi_cond4 = pd.read_gbq(gi_incident_sql, dialect="standard")
```

In [73]:

```
gi_cond4
```

Out[73]:

	person_id	condition_name	condition_concept_id	condition_start_date	condition_end_date



5730 rows × 26 columns

Condition 5 Incidents

Platelet Transfusion

In [75]:

```
# SQL for getting gastrointestinal hemorrhage conditions
# NOTE: source codes (ICD9 and ICD10) should be under condition_source_concept_i
# but there is an error in the database where it sometimes gets coded under cond
# my solution has been to pull both columns
# use condition_source by default, but if its coded under condition_status, use

cond5_codes = [4130829]
insert_cond5_sql0 = ",".join([f"('{x}')" for x in cond5_codes])
gi_incident_sql = f"""
-- temp SQL table for procedure codes for faster lookup using join
CREATE TEMP TABLE temp
(
  procedure_codes STRING
);
INSERT INTO temp (
    procedure_codes
)
VALUES {insert_cond5_sql0};
```

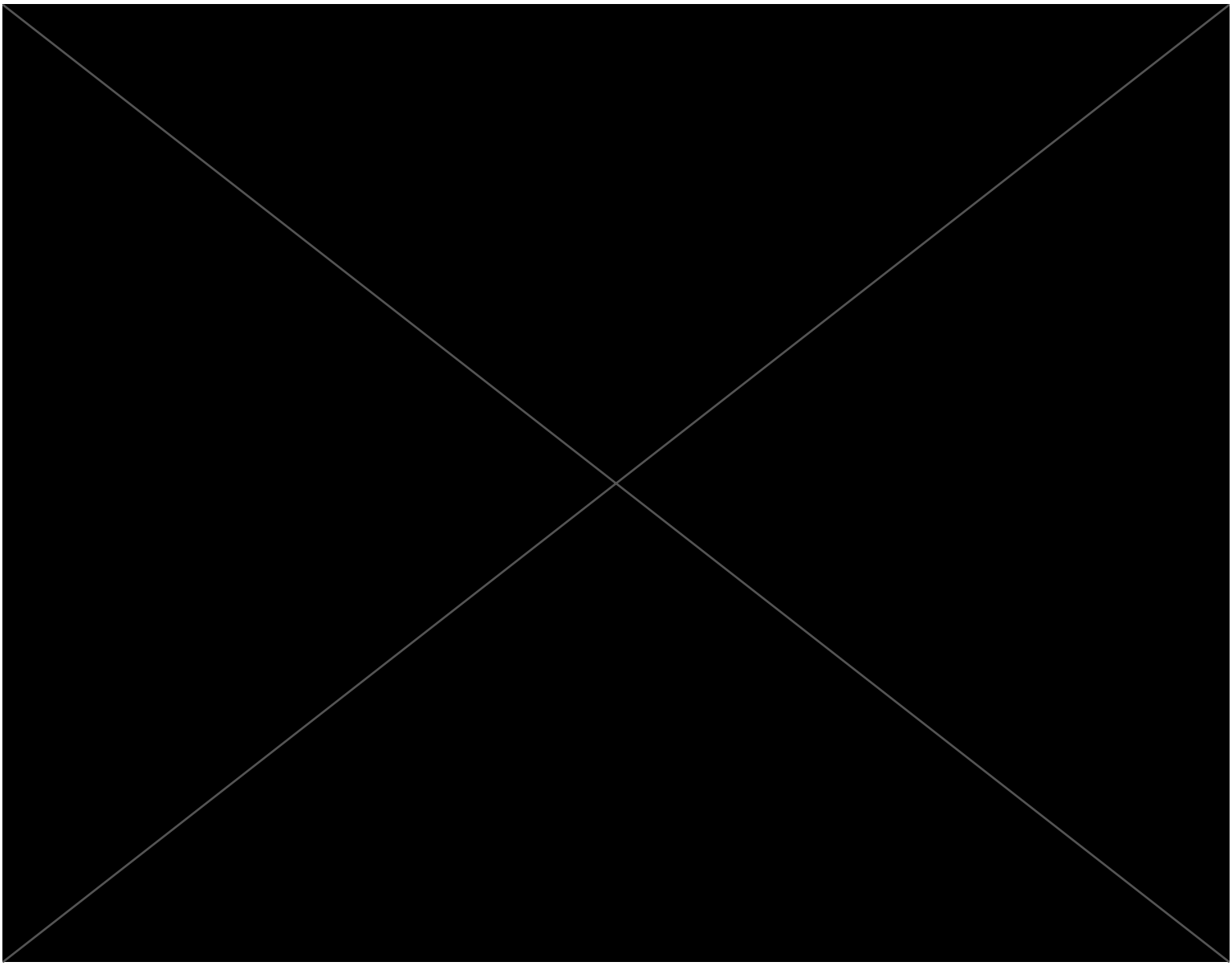
```
-- create official table with correct integer cast
CREATE TEMP TABLE search_codes (
    procedure_codes INT64
);

INSERT INTO search_codes
SELECT CAST(procedure_codes as INT64) as procedure_codes FROM temp;

SELECT * FROM search_codes;

-- get procedure occurrences
with co as (
    select
    concept.concept_name as procedure_name,
    a.person_id as person_id,
    a.procedure_concept_id,
    a.procedure_date,
    a.procedure_type_concept_id,
    a.visit_occurrence_id,
    v.visit_occurrence_id as visit_occurrence_id_1,
    v.person_id as person_id_1,
    v.visit_concept_id,
    v.visit_start_date,
    v.visit_start_datetime,
    v.visit_end_date,
    v.visit_end_datetime,
    v.visit_type_concept_id,
    v.visit_source_value,
    v.visit_source_concept_id,
    v.admitting_source_concept_id,
    v.admitting_source_value,
    v.discharge_to_concept_id,
    v.discharge_to_source_value,
    v.preceding_visit_occurrence_id,
    a.procedure_source_concept_id
    from {os.environ["WORKSPACE_CDR"]}.procedure_occurrence a

    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept concept
        on a.procedure_concept_id = concept.concept_id
    LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.visit_occurrence v
        on a.visit_occurrence_id = v.visit_occurrence_id
),

search_all as(
SELECT *
FROM search_codes s
LEFT JOIN
        {os.environ["WORKSPACE_CDR"]}.concept_ancestor ca
        on s.procedure_codes = ca.ancestor_concept_id
),
-- use inner join to get only conditions that we care about
target_conditions as (
    select distinct a.* from co a
    inner join search_all
    on a.procedure_concept_id = search_all.descendant_concept_id
)
select * from target_conditions;
"""
```

```
gi_cond5 = pd.read_gbq(gi_incident_sql, dialect="standard")
```

In [77]:
```
gi_cond5
```

Out[77]:

	procedure_name	person_id	procedure_concept_id	procedure_date	procedure_type_concep



778 rows × 22 columns

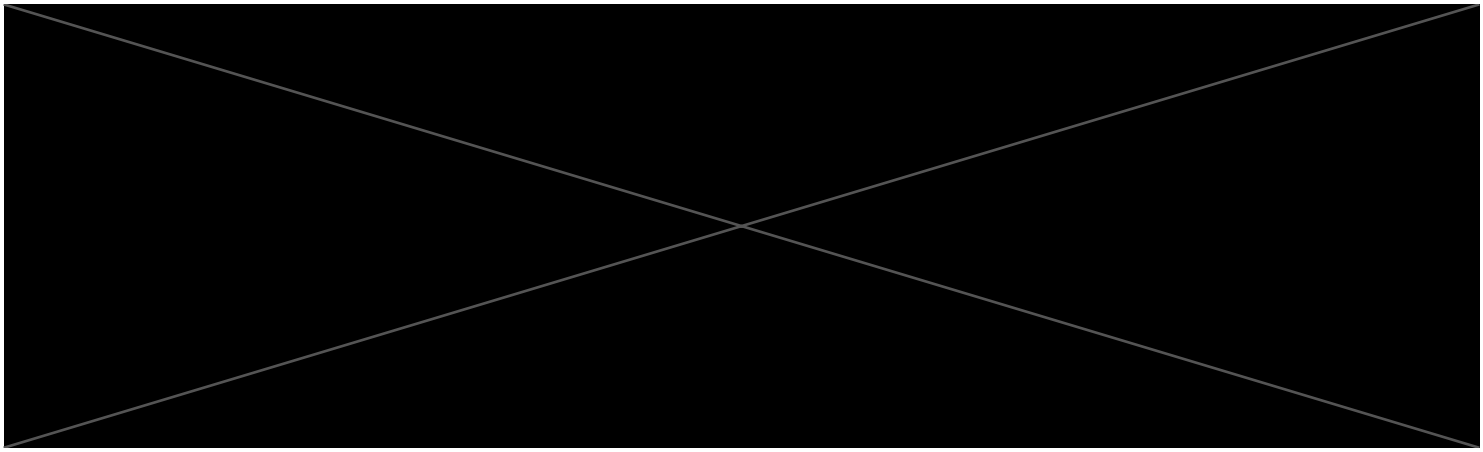
Overlap between Drug Exposures and Condition Occurrences

In [79]:
```
clopidogrel_3.set_index('person_id').head(3)
```

Out[79]:

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll

	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id						



3 rows × 36 columns

Instance Cohort

This records all patients who are both in the drug exposure and the condition tables, looking specifically for patients whose `condition_start_date` is before their `index_date` yet after the `follow_up_end_date` . Note that this table is only looking looking at the first time a patient has a visit in relation to a condition after their drug-exposure, not all visits.

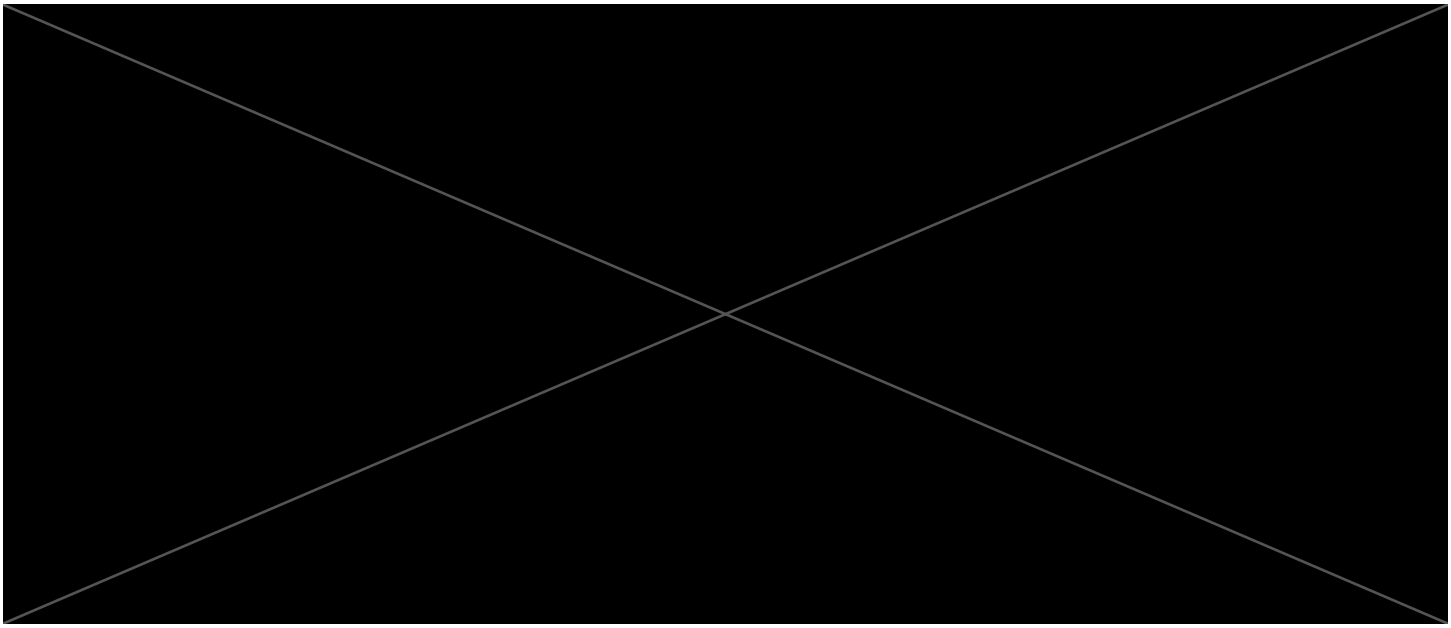
Condition 1

Overlap for Instance Cohort
Gastrointestinal Bleeding

In [81]:
```
# join drug exposures with condition_occurrences
overlap = clopidogrel_3.set_index('person_id').join(
    gi_cond1.set_index('person_id'), on='person_id', how='left').reset_index()
```

In [83]:
```
# get rows where condition starts before drug exposure index date
# NOTE: patient can only be in dataset once now

#index_date = time patient was given drug (Clopidogrel)

a = overlap[overlap['index_date'] < overlap['condition_start_date']]
a = a[a['follow_up_end_date'] > a['condition_start_date']]

b = a.groupby('person_id').apply(lambda x: x.sort_values('condition_start_date',

overlap_df1 = b.reset_index(drop = True).set_index("person_id")
overlap_df1_er = overlap_df1[(overlap_df1['visit_concept_id'] == 262.0) | (overl

assert len(a.person_id.unique())==len(b.person_id.unique())
display(overlap_df1.head(5))
create_download_link(overlap_df1, "Download Overlap CSV", "overlap(gi_bleeding).
```

	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id						



5 rows × 61 columns

Out[83]: Download Overlap CSV

In [85]:
```
overlap_df1.columns
```

Out[85]:
```
Index(['drug_name', 'before_visit_start_date', 'index_date', 'last_dispensed',
       'last_days_supply', 'follow_up_end_date', 'follow_up_period',
       'visit_after', 'avg_days_bw_records', 'dispensed_count',
       'avg_days_supply', 'gender_concept_id', 'year_of_birth',
       'month_of_birth', 'day_of_birth', 'birth_datetime', 'race_concept_id',
       'ethnicity_concept_id', 'location_id', 'provider_id', 'care_site_id',
       'person_source_value', 'gender_source_value',
       'gender_source_concept_id', 'race_source_value',
       'race_source_concept_id', 'ethnicity_source_value',
       'ethnicity_source_concept_id', 'sex_at_birth_concept_id',
       'sex_at_birth_source_concept_id', 'sex_at_birth_source_value', 'race',
       'gender', 'ethnicity', 'sex_at_birth', 'care_sites', 'condition_name',
       'condition_concept_id', 'condition_start_date', 'condition_end_date',
       'condition_type_concept_id', 'visit_occurrence_id', 'source_concept_id',
       'source_name', 'source_code', 'source_vocab', 'visit_occurrence_id_1',
       'person_id_1', 'visit_concept_id', 'visit_start_date',
       'visit_start_datetime', 'visit_end_date', 'visit_end_datetime',
       'visit_type_concept_id', 'visit_source_value',
       'visit_source_concept_id', 'admitting_source_concept_id',
       'admitting_source_value', 'discharge_to_concept_id',
       'discharge_to_source_value', 'preceding_visit_occurrence_id'],
      dtype='object')
```

Condition 2

Overlap for Instance Cohort

Gastrointestinal Hemorrhage

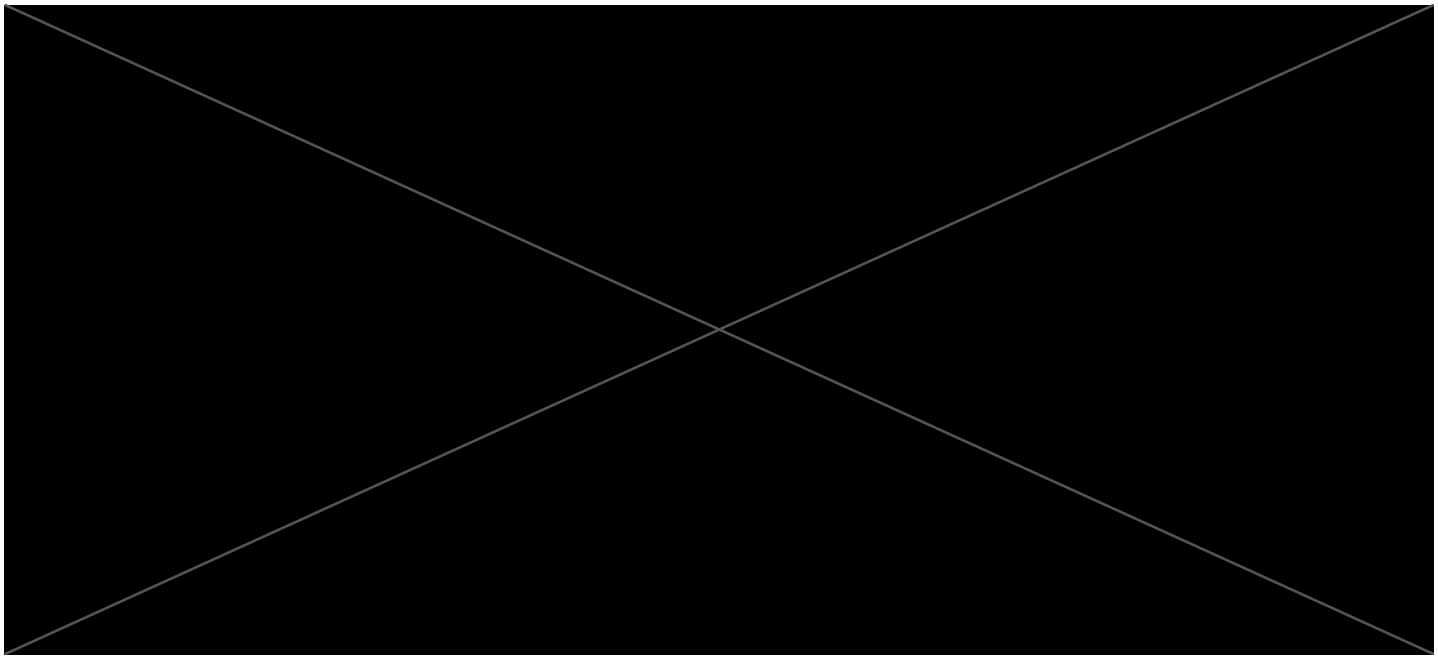
In [87]:
```
# join drug exposures with condition_occurrences
```

```
overlap = clopidogrel_3.set_index('person_id').join(gi_cond2.set_index('person_i
```

In [89]:

```
# get rows where condition starts before drug exposure index date
# NOTE: patient can only be in dataset once now

a = overlap[overlap['index_date'] < overlap['condition_start_date']]
a = a[a['follow_up_end_date'] > a['condition_start_date']]

b = a.groupby('person_id').apply(lambda x: x.sort_values('condition_start_date',

overlap_df2 = b.reset_index(drop = True).set_index("person_id")
overlap_df2_er = overlap_df2[(overlap_df2['visit_concept_id'] == 262.0) | (overl

assert len(a.person_id.unique())==len(b.person_id.unique())
display(overlap_df2.head(5))
create_download_link(overlap_df2, "Download Overlap CSV", "overlap2(gi_hemorrhag
```

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



5 rows × 61 columns

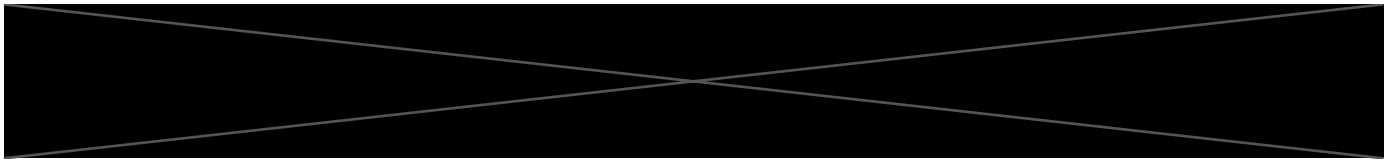
Out[89]: Download Overlap CSV

In [91]:

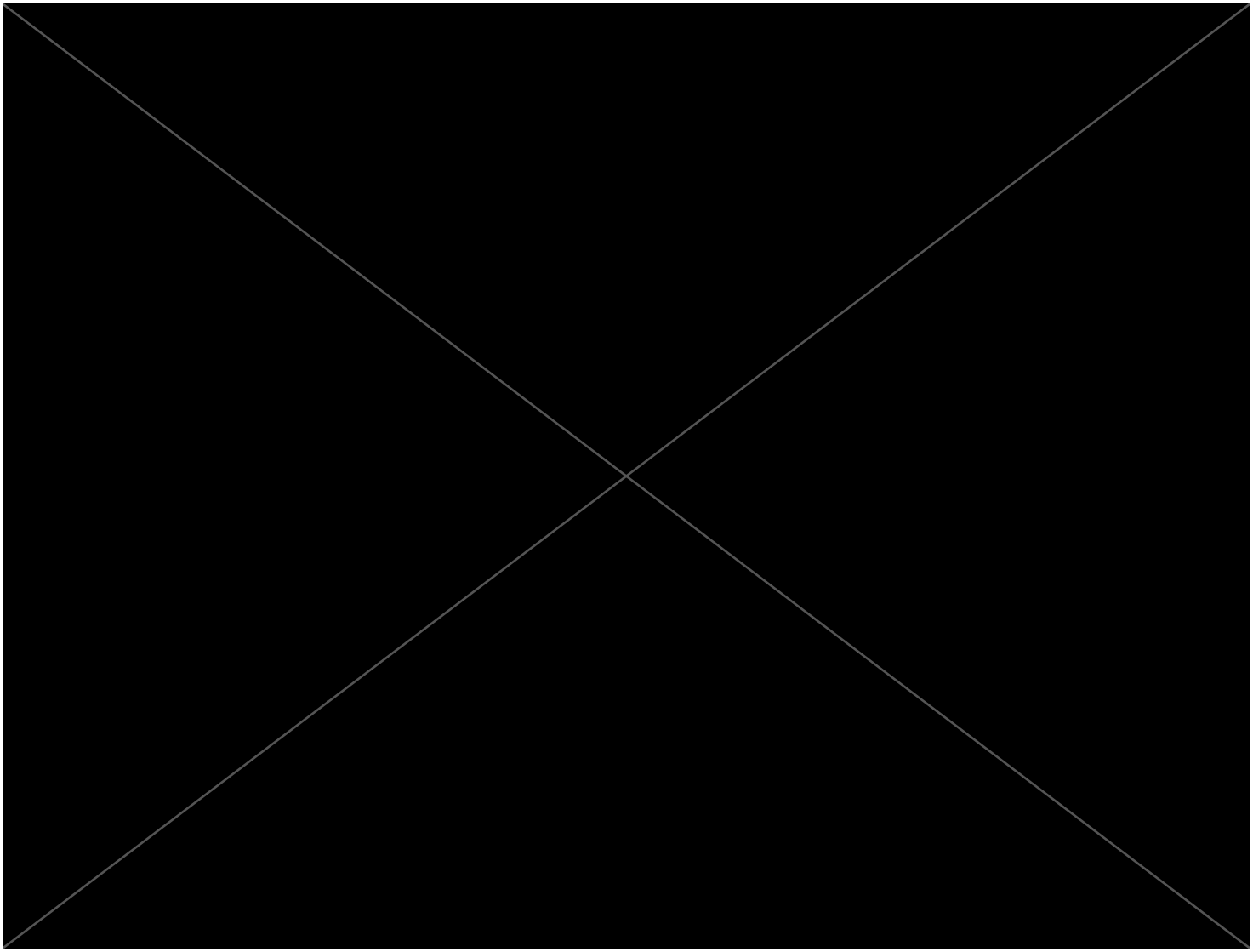
```
overlap_df2
```

Out[91]:

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



197 rows × 61 columns

Condition 3

Overlap for Instance Cohort

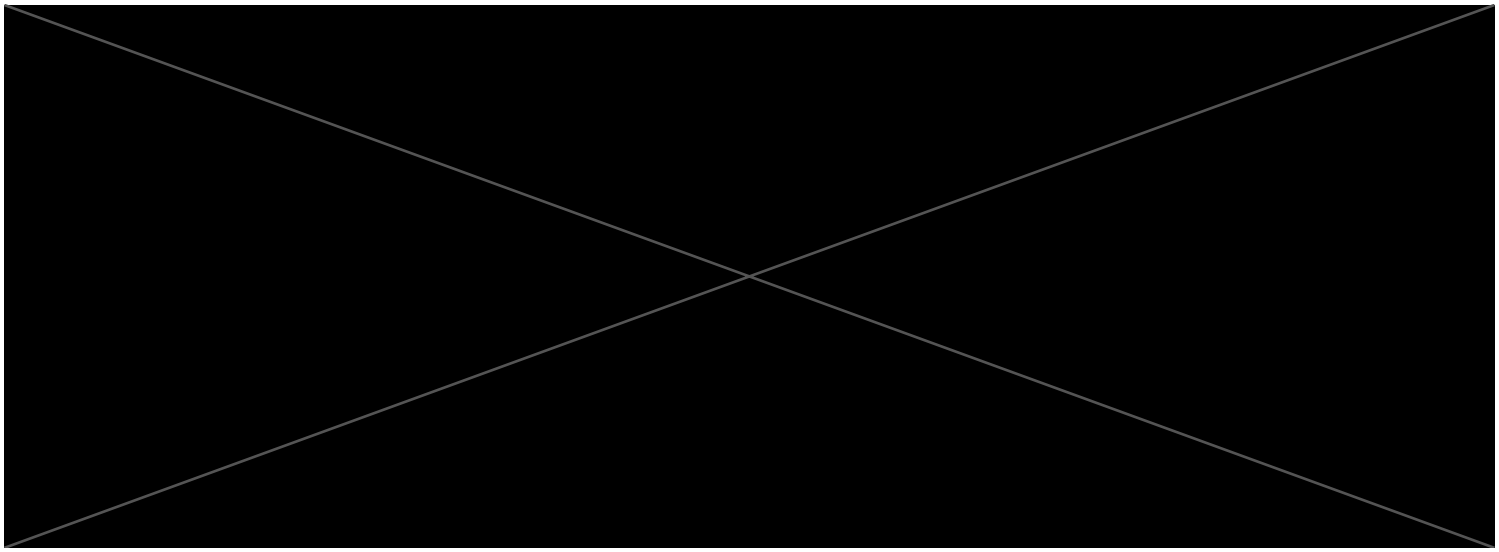
Intracranial Hemorrhage

In [93]:
```
# join drug exposures with condition_occurrences
overlap = clopidogrel_3.set_index('person_id').join(gi_cond3.set_index('person_i
```

In [95]:
```
# get rows where condition starts before drug exposure index date
# NOTE: patient can only be in dataset once now

a = overlap[overlap['index_date'] < overlap['condition_start_date']]
a = a[a['follow_up_end_date'] > a['condition_start_date']]

b = a.groupby('person_id').apply(lambda x: x.sort_values('condition_start_date',
```

```
overlap_df3 = b.reset_index(drop = True).set_index("person_id")
overlap_df3_er = overlap_df3[(overlap_df3['visit_concept_id'] == 262.0) | (overl

assert len(a.person_id.unique())==len(b.person_id.unique())
display(overlap_df3.head(5))
create_download_link(overlap_df3, "Download Overlap CSV", "overlap3(strokeEvents
```

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



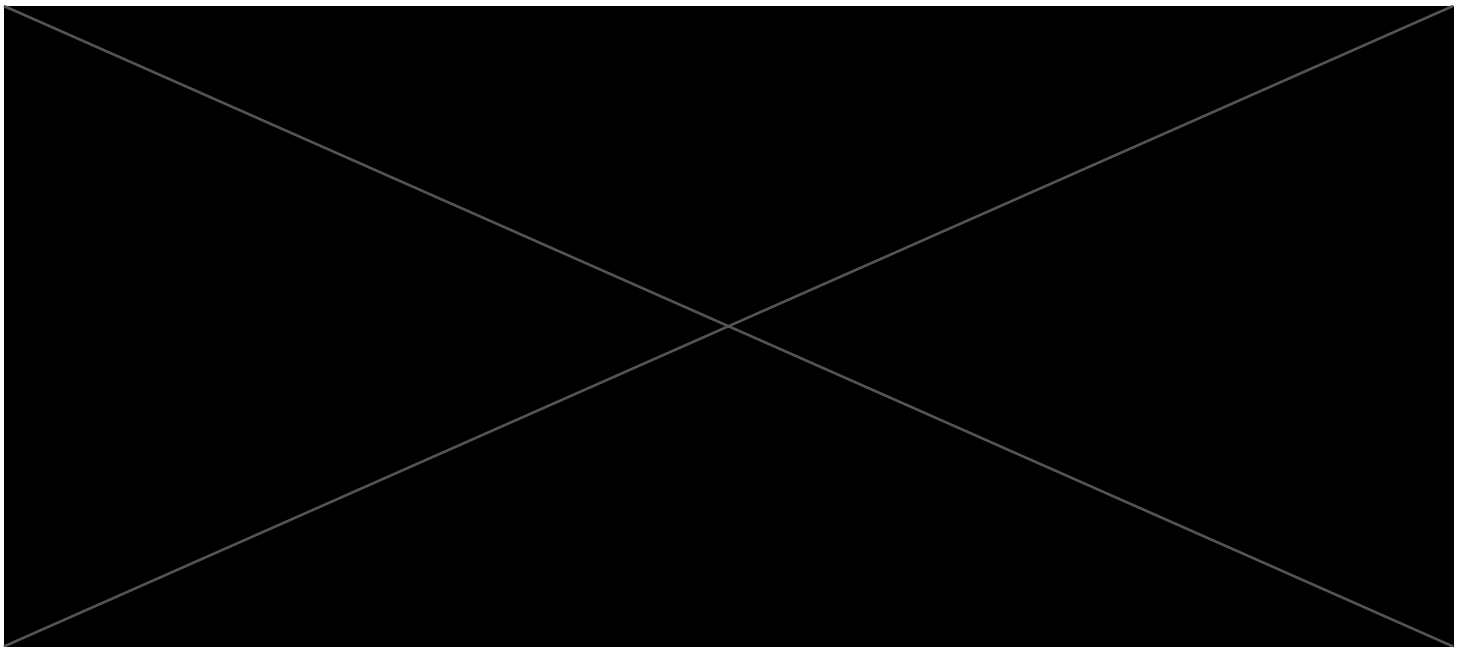
5 rows × 61 columns

Out[95]: Download Overlap CSV

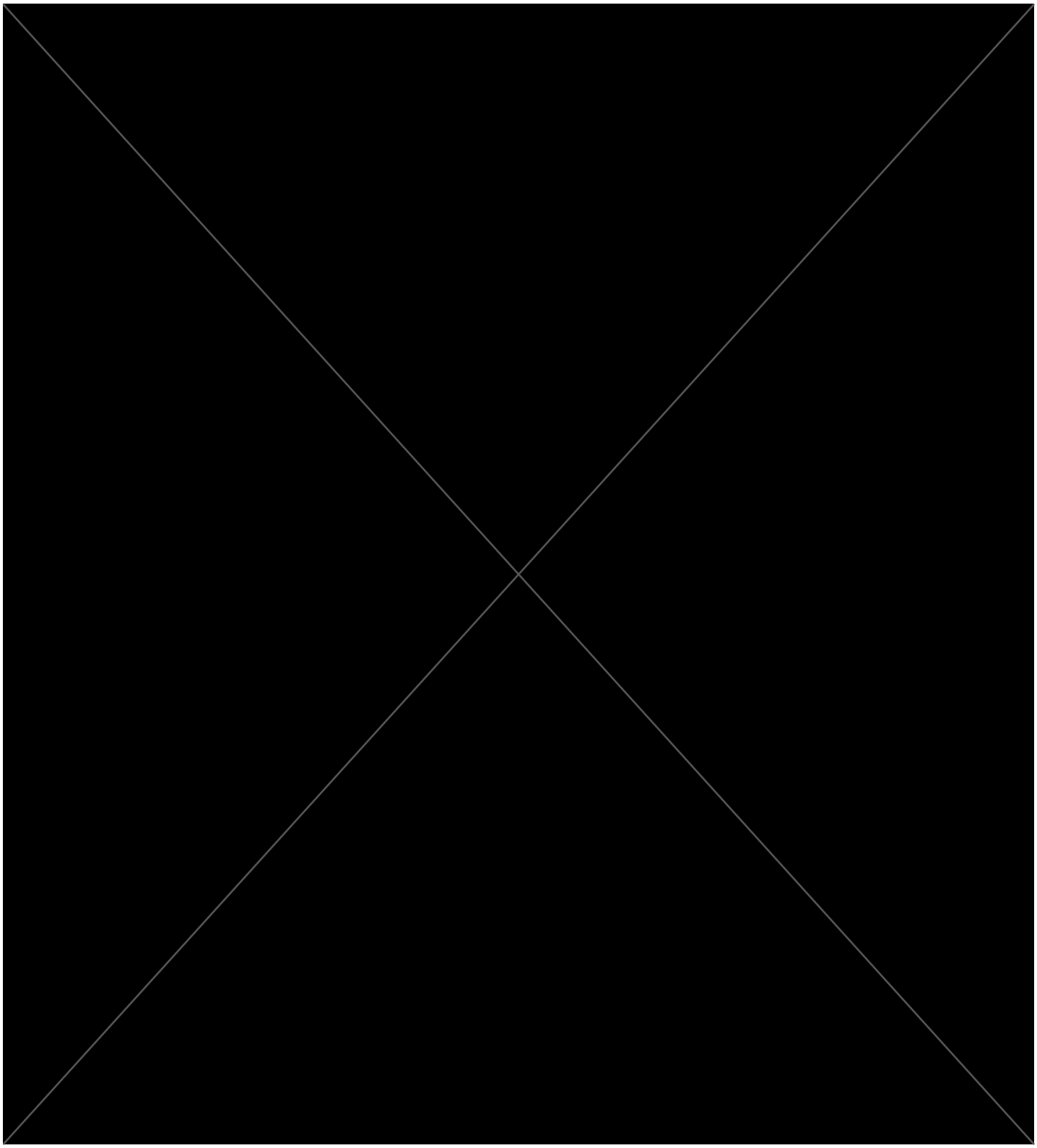
In [97]:
```
overlap_df3
```

Out[97]:

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



19 rows × 61 columns

Condition 4

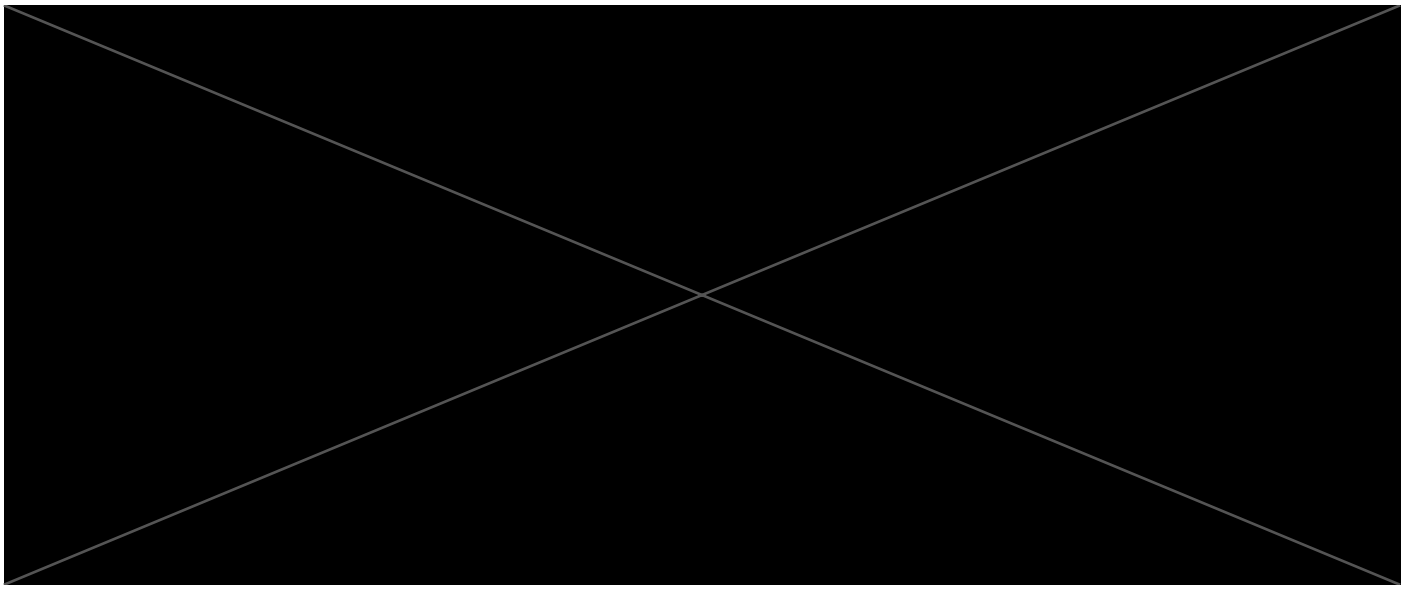
Overlap for Instance Cohort

Intracranial Hemorrhage (Minimized)

In [99]:
```
# join drug exposures with condition_occurrences
overlap = clopidogrel_3.set_index('person_id').join(gi_cond4.set_index('person_i
```

In [101…
```
# get rows where condition starts before drug exposure index date
# NOTE: patient can only be in dataset once now

a = overlap[overlap['index_date'] < overlap['condition_start_date']]
a = a[a['follow_up_end_date'] > a['condition_start_date']]

b = a.groupby('person_id').apply(lambda x: x.sort_values('condition_start_date',

overlap_df4 = b.reset_index(drop = True).set_index("person_id")
overlap_df4_er = overlap_df4[(overlap_df4['visit_concept_id'] == 262.0) | (overl

assert len(a.person_id.unique())==len(b.person_id.unique())
display(overlap_df4.head(5))
create_download_link(overlap_df4, "Download Overlap CSV", "overlap4(strokeEvents
```

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



5 rows × 61 columns

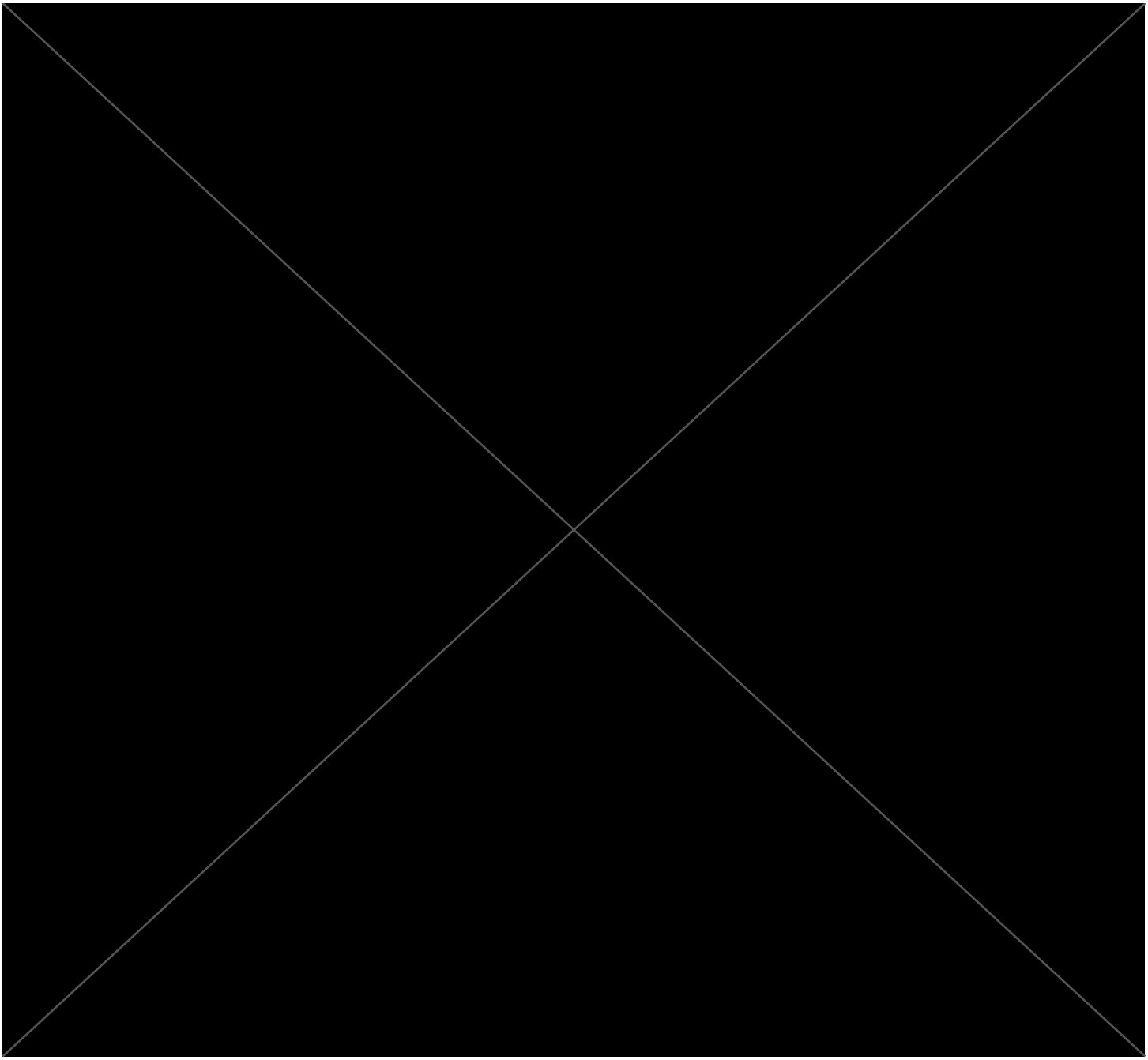
Out[101… Download Overlap CSV

In [103…
```
overlap_df4
```

Out[103…

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll

	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id						



11 rows × 61 columns

Contion 5

Overlap for Instance Cohort

Platelet Transfusion

In [105…
```
gi_cond5.columns
```

Out[105…
```
Index(['procedure_name', 'person_id', 'procedure_concept_id', 'procedure_date',
       'procedure_type_concept_id', 'visit_occurrence_id',
```

```
       'visit_occurrence_id_1', 'person_id_1', 'visit_concept_id',
       'visit_start_date', 'visit_start_datetime', 'visit_end_date',
       'visit_end_datetime', 'visit_type_concept_id', 'visit_source_value',
       'visit_source_concept_id', 'admitting_source_concept_id',
       'admitting_source_value', 'discharge_to_concept_id',
       'discharge_to_source_value', 'preceding_visit_occurrence_id',
       'procedure_source_concept_id'],
      dtype='object')
```

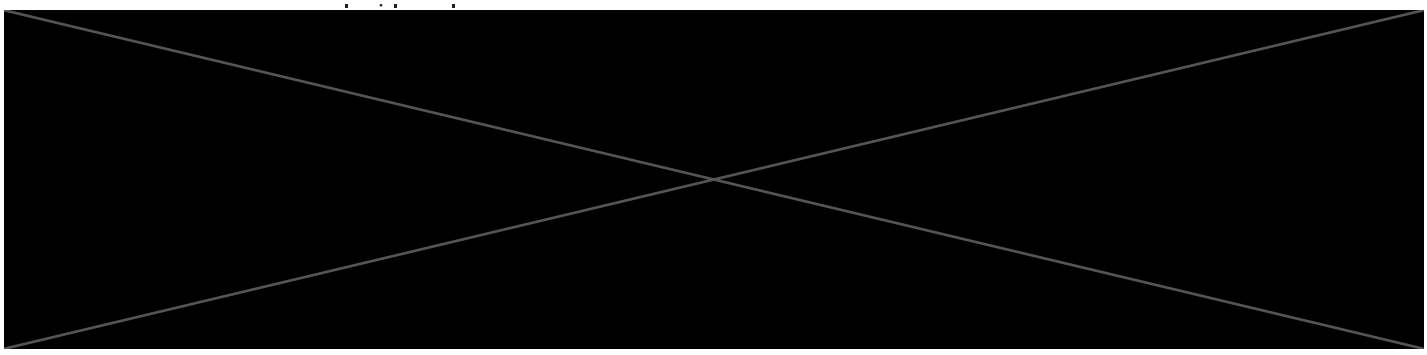
In [107…

```
# join drug exposures with condition_occurrences
overlap = clopidogrel_3.set_index('person_id').join(
    gi_cond5.set_index('person_id'), on='person_id', how='left').reset_index()
```

In [109…

```
# get rows where condition starts before drug exposure index date
# NOTE: patient can only be in dataset once now

a = overlap[overlap['index_date'] < overlap['procedure_date']]
a = a[a['follow_up_end_date'] > a['procedure_date']]

b = a.groupby('person_id').apply(lambda x: x.sort_values('procedure_date', ascen

overlap_df5 = b.reset_index(drop = True).set_index("person_id")
overlap_df5_er = overlap_df5[
    (overlap_df5['visit_concept_id'] == 262.0) |
    (overlap_df5['visit_concept_id'] == 9201.0)|
    (overlap_df5['visit_concept_id'] == 9203.0)]

assert len(a.person_id.unique())==len(b.person_id.unique())
display(overlap_df5.head(5))
create_download_link(overlap_df5, "Download Overlap CSV", "overlap5(TransfusionP
```

	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id						



3 rows × 57 columns

Out[109… Download Overlap CSV

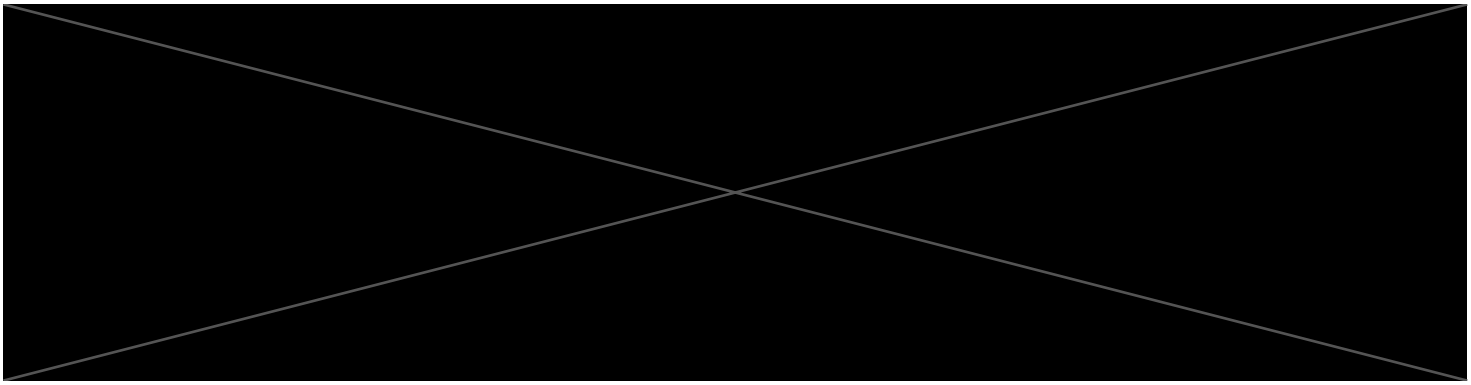
In [111…

```
overlap_df5
```

Out[111…

	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id						

	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id						



3 rows × 57 columns

Patient Counts per Condition for Instance Cohort

In [113…

```
d = {'Condition': ['1 (Bleeding)', '2 (Hemorrhage)', '3 (ICH)', '4 (ICH Min)', '
     'Patient Count': [overlap_df1.shape[0], overlap_df2.shape[0], overlap_df3.s
cond_counts_inst = pd.DataFrame(data=d)
cond_counts_inst
```

Out[113…

	Condition	Patient Count
0	1 (Bleeding)	194
1	2 (Hemorrhage)	197
2	3 (ICH)	19
3	4 (ICH Min)	11
4	5 (Transfusion Procedure)	3

Recurring Cohort

This table records all patients who are both in the drug exposure and the condition tables, looking specifically for patients whose `condition_start_date` is before their `index_date` yet after the `follow_up_end_date` . Note that this table, unlike that in section 4.1, is looking at every time a patient has a visit in relation to a condition after their drug-exposure, not just their first instance visit.

Condition 1

Overlap for Recurring Cohort
Gastrointestinal Bleeding

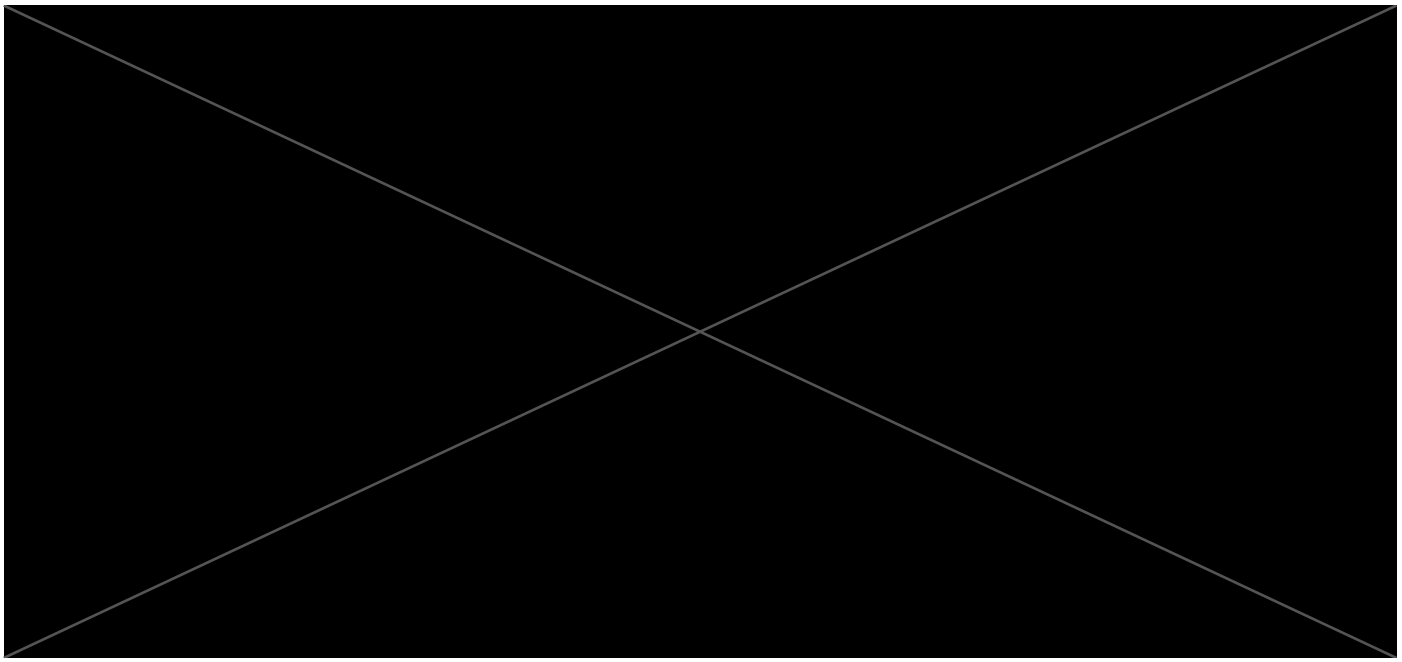
In [115…

```
# join drug exposures with condition_occurrences
overlap = clopidogrel_3.set_index('person_id').join(
    gi_cond1.set_index('person_id'), on='person_id', how='left').reset_index()
```

In [117…

```
# get rows where condition starts before drug exposure index date
# NOTE: patient can only be in dataset once now

#index_date = time patient was given drug (Clopidogrel)

a = overlap[overlap['index_date'] < overlap['condition_start_date']]
a = a[a['follow_up_end_date'] > a['condition_start_date']]

b = a.groupby('person_id').apply(lambda x: x.sort_values('condition_start_date',

overlap_df1_all = b.reset_index(drop = True).set_index("person_id")
overlap_df1_all_er = overlap_df1[(overlap_df1['visit_concept_id'] == 262.0) | (o

assert len(a.person_id.unique())==len(b.person_id.unique())
display(overlap_df1.head(5))
create_download_link(overlap_df1, "Download Overlap CSV", "overlap(gi_bleeding)_
```

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



5 rows × 61 columns

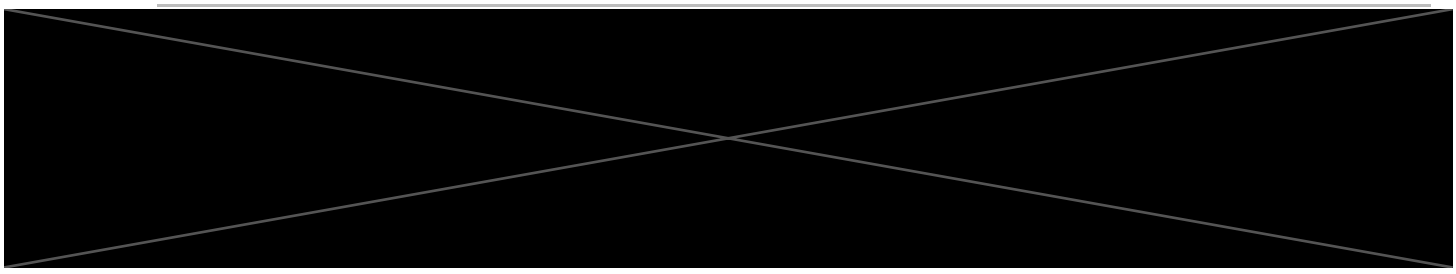
Out[117… Download Overlap CSV

In [119…

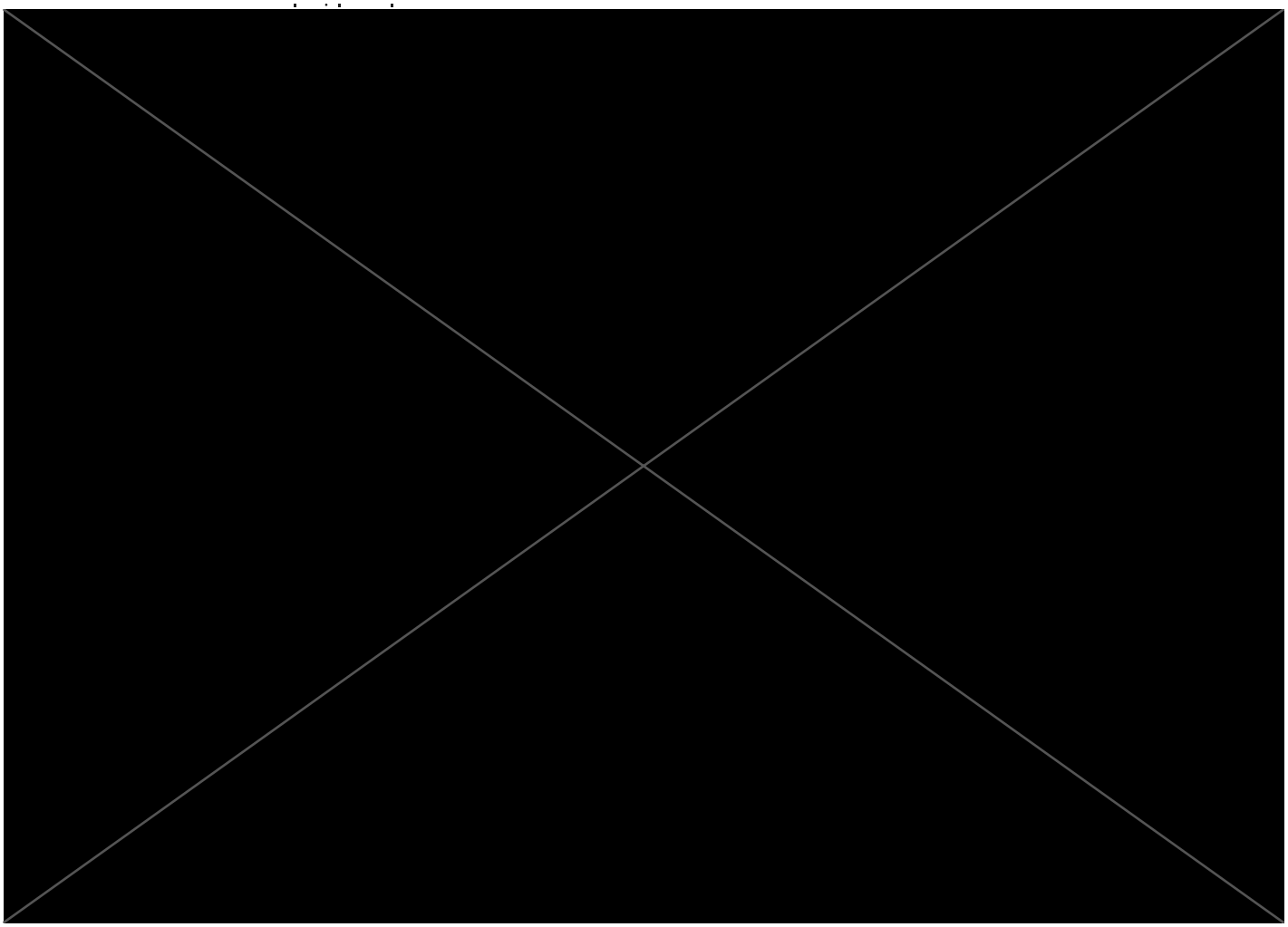
```
overlap_df1_all
```

Out[119…

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id						



746 rows × 61 columns

Condition 2

Overlap for Recurring Cohort

Gastrointestinal Hemorrhage

In [121…

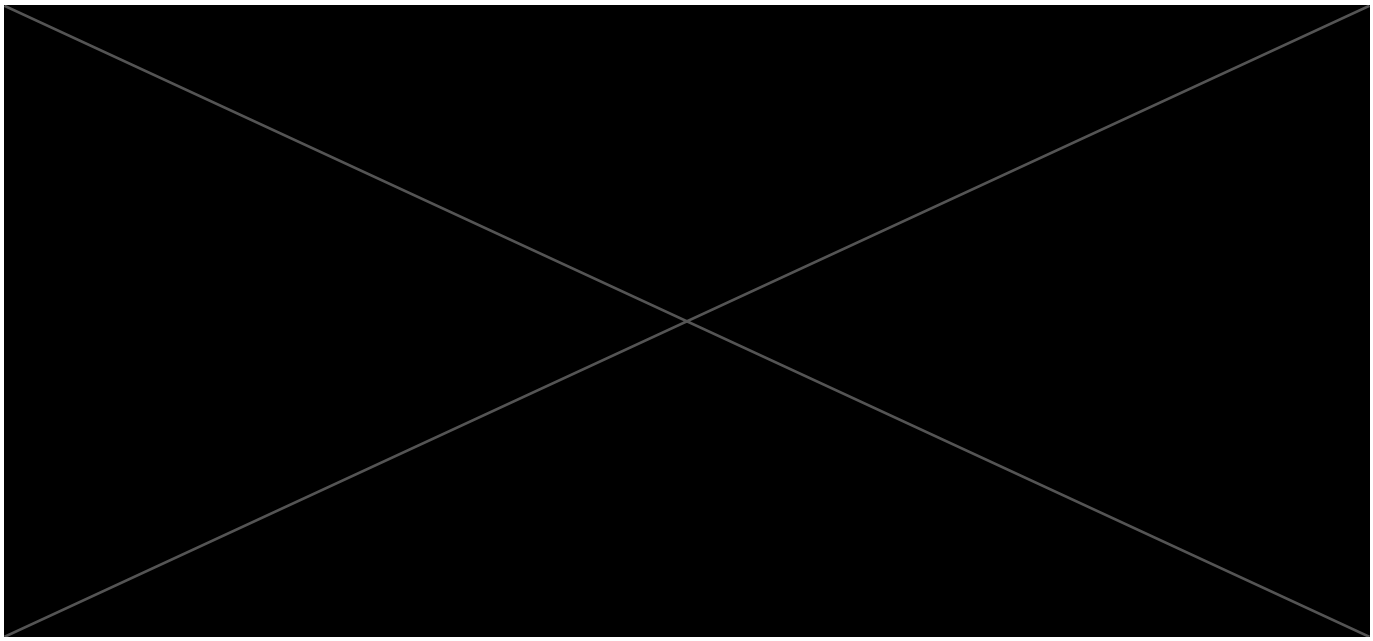
```
# join drug exposures with condition_occurrences
overlap = clopidogrel_3.set_index('person_id').join(
    gi_cond2.set_index('person_id'), on='person_id', how='left').reset_index()
```

In [123…

```
# get rows where condition starts before drug exposure index date
# NOTE: patient can only be in dataset once now

#index_date = time patient was given drug (Clopidogrel)

a = overlap[overlap['index_date'] < overlap['condition_start_date']]
a = a[a['follow_up_end_date'] > a['condition_start_date']]

b = a.groupby('person_id').apply(lambda x: x.sort_values('condition_start_date',
```

```
overlap_df2_all = b.reset_index(drop = True).set_index("person_id")
overlap_df2_all_er = overlap_df2[(overlap_df2['visit_concept_id'] == 262.0) | (o

assert len(a.person_id.unique())==len(b.person_id.unique())
display(overlap_df2.head(5))
create_download_link(overlap_df2, "Download Overlap CSV", "overlap(gi_hemorrhage
```

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



5 rows × 61 columns

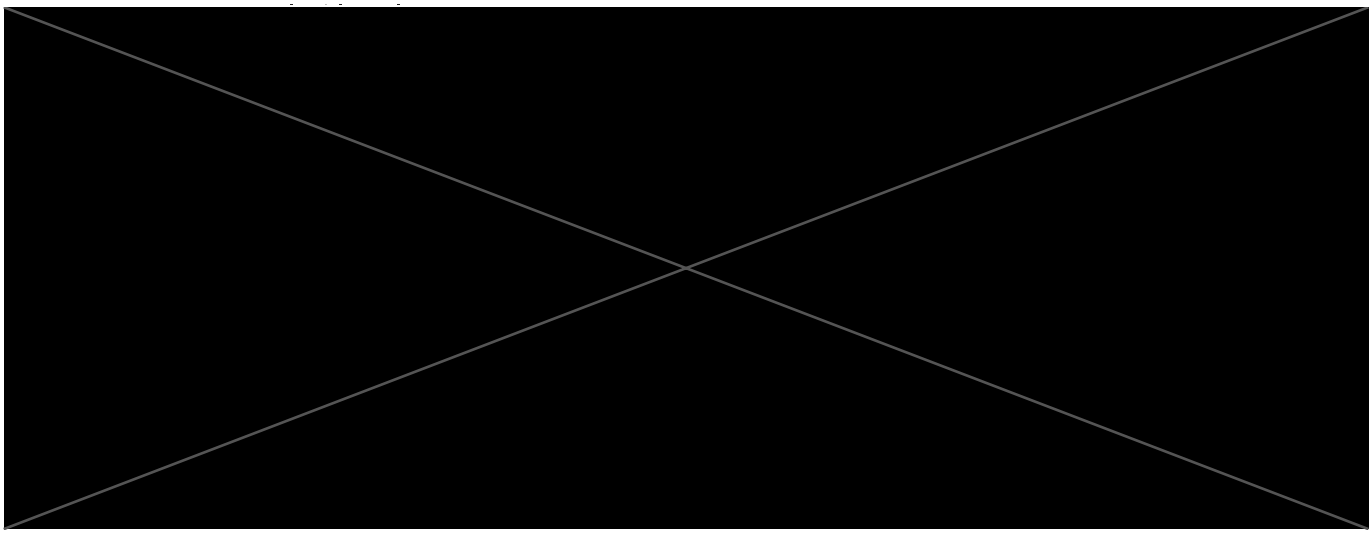
Out[123… Download Overlap CSV

In [125…

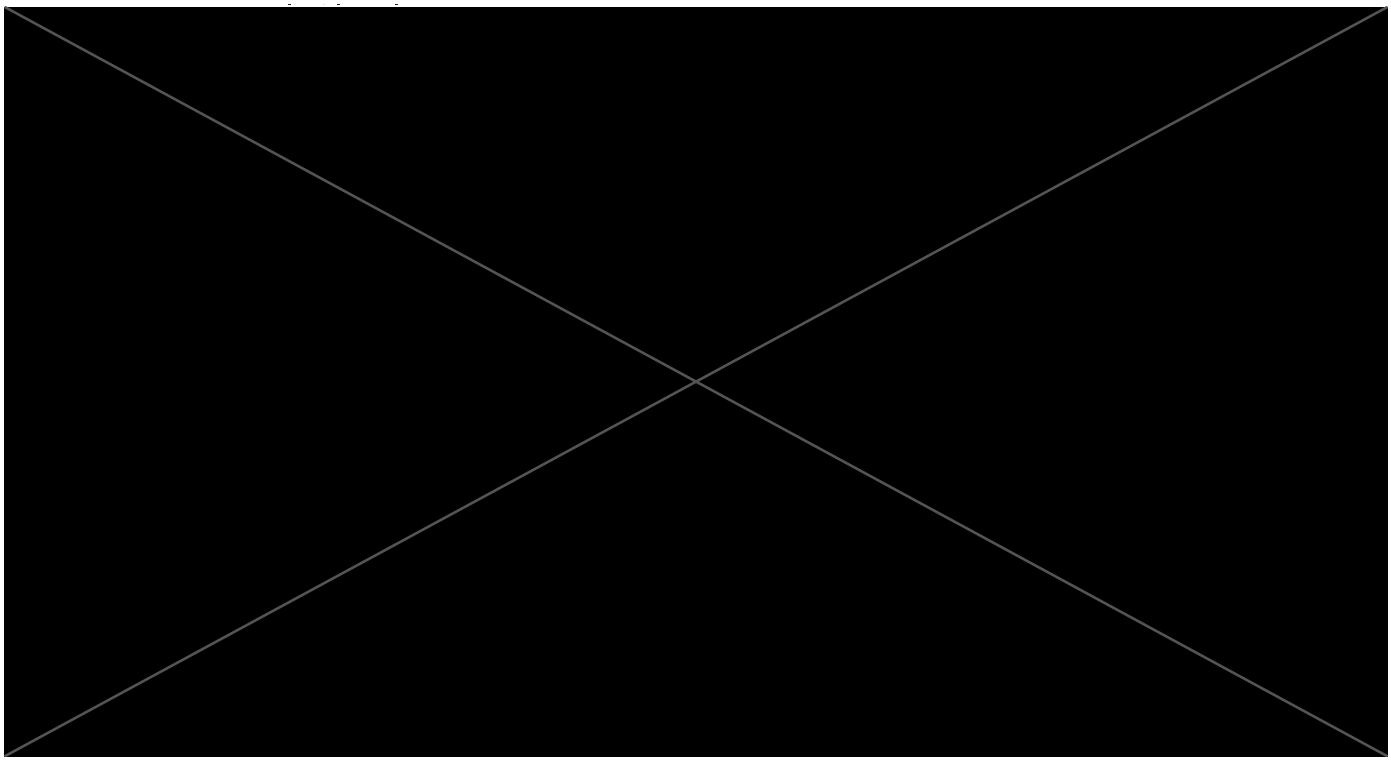
```
overlap_df2_all
```

Out[125…

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id						



701 rows × 61 columns

Condition 3

Overlap for Recurring Cohort

Intracranial Hemorrhage

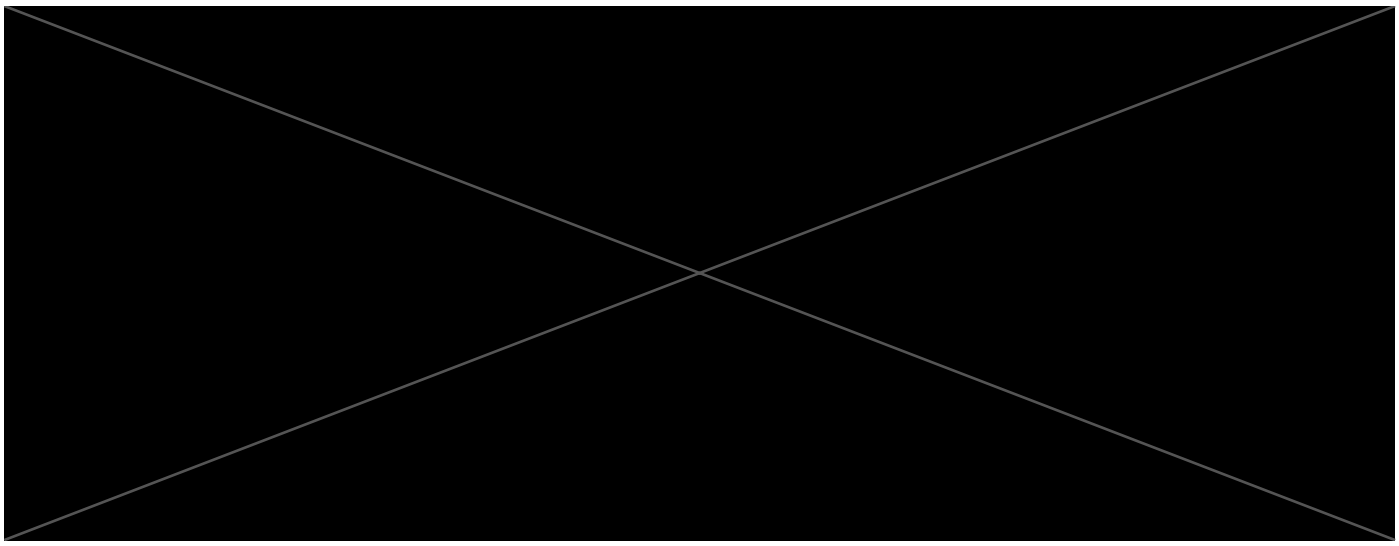
In [127…

```
# join drug exposures with condition_occurrences
overlap = clopidogrel_3.set_index('person_id').join(
    gi_cond3.set_index('person_id'), on='person_id', how='left').reset_index()
```

In [129…

```
# get rows where condition starts before drug exposure index date
# NOTE: patient can only be in dataset once now

#index_date = time patient was given drug (Clopidogrel)

a = overlap[overlap['index_date'] < overlap['condition_start_date']]
a = a[a['follow_up_end_date'] > a['condition_start_date']]

b = a.groupby('person_id').apply(lambda x: x.sort_values('condition_start_date',

overlap_df3_all = b.reset_index(drop = True).set_index("person_id")
overlap_df3_all_er = overlap_df3[(overlap_df3['visit_concept_id'] == 262.0) | (o

assert len(a.person_id.unique())==len(b.person_id.unique())
display(overlap_df3.head(5))
create_download_link(overlap_df3, "Download Overlap CSV", "overlap(strokeEvents)
```

	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id						



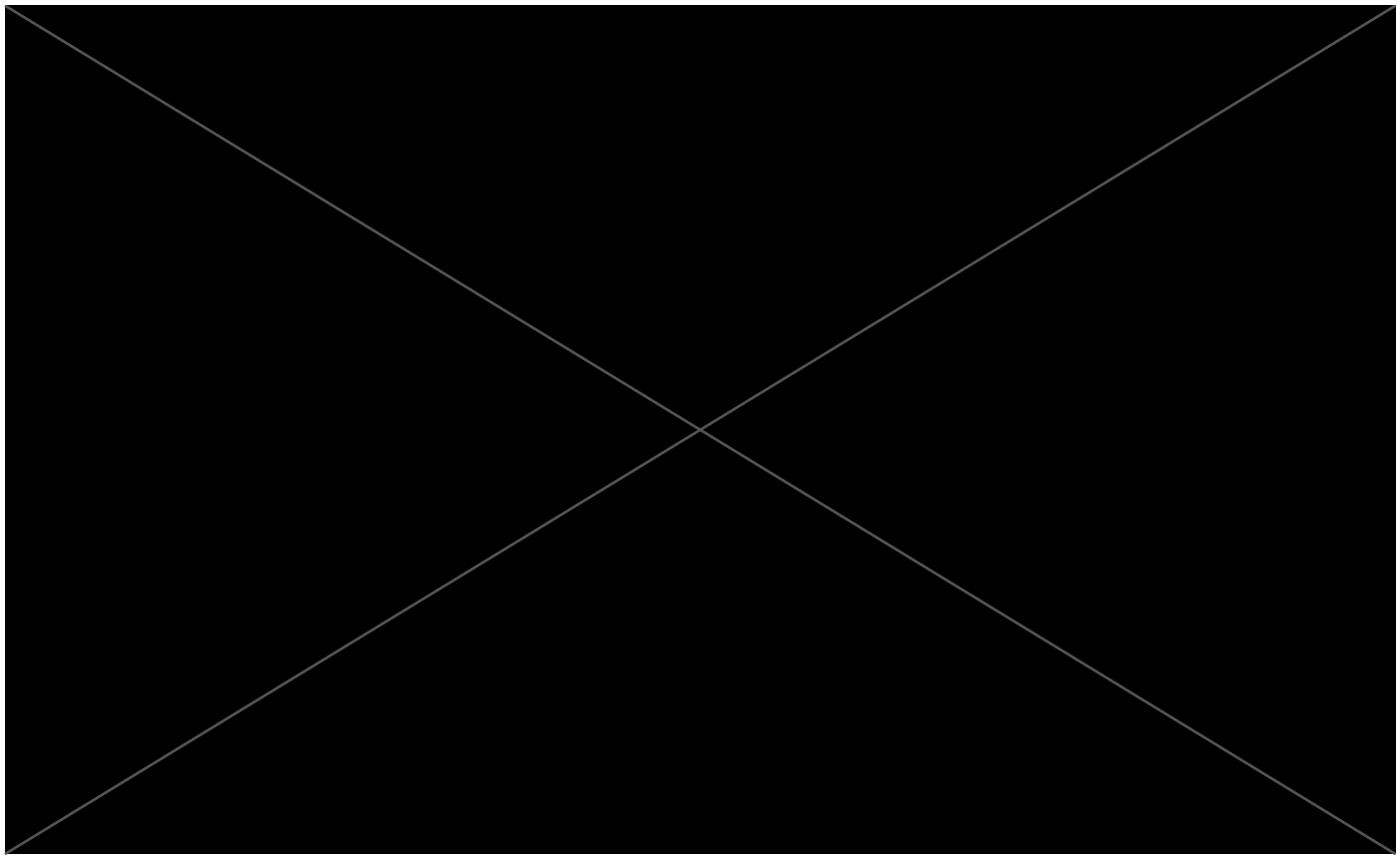
5 rows × 61 columns

Out[129… Download Overlap CSV

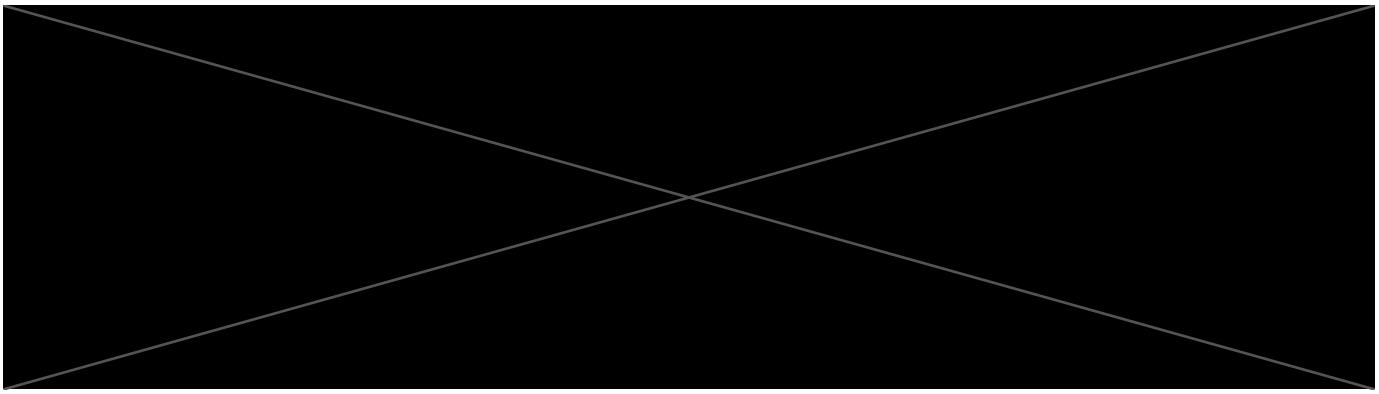
In [131…
```
overlap_df3_all
```

Out[131…

	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id						



person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



89 rows × 61 columns

Condition 4

Overlap for Recurring Cohort

Intracranial Hemorrhage (Minimized)

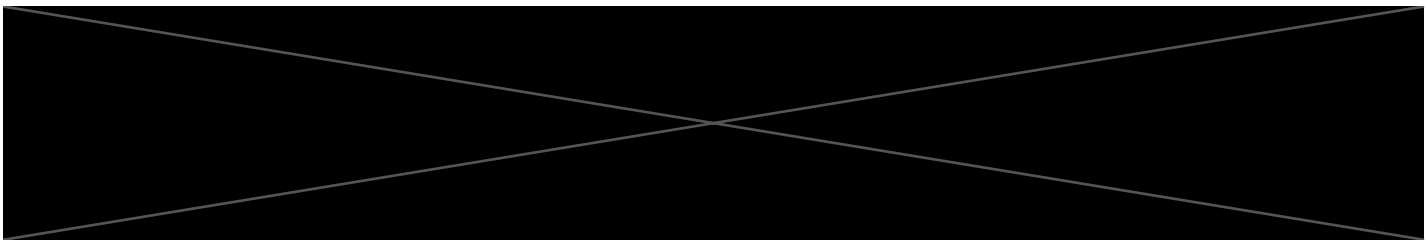
In [133…

```
# join drug exposures with condition_occurrences
overlap = clopidogrel_3.set_index('person_id').join(
    gi_cond4.set_index('person_id'), on='person_id', how='left').reset_index()
```

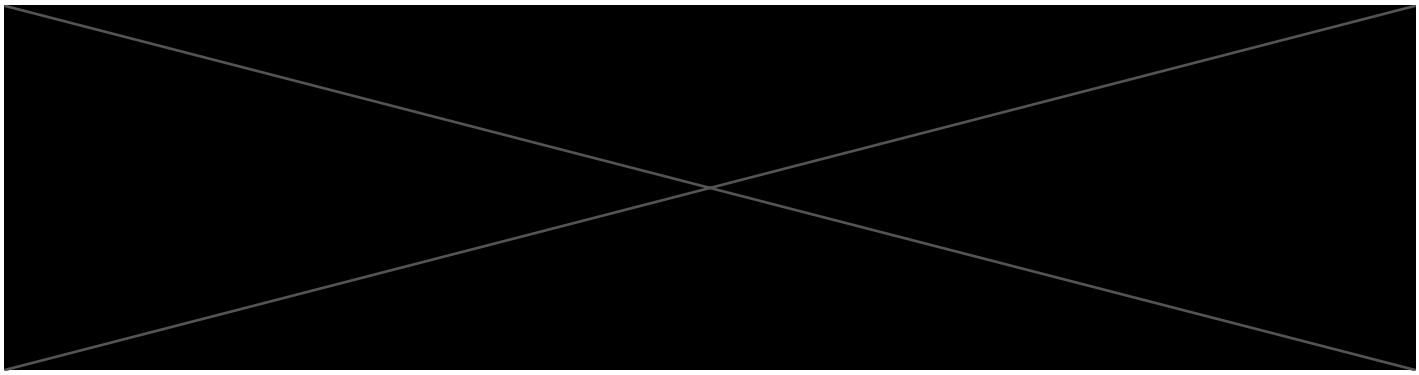
In [135…

```
# get rows where condition starts before drug exposure index date
# NOTE: patient can only be in dataset once now

#index_date = time patient was given drug (Clopidogrel)

a = overlap[overlap['index_date'] < overlap['condition_start_date']]
a = a[a['follow_up_end_date'] > a['condition_start_date']]

b = a.groupby('person_id').apply(lambda x: x.sort_values('condition_start_date',

overlap_df4_all = b.reset_index(drop = True).set_index("person_id")
overlap_df4_all_er = overlap_df4[(overlap_df4['visit_concept_id'] == 262.0) | (o

assert len(a.person_id.unique())==len(b.person_id.unique())
display(overlap_df4.head(5))
create_download_link(overlap_df4, "Download Overlap CSV", "overlap(strokeEventsM
```

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id						



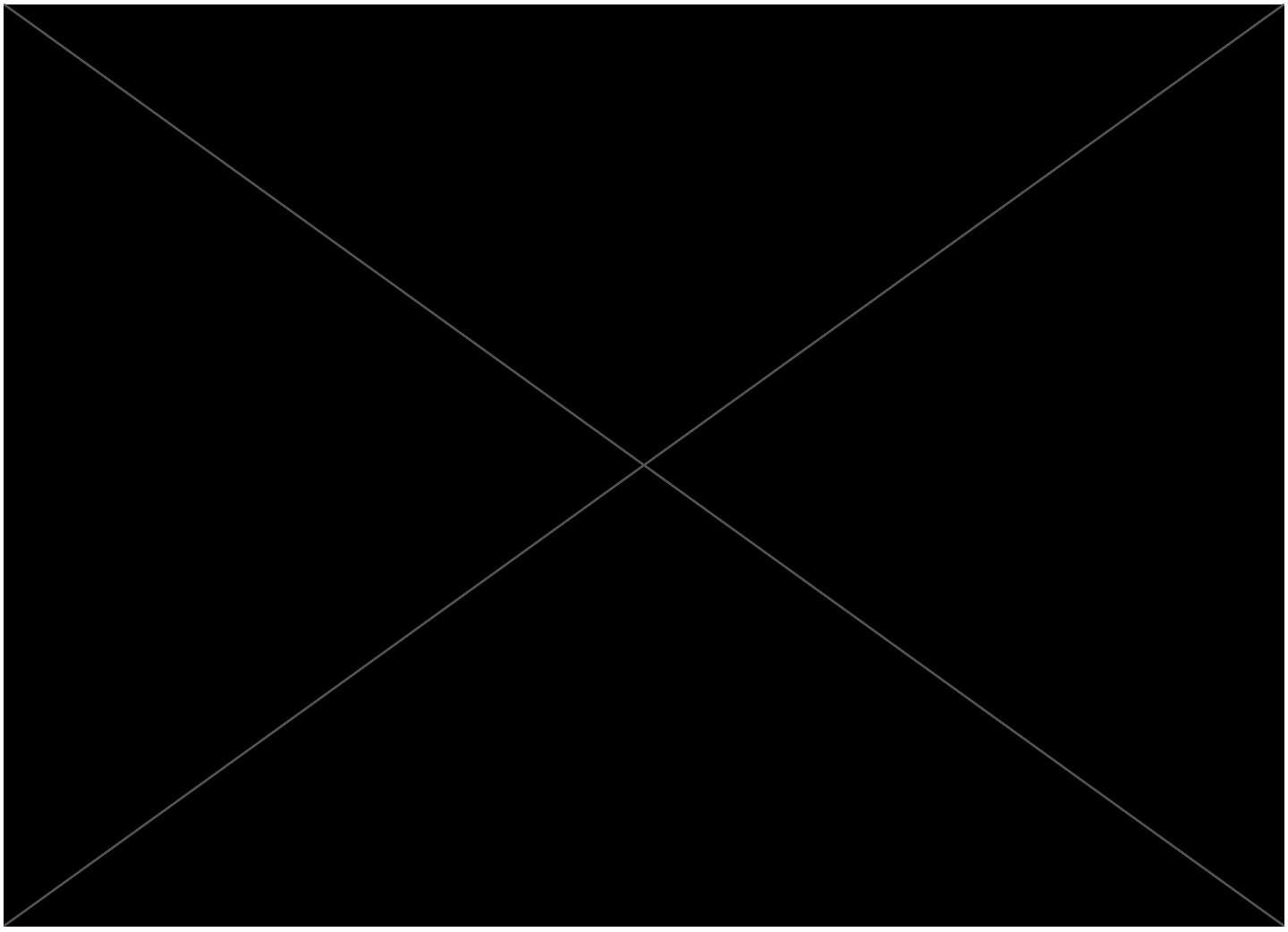
5 rows × 61 columns

Out[135… Download Overlap CSV

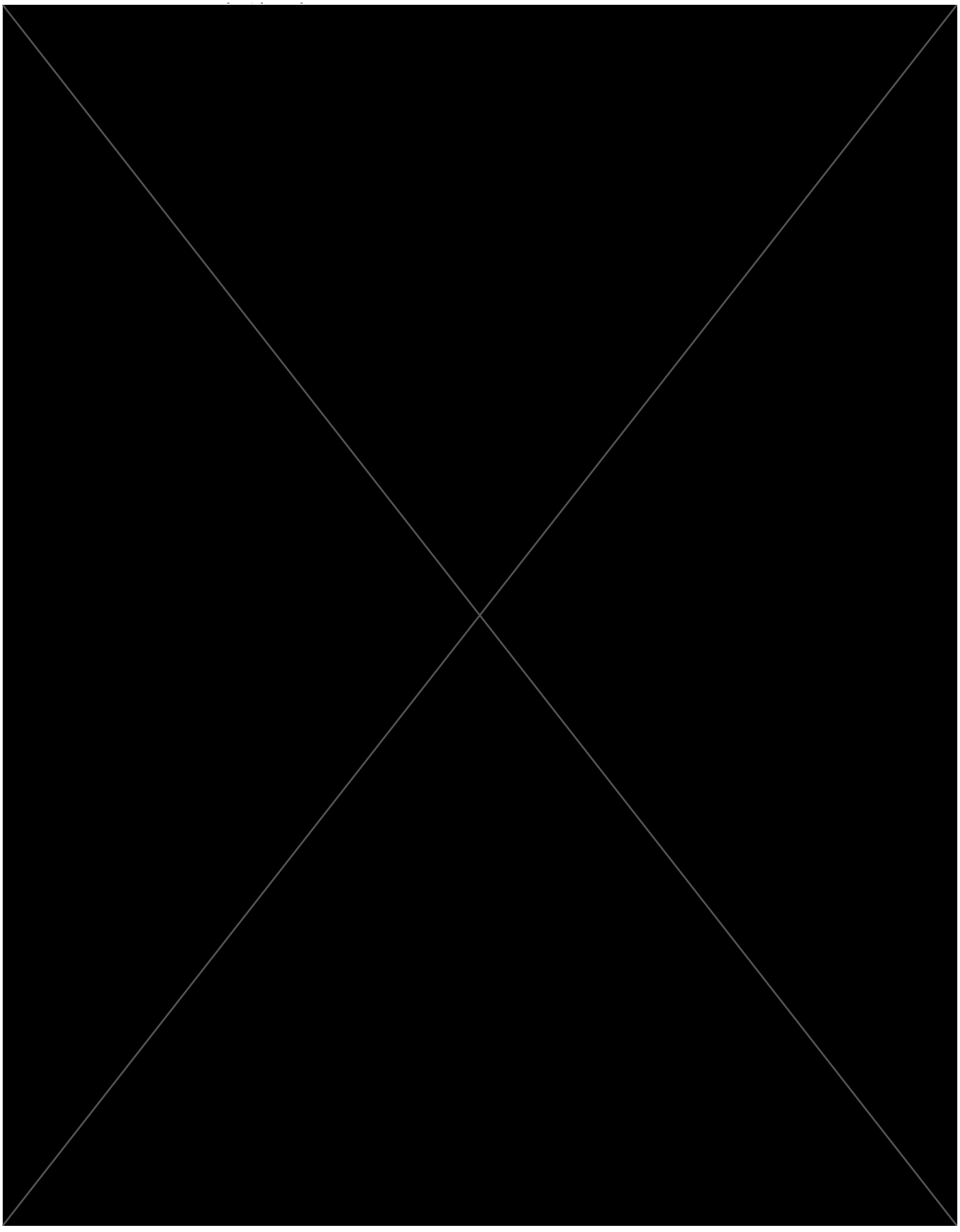
In [137…
```
overlap_df4_all
```

Out[137…

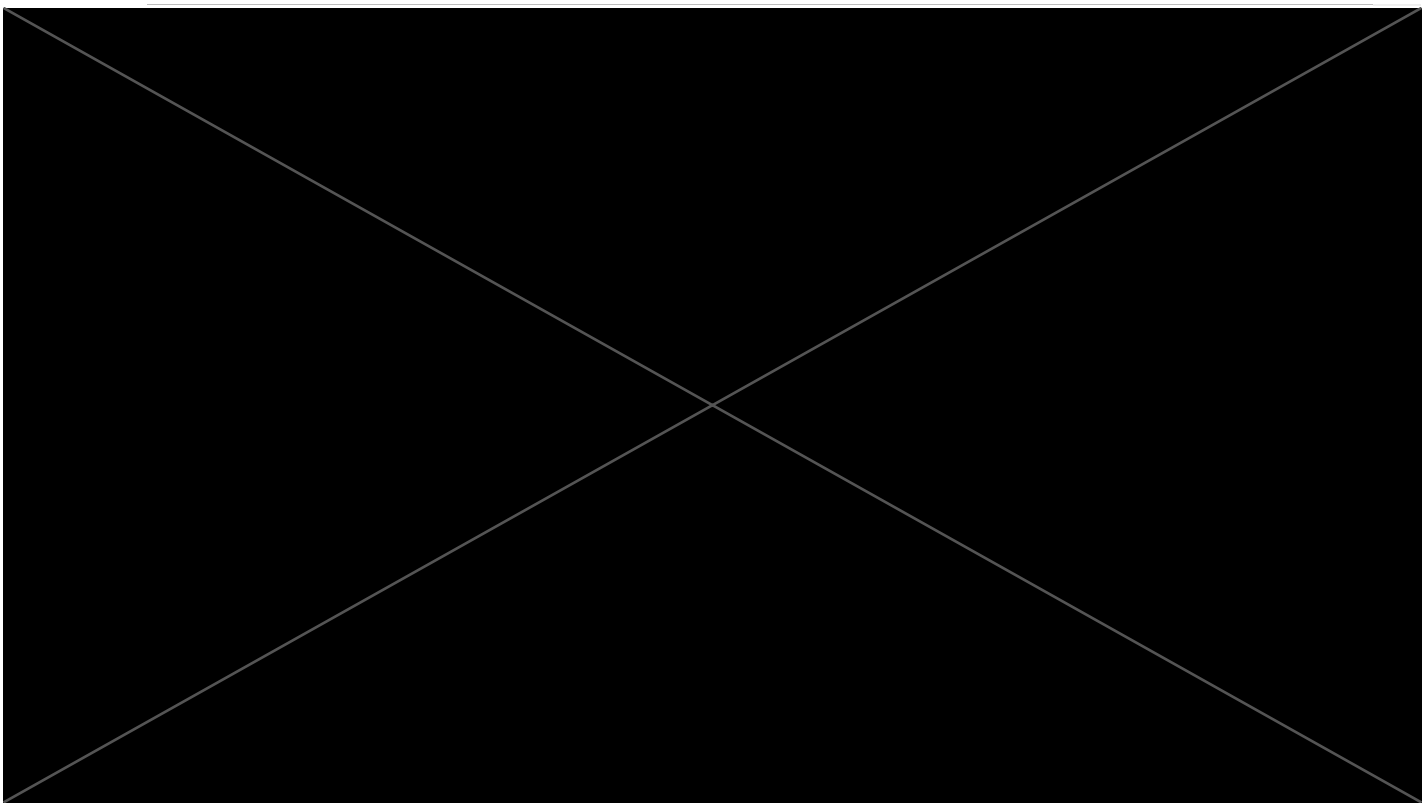
	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id						



drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll
person_id					



person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



29 rows × 61 columns

Condition 5

Overlap for Recurring Cohort

Platelet Transfusion

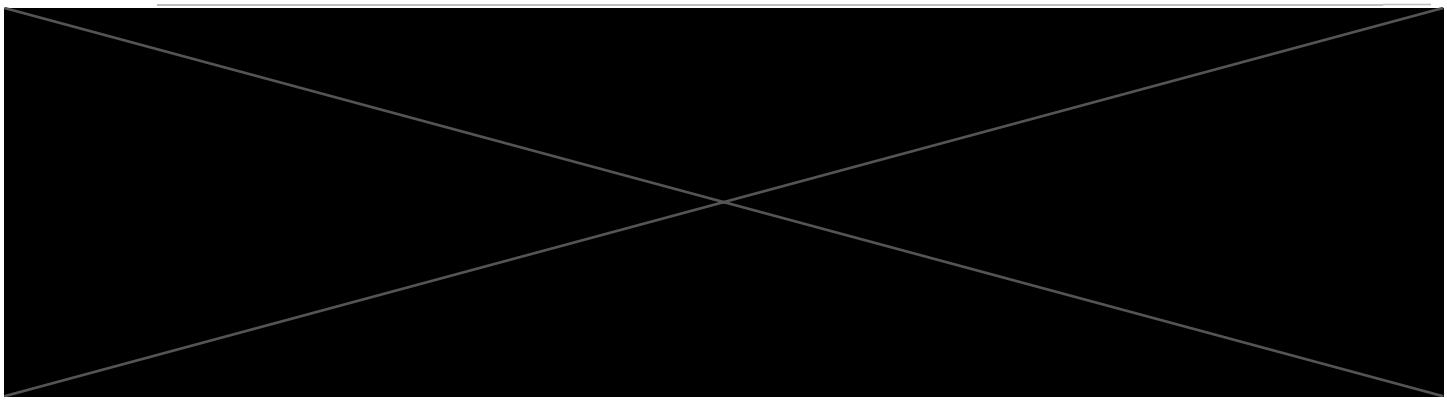
In [139…

```
# join drug exposures with condition_occurrences
overlap = clopidogrel_3.set_index('person_id').join(
    gi_cond5.set_index('person_id'), on='person_id', how='left').reset_index()
```

In [141…

```
# get rows where condition starts before drug exposure index date
# NOTE: patient can only be in dataset once now

#index_date = time patient was given drug (Clopidogrel)

a = overlap[overlap['index_date'] < overlap['procedure_date']]
a = a[a['follow_up_end_date'] > a['procedure_date']]

b = a.groupby('person_id').apply(lambda x: x.sort_values('procedure_date', ascen

overlap_df5_all = b.reset_index(drop = True).set_index("person_id")
overlap_df5_all_er = overlap_df5[(overlap_df5['visit_concept_id'] == 262.0) | (o

assert len(a.person_id.unique())==len(b.person_id.unique())
display(overlap_df5.head(5))
create_download_link(overlap_df5, "Download Overlap CSV", "overlap(platelet_tran
```

person_id	drug_name	before_visit_start_date	index_date	last_dispensed	last_days_supply	foll



3 rows × 57 columns

Out[141… Download Overlap CSV

In [143…
```
overlap_df5_all.visit_concept_id
```

Out[143…
```
person_id
1558103    9201.0
2507219    9201.0
2507219    9201.0
3183454    9201.0
Name: visit_concept_id, dtype: float64
```

Patient Counts per Condition for Recurring Cohort

In [145…
```
d = {'Condition': ['1 (Bleeding)', '2 (Hemorrhage)', '3 (ICH)', '4 (ICH Min)', '
     'Patient Count': [overlap_df1_all.shape[0], overlap_df2_all.shape[0], overl
cond_counts = pd.DataFrame(data=d)
cond_counts
```

Out[145…

	Condition	Patient Count
0	1 (Bleeding)	746
1	2 (Hemorrhage)	701
2	3 (ICH)	89
3	4 (ICH Min)	29
4	5 (Transfusion Procedure)	4

In [147…
```
overlap_df5.columns
```

Out[147…
```
Index(['drug_name', 'before_visit_start_date', 'index_date', 'last_dispensed',
       'last_days_supply', 'follow_up_end_date', 'follow_up_period',
       'visit_after', 'avg_days_bw_records', 'dispensed_count',
       'avg_days_supply', 'gender_concept_id', 'year_of_birth',
       'month_of_birth', 'day_of_birth', 'birth_datetime', 'race_concept_id',
       'ethnicity_concept_id', 'location_id', 'provider_id', 'care_site_id',
       'person_source_value', 'gender_source_value',
       'gender_source_concept_id', 'race_source_value',
```

```
       'race_source_concept_id', 'ethnicity_source_value',
       'ethnicity_source_concept_id', 'sex_at_birth_concept_id',
       'sex_at_birth_source_concept_id', 'sex_at_birth_source_value', 'race',
       'gender', 'ethnicity', 'sex_at_birth', 'care_sites', 'procedure_name',
       'procedure_concept_id', 'procedure_date', 'procedure_type_concept_id',
       'visit_occurrence_id', 'visit_occurrence_id_1', 'person_id_1',
       'visit_concept_id', 'visit_start_date', 'visit_start_datetime',
       'visit_end_date', 'visit_end_datetime', 'visit_type_concept_id',
       'visit_source_value', 'visit_source_concept_id',
       'admitting_source_concept_id', 'admitting_source_value',
       'discharge_to_concept_id', 'discharge_to_source_value',
       'preceding_visit_occurrence_id', 'procedure_source_concept_id'],
      dtype='object')
```

Final Patient Counts

In [149…

```
d = {'Condition': ['2 (GI Hemorrhage)', '3 (ICH)', '5 (Platelet Transfusion)'],
     'Patient Count': [overlap_df2.shape[0], overlap_df3.shape[0], overlap_df5.s
     'Patient Count (ER/Inpatient Only)': [overlap_df2_er.shape[0], overlap_df3_
     'Recurring Patient Count': [overlap_df2_all.shape[0], overlap_df3_all.shape
     'Recurring Patient Count (ER/Inpatient Only)': [overlap_df2_all_er.shape[0]
cond_counts_inst = pd.DataFrame(data=d)
cond_counts_inst
```

Out[149…

	Condition	Patient Count	Patient Count (ER/Inpatient Only)	Recurring Patient Count	Recurring Patient Count (ER/Inpatient Only)
0	2 (GI Hemorrhage)	197	89	701	89
1	3 (ICH)	19	5	89	5
2	5 (Platelet Transfusion)	3	3	4	3

Visualizations

Percent Patients with ADE Conditions

In [151…

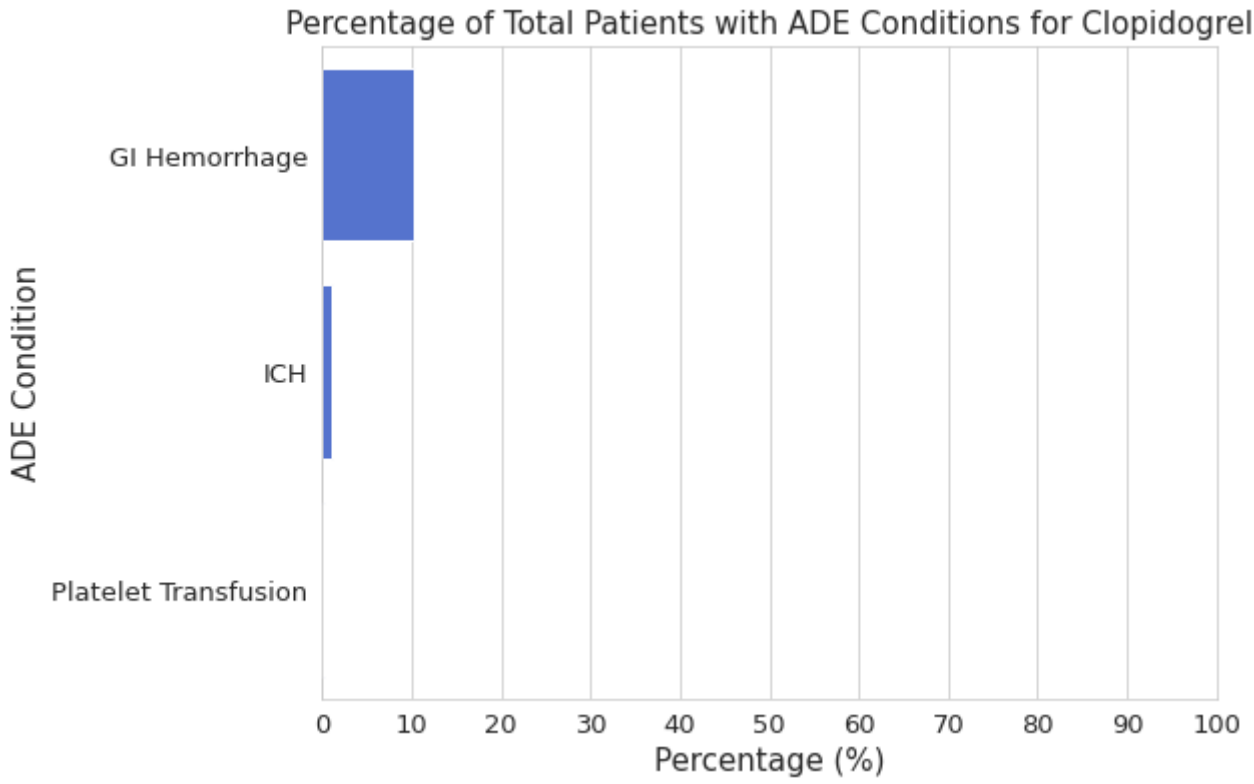
```
cond_counts_inst['Total'] = [1907]*3
cond_counts_inst['Percentage'] = 100*cond_counts_inst['Patient Count']/cond_coun
cond_counts_inst
```

Out[151…

	Condition	Patient Count	Patient Count (ER/Inpatient Only)	Recurring Patient Count	Recurring Patient Count (ER/Inpatient Only)	Total	Percentage
0	2 (GI Hemorrhage)	197	89	701	89	1907	10.330362
1	3 (ICH)	19	5	89	5	1907	0.996329
2	5 (Platelet Transfusion)	3	3	4	3	1907	0.157315

In [153…

```
sns.set_style("whitegrid")
fig, ax = plt.subplots(1, 1, figsize=(8,6))
# fig.suptitle('Visit Type Frequency for All Events', fontsize = 20)
# fig.subplots_adjust(hspace=0.4, wspace = 1)

# Condition 1
sns.barplot(ax = ax, data = cond_counts_inst, x='Percentage', y='Condition', col
ax.set_title('Percentage of Total Patients with ADE Conditions for Clopidogrel',
ax.set_xlabel('Percentage (%)', fontsize = 15)
ax.set_ylabel('ADE Condition', fontsize = 15)
ax.set_xticks(range(0,110,10))
ax.tick_params(labelsize=13)
ax.set_yticklabels(['GI Hemorrhage', 'ICH', 'Platelet Transfusion'])

plt.show()
```



Popular Condition Codes Frequency

Data Wrangle

In [155…

```
overlaps = [overlap_df2, overlap_df3, overlap_df5]
cond_all_ct = []
cond_show_tbl = []

for ov in overlaps:
    if 'condition_concept_id' in ov.columns:
        cond_ct = ov.groupby([
            'condition_concept_id', 'condition_name']).count().reset_index()[[
            'condition_concept_id', 'condition_name', 'drug_name']]
        cond_ct.columns = ['condition_concept_id', 'condition_name', 'count']
        cond_ct.condition_concept_id[:cond_ct.shape[0]] = (
            cond_ct.condition_concept_id[:cond_ct.shape[0]].astype(int).map(str)
    else:
```

```
        cond_ct = ov.groupby([
            'procedure_concept_id', 'procedure_name']).count().reset_index()[[
            'procedure_concept_id', 'procedure_name', 'drug_name']]
        cond_ct.columns = ['procedure_concept_id', 'procedure_name', 'count']
        cond_ct.procedure_concept_id[:cond_ct.shape[0]] = (
            cond_ct.procedure_concept_id[:cond_ct.shape[0]].astype(int).map(str)
    cond_ct = cond_ct.sort_values(by = ["count"], ascending = False)
    cond_show_tbl.append(cond_ct.reset_index(drop = True))
    cond_ct = cond_ct.head(8)
    cond_all_ct.append(cond_ct)
```

In [157…

```
# overlap_df1_ip_only = overlap_df1[(overlap_df1['visit_concept_id'] == 9201.0)]

# overlap_df2_ip_only = overlap_df2[(overlap_df2['visit_concept_id'] == 9201.0)]

# overlap_df3_ip_only = overlap_df3[(overlap_df3['visit_concept_id'] == 9201.0)]

# overlap_df4_ip_only = overlap_df4[(overlap_df4['visit_concept_id'] == 9201.0)]

# overlap_df5_ip_only = overlap_df5[(overlap_df5['visit_concept_id'] == 9201.0)]
```

In [159…

```
# overlap_df1_erip_only = overlap_df1[(overlap_df1['visit_concept_id'] == 262.0)

# overlap_df2_erip_only = overlap_df2[(overlap_df2['visit_concept_id'] == 262.0)

# overlap_df3_erip_only = overlap_df3[(overlap_df3['visit_concept_id'] == 262.0)

# overlap_df4_erip_only = overlap_df4[(overlap_df4['visit_concept_id'] == 262.0)

# overlap_df5_erip_only = overlap_df5[(overlap_df5['visit_concept_id'] == 262.0)
```

In [161…

```
# d = {'condition': ["GI Hemorrhage", "Intracranial Hemorrhage", "Platelet Trans
#      'patientCounts': [overlap_df2.shape[0],overlap_df3.shape[0],overlap_df4.s
#      'ER Visit Counts' : [overlap_df1_er_only.shape[0],overlap_df2_er_only.sha
#     'Inpatient Visit Counts' : [overlap_df1_ip_only.shape[0],overlap_df2_ip_on
#     'ER/Inpatient Counts': [overlap_df1_erip_only.shape[0], overlap_df2_erip_o
#      'Total ER and Inpatient Visit Counts' : [overlap_df1_er.shape[0],overlap_
# conditionCounts = pd.DataFrame(data=d)
# conditionCounts.set_index("condition", inplace = True)
```

In [163…

```
# conditionCounts
```

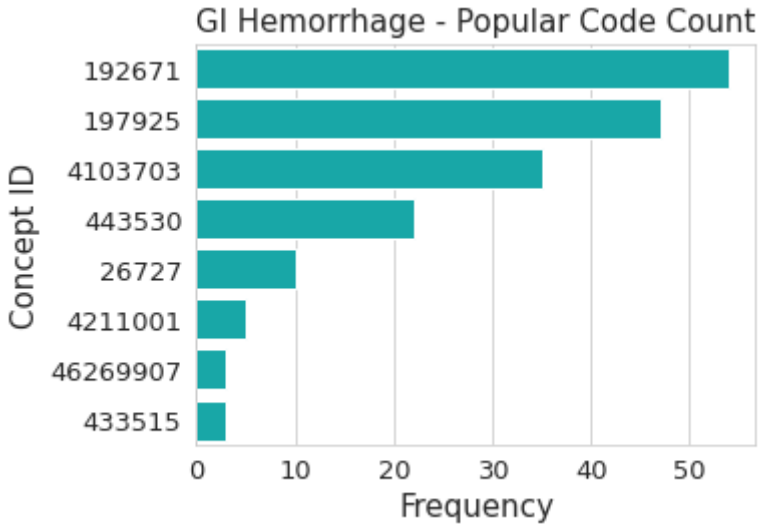
Plots

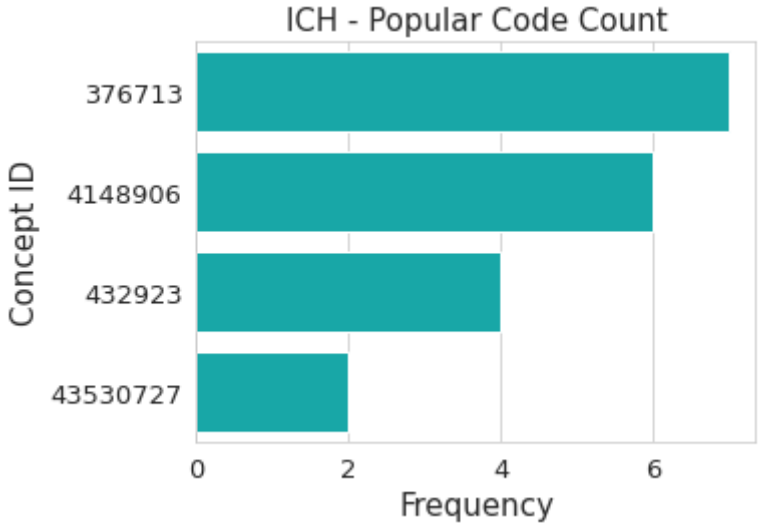
In [165…

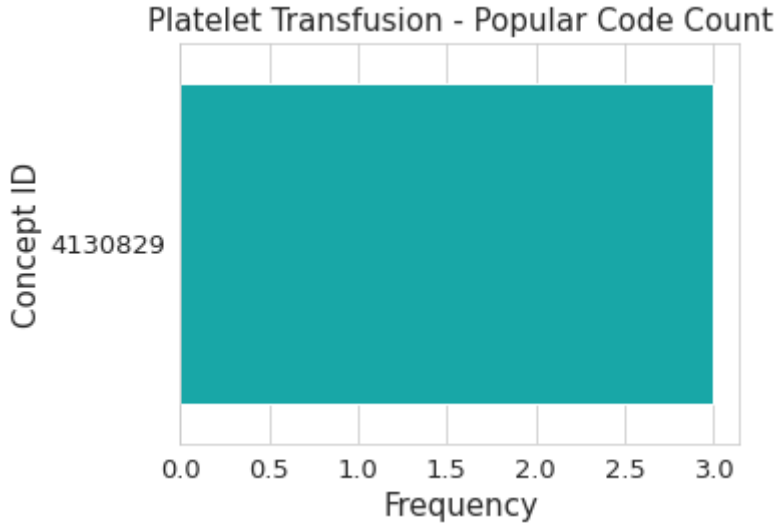
```
# fig, axes = plt.subplots(1, 1, figsize=(10,8))

# #Condition1 Plot
# a = sns.barplot(data = cond1allCount, x='condition_concept_id', y='count', ord
# axes.set_title('Condition 1 (GI Bleeding) - Frequency of Common Codes', fontsi
# axes.set_xlabel('Condition Concept ID', fontsize = 15)
# axes.set_ylabel('Frequency', fontsize = 15)
# axes.tick_params(labelsize=13)

# axes.set(ylim = (0,100))


# for p in a.patches:
#     a.annotate(format(p.get_height(), '.0f'),
#                    (p.get_x() + p.get_width() / 2., p.get_height()),
#                    ha = 'center', va = 'center',
#                    xytext = (0, 9),
#                    textcoords = 'offset points', fontsize = 12)

# plt.show()
```

In [167…

```
fig, axes = plt.subplots(3, 1, figsize=(5,14))
fig.subplots_adjust(hspace=0.4, wspace = 1)

titles = ['GI Hemorrhage - Popular Code Count',
          'ICH - Popular Code Count',
          'Platelet Transfusion - Popular Code Count']

for i in range(len(overlaps)):
    if 'condition_concept_id' in cond_all_ct[i]:
        sns.barplot(ax = axes[i], data = cond_all_ct[i], x='count', y='condition
                    order = cond_all_ct[i].condition_concept_id.tolist(), co
        axes[i].set_title(titles[i], fontsize = 15)
        axes[i].set_xlabel('Frequency', fontsize = 15 )
        axes[i].set_ylabel('Concept ID', fontsize = 15)
        axes[i].tick_params(labelsize=13)
    else:
        sns.barplot(ax = axes[i], data = cond_all_ct[i], x='count', y='procedure
                    order = cond_all_ct[i].procedure_concept_id.tolist(), co
        axes[i].set_title(titles[i], fontsize = 15)
        axes[i].set_xlabel('Frequency', fontsize = 15 )
        axes[i].set_ylabel('Concept ID', fontsize = 15)
        axes[i].tick_params(labelsize=13)

plt.show()
```







In [169…

```
cond_show_tbl[0]
```

Out[169…

	condition_concept_id	condition_name	count
0	192671	Gastrointestinal hemorrhage	54
1	197925	Hemorrhage of rectum and anus	47

	condition_concept_id	condition_name	count
2	4103703	Melena	35
3	443530	Hematochezia	22
4	26727	Hematemesis	10
5	4211001	Chronic gastric ulcer with hemorrhage	5
6	46269907	Intestinal hemorrhage with diverticulosis	3
7	433515	Chronic gastrojejunal ulcer with hemorrhage	3
8	442190	Hemorrhage of colon	2
9	26441	Bleeding ulcer of esophagus	2
10	316457	Mallory-Weiss syndrome	2
11	193250	Gastric hemorrhage	2
12	198798	Dieulafoy's vascular malformation	1
13	436148	Chronic duodenal ulcer with hemorrhage but wit...	1
14	197018	Chronic gastric ulcer with hemorrhage but with...	1
15	193249	Acute hemorrhagic gastritis	1
16	30770	Chronic peptic ulcer with hemorrhage but witho...	1
17	4231580	Acute gastric ulcer with hemorrhage	1
18	4232181	Chronic duodenal ulcer with hemorrhage	1
19	45757543	Hemorrhage of colon due to diverticulosis	1
20	45757654	Intestinal hemorrhage due to angiodysplasia of...	1
21	434402	Acute duodenal ulcer with hemorrhage but witho...	1

In [171…
```
cond_show_tbl[1]
```

Out[171…

	condition_concept_id	condition_name	count
0	376713	Cerebral hemorrhage	7
1	4148906	Spontaneous subarachnoid hemorrhage	6
2	432923	Subarachnoid hemorrhage	4
3	43530727	Spontaneous cerebral hemorrhage	2

In [173…
```
cond_show_tbl[2]
```

Out[173…

	procedure_concept_id	procedure_name	count
0	4130829	Platelet transfusion	3

In [175…
```
# # rename cond5 table
# cond5allCount.columns = cond4allCount.columns
```

```
# # create condition column in all 5 dfs
# # cond1allCount['Condition'] = [1]*cond1allCount.shape[0]
# cond2allCount['Condition'] = [2]*cond2allCount.shape[0]
# cond3allCount['Condition'] = [3]*cond3allCount.shape[0]
# # cond4allCount['Condition'] = [4]*cond4allCount.shape[0]
# cond5allCount['Condition'] = [5]*cond5allCount.shape[0]

# cond12count = pd.concat([cond1allCount, cond2allCount])
# cond34count = pd.concat([cond3allCount, cond4allCount])
# cond34count['condition_concept_id'] = cond34count['condition_concept_id'].asty
# cond12count['condition_concept_id'] = cond12count['condition_concept_id'].map(
# cond34count['condition_concept_id'] = cond34count['condition_concept_id'].map(
```

Visit Type Frequency (2 plots)

Create visit count table for each condition

In [177…

```
overlap_dfs = [overlap_df2_all, overlap_df3_all, overlap_df5_all]

vis_id_dfs = []
for i in range(len(overlap_dfs)):
    vis_id_cts = overlap_dfs[i].groupby('visit_concept_id').count().reset_index(
    vis_id_cts.columns = ['visit_concept_id', 'count']
    vis_id_cts.sort_values('count', ascending = False, inplace = True)
    vis_id_cts.reset_index(drop = True, inplace = True)
    vis_id_cts[:vis_id_cts.shape[0]] = vis_id_cts[:vis_id_cts.shape[0]].astype(i
    vis_id_dfs.append(vis_id_cts)
```

Create dictionary to convert visit_concept_id to visit_type

In [179…

```
vis_id_lst = pd.concat([overlap_df2.visit_concept_id,
                        overlap_df3.visit_concept_id,
                        overlap_df5.visit_concept_id
                       ]).drop_duplicates().dropna().astype(int).tolist()

insert_visid = ",".join([f"('{x}')" for x in vis_id_lst])

vis_conv_sql = f"""
-- temp SQL table for condition codes for faster lookup using join
CREATE TEMP TABLE temp
(
  visit_concept_id STRING
);
INSERT INTO temp (
    visit_concept_id
)
VALUES {insert_visid};

-- create official table with correct integer cast
CREATE TEMP TABLE search_codes (
    visit_concept_id INT64
);

INSERT INTO search_codes
SELECT CAST(visit_concept_id as INT64) as visit_concept_id FROM temp;
```

```
--SELECT * FROM search_codes;

SELECT s.visit_concept_id, c.concept_name
FROM search_codes s
LEFT JOIN {os.environ["WORKSPACE_CDR"]}.concept c
        on s.visit_concept_id = c.concept_id ;
"""
vis_id_conv = pd.read_gbq(vis_conv_sql, dialect="standard")
```

Add visit type to vis_id_cts tables and groupby visit type

In [181…

```
vis_type_dfs = []
for i in range(len(vis_id_dfs)):
    vis_type_ct = vis_id_dfs[i].join(
        vis_id_conv.set_index('visit_concept_id'), on = 'visit_concept_id')[['co
    vis_type_ct = vis_type_ct.groupby('concept_name', dropna=False).sum().reset_
    vis_type_ct.sort_values('count', ascending = False, inplace = True)
    vis_type_ct['concept_name'][
        vis_type_ct['concept_name'] == 'Emergency Room and Inpatient Visit'] = [
        'ER and Inpatient Visit']*vis_type_ct[vis_type_ct[
        'concept_name'] == 'Emergency Room and Inpatient Visit'].shape[0]
    vis_type_dfs.append(vis_type_ct)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  # Remove the CWD from sys.path while we load stuff.
```

In [183…

```
# # wrangle data to suit a join
vis_type_jn1 = vis_type_dfs[0].copy()
vis_type_jn2 = vis_type_dfs[1].copy()
vis_type_jn3 = vis_type_dfs[2].copy()

vis_type_jn1.columns = ['concept_name', 'GI_Hemorr']
vis_type_jn2.columns = ['concept_name', 'ICH']
vis_type_jn3.columns = ['concept_name', 'Platelet']

# join all 4 conditions by visit type
vis_type_cts_all = vis_type_jn1.join(
    vis_type_jn2.set_index('concept_name'), on = 'concept_name').join(
    vis_type_jn3.set_index('concept_name'), on = 'concept_name')

# convert all columns to integers
vis_type_cts_all = vis_type_cts_all.fillna(0)
vis_type_cts_all.GI_Hemorr[
    :vis_type_cts_all.shape[0]] = vis_type_cts_all.GI_Hemorr[:vis_type_cts_all.s
vis_type_cts_all.ICH[
    :vis_type_cts_all.shape[0]] = vis_type_cts_all.ICH[:vis_type_cts_all.shape[0
vis_type_cts_all.Platelet[
    :vis_type_cts_all.shape[0]] = vis_type_cts_all.Platelet[:vis_type_cts_all.sh

vis_type_cts_all.columns = ['Visit_Type', 'GI_Hemorr', 'ICH', 'Platelet']
vis_type_cts_all
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: SettingWithCopy
Warning:
```

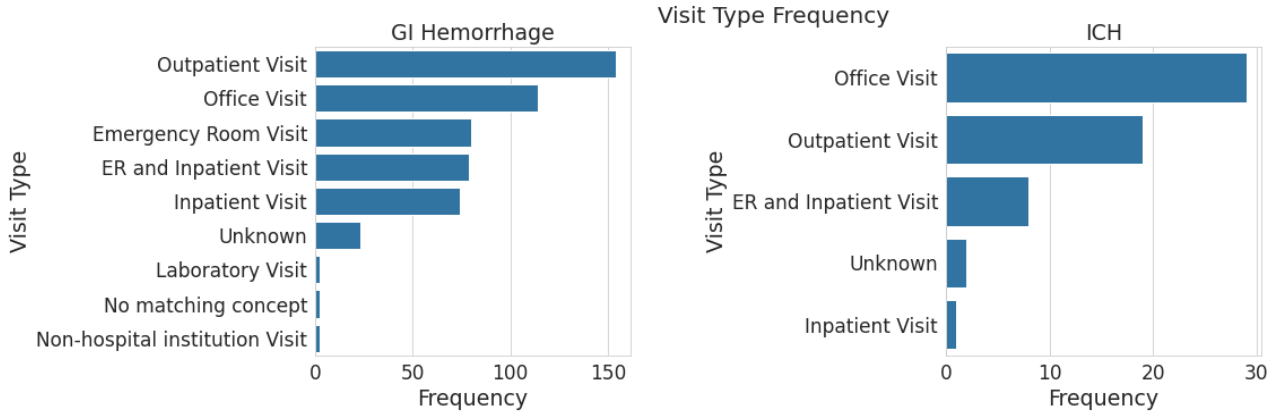
```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:20: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:22: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[183…

	Visit_Type	GI_Hemorr	ICH	Platelet
7	Outpatient Visit	154	19	0
6	Office Visit	114	29	0
0	Emergency Room Visit	80	0	0
1	ER and Inpatient Visit	79	8	0
2	Inpatient Visit	74	1	4
8	Unknown	23	2	0
3	Laboratory Visit	2	0	0
4	No matching concept	2	0	0
5	Non-hospital institution Visit	2	0	0

In [185…

```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))
fig.suptitle('Visit Type Frequency', fontsize = 20)
fig.subplots_adjust(hspace=0.4, wspace = 1)
titles = ['GI Hemorrhage', 'ICH', 'Platelet Transfusion']


sns.barplot(ax=axes[0], data = vis_type_dfs[0], x='count', y='concept_name', col
axes[0].set_title(titles[0], fontsize = 19)
axes[0].set_xlabel('Frequency', fontsize = 19)
axes[0].set_ylabel('Visit Type', fontsize = 19)
axes[0].tick_params(labelsize=17)

sns.barplot(ax=axes[1], data = vis_type_dfs[1], x='count', y='concept_name', col
axes[1].set_title(titles[1], fontsize = 19)
axes[1].set_xlabel('Frequency', fontsize = 19)
axes[1].set_ylabel('Visit Type', fontsize = 19)
axes[1].tick_params(labelsize=17)

plt.show()

# Condition 5 plot is not included as visit type is all procedure due to the eve
# unnecessary plot.
```



Subplots of Visit Type Frequency

In [187…

```
# vis_type_ct4
```

In [189…

```
# fig, axes = plt.subplots(1, 2, figsize=(18,8))
# fig.suptitle('Visit Type Frequency for All Events', fontsize = 20)
# fig.subplots_adjust(hspace=0.4, wspace = 1)

# # # Condition 1
# # sns.barplot(ax = axes[0, 0], data = vis_type_ct1, x='count', y='concept_name
# # axes[0, 0].set_title('Condition 1 (GI Bleeding)', fontsize = 17)
# # axes[0, 0].set_xlabel('Frequency', fontsize = 17)
# # axes[0, 0].set_ylabel('Visit Type', fontsize = 17)
# # axes[0, 0].tick_params(labelsize=15)
# # axes[0, 0].set(xlim=(0, 160))

# # Condition 2
# sns.barplot(ax=axes[0], data = vis_type_ct2, x='count', y='concept_name', colo
# axes[0].set_title('Condition 2 (GI Hemorrhage)', fontsize = 17)
# axes[0].set_xlabel('Frequency', fontsize = 17)
# axes[0].set_ylabel('Visit Type', fontsize = 17)
# axes[0].set_xticks(range(0, 130, 20))
# axes[0].tick_params(labelsize=15)
# axes[0].set(xlim=(0, 160))

# # Condition 3
# sns.barplot(ax=axes[1], data = vis_type_ct3, x='count', y='concept_name', colo
# axes[1].set_title('Condition 3 (Intracranial Hemorrhage)', fontsize = 17)
# axes[1].set_xlabel('Frequency', fontsize = 17)
# axes[1].set_ylabel('Visit Type', fontsize = 17)
# axes[1].tick_params(labelsize=15)
# axes[1].set(xlim=(0, 30))

# # # Condition 4
# # sns.barplot(ax=axes[1, 1], data = vis_type_ct4, x='count', y='concept_name',
# # axes[1, 1].set_title('Condition 4 (Intracranial Bleed Min)', fontsize = 17)
# # axes[1, 1].set_xlabel('Frequency', fontsize = 17)
# # axes[1, 1].set_ylabel('Visit Type', fontsize = 17)
# # axes[1, 1].tick_params(labelsize=15)
# # axes[1, 1].set_xticks(range(0, 11, 1))
# # axes[1, 1].set(xlim=(0, 10))
# # plt.show()
```

```
# # Condition 5 plot is not included as visit type is all procedure due to the e
# # unnecessary plot.
```

In [191…

```
# # # wrangle data to suit a join
# # vis_type_jn1 = vis_type_ct1.copy()
# vis_type_jn2 = vis_type_ct2.copy()
# vis_type_jn3 = vis_type_ct3.copy()
# # vis_type_jn4 = vis_type_ct4.copy()


# # vis_type_jn1.columns = ['concept_name', 'cond_1_count']
# vis_type_jn2.columns = ['concept_name', 'cond_2_count']
# vis_type_jn3.columns = ['concept_name', 'cond_3_count']
# # vis_type_jn4.columns = ['concept_name', 'cond_4_count']

# # join all 4 conditions by visit type
# vis_type_cts_all = vis_type_jn2.join(vis_type_jn3.set_index('concept_name'), o

# # convert all columns to integers
# vis_type_cts_all = vis_type_cts_all.fillna(0)
# vis_type_cts_all.cond_3_count[
#     :vis_type_cts_all.shape[0]] = vis_type_cts_all.cond_3_count[:vis_type_cts_
# # vis_type_cts_all.cond_4_count[
# #     :vis_type_cts_all.shape[0]] = vis_type_cts_all.cond_4_count[:vis_type_ct
# vis_type_cts_all.columns = ['Visit Type', 'GI Hemorrhage Count', 'ICH Count']
```

In [193…

```
# vis_type_cts_all
```

Two Subplots of Comparisons of Visit Type Frequencies

Create combined dataframes

In [195…

```
# vis_type_ct1['Condition'] = [1]*vis_type_ct1.shape[0]
# vis_type_ct2['Condition'] = [2]*vis_type_ct2.shape[0]
# vis_type_ct3['Condition'] = [3]*vis_type_ct3.shape[0]
# vis_type_ct4['Condition'] = [4]*vis_type_ct4.shape[0]

# vis_cts12 = pd.concat([vis_type_ct1, vis_type_ct2])
# vis_cts34 = pd.concat([vis_type_ct3, vis_type_ct4])
```

In [197…

```
# vis_cts34
```

In [199…

```
# fig, axes = plt.subplots(1, 2, figsize=(10,5), sharey = True)
# fig.suptitle('Compare Visit Type frequencies between Conditions', fontsize = 1
# sns.set_palette("Paired")

# # Condition 1 vs 2
# sns.barplot(ax = axes[0], data = vis_cts12, x='count', y='concept_name', hue =
# axes[0].set_xlabel('Frequency', fontsize = 15)
# axes[0].set_ylabel('Visit Type', fontsize = 15)
# axes[0].set_title('Bleeding: Condition 1 vs. 2', fontsize = 15)
# axes[0].tick_params(labelsize = 13)
# axes[0].legend(fontsize = 'xx-large', loc = 4)
```

```
# # Condition 3 vs 4
# sns.barplot(ax = axes[1], data = vis_cts34, x='count', y='concept_name', hue =
# axes[1].set_title('ICH: Condition 3 vs. 4', fontsize = 15)
# axes[1].set_xlabel('Frequency', fontsize = 15)
# axes[1].set_ylabel('Visit Type', fontsize = 15)
# axes[1].tick_params(labelsize = 13)
# # axes[1].set(xlabel = 'Frequency', ylabel = 'Visit Type', title = 'ICH: Condi
# axes[1].legend(loc=4, fontsize = 'xx-large')
# plt.show()
```

Cohort Overlap Visualization (Percent)

All Visit Types

In [201…

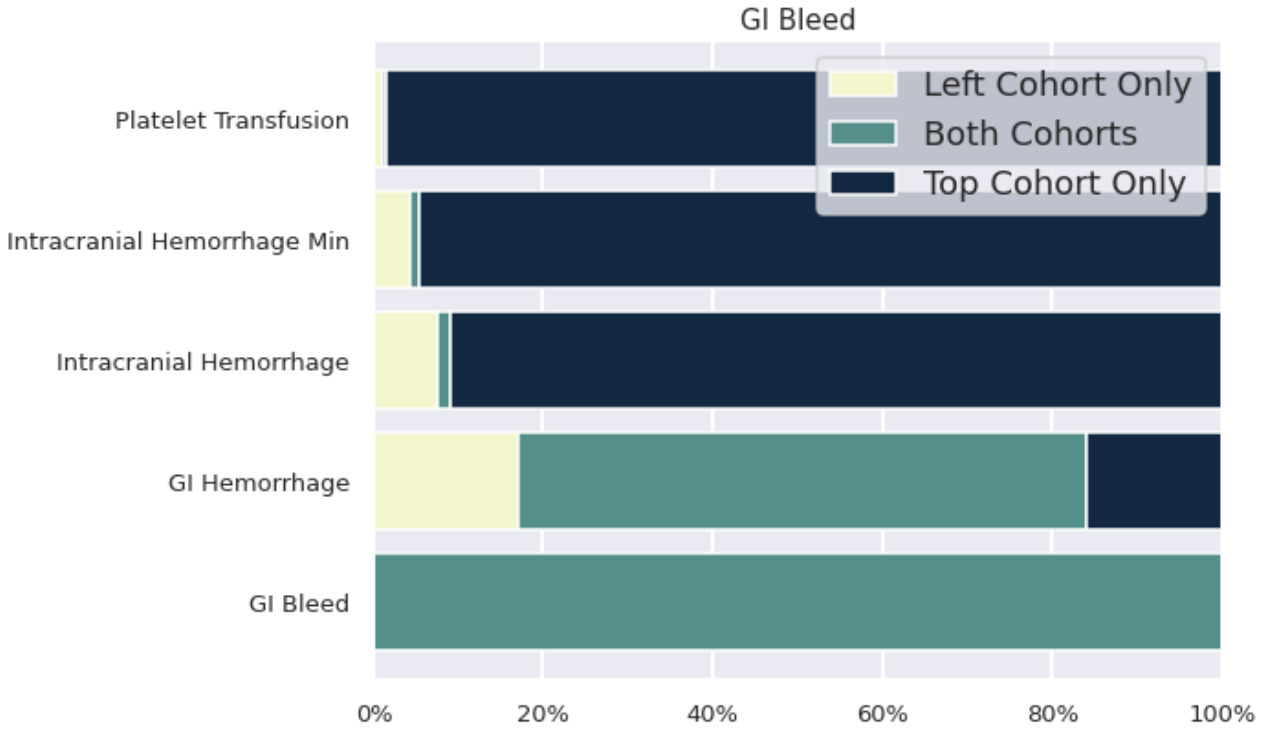
```
temp1 = overlap_df1.reset_index()
temp2 = overlap_df2.reset_index()
temp3 = overlap_df3.reset_index()
temp4 = overlap_df4.reset_index()
temp5 = overlap_df5.reset_index()

temp1_er = overlap_df1_er.reset_index()
temp2_er = overlap_df2_er.reset_index()
temp3_er = overlap_df3_er.reset_index()
temp4_er = overlap_df4_er.reset_index()
temp5_er = overlap_df5_er.reset_index()

cond = ['GI Bleed', 'GI Hemorrhage', 'Intracranial Hemorrhage', 'Intracranial He
temps = [temp1, temp2, temp3, temp4, temp5]
temps_er = [temp1_er, temp2_er, temp3_er, temp4_er, temp5_er]
```

In [203…

```
# Condition 1
fig, axes = plt.subplots(1,1, figsize = (10,6))

for i in range(1):
    top_list, left_list, both_list = [], [], []
    for j in range(len(cond)):
        x_y = list(temps[i]['person_id'].isin(temps[j]['person_id']).value_count
        y_x = list(temps[j]['person_id'].isin(temps[i]['person_id']).value_count

        total = sum(set(x_y + y_x))
        if len(set(x_y).intersection(y_x)) != 0:
            both = set(x_y).intersection(y_x).pop()
            x_y.remove(both)
            y_x.remove(both)
        else:
            both = 0
        both = round(both/total*100,2)
        top = 0.00 if len(x_y) == 0 else round(x_y[0]/total*100,2)
        left = 0.00 if len(y_x) == 0 else round(y_x[0]/total*100,2)

        top_list.append(top)
        left_list.append(left)
        both_list.append(both)

    left_both = np.add(left_list, both_list).tolist()

    # Left Cohort
```

```
    axes.barh(cond, left_list, color='#F3F6CC')

    # Both Cohort
    axes.barh(cond, both_list, left=left_list, color='#558F8B')

    # Top Co
    axes.barh(cond, top_list, left=left_both, color='#122740')

    axes.grid(True, axis='x', linewidth=2, color='white')
    axes.set_xticklabels(['0%', '20%', '40%', '60%', '80%', '100%'])
    axes.tick_params('y', labelsize= 13)
    axes.set_title(cond[i], fontsize= 15)
    axes.tick_params('x', labelsize = 13)
    axes.legend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc = '
    #plt.figlegend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc

#fig.suptitle('Cohort Overlap', fontsize=30, verticalalignment='top')
fig.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:37: UserWarning: Fi
xedFormatter should only be used together with FixedLocator
```

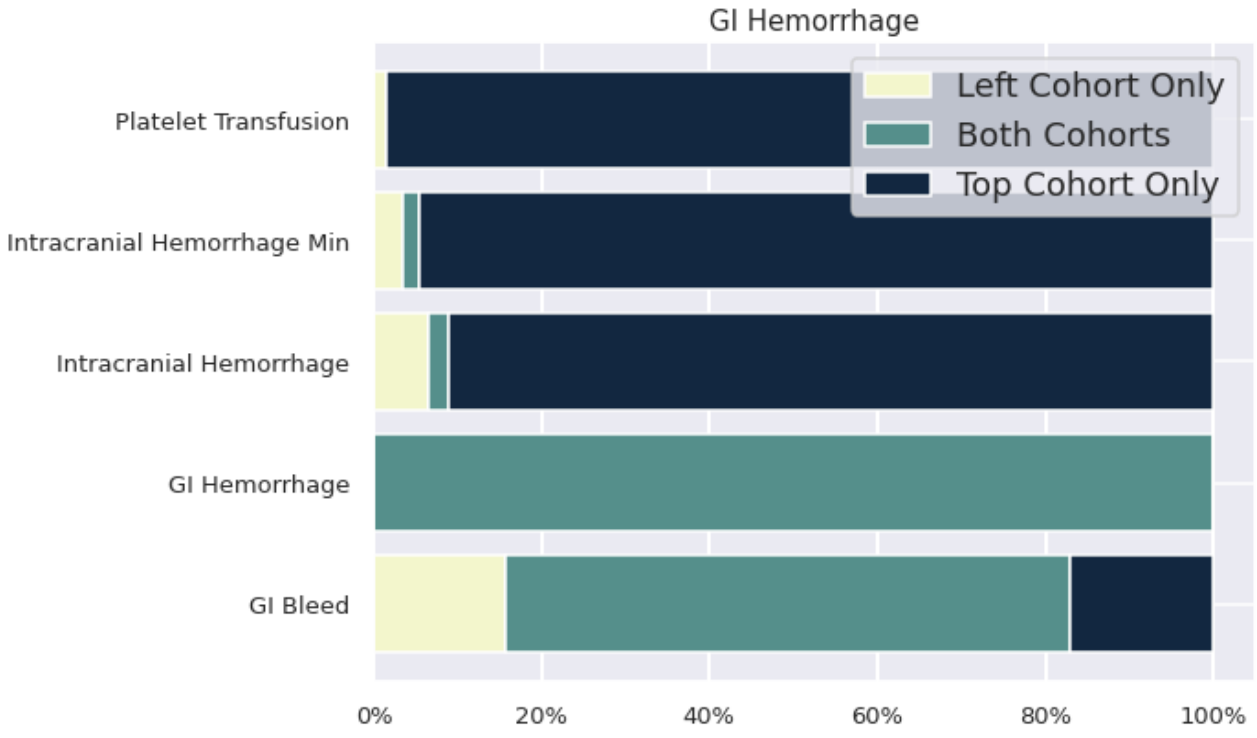


In [205…

```
# Condition 2
fig, axes = plt.subplots(1,1, figsize = (10,6))

for i in range(1,2):
    top_list, left_list, both_list = [], [], []
    for j in range(len(cond)):
        x_y = list(temps[i]['person_id'].isin(temps[j]['person_id']).value_count
        y_x = list(temps[j]['person_id'].isin(temps[i]['person_id']).value_count

        total = sum(set(x_y + y_x))
        if len(set(x_y).intersection(y_x)) != 0:
            both = set(x_y).intersection(y_x).pop()
            x_y.remove(both)
```

```
            y_x.remove(both)
        else:
            both = 0
        both = round(both/total*100,2)
        top = 0.00 if len(x_y) == 0 else round(x_y[0]/total*100,2)
        left = 0.00 if len(y_x) == 0 else round(y_x[0]/total*100,2)

        top_list.append(top)
        left_list.append(left)
        both_list.append(both)

    left_both = np.add(left_list, both_list).tolist()

    # Left Cohort
    axes.barh(cond, left_list, color='#F3F6CC')

    # Both Cohort
    axes.barh(cond, both_list, left=left_list, color='#558F8B')

    # Top Co
    axes.barh(cond, top_list, left=left_both, color='#122740')

    axes.grid(True, axis='x', linewidth=2, color='white')
    axes.set_xticklabels(['0%', '20%', '40%', '60%', '80%', '100%'])
    axes.tick_params('y', labelsize=13)
    axes.set_title(cond[i], fontsize=15)
    axes.tick_params('x', labelsize = 13)
    axes.legend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc = '
    #plt.figlegend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc

#fig.suptitle('Cohort Overlap', fontsize=30, verticalalignment='top')
fig.tight_layout()
plt.show()
```

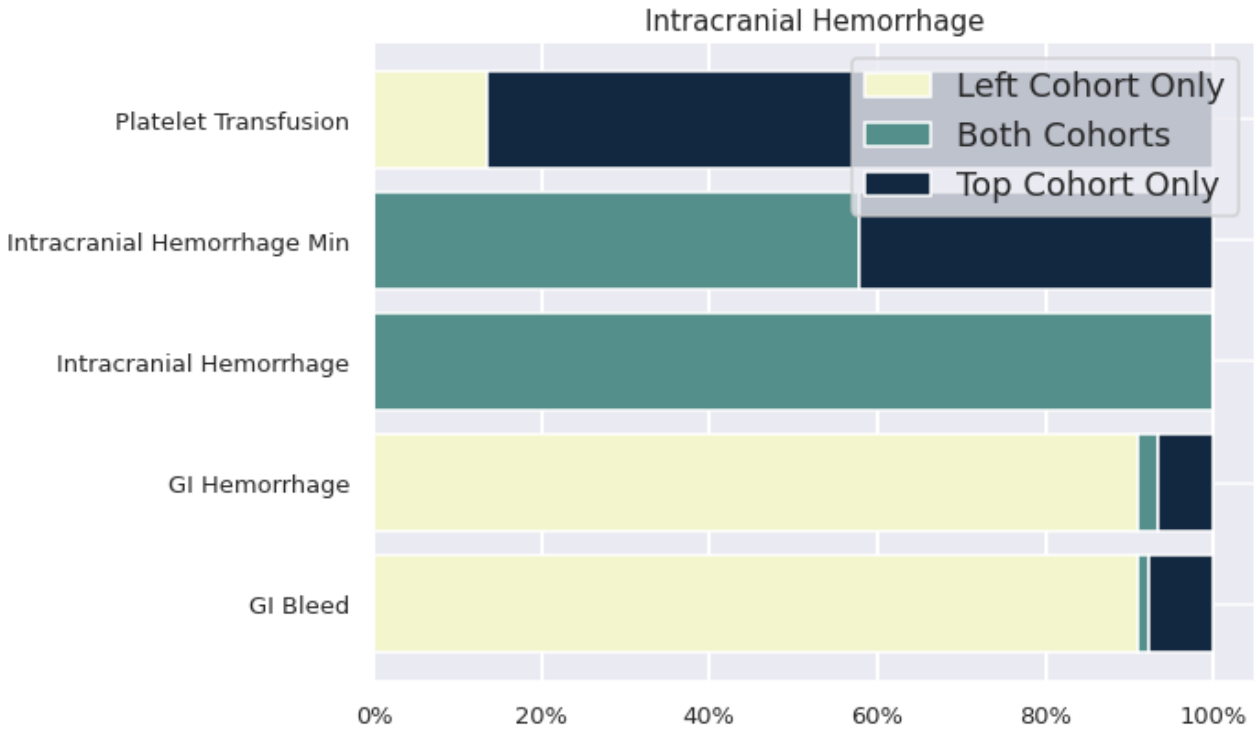
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:37: UserWarning: Fi
xedFormatter should only be used together with FixedLocator
```



In [207…

```
# Condition 3
fig, axes = plt.subplots(1,1, figsize = (10,6))

for i in range(2,3):
    top_list, left_list, both_list = [], [], []
    for j in range(len(cond)):
        x_y = list(temps[i]['person_id'].isin(temps[j]['person_id']).value_count
        y_x = list(temps[j]['person_id'].isin(temps[i]['person_id']).value_count

        total = sum(set(x_y + y_x))
        if len(set(x_y).intersection(y_x)) != 0:
            both = set(x_y).intersection(y_x).pop()
            x_y.remove(both)
            y_x.remove(both)
        else:
            both = 0
        both = round(both/total*100,2)
        top = 0.00 if len(x_y) == 0 else round(x_y[0]/total*100,2)
        left = 0.00 if len(y_x) == 0 else round(y_x[0]/total*100,2)

        top_list.append(top)
        left_list.append(left)
        both_list.append(both)

    left_both = np.add(left_list, both_list).tolist()

    # Left Cohort
    axes.barh(cond, left_list, color='#F3F6CC')

    # Both Cohort
    axes.barh(cond, both_list, left=left_list, color='#558F8B')

    # Top Co
    axes.barh(cond, top_list, left=left_both, color='#122740')

    axes.grid(True, axis='x', linewidth=2, color='white')
    axes.set_xticklabels(['0%', '20%', '40%', '60%', '80%', '100%'])
    axes.tick_params('y', labelsize=13)
    axes.set_title(cond[i], fontsize=15)
    axes.tick_params('x', labelsize = 13)
    axes.legend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc = '
    #plt.figlegend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc

#fig.suptitle('Cohort Overlap', fontsize=30, verticalalignment='top')
fig.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:37: UserWarning: Fi
xedFormatter should only be used together with FixedLocator
```

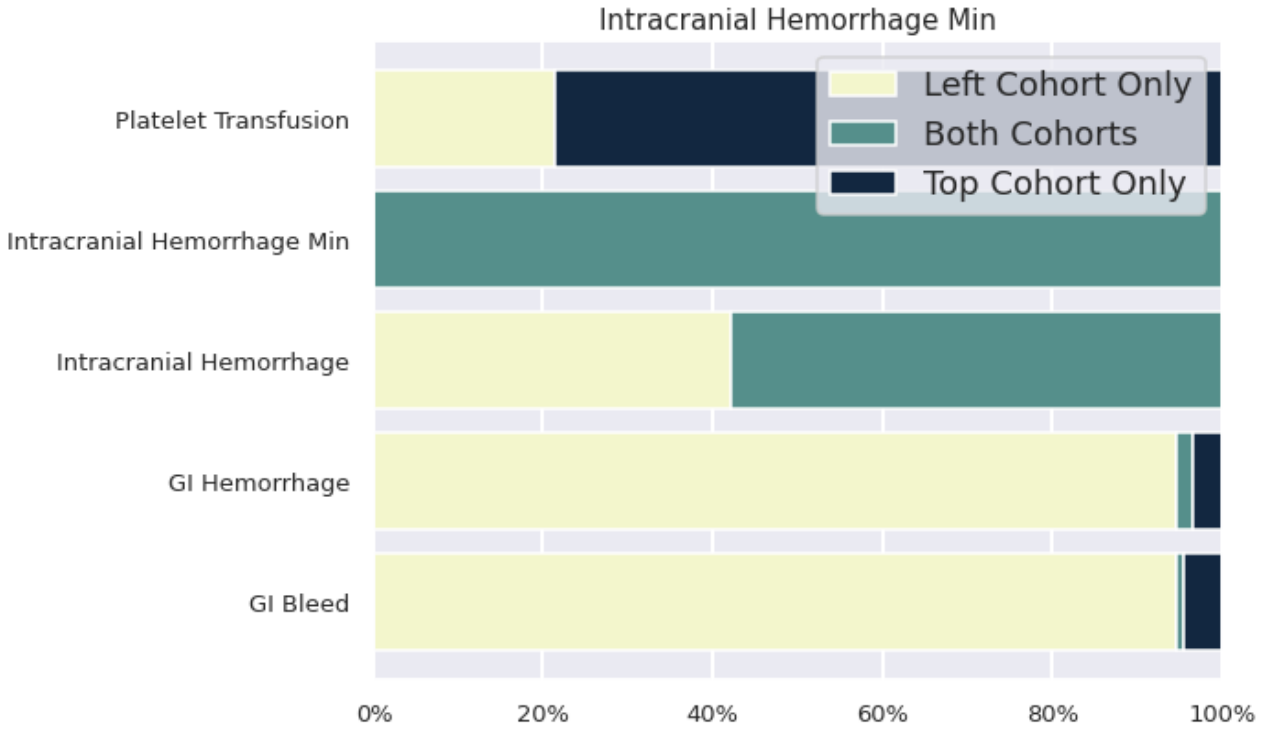


In [209…

```
# Condition 4
fig, axes = plt.subplots(1,1, figsize = (10,6))

for i in range(3,4):
    top_list, left_list, both_list = [], [], []
    for j in range(len(cond)):
        x_y = list(temps[i]['person_id'].isin(temps[j]['person_id']).value_count
        y_x = list(temps[j]['person_id'].isin(temps[i]['person_id']).value_count

        total = sum(set(x_y + y_x))
        if len(set(x_y).intersection(y_x)) != 0:
            both = set(x_y).intersection(y_x).pop()
            x_y.remove(both)
            y_x.remove(both)
        else:
            both = 0
        both = round(both/total*100,2)
        top = 0.00 if len(x_y) == 0 else round(x_y[0]/total*100,2)
        left = 0.00 if len(y_x) == 0 else round(y_x[0]/total*100,2)

        top_list.append(top)
        left_list.append(left)
        both_list.append(both)

    left_both = np.add(left_list, both_list).tolist()

    # Left Cohort
    axes.barh(cond, left_list, color='#F3F6CC')

    # Both Cohort
    axes.barh(cond, both_list, left=left_list, color='#558F8B')

    # Top Co
    axes.barh(cond, top_list, left=left_both, color='#122740')

    axes.grid(True, axis='x', linewidth=2, color='white')
```

```
    axes.set_xticklabels(['0%', '20%', '40%', '60%', '80%', '100%'])
    axes.tick_params('y', labelsize=13)
    axes.set_title(cond[i], fontsize=15)
    axes.tick_params('x', labelsize = 13)
    axes.legend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc = '
    #plt.figlegend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc

#fig.suptitle('Cohort Overlap', fontsize=30, verticalalignment='top')
fig.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:37: UserWarning: Fi
xedFormatter should only be used together with FixedLocator
```

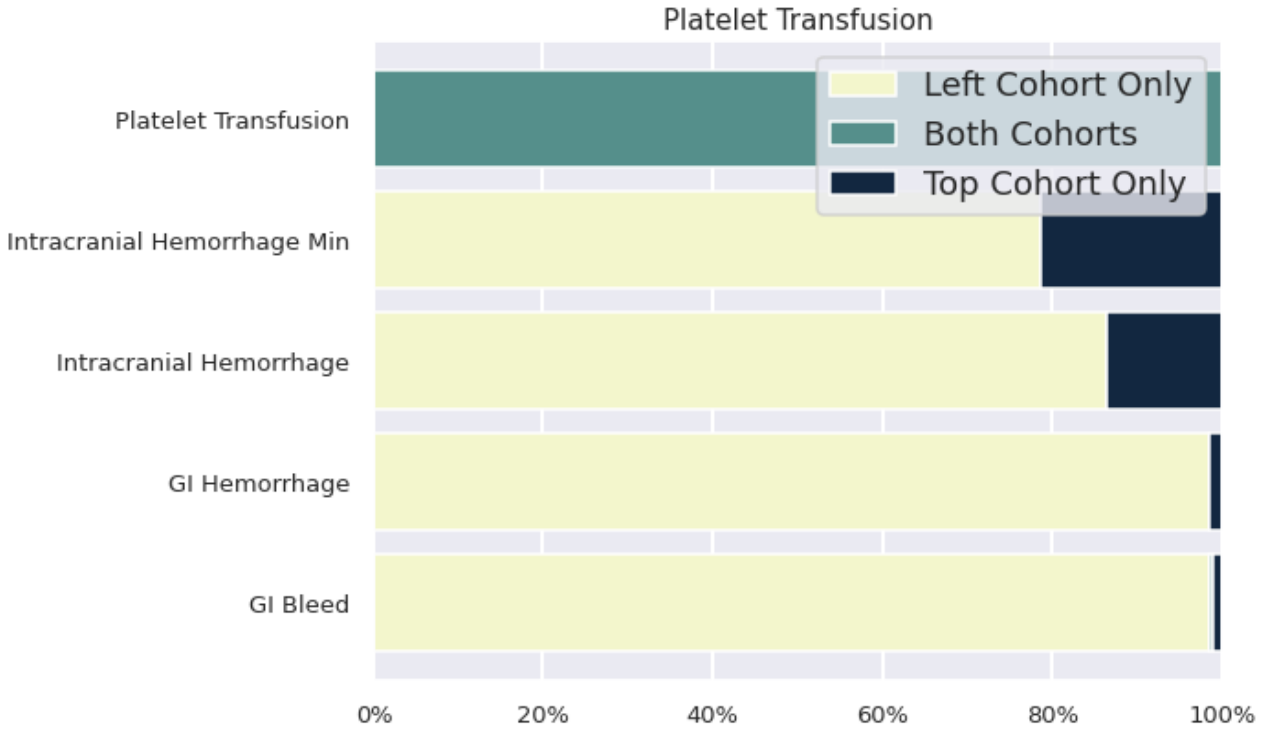


In [211…

```
# Condition 5
fig, axes = plt.subplots(1,1, figsize = (10,6))

for i in range(4,5):
    top_list, left_list, both_list = [], [], []
    for j in range(len(cond)):
        x_y = list(temps[i]['person_id'].isin(temps[j]['person_id']).value_count
        y_x = list(temps[j]['person_id'].isin(temps[i]['person_id']).value_count

        total = sum(set(x_y + y_x))
        if len(set(x_y).intersection(y_x)) != 0:
            both = set(x_y).intersection(y_x).pop()
            x_y.remove(both)
            y_x.remove(both)
        else:
            both = 0
        both = round(both/total*100,2)
        top = 0.00 if len(x_y) == 0 else round(x_y[0]/total*100,2)
        left = 0.00 if len(y_x) == 0 else round(y_x[0]/total*100,2)

        top_list.append(top)
        left_list.append(left)
```

```
        both_list.append(both)

    left_both = np.add(left_list, both_list).tolist()

    # Left Cohort
    axes.barh(cond, left_list, color='#F3F6CC')

    # Both Cohort
    axes.barh(cond, both_list, left=left_list, color='#558F8B')

    # Top Co
    axes.barh(cond, top_list, left=left_both, color='#122740')

    axes.grid(True, axis='x', linewidth=2, color='white')
    axes.set_xticklabels(['0%', '20%', '40%', '60%', '80%', '100%'])
    axes.tick_params('y', labelsize=13)
    axes.set_title(cond[i], fontsize=15)
    axes.tick_params('x', labelsize = 13)
    axes.legend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc = '
    #plt.figlegend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc

#fig.suptitle('Cohort Overlap', fontsize=30, verticalalignment='top')
fig.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:37: UserWarning: Fi
xedFormatter should only be used together with FixedLocator
```



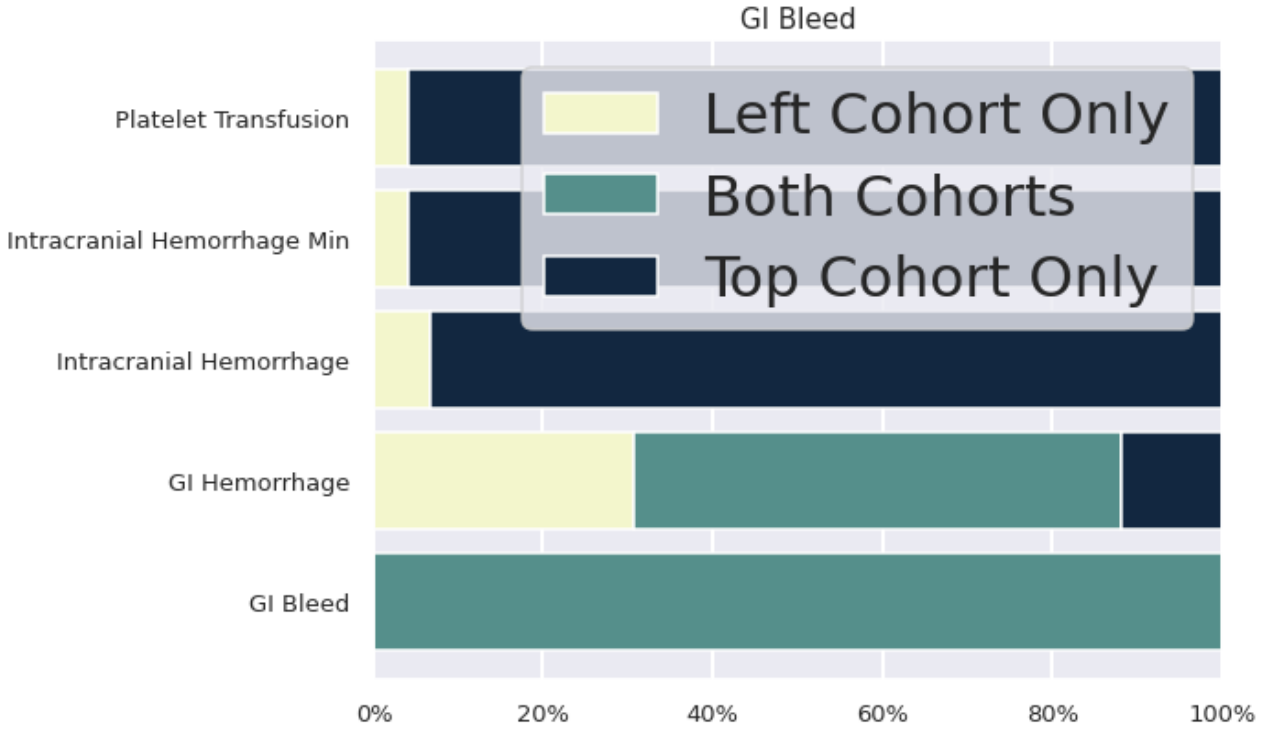
Only ER and Inpatient Visit Types

In [213…

```
# Condition 1
fig, axes = plt.subplots(1,1, figsize = (10,6))

for i in range(1):
    top_list, left_list, both_list = [], [], []
    for j in range(len(cond)):
```

```
            x_y = list(temps_er[i]['person_id'].isin(temps_er[j]['person_id']).value
            y_x = list(temps_er[j]['person_id'].isin(temps_er[i]['person_id']).value

            if len(set(x_y).intersection(y_x)) != 0:
                both = set(x_y).intersection(y_x).pop()
                x_y.remove(both)
                y_x.remove(both)
            else:
                both = 0
            top = 0.00 if len(x_y) == 0 else x_y[0]
            left = 0.00 if len(y_x) == 0 else y_x[0]
            total = both + left + top

            both = round(both/total*100,2)
            top = round(top/total*100,2)
            left = round(left/total*100,2)

            top_list.append(top)
            left_list.append(left)
            both_list.append(both)

        left_both = np.add(left_list, both_list).tolist()

        # Left Cohort
        axes.barh(cond, left_list, color='#F3F6CC')

        # Both Cohort
        axes.barh(cond, both_list, left=left_list, color='#558F8B')

        # Top Co
        axes.barh(cond, top_list, left=left_both, color='#122740')

        axes.grid(True, axis='x', linewidth=2, color='white')
        axes.set_xticklabels(['0%', '20%', '40%', '60%', '80%', '100%'])
        axes.tick_params('y', labelsize=13)
        axes.set_title(cond[i], fontsize=15)
        axes.tick_params('x', labelsize = 13)
        axes.legend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc = '

        #fig.suptitle('Cohort Overlap (Inpatient/ER Only)', fontsize=25)

    fig.tight_layout()
    plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:40: UserWarning: Fi
xedFormatter should only be used together with FixedLocator
```

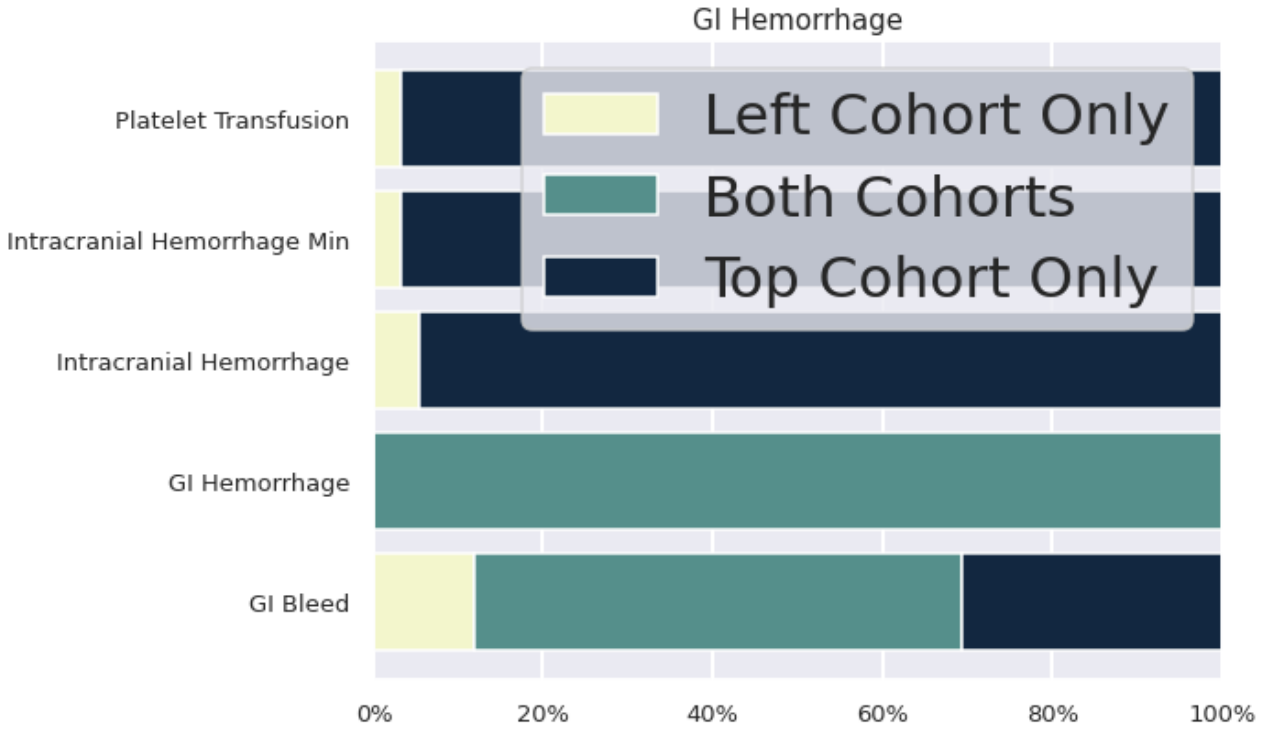


In [215…

```
# Condition 2
fig, axes = plt.subplots(1,1, figsize = (10,6))

for i in range(1,2):
    top_list, left_list, both_list = [], [], []
    for j in range(len(cond)):
        x_y = list(temps_er[i]['person_id'].isin(temps_er[j]['person_id']).value
        y_x = list(temps_er[j]['person_id'].isin(temps_er[i]['person_id']).value

        if len(set(x_y).intersection(y_x)) != 0:
            both = set(x_y).intersection(y_x).pop()
            x_y.remove(both)
            y_x.remove(both)
        else:
            both = 0
        top = 0.00 if len(x_y) == 0 else x_y[0]
        left = 0.00 if len(y_x) == 0 else y_x[0]
        total = both + left + top

        both = round(both/total*100,2)
        top = round(top/total*100,2)
        left = round(left/total*100,2)

        top_list.append(top)
        left_list.append(left)
        both_list.append(both)

    left_both = np.add(left_list, both_list).tolist()

    # Left Cohort
    axes.barh(cond, left_list, color='#F3F6CC')

    # Both Cohort
    axes.barh(cond, both_list, left=left_list, color='#558F8B')

    # Top Co
```

```
    axes.barh(cond, top_list, left=left_both, color='#122740')

    axes.grid(True, axis='x', linewidth=2, color='white')
    axes.set_xticklabels(['0%', '20%', '40%', '60%', '80%', '100%'])
    axes.tick_params('y', labelsize=13)
    axes.set_title(cond[i], fontsize=15)
    axes.tick_params('x', labelsize = 13)
    axes.legend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc = '

    #fig.suptitle('Cohort Overlap (Inpatient/ER Only)', fontsize=25)

fig.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:40: UserWarning: Fi
xedFormatter should only be used together with FixedLocator
```

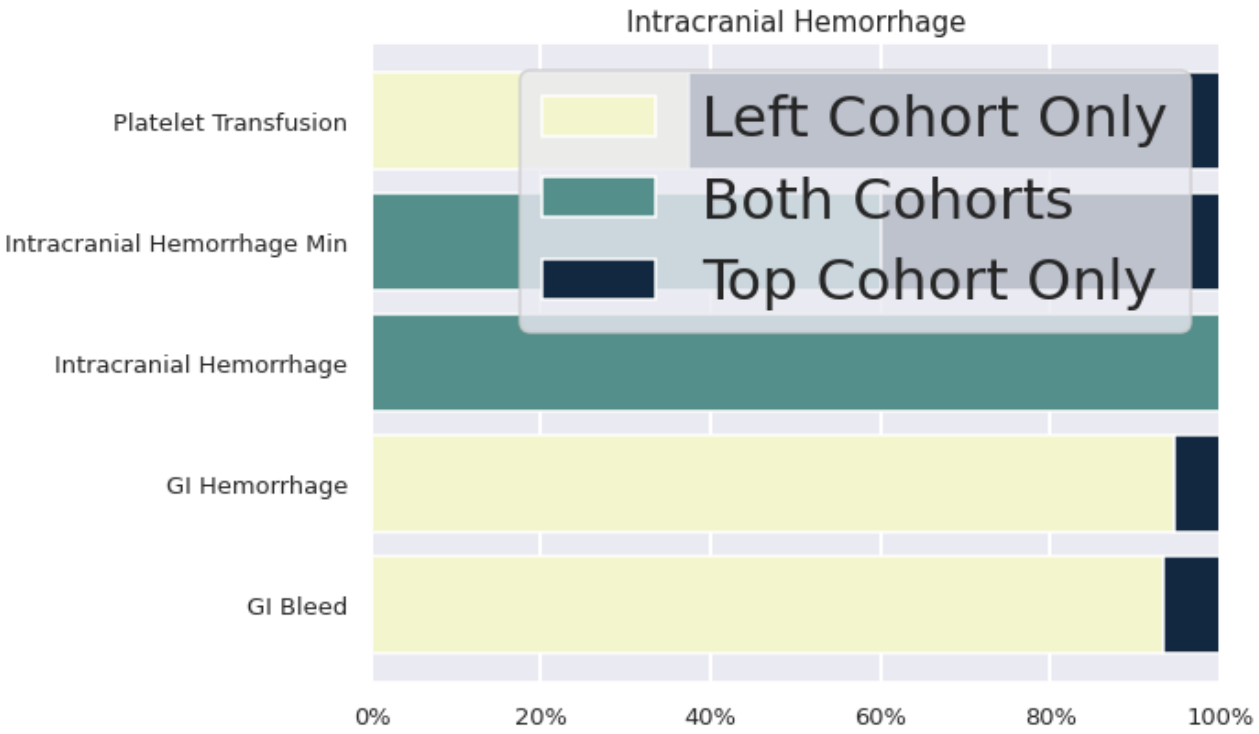


In [217…

```
# Condition 3
fig, axes = plt.subplots(1,1, figsize = (10,6))

for i in range(2,3):
    top_list, left_list, both_list = [], [], []
    for j in range(len(cond)):
        x_y = list(temps_er[i]['person_id'].isin(temps_er[j]['person_id']).value
        y_x = list(temps_er[j]['person_id'].isin(temps_er[i]['person_id']).value

        if len(set(x_y).intersection(y_x)) != 0:
            both = set(x_y).intersection(y_x).pop()
            x_y.remove(both)
            y_x.remove(both)
        else:
            both = 0
        top = 0.00 if len(x_y) == 0 else x_y[0]
        left = 0.00 if len(y_x) == 0 else y_x[0]
        total = both + left + top
```

```
        both = round(both/total*100,2)
        top = round(top/total*100,2)
        left = round(left/total*100,2)

        top_list.append(top)
        left_list.append(left)
        both_list.append(both)

    left_both = np.add(left_list, both_list).tolist()

    # Left Cohort
    axes.barh(cond, left_list, color='#F3F6CC')

    # Both Cohort
    axes.barh(cond, both_list, left=left_list, color='#558F8B')

    # Top Co
    axes.barh(cond, top_list, left=left_both, color='#122740')

    axes.grid(True, axis='x', linewidth=2, color='white')
    axes.set_xticklabels(['0%', '20%', '40%', '60%', '80%', '100%'])
    axes.tick_params('y', labelsize=13)
    axes.set_title(cond[i], fontsize=15)
    axes.tick_params('x', labelsize = 13)
    axes.legend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc = '

    #fig.suptitle('Cohort Overlap (Inpatient/ER Only)', fontsize=25)

fig.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:40: UserWarning: Fi
xedFormatter should only be used together with FixedLocator
```

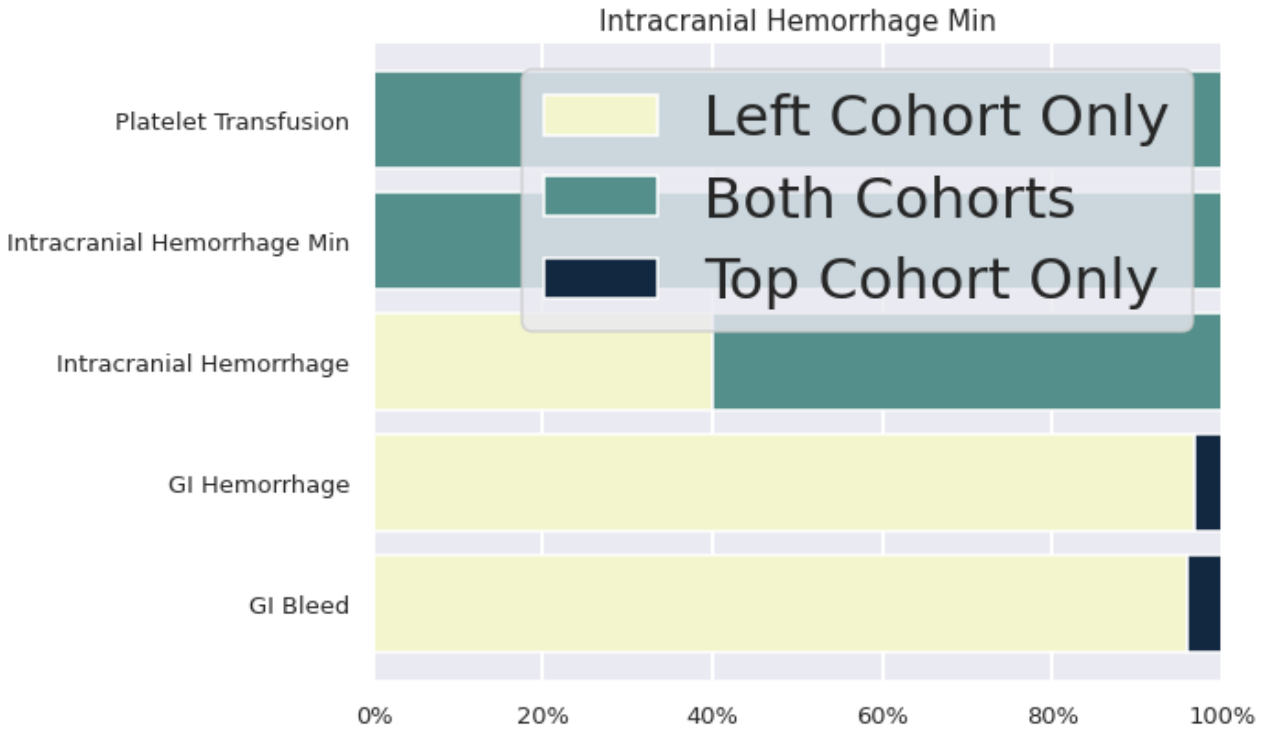


In [219…

```
# Condition 4
fig, axes = plt.subplots(1,1, figsize = (10,6))
```

```
for i in range(3,4):
    top_list, left_list, both_list = [], [], []
    for j in range(len(cond)):
        x_y = list(temps_er[i]['person_id'].isin(temps_er[j]['person_id']).value
        y_x = list(temps_er[j]['person_id'].isin(temps_er[i]['person_id']).value

        if len(set(x_y).intersection(y_x)) != 0:
            both = set(x_y).intersection(y_x).pop()
            x_y.remove(both)
            y_x.remove(both)
        else:
            both = 0
        top = 0.00 if len(x_y) == 0 else x_y[0]
        left = 0.00 if len(y_x) == 0 else y_x[0]
        total = both + left + top

        both = round(both/total*100,2)
        top = round(top/total*100,2)
        left = round(left/total*100,2)

        top_list.append(top)
        left_list.append(left)
        both_list.append(both)

    left_both = np.add(left_list, both_list).tolist()

    # Left Cohort
    axes.barh(cond, left_list, color='#F3F6CC')

    # Both Cohort
    axes.barh(cond, both_list, left=left_list, color='#558F8B')

    # Top Co
    axes.barh(cond, top_list, left=left_both, color='#122740')

    axes.grid(True, axis='x', linewidth=2, color='white')
    axes.set_xticklabels(['0%', '20%', '40%', '60%', '80%', '100%'])
    axes.tick_params('y', labelsize=13)
    axes.set_title(cond[i], fontsize=15)
    axes.tick_params('x', labelsize = 13)
    axes.legend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc = '

    #fig.suptitle('Cohort Overlap (Inpatient/ER Only)', fontsize=25)

fig.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:40: UserWarning: Fi
xedFormatter should only be used together with FixedLocator
```

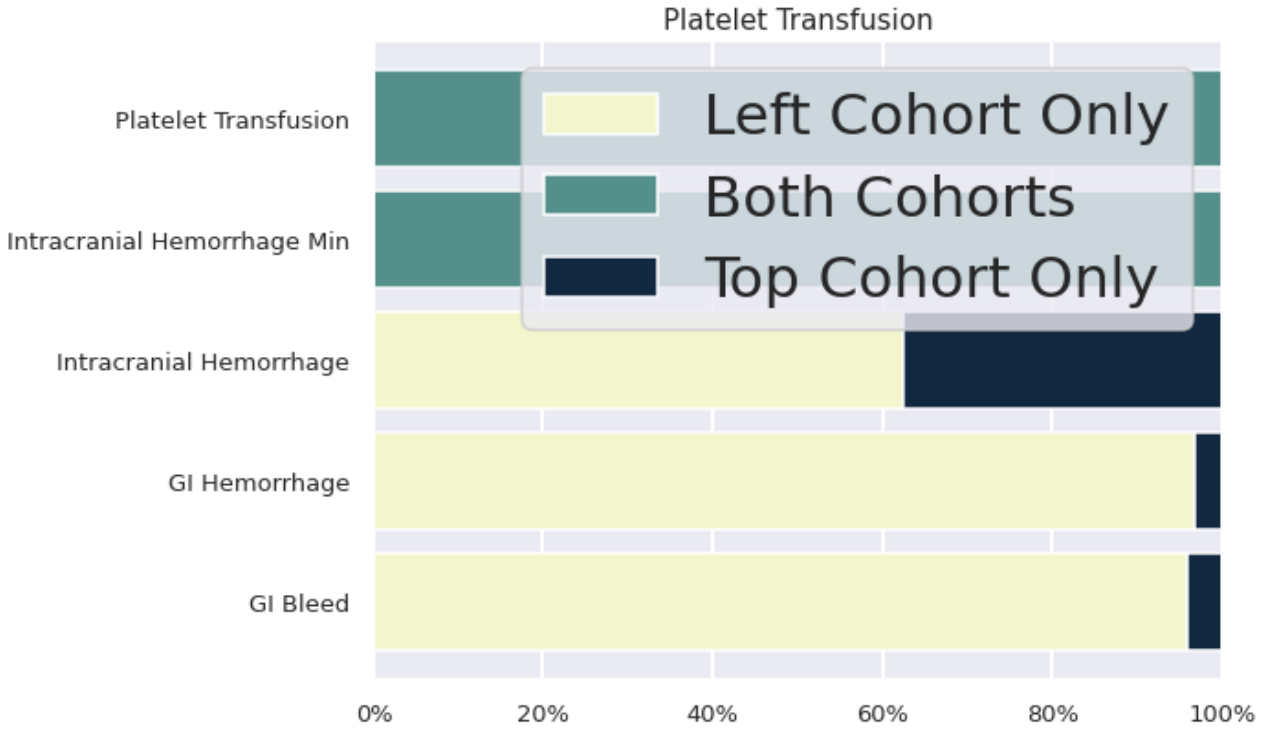


In [221…

```
# Condition 5
fig, axes = plt.subplots(1,1, figsize = (10,6))

for i in range(4,5):
    top_list, left_list, both_list = [], [], []
    for j in range(len(cond)):
        x_y = list(temps_er[i]['person_id'].isin(temps_er[j]['person_id']).value
        y_x = list(temps_er[j]['person_id'].isin(temps_er[i]['person_id']).value

        if len(set(x_y).intersection(y_x)) != 0:
            both = set(x_y).intersection(y_x).pop()
            x_y.remove(both)
            y_x.remove(both)
        else:
            both = 0
        top = 0.00 if len(x_y) == 0 else x_y[0]
        left = 0.00 if len(y_x) == 0 else y_x[0]
        total = both + left + top

        both = round(both/total*100,2)
        top = round(top/total*100,2)
        left = round(left/total*100,2)

        top_list.append(top)
        left_list.append(left)
        both_list.append(both)

    left_both = np.add(left_list, both_list).tolist()

    # Left Cohort
    axes.barh(cond, left_list, color='#F3F6CC')

    # Both Cohort
    axes.barh(cond, both_list, left=left_list, color='#558F8B')

    # Top Co
```

```
    axes.barh(cond, top_list, left=left_both, color='#122740')

    axes.grid(True, axis='x', linewidth=2, color='white')
    axes.set_xticklabels(['0%', '20%', '40%', '60%', '80%', '100%'])
    axes.tick_params('y', labelsize=13)
    axes.set_title(cond[i], fontsize=15)
    axes.tick_params('x', labelsize = 13)
    axes.legend(['Left Cohort Only', 'Both Cohorts', 'Top Cohort Only'], loc = '

    #fig.suptitle('Cohort Overlap (Inpatient/ER Only)', fontsize=25)

fig.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:40: UserWarning: Fi
xedFormatter should only be used together with FixedLocator
```



In [223…

```
# l = []
# for i in range(len(cond)):

#     for j in range(len(cond)):
#         x_y = list(temps[i]['person_id'].isin(temps[j]['person_id']).value_cou
#         y_x = list(temps[j]['person_id'].isin(temps[i]['person_id']).value_cou

#         total = sum(set(x_y + y_x))
#         if len(set(x_y).intersection(y_x)) != 0:
#             both = set(x_y).intersection(y_x).pop()
#             x_y.remove(both)
#             y_x.remove(both)
#         else:
#             both = 0
#         both_count = both
#         top_count = 0 if len(x_y) == 0 else x_y[0]
#         left_count = 0 if len(y_x) == 0 else y_x[0]
#         both = round(both/total*100,2)
#         top = 0.00 if len(x_y) == 0 else round(x_y[0]/total*100,2)
```

```
#         left = 0.00 if len(y_x) == 0 else round(y_x[0]/total*100,2)

#         l.append({'Top Cohort': cond[i], 'Left Cohort': cond[j], 'count': top_
#         l.append({'Top Cohort': cond[i], 'Left Cohort': cond[j], 'count': both
#         l.append({'Top Cohort': cond[i], 'Left Cohort': cond[j], 'count': left
```

In [225…

```
# cohort_overlap = pd.DataFrame(l)
# create_download_link(cohort_overlap, "Download cohort_overlap CSV", "cohort_ov
```

In [227…

```
# l_er = []
# for i in range(len(cond)):

#     for j in range(len(cond)):
#         x_y = list(temps_er[i]['person_id'].isin(temps_er[j]['person_id']).val
#         y_x = list(temps_er[j]['person_id'].isin(temps_er[i]['person_id']).val

#         if len(set(x_y).intersection(y_x)) != 0:
#             both = set(x_y).intersection(y_x).pop()
#             x_y.remove(both)
#             y_x.remove(both)
#         else:
#             both = 0

#         both_count = both
#         top_count = 0.00 if len(x_y) == 0 else x_y[0]
#         left_count = 0.00 if len(y_x) == 0 else y_x[0]
#         total = both + left + top

#         both = round(both/total*100,2)
#         top = round(top_count/total*100,2)
#         left = round(left_count/total*100,2)

#         l_er.append({'Top Cohort': cond[i], 'Left Cohort': cond[j], 'count': t
#         l_er.append({'Top Cohort': cond[i], 'Left Cohort': cond[j], 'count': b
#         l_er.append({'Top Cohort': cond[i], 'Left Cohort': cond[j], 'count': l
```

In [229…

```
# cohort_overlap_er = pd.DataFrame(l_er)
# create_download_link(cohort_overlap_er, "Download cohort_overlap_er CSV", "coh
```

Multidimensional Plots

Race, Sex, Age and Dispension Count Distribution by ADE Classification

In [231…

```
ade2_min = overlap_df2.reset_index()[overlap_df2.reset_index().columns[:39]]
clopidogrel_3['ADE_Classification'] = pd.Series(['No']*1907)
ade2_min['ADE_Classification'] = pd.Series(['Yes']*overlap_df2.shape[0])
```

In [233…

```
clop_ade = pd.concat([clopidogrel_3, ade2_min], ignore_index=True).groupby(
    'person_id', as_index = False).apply(lambda x: x.sort_values(
    'ADE_Classification', ascending = False).head(1)).reset_index(drop = True)
```

In [ ]:

In [235…
```
race_dic = {'Another single population': 'Other',
            'I prefer not to answer': 'Unknown',
            'No matching concept': 'Unknown',
            'None of these': 'Other',
            'None Indicated': 'Unknown',
            'PMI: Skip' : 'Unknown',
            'More than one population': 'Other'}
clop_ade.race = clop_ade.race.replace(race_dic)
race_grp = clop_ade.groupby(['race', 'ADE_Classification']).count().reset_index(
    'race', 'ADE_Classification', 'person_id']]

race_tot = race_grp.groupby('race').sum('person_id').reset_index()['person_id'].

len(race_tot)
race_dbl = [0]*len(race_tot)*2
for i in range(len(race_dbl)):
    race_dbl[i] = race_tot[i//2]

race_grp['total'] = race_dbl
race_grp['percentage'] = 100*race_grp['person_id']/race_grp['total']
```

In [237…
```
sex_dic = {'Not male, not female, prefer not to answer, or skipped': 'Other',
           'No matching concept': 'Other'}
clop_ade.sex_at_birth = clop_ade.sex_at_birth.replace(sex_dic)
sex_grp = clop_ade.groupby(['sex_at_birth', 'ADE_Classification']).count().reset
    'sex_at_birth', 'ADE_Classification', 'person_id']]
sex_tot = sex_grp.groupby('sex_at_birth').sum('person_id').reset_index()['person

len(sex_tot)
sex_dbl = [0]*len(sex_tot)*2
for i in range(len(sex_dbl)):
    sex_dbl[i] = sex_tot[i//2]

sex_grp['total'] = sex_dbl
sex_grp['percentage'] = 100*sex_grp['person_id']/sex_grp['total']
```

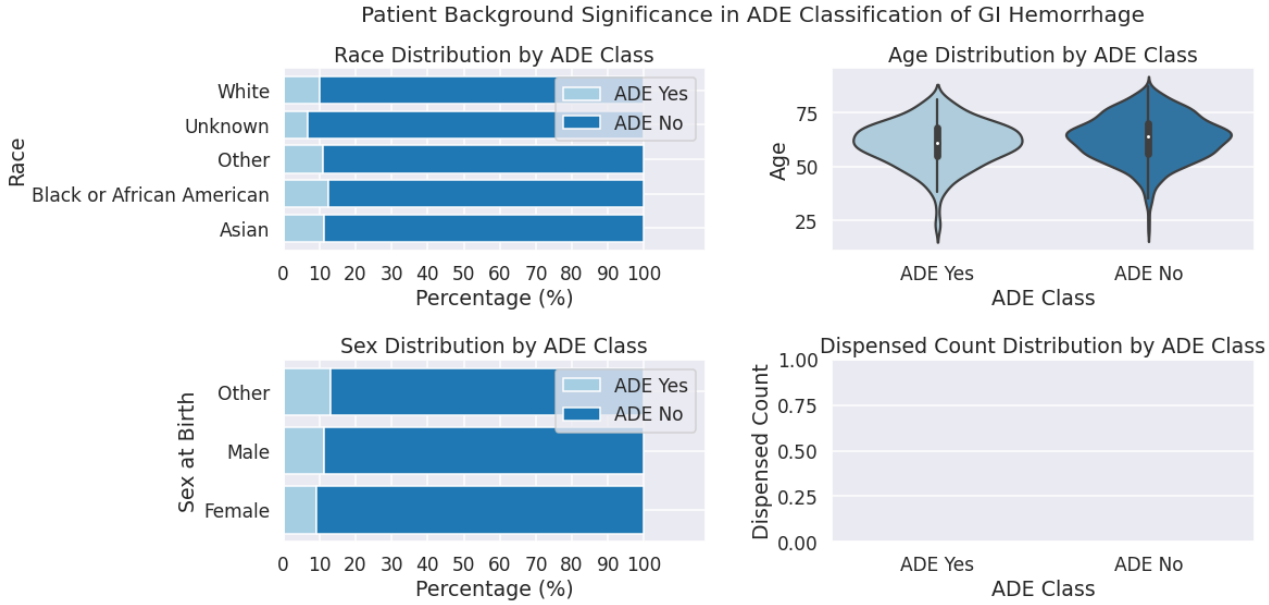
In [239…
```
clop_ade['age'] = pd.DatetimeIndex(clop_ade.index_date).year - clop_ade.year_of_
age_grp = clop_ade[['age', 'ADE_Classification']]
```

In [241…
```
clop_ade.dispensed_count
```

Out[241…
```
0       19
1        6
2       14
3       30
4       10
        ..
1902     2
1903    12
1904     3
1905     2
1906    24
Name: dispensed_count, Length: 1907, dtype: int64
```

In [243…

```
sns.set_palette('Paired')
fig, axes = plt.subplots(2, 2, figsize=(16,8))
fig.suptitle('Patient Background Significance in ADE Classification of GI Hemorr
fig.subplots_adjust(hspace=0.6, wspace = 0.3)

# race
axes[0,0].barh(race_grp['race'].drop_duplicates().tolist(),
               race_grp[race_grp['ADE_Classification'] == 'Yes']['percentage'].t
              label = 'ADE Yes')
axes[0,0].barh(race_grp['race'].drop_duplicates().tolist(),
               race_grp[race_grp['ADE_Classification'] == 'No']['percentage'].to
               left = race_grp[race_grp['ADE_Classification'] == 'Yes']['percent
              label = 'ADE No')
axes[0, 0].set_title('Race Distribution by ADE Class', fontsize = 19)
axes[0, 0].set_xlabel('Percentage (%)', fontsize = 19)
axes[0, 0].set_ylabel('Race', fontsize = 19)
axes[0, 0].tick_params(labelsize = 17)
axes[0, 0].set_xticks(range(0,110,10))
axes[0, 0].set_xlim(0,117)
axes[0, 0].legend(fontsize=17)

# sex
axes[1, 0].barh(sex_grp['sex_at_birth'].drop_duplicates().tolist(),
               sex_grp[sex_grp['ADE_Classification'] == 'Yes']['percentage'].tol
               label = 'ADE Yes')
axes[1, 0].barh(sex_grp['sex_at_birth'].drop_duplicates().tolist(),
               sex_grp[sex_grp['ADE_Classification'] == 'No']['percentage'].toli
               left = sex_grp[sex_grp['ADE_Classification'] == 'Yes']['percentag
               label = 'ADE No')
axes[1, 0].set_title('Sex Distribution by ADE Class', fontsize = 19)
axes[1, 0].set_xlabel('Percentage (%)', fontsize = 19)
axes[1, 0].set_ylabel('Sex at Birth', fontsize = 19)
axes[1, 0].tick_params(labelsize = 17)
axes[1, 0].set_xlim(0,117)
axes[1, 0].set_xticks(range(0,110,10))
axes[1, 0].legend(fontsize=17, loc="upper right")

# age
sns.violinplot(ax = axes[0, 1], x="ADE_Classification", y="age", data=age_grp, o
axes[0, 1].set_title('Age Distribution by ADE Class', fontsize = 19)
axes[0, 1].set_xlabel('ADE Class', fontsize = 19)
axes[0, 1].set_ylabel('Age', fontsize = 19)
axes[0, 1].set_xticklabels(labels = ['ADE Yes', 'ADE No'])
axes[0, 1].tick_params(labelsize = 17)

# dispensed amount
sns.violinplot(ax = axes[1, 1], x = 'ADE_Classification', y = 'dispensed_count',
               data = clop_ade, order = [1, 0])
axes[1, 1].set_title('Dispensed Count Distribution by ADE Class', fontsize = 19)
axes[1, 1].set_xlabel('ADE Class', fontsize = 19)
axes[1, 1].set_ylabel('Dispensed Count', fontsize = 19)
axes[1, 1].set_xticklabels(labels = ['ADE Yes', 'ADE No'])
axes[1, 1].tick_params(labelsize = 17)
plt.show()
```



Chi-Squared p-values

In [245…

```
ade2_min = overlap_df2.reset_index()[overlap_df2.reset_index().columns[:39]]
clopidogrel_3['ADE_Classification'] = pd.Series(['No']*clopidogrel_3.shape[0])
ade2_min['ADE_Classification'] = pd.Series(['Yes']*overlap_df2.shape[0])

ade3_min = overlap_df3.reset_index()[overlap_df3.reset_index().columns[:39]]
clopidogrel_3['ADE_Classification'] = pd.Series(['No']*clopidogrel_3.shape[0])
ade3_min['ADE_Classification'] = pd.Series(['Yes']*overlap_df3.shape[0])


clop_ade2 = pd.concat([clopidogrel_3, ade2_min], ignore_index=True).groupby(
    'person_id', as_index = False).apply(lambda x: x.sort_values(
    'ADE_Classification', ascending = False).head(1)).reset_index(drop = True)

clop_ade3 = pd.concat([clopidogrel_3, ade3_min], ignore_index=True).groupby(
    'person_id', as_index = False).apply(lambda x: x.sort_values(
    'ADE_Classification', ascending = False).head(1)).reset_index(drop = True)


clop_ades = [clop_ade2, clop_ade3]
```

In [247…

```
'''
Age Bins Classification:
0-17: 0
18-24: 1
25-34: 2
35-44: 3
45-54: 4
55-64: 5
65+: 6
'''

for i in range(len(clop_ades)):
    clop_ades[i]['age'] = pd.DatetimeIndex(clop_ades[i].index_date).year - clop_
    clop_ades[i]['age_bins'] = (clop_ades[i]['age'] - 5)//10
```

In [249…
```
pval_dfs = []
pvalTitles = ['GI Hemorr', 'ICH']

for i in range(len(clop_ades)):
    ft_lst = ['care_sites', 'race',
          'sex_at_birth', 'gender', 'ADE_Classification', 'age_bins', 'ethnicity
    p_vals = [
        chi2_contingency(pd.crosstab(clop_ades[i]['care_sites'],
                                     clop_ades[i]['ADE_Classification'],
                                     margins = False))[1],

        chi2_contingency(pd.crosstab(clop_ades[i]['race'],
                                     clop_ades[i]['ADE_Classification'],
                                     margins = False))[1],

        chi2_contingency(pd.crosstab(clop_ades[i]['sex_at_birth'],
                                     clop_ades[i]['ADE_Classification'],
                                     margins = False))[1],

        chi2_contingency(pd.crosstab(clop_ades[i]['gender'],
                                     clop_ades[i]['ADE_Classification'],
                                     margins = False))[1],

        chi2_contingency(pd.crosstab(clop_ades[i]['ADE_Classification'],
                                     clop_ades[i]['ADE_Classification'],
                                     margins = False))[1],

        chi2_contingency(pd.crosstab(clop_ades[i]['age_bins'],
                                     clop_ades[i]['ADE_Classification'],
                                     margins = False))[1],

        chi2_contingency(pd.crosstab(clop_ades[i]['ethnicity'],
                                     clop_ades[i]['ADE_Classification'],
                                     margins = False))[1],

    ]
    if i == len(clop_ades) - 1:
        p_str = pvalTitles[-1]
    else:
        p_str = pvalTitles[i]
    pval_df = pd.DataFrame({'Feature': ft_lst, p_str: p_vals})
    pval_dfs.append(pval_df)
```

In [251…
```
adf = pval_dfs[0].merge(pval_dfs[1], left_on ='Feature', right_on = 'Feature'
                  )
create_download_link(adf, 'psqwarf', 'psqwarf.csv')
```

Out[251… psqwarf

In [253…
```
adf
```

Out[253…

	Feature	GI Hemorr	ICH
0	care_sites	0.836752	0.871127
1	race	0.846859	0.478102
2	sex_at_birth	0.381585	0.733668

	Feature	GI Hemorr	ICH
3	gender	0.336051	0.959906
4	ADE_Classification	0.000000	0.000000
5	age_bins	0.044983	0.139058
6	ethnicity	0.523592	0.100981

In [255…

```
cat_var = ['race', 'ethnicity', 'sex_at_birth']

pval_df = pd.DataFrame(columns = ['cat1', 'cat2', 'P_value'])
for i in range(len(cat_var)):
    for j in range(len(cat_var)):

        pval_df = pval_df.append({'cat1': cat_var[i], 'cat2': cat_var[j],
        'P_value': chi2_contingency(pd.crosstab(clop_ade2[cat_var[i]],
                                    clop_ade2[cat_var[j]],
                                    margins = False))[1]
                                }, ignore_index = True)

#     if i == len(clop_ades) - 1:
#         p_str = pvalTitles[-1]
#     else:
#         p_str = pvalTitles[i]
#     pval_df = pd.DataFrame({'Feature': ft_lst, p_str: p_vals})
#     pval_dfs.append(pval_df)
```

In [257…

```
pval_df.pivot(index = 'cat1', columns = 'cat2', values = 'P_value')
```

Out[257…

cat1 \ cat2	ethnicity	race	sex_at_birth
ethnicity	0.000000e+00	0.000000e+00	7.374136e-11
race	0.000000e+00	0.000000e+00	4.229686e-206
sex_at_birth	7.374136e-11	4.229686e-206	0.000000e+00

Age at ADE Classification Distribution

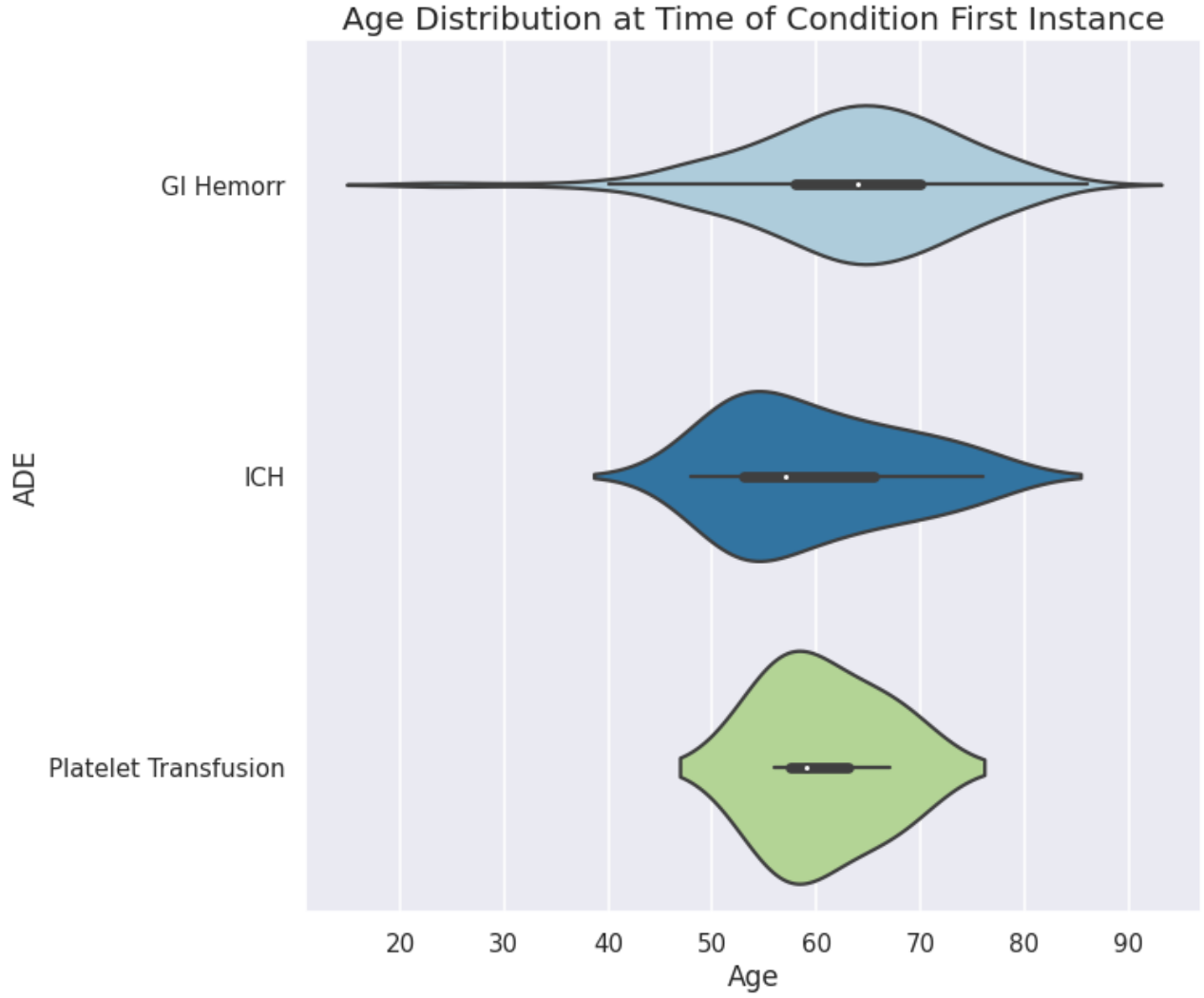
In [259…

```
age_condition1 = overlap_df2.reset_index()
age_condition1['age'] = pd.DatetimeIndex(age_condition1.condition_start_date).ye
age_condition1['condition'] = 'GI Hemorr'
age_condition1 = age_condition1[['age', 'condition']]

age_condition2 = overlap_df3.reset_index()
age_condition2['age'] = pd.DatetimeIndex(age_condition2.condition_start_date).ye
age_condition2['condition'] = 'ICH'
age_condition2 = age_condition2[['age', 'condition']]

age_condition3 = overlap_df5.reset_index()
age_condition3['age'] = pd.DatetimeIndex(age_condition3.procedure_date).year - a
age_condition3['condition'] = 'Platelet Transfusion'
age_condition3 = age_condition3[['age', 'condition']]
```

```
age_condition = age_condition1.append(age_condition2, ignore_index = True).appen
    age_condition3, ignore_index = True)
```

In [261…

```
fig,ax = plt.subplots(1,1, figsize = (10,10))
sns.set_style("darkgrid")
sns.violinplot(x="age", y="condition", data=age_condition)
ax.set_title('Age Distribution at Time of Condition First Instance', fontsize =
ax.set_ylabel('ADE', fontsize = 17)
ax.set_xlabel('Age', fontsize = 17)
ax.tick_params(labelsize = 15)
plt.show()
```



ADE Classification Rate

In [263…

```
overlap_dfs = [overlap_df2, overlap_df3, overlap_df5]
period_ct_dfs = []
for i in range(len(overlap_dfs)):
    if i!=2:
        overlap_dfs[i]['period'] = (overlap_dfs[i]['condition_start_date'] - ove
                                    )/np.timedelta64(1, 'M')

        period_cts = overlap_dfs[i].groupby('period').count().reset_index()[['pe
```

```
        period_cts['cum_cts'
                   ] = overlap_dfs[i].groupby('period').count().reset_index()['dr
        period_cts.columns = ['period', 'count', 'cum_cts']
        period_ct_dfs.append(period_cts)
    else:
        overlap_dfs[i]['period'] = (pd.to_datetime(overlap_dfs[i]['procedure_dat
                                                   ) - overlap_dfs[i]['index_date
                                    )/np.timedelta64(1, 'M')

        period_cts = overlap_dfs[i].groupby('period').count().reset_index()[['pe
        period_cts['cum_cts'] = overlap_dfs[i].groupby('period').count().reset_i

        period_cts.columns = ['period', 'count', 'cum_cts']
        period_ct_dfs.append(period_cts)
```

In [265…

```
# overlap_dfs = [overlap_df2, overlap_df3, overlap_df5]
# period_ct_dfs = []
# for i in range(len(overlap_dfs)):
#     if i!=2:
overlap_df2['period'] = (overlap_df2['condition_start_date'] - overlap_df2['inde
                         )/np.timedelta64(1, 'M')

period_cts = overlap_df2.groupby(['period', 'sex_at_birth']).count().reset_index


# period_cts['cum_cts'
#             ] = overlap_dfs[i].groupby('period').count().reset_index()['drug_n
# period_cts.columns = ['period', 'count', 'cum_cts']
# period_ct_dfs.append(period_cts)
per_male = period_cts[period_cts.sex_at_birth == 'Male']
per_female = period_cts[period_cts.sex_at_birth == 'Female']
per_other = period_cts[period_cts.sex_at_birth != 'Male'][period_cts.sex_at_birt

male_cum = overlap_df2.groupby(['period','sex_at_birth']).count().reset_index()[
per_male['cum_cts'] = male_cum[male_cum.sex_at_birth == 'Male']['drug_name'].cum

female_cum = overlap_df2.groupby(['period','sex_at_birth']).count().reset_index(

per_female['cum_cts'] = female_cum[female_cum.sex_at_birth == 'Female']['drug_na

other_cum = overlap_df2.groupby(['period','sex_at_birth']).count().reset_index()

per_other['cum_cts'] = female_cum[female_cum.sex_at_birth != 'Female'][female_cu
    'drug_name'].cumsum().tolist()
```

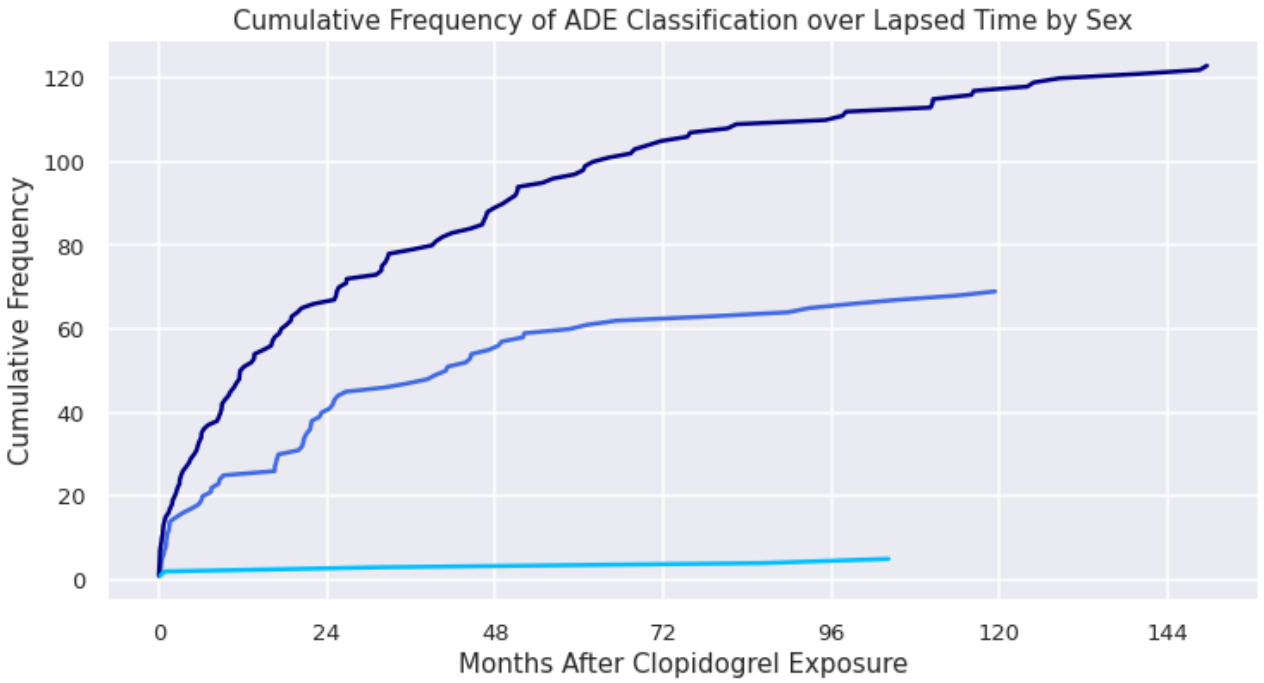
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:17: UserWarning: Bo
olean Series key will be reindexed to match DataFrame index.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:20: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:24: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
```

```
le/user_guide/indexing.html#returning-a-view-versus-a-copy
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: UserWarning: Bo
olean Series key will be reindexed to match DataFrame index.
```

In [ ]:

In [267…

```
sns.set_style("darkgrid")
sns.set_context("talk")
fig, ax = plt.subplots(1, 1, figsize=(12,6))

sns.lineplot(ax = ax, data=per_female, x="period", y="cum_cts", linewidth=2.5, c
sns.lineplot(ax = ax, data=per_male, x="period", y="cum_cts", linewidth=2.5, col
sns.lineplot(ax = ax, data=per_other, x="period", y="cum_cts", linewidth=2.5, co

ax.set_xlabel('Months After Clopidogrel Exposure', fontsize = 15)
ax.set_ylabel('Cumulative Frequency', fontsize = 15)
ax.set_title('Cumulative Frequency of ADE Classification over Lapsed Time by Sex
ax.tick_params(labelsize = 13)
ax.set_xticks(range(0,160,24))

plt.show()
```



In [269…

```
gi_hem = period_ct_dfs[0].describe()[['cum_cts']]
gi_hem.reset_index(inplace = True)
gi_hem.columns = ['Statistic', 'GI_Hemorr']

ich = period_ct_dfs[1].describe()[['cum_cts']]
ich.reset_index(inplace = True)
ich.columns = ['Statistic', 'ICH']


platelet = period_ct_dfs[2].describe()[['cum_cts']]
platelet.reset_index(inplace = True)
platelet.columns = ['Statistic', 'Platelet_Transusion']
```

```
month_stats = gi_hem.join(ich.set_index('Statistic'), on = 'Statistic').join(
    platelet.set_index('Statistic'), on = 'Statistic')

month_stats
```

Out[269…

	Statistic	GI_Hemorr	ICH	Platelet_Transusion
0	count	183.000000	17.000000	3.0
1	mean	103.278689	10.470588	2.0
2	std	55.945754	5.767556	1.0
3	min	1.000000	1.000000	1.0
4	25%	56.500000	6.000000	1.5
5	50%	105.000000	11.000000	2.0
6	75%	151.500000	15.000000	2.5
7	max	197.000000	19.000000	3.0

In [271…

```
# sns.set_style("darkgrid")
# sns.set_context("talk")
# fig, ax = plt.subplots(1, 1, figsize=(12,6))

# sns.lineplot(ax = ax, data=period_cts, x="period", y="cum_cts", linewidth=2.5,
# ax.set_xlabel('Months After Clopidogrel Exposure', fontsize = 15)
# ax.set_ylabel('Cumulative Frequency', fontsize = 15)
# ax.set_title('Cumulative Frequency of ADE Classification by Lapsed Time', font
# ax.tick_params(labelsize = 13)
# # plt.setp(l,linewidth=5)

# plt.show()
```

In [ ]:

Survey Data

In [273…

```
# create test cohort to test functions
test_cohort_sql = f"""
    with cohort as (
        select * from {os.environ["WORKSPACE_CDR"]}.person limit 1000
    )
    select distinct person_id from cohort
    """
test_cohort = list(pd.read_gbq(test_cohort_sql, dialect="standard")['person_id']
#     insert_values_sql0 = ",".join([f"({x})" for x in test_cohort])

# insert_values_sql1 = ",".join([f"({x})" for x in all_comorbidities_condition_c
# cohort_sql = f"""
# CREATE TEMP TABLE cohort
# (
#   person_id INT64
# );
# INSERT INTO cohort
```

```
# VALUES {insert_values_sql0};


# SELECT * FROM cohort;

# """
# cohort = pd.read_gbq(cohort_sql, dialect="standard")

# test_cohort
```

In [275…

```
p_id = clopidogrel_3.person_id.to_list()
```

In [277…

```
# Define PROMIS_health_concept_set_sql
questions_concept_set_sql = f"""
    question_concept_id IN (
                1585723, 1585741, 1585747, 1585748 -- Physical Health
                , 1585760, 1585729, 1585717, 1585735 -- Mental Health
                , 1585711
                , 1585811, 1585784 -- Menstrual/Pregnant
                , 1585803, 1585806
                --Lifestyle Qs
                , 1585857, 1585860, 1585864, 1585867, 1585870
                , 1585873, 1586159, 1586162, 1586166, 1586174
                , 1586177, 1586182, 1586185, 1586190, 1586193 -- smoking
                , 1586198, 1586201, 1586207, 1586213, 1585636
                , 1585650, 1585656, 1585662, 1585668, 1585674
                , 1585680, 1585686, 1585692, 1585698, 1585704
                --Basics
                , 1585940, 1585892, 1585386
            )
"""
```

In [279…

```
# Define SQL query for add Overall Health survey
add_health_survey_sql = f"""
    with_health_survey as (
        select a.person_id, answer,
        cast(cast(question_concept_id as int64) as string) as question_concept_i
        survey_date,
        cast(cast(answer_concept_id as int64) as string) as answer_concept_id,
        question from cohort a
        inner join (
            SELECT
                answer.person_id, answer.answer,
                answer.question_concept_id, extract(date from answer.survey_date
                answer.answer_concept_id, answer.question
            FROM {os.environ["WORKSPACE_CDR"]}.ds_survey answer
            WHERE
                (
                    {questions_concept_set_sql}
                )
            )  b
            on a.person_id = b.person_id
    )
"""
```

In [281…

```
# cohort is list of person_ids
def get_survey_data_for_cohort(cohort):
    insert_values_sql0 = ",".join([f"({x})" for x in cohort])
    health_sql = f"""
    CREATE TEMP TABLE cohort
    (
      person_id INT64
    );
    INSERT INTO cohort
    VALUES {insert_values_sql0};


    SELECT * FROM cohort;
    with
    {add_health_survey_sql}
    select * from with_health_survey
    """
    health = pd.read_gbq(health_sql, dialect="standard")
    return health
```

In [283…

```
survey_data1 = get_survey_data_for_cohort(p_id[:1000])
survey_data2 = get_survey_data_for_cohort(p_id[1000:])
```

In [285…

```
survey_data2.shape
```

Out[285… (23249, 6)

In [287…

```
# 25752 + 23286 = 49038
survey_data = survey_data1.append(survey_data2)
```

In [289…

```
# define helper functions
def get_questions(df):
    questions = list(df['question'].unique())
    this_df = df[['person_id','question']][df[['person_id','question']].duplicat
    multiple_choice = list(this_df['question'].unique())
    single_choice = [x for x in questions if x not in multiple_choice]
    return (multiple_choice, single_choice)

def create_single_choice_columns(df, single_choice):
    this_df = df.copy()
    this_df = this_df[['person_id','question', 'answer']]
    this_df = this_df[this_df['question'].isin(single_choice)]
    this_df = this_df.pivot(index='person_id', columns="question", values="answe
    return this_df

def create_multiple_choice_columns(df, multiple_choice):
    this_df = df.copy()
    this_df = this_df[['person_id','question', 'answer']]
    this_df = this_df[this_df['question'].isin(multiple_choice)]
    this_df['question (answer)'] = this_df['question'] + " = " + this_df['answer
#     this_df['question (answer)'] = this_df['answer']
    this_df = this_df[['person_id','question (answer)']]
    this_df['indicator'] = 1
    this_df = this_df.pivot(index='person_id', columns="question (answer)", valu
    this_df = this_df.fillna(0)
```

```
    this_df = this_df.astype(int)
    return(this_df)

def create_indicator_df(df):
    mult_choice, sing_choice = get_questions(df)
    sing_df = create_single_choice_columns(df, sing_choice)
    mult_df = create_multiple_choice_columns(df, mult_choice)
    output_df = sing_df.join(mult_df, on='person_id', how='outer')
    return(output_df)
```

In [291…
```
survey_temp = survey_data[['person_id', 'answer', 'question']]
```

In [293…
```
survey_temp = survey_temp.pivot_table(index="person_id", columns='question', val
```

In [295…
```
survey_temp
```

Out[295…

question	Alcohol: 6 or More Drinks Occurrence	Alcohol: Alcohol Participant	Alcohol: Average Daily Drink Count	Alcohol: Drink Frequency Past Year	Attempt Quit Smoking: Completely Quit Age	Cigar Smoking: Cigar Smoke Participant	Cigar Smoking: Current Cigar Frequency	Edu H
person_id								
[redacted]	6 or More Drinks Occurrence: Never In Last Year	Alcohol Participant: Yes	Average Daily Drink Count: 1 or 2	Drink Frequency Past Year: Monthly Or Less	NaN	Cigar Smoke Participant: Yes	Current Cigar Frequency: Not At All	(gr ad
	6 or More Drinks Occurrence: Never In Last Year	Alcohol Participant: Yes	Average Daily Drink Count: 1 or 2	Drink Frequency Past Year: 2 to 4 Per Month	NaN	Cigar Smoke Participant: No	NaN	(gr ad
	6 or More Drinks Occurrence: Never In Last Year	Alcohol Participant: Yes	Average Daily Drink Count: 1 or 2	Drink Frequency Past Year: 4 or More Per Week	NaN	Cigar Smoke Participant: Yes	Current Cigar Frequency: Not At All	H (
	NaN	Alcohol Participant: Yes	NaN	Drink Frequency Past Year: Never	NaN	Cigar Smoke Participant: Yes	Current Cigar Frequency: Not At All	H (
	NaN	Alcohol Participant: Yes	NaN	Drink Frequency Past Year: Never	NaN	Cigar Smoke Participant: Yes	PMI: Dont Know	H Tw
	...	...	...	...	...	...	...	

question	Alcohol: 6 or More Drinks Occurrence	Alcohol: Alcohol Participant	Alcohol: Average Daily Drink Count	Alcohol: Drink Frequency Past Year	Attempt Quit Smoking: Completely Quit Age	Cigar Smoking: Cigar Smoke Participant	Cigar Smoking: Current Cigar Frequency	Edu H
person_id								
[redacted]	6 or More Drinks Occurrence: Never In Last Year	Alcohol Participant: Yes	Average Daily Drink Count: 3 or 4	Drink Frequency Past Year: Monthly Or Less	NaN	Cigar Smoke Participant: Yes	Current Cigar Frequency: Not At All	Tw
	6 or More Drinks Occurrence: Never In Last Year	Alcohol Participant: Yes	Average Daily Drink Count: 1 or 2	Drink Frequency Past Year: 2 to 4 Per Month	NaN	Cigar Smoke Participant: No	NaN	gr ad
	6 or More Drinks Occurrence: Weekly	Alcohol Participant: Yes	Average Daily Drink Count: 5 or 6	Drink Frequency Past Year: 4 or More Per Week	NaN	Cigar Smoke Participant: No	NaN	gr ad
	6 or More Drinks Occurrence: Never In Last Year	Alcohol Participant: Yes	Average Daily Drink Count: 1 or 2	Drink Frequency Past Year: Monthly Or Less	NaN	Cigar Smoke Participant: No	NaN	gr ad
	NaN	Alcohol Participant: Yes	NaN	Drink Frequency Past Year: Never	NaN	Cigar Smoke Participant: Yes	Current Cigar Frequency: Not At All	Tw

1906 rows × 43 columns

In [297…
```
survey_temp_col = list(survey_temp.columns)
to_delete = list()
for i in survey_temp_col:
    sum = survey_temp[survey_temp[i].isnull()].shape[0]
    sum += survey_temp[survey_temp[i] == 'PMI: Prefer Not To Answer'].shape[0]
    sum += survey_temp[survey_temp[i] == 'PMI: Skip'].shape[0]
    if ((sum/survey_temp.shape[0]) >= 0.5):
        to_delete.append(i)
```

In [299…
```
for i in to_delete:
    del survey_temp[i]
```

In [301…
```
survey_temp_col = list(survey_temp.columns)
to_delete = list()
for i in survey_temp_col:
    sum = survey_temp[survey_temp[i].isnull()].shape[0]
    sum += survey_temp[survey_temp[i] == 'PMI: Prefer Not To Answer'].shape[0]
```

```
    sum += survey_temp[survey_temp[i] == 'PMI: Skip'].shape[0]
    if ((sum/survey_temp.shape[0]) >= 0.2):
        to_delete.append((i, sum/1906))
survey_temp = survey_temp.replace({'PMI: Skip': np.nan, 'PMI: Prefer Not To Answ
                                   'PMI: Dont Know': np.nan})
```

In [303…

```
del survey_temp['Alcohol: 6 or More Drinks Occurrence']
del survey_temp['Smoking: Serious Quit Attempt']
```

In [305…

```
survey_temp.reset_index(inplace=True)
survey_col = list(survey_temp.columns)
```

In [307…

```
# Change the following columns to ordered variables

# 'Alcohol: Drink Frequency Past Year'
dict1 = {'PMI: Skip': np.nan, 'PMI: Prefer Not To Answer': np.nan, 'NA': np.nan,
         'Drink Frequency Past Year: Never':0, 'Drink Frequency Past Year: Month
         'Drink Frequency Past Year: 2 to 4 Per Month': 2, 'Drink Frequency Past
         'Drink Frequency Past Year: 4 or More Per Week': 4
        }
survey_temp = survey_temp.replace({'Alcohol: Drink Frequency Past Year': dict1})

# 'Education Level: Highest Grade'
dict1 = {'PMI: Skip': np.nan, 'PMI: Prefer Not To Answer': np.nan,
         'Less than a high school degree or equivalent':0, 'Highest Grade: Twelv
         'Highest Grade: College One to Three': 2, 'College graduate or advanced
        }
survey_temp = survey_temp.replace({'Education Level: Highest Grade': dict1})

# Overall Health: Average Fatigue 7 Days
dict1 = {'PMI: Skip': np.nan, 'Average Fatigue 7 Days: None': 0, 'Average Fatigu
         'Average Fatigue 7 Days: Moderate':2, 'Average Fatigue 7 Days: Severe':
         'Average Fatigue 7 Days: Very Severe': 4
        }
survey_temp = survey_temp.replace({'Overall Health: Average Fatigue 7 Days': dic

# Replace: Overall Health: Emotional Problem 7 Days
dict1 = {'PMI: Skip': np.nan, 'Emotional Problem 7 Days: Never': 0, 'Emotional P
         'Emotional Problem 7 Days: Sometimes': 2, 'Emotional Problem 7 Days: Of
         'Emotional Problem 7 Days: Always': 4
        }
survey_temp = survey_temp.replace({'Overall Health: Emotional Problem 7 Days': d

# Replace: Overall Health: General Health
dict1 = {'PMI: Skip': np.nan, 'General Health: Poor': 0, 'General Health: Fair':
         'General Health: Good': 2, 'General Health: Very Good': 3,
         'General Health: Excellent': 4
        }
survey_temp = survey_temp.replace({'Overall Health: General Health': dict1})

# Replace: Overall Health: General Mental Health
dict1 = {'PMI: Skip': np.nan, 'General Mental Health: Poor': 0, 'General Mental
         'General Mental Health: Good': 2, 'General Mental Health: Very Good': 3
         'General Mental Health: Exclent': 4, 'General Mental Health: Excellent
        }
survey_temp = survey_temp.replace({'Overall Health: General Mental Health': dict
```

```
# Overall Health: General Quality
dict1 = {'PMI: Skip': np.nan, 'General Quality: Poor': 0, 'General Quality: Fair
         'General Quality: Good': 2, 'General Quality: Very Good': 3,
         'General Quality: Excellent': 4
        }
survey_temp = survey_temp.replace({'Overall Health: General Quality': dict1})

# 'Overall Health: Average Pain 7 Days'
dict1 = {'PMI: Skip': np.nan}
survey_temp = survey_temp.replace({'Overall Health: Average Pain 7 Days': dict1}
survey_temp['Overall Health: Average Pain 7 Days'] = survey_temp[
    'Overall Health: Average Pain 7 Days'].astype(float)

# 'Overall Health: Everyday Activities'
survey_temp = survey_temp.replace({'Overall Health: Everyday Activities':
                {'Everyday Activities: Completely': 4,
                 'Everyday Activities: Mostly': 3,
                 'Everyday Activities: Moderately': 2,
                 'Everyday Activities: A Little': 1,
                 'Everyday Activities: Not At All': 0,
                 'PMI: Skip': -1}})

# 'Overall Health: General Physical Health'
survey_temp = survey_temp.replace({
    'Overall Health: General Physical Health': {
        'General Physical Health: Very Good': 3,
        'General Physical Health: Good': 2,
        'PMI: Skip': np.nan,
        'General Physical Health: Fair': 1,
        'General Physical Health: Poor': 0,
        'General Physical Health: Excellent': 4}
})

# "Overall Health: Social Satisfaction"
xdict1 = {
    "Social Satisfaction: Very Good" : 4,
    "Social Satisfaction: Good" : 3,
    "Social Satisfaction: Excellent" : 2,
    "Social Satisfaction: Fair" : 1,
    "Social Satisfaction: Poor" : 0,
    "PMI: Skip" : np.nan,
    'NA': np.nan
}

survey_temp = survey_temp.replace({"Overall Health: Social Satisfaction": xdict1

# 'Alcohol: Average Daily Drink Count'
dict1 = {'PMI: Skip': -1, 'PMI: Prefer Not To Answer': np.nan, 'NA': np.nan,
         'Average Daily Drink Count: 1 or 2':0, 'Average Daily Drink Count: 3 or
         'Average Daily Drink Count: 5 or 6': 2, 'Average Daily Drink Count: 7 t
         'Average Daily Drink Count: 10 or More': 4
        }
survey_temp = survey_temp.replace({'Alcohol: Average Daily Drink Count': dict1})
```

In [309…

```
survey_temp
```

Out[309…

question	person_id	Alcohol: Alcohol Participant	Alcohol: Average Daily Drink Count	Alcohol: Drink Frequency Past Year	Cigar Smoking: Cigar Smoke Participant	Education Level: Highest Grade	Electronic Smoking: Electric Smoke Participant	Hoo Smok Hoo Sm Particip
0	[redacted]	Alcohol Participant: Yes	0.0	1.0	Cigar Smoke Participant: Yes	3.0	Electric Smoke Participant: No	Hoo Sm Particip
1	[redacted]	Alcohol Participant: Yes	0.0	2.0	Cigar Smoke Participant: No	3.0	Electric Smoke Participant: No	Hoo Sm Particip
2	[redacted]	Alcohol Participant: Yes	0.0	4.0	Cigar Smoke Participant: Yes	2.0	Electric Smoke Participant: No	Hoo Sm Particip
3	[redacted]	Alcohol Participant: Yes	NaN	0.0	Cigar Smoke Participant: Yes	2.0	Electric Smoke Participant: No	Hoo Sm Particip
4	[redacted]	Alcohol Participant: Yes	NaN	0.0	Cigar Smoke Participant: Yes	1.0	Electric Smoke Participant: No	Hoo Sm Particip
...	[redacted]	...	...	...	...	...	...	
1901	[redacted]	Alcohol Participant: Yes	1.0	1.0	Cigar Smoke Participant: Yes	1.0	Electric Smoke Participant: Yes	Hoo Sm Particip
1902	[redacted]	Alcohol Participant: Yes	0.0	2.0	Cigar Smoke Participant: No	3.0	Electric Smoke Participant: No	Hoo Sm Particip
1903	[redacted]	Alcohol Participant: Yes	2.0	4.0	Cigar Smoke Participant: No	3.0	Electric Smoke Participant: Yes	Hoo Sm Particip
1904	[redacted]	Alcohol Participant: Yes	0.0	1.0	Cigar Smoke Participant: No	3.0	Electric Smoke Participant: No	Hoo Sm Particip
1905	[redacted]	Alcohol Participant: Yes	NaN	0.0	Cigar Smoke Participant: Yes	1.0	Electric Smoke Participant: No	Hoo Sm Particip

1906 rows × 24 columns

In [311…
```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, SimpleImputer
imp_mean = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
imp_mean.fit(survey_temp)
survey_imp = imp_mean.transform(survey_temp)

survey_filled = pd.DataFrame(survey_imp, columns = survey_col)
#imp.fit(health.drop(columns = 'Marital Status: Current Marital Status'))
#IterativeImputer(random_state=0)
#health_np = np.round(imp.transform(health.drop(columns = 'Marital Status: Curre
```

In [313…
```
health = create_indicator_df(survey_data)
health.columns
```

Out[313…
```
Index(['Alcohol: 6 or More Drinks Occurrence', 'Alcohol: Alcohol Participant',
       'Alcohol: Average Daily Drink Count',
       'Alcohol: Drink Frequency Past Year',
       'Attempt Quit Smoking: Completely Quit Age',
       'Cigar Smoking: Cigar Smoke Participant',
       'Cigar Smoking: Current Cigar Frequency',
       'Education Level: Highest Grade',
       'Electronic Smoking: Electric Smoke Participant',
       'Hookah Smoking: Current Hookah Frequency',
       'Hookah Smoking: Hookah Smoke Participant',
       'Insurance: Health Insurance', 'Marital Status: Current Marital Status',
       'Overall Health: Average Fatigue 7 Days',
       'Overall Health: Average Pain 7 Days',
       'Overall Health: Emotional Problem 7 Days',
       'Overall Health: Everyday Activities', 'Overall Health: General Health',
       'Overall Health: General Mental Health',
       'Overall Health: General Physical Health',
       'Overall Health: General Quality', 'Overall Health: Menstrual Stopped',
       'Overall Health: Organ Transplant',
       'Overall Health: Social Satisfaction',
       'Past 3 Month Use Frequency: Cocaine 3 Month Use',
       'Past 3 Month Use Frequency: Inhalant 3 Month Use',
       'Past 3 Month Use Frequency: Marijuana 3 Month Use',
       'Past 3 Month Use Frequency: Other 3 Month Use',
       'Past 3 Month Use Frequency: Other Stimulant 3 Month Use',
       'Past 3 Month Use Frequency: Prescription Opioid 3 Month Use',
       'Past 3 Month Use Frequency: Sedative 3 Month Use',
       'Pregnancy: 1 Pregnancy Status',
       'Smokeless Tobacco: Smokeless Tobacco Frequency',
       'Smokeless Tobacco: Smokeless Tobacco Participant',
       'Smoking: 100 Cigs Lifetime', 'Smoking: Average Daily Cigarette Number',
       'Smoking: Current Daily Cigarette Number',
       'Smoking: Daily Smoke Starting Age', 'Smoking: Number Of Years',
       'Smoking: Serious Quit Attempt', 'Smoking: Smoke Frequency',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Blood Vessels',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Bone',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Cornea',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Heart',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Kidney',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Liver',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Lung',
```

```
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Other Tissue',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Skin',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Valve',
       'Organ Transplant: Organ Transplant Description = PMI: Skip',
       'Recreational Drug Use: Which Drugs Used = PMI: Prefer Not To Answer',
       'Recreational Drug Use: Which Drugs Used = PMI: Skip',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Cocaine Us
e',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Hallucinogen
s Use',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Inhalants Us
e',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Marijuana Us
e',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Methamphetam
ine Use',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: None Of Thes
e Drugs',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Other Specif
y',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Prescription
Opioids Use',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Prescription
Stimulants Use',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Sedatives Us
e',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Street Opioi
ds Use'],
      dtype='object')
```

In [315…

```
# Combine 'Recreational Drug Use: Which Drugs Used = PMI: Prefer Not To Answer'
# 'Recreational Drug Use: Which Drugs Used = PMI: Skip'
health['Recreational Drug Use: Which Drugs Used = Null'
      ] = health['Recreational Drug Use: Which Drugs Used = PMI: Prefer Not To A
                 ] + health['Recreational Drug Use: Which Drugs Used = PMI: Skip'
health[['Recreational Drug Use: Which Drugs Used = Null',
        'Recreational Drug Use: Which Drugs Used = PMI: Prefer Not To Answer',
        'Recreational Drug Use: Which Drugs Used = PMI: Skip']]
health.drop(columns = 'Recreational Drug Use: Which Drugs Used = PMI: Prefer Not
health.drop(columns = 'Recreational Drug Use: Which Drugs Used = PMI: Skip', inp

# Column: 'Cigar Smoking: Current Cigar Frequency'
health.drop(columns = 'Cigar Smoking: Current Cigar Frequency', inplace = True)

def binaryConv(colName, health = health):
    bin_dic = {}
    y_str = health[health[colName].str.contains("Yes")
                   ][colName].tolist()[1]
    n_str = health[health[colName].str.contains("No")
                   ][colName].tolist()[1]
    for i in health[colName].unique():
        if i != y_str and i!= n_str:
            bin_dic[i] = -1
    bin_dic[y_str] = 1
    bin_dic[n_str] = 0
    health = health.replace({colName: bin_dic})
    return health

# Column: 'Cigar Smoking: Cigar Smoke Participant'
```

```
health = binaryConv('Cigar Smoking: Cigar Smoke Participant', health)

# # Column: 'Overall Health: Average Pain 7 Days'
# health = health.replace({'Overall Health: Average Pain 7 Days': {'PMI: Skip':

dict1 = {'PMI: Skip': -1}
health = health.replace({'Overall Health: Average Pain 7 Days': dict1})

# Column: 'Overall Health: Everyday Activities'
health = health.replace({'Overall Health: Everyday Activities':
                {'Everyday Activities: Completely': 4,
                 'Everyday Activities: Mostly': 3,
                 'Everyday Activities: Moderately': 2,
                 'Everyday Activities: A Little': 1,
                 'Everyday Activities: Not At All': 0,
                 'PMI: Skip': -1}})

# Column: 'Overall Health: General Physical Health'
health = health.replace({
    'Overall Health: General Physical Health': {
        'General Physical Health: Very Good': 3,
        'General Physical Health: Good': 2,
        'PMI: Skip': -1,
        'General Physical Health: Fair': 1,
        'General Physical Health: Poor': 0,
        'General Physical Health: Excellent': 4}
})

# Column: 'Past 3 Month Use Frequency: Sedative 3 Month Use'
health.drop(columns = 'Past 3 Month Use Frequency: Sedative 3 Month Use', inplac

# Column: 'Smoking: '100 Cigs Lifetime'
health = binaryConv('Smoking: 100 Cigs Lifetime', health)

# Column: 'Smoking: 'Average Daily Cigarette Number'
health.drop(columns = 'Smoking: Average Daily Cigarette Number', inplace = True)

# Column: 'Smoking: 'Daily Smoke Starting Age'
health.drop(columns = 'Smoking: Daily Smoke Starting Age', inplace = True)

# Column: 'Smoking: 'Smoking: Number Of Years'
health.drop(columns = 'Smoking: Number Of Years', inplace = True)

# Column: 'Smoking: 'Smoking: Smoke Frequency'
health.drop(columns = 'Smoking: Smoke Frequency', inplace = True)
```

In [317…

```
# Delete given columns
health = health.drop(['Overall Health: Menstrual Stopped',
              'Hookah Smoking: Current Hookah Frequency',
              'Attempt Quit Smoking: Completely Quit Age',
              'Alcohol: 6 or More Drinks Occurrence',
              'Alcohol: Average Daily Drink Count'], axis=1)

# fill NaN values as NA
health = health.fillna('NA')

# Replace: Alcohol: Alcohol Participant
dict1 = {'Alcohol Participant: Yes': 1,
      'Alcohol Participant: No': 0,
```

```
      'PMI: Skip': -1,
      'PMI: Prefer Not To Answer': -1}
health = health.replace({'Alcohol: Alcohol Participant': dict1})

# Replace: Alcohol: Drink Frequency Past Year
dict1 = {'PMI: Skip': -1, 'PMI: Prefer Not To Answer': -1, 'NA': -1,
         'Drink Frequency Past Year: Never':0, 'Drink Frequency Past Year: Month
         'Drink Frequency Past Year: 2 to 4 Per Month': 2, 'Drink Frequency Past
         'Drink Frequency Past Year: 4 or More Per Week': 4
        }
health = health.replace({'Alcohol: Drink Frequency Past Year': dict1})

# Replace: Education Level: Highest Grade
dict1 = {'PMI: Skip': -1, 'PMI: Prefer Not To Answer': -1,
         'Less than a high school degree or equivalent':0, 'Highest Grade: Twelv
         'Highest Grade: College One to Three': 2, 'College graduate or advanced
        }
health = health.replace({'Education Level: Highest Grade': dict1})

# Replace: Electronic Smoking: Electric Smoke Participant
dict1 = {'Electric Smoke Participant: Yes': 1,
      'Electric Smoke Participant: No': 0,
      'PMI: Skip': -1,
      'PMI: Prefer Not To Answer': -1,
        'PMI: Dont Know':-1}
health = health.replace({'Electronic Smoking: Electric Smoke Participant': dict1

# Replace: Hookah Smoking: Hookah Smoke Participant'
dict1 = {'Hookah Smoke Participant: Yes': 1,
      'Hookah Smoke Participant: No': 0,
      'PMI: Skip': -1,
      'PMI: Prefer Not To Answer': -1,
        'PMI: Dont Know':-1}
health = health.replace({'Hookah Smoking: Hookah Smoke Participant': dict1})

# Replace: Insurance: Health Insurance
dict1 = {'Health Insurance: Yes': 1,
      'Health Insurance: No': 0,
      'PMI: Skip': -1,
      'PMI: Prefer Not To Answer': -1,
        'PMI: Dont Know':-1,
        'NA':-1}
health = health.replace({'Insurance: Health Insurance': dict1})

# Replace: Marital Status: Current Marital Status
dict1 = {'Current Marital Status: Married': 'Married',
      'Current Marital Status: Widowed': 'Widowed',
      'Current Marital Status: Divorced': 'Divorced',
      'Current Marital Status: Living With Partner': 'Living With Partner',
        'Current Marital Status: Never Married': 'Never Married',
        'Current Marital Status: Separated':'Separated',
        'PMI: Prefer Not To Answer': 'Null',
        'PMI: Skip': 'Null'}
health = health.replace({'Marital Status: Current Marital Status': dict1})

# Replace: Overall Health: Average Fatigue 7 Days
dict1 = {'PMI: Skip': -1, 'Average Fatigue 7 Days: None': 0, 'Average Fatigue 7
         'Average Fatigue 7 Days: Moderate':2, 'Average Fatigue 7 Days: Severe':
         'Average Fatigue 7 Days: Very Severe': 4
        }
health = health.replace({'Overall Health: Average Fatigue 7 Days': dict1})
```

```
# Replace: Overall Health: Emotional Problem 7 Days
dict1 = {'PMI: Skip': -1, 'Emotional Problem 7 Days: Never': 0, 'Emotional Probl
         'Emotional Problem 7 Days: Sometimes': 2, 'Emotional Problem 7 Days: Of
         'Emotional Problem 7 Days: Always': 4
        }
health = health.replace({'Overall Health: Emotional Problem 7 Days': dict1})

# Replace: Overall Health: General Health
dict1 = {'PMI: Skip': -1, 'General Health: Poor': 0, 'General Health: Fair': 1,
         'General Health: Good': 2, 'General Health: Very Good': 3,
         'General Health: Excellent': 4
        }
health = health.replace({'Overall Health: General Health': dict1})

# Replace: Overall Health: General Mental Health
dict1 = {'PMI: Skip': -1, 'General Mental Health: Poor': 0, 'General Mental Heal
         'General Mental Health: Good': 2, 'General Mental Health: Very Good': 3
         'General Mental Health: Exclent': 4, 'General Mental Health: Excellent
        }
health = health.replace({'Overall Health: General Mental Health': dict1})

# Replace: Overall Health: General Quality
dict1 = {'PMI: Skip': -1, 'General Quality: Poor': 0, 'General Quality: Fair': 1
         'General Quality: Good': 2, 'General Quality: Very Good': 3,
         'General Quality: Excellent': 4
        }
health = health.replace({'Overall Health: General Quality': dict1})
```

In [319…

```
# *'Overall Health: Organ Transplant'*

xdict = {
    "Organ Transplant: No" : 0,
    "Organ Transplant: Yes" : 1,
    "PMI: Skip" : -1,
    "PMI: Dont Know" : -1,
    'NA': -1
}

health = health.replace({"Overall Health: Organ Transplant": xdict})



#### * 'Overall Health: Social Satisfaction'*

xdict1 = {
    "Social Satisfaction: Very Good" : 4,
    "Social Satisfaction: Good" : 3,
    "Social Satisfaction: Excellent" : 2,
    "Social Satisfaction: Fair" : 1,
    "Social Satisfaction: Poor" : 0,
    "PMI: Skip" : -1,
    'NA': -1
}

health = health.replace({"Overall Health: Social Satisfaction": xdict1})
```

```
#### * 'Smokeless Tobacco: Smokeless Tobacco Participant' *

xdict1 = {
    "Smokeless Tobacco Participant: No" : 0,
    "Smokeless Tobacco Participant: Yes" : 1,
    "PMI: Skip" : -1,
    "PMI: Prefer Not To Answer" : -1,
    "PMI: Dont Know" : -1,
    'NA': -1
}

health = health.replace({'Smokeless Tobacco: Smokeless Tobacco Participant': xdi



#### * 'Smoking: Serious Quit Attempt'*

xdict1 = {
    "Serious Quit Attempt: Attempt Quit Smoking" : 1,
    "Serious Quit Attempt: No Attempt Quit Smoking" : 0,
    "PMI: Dont Know" : -1,
    "PMI: Prefer Not To Answer" : -1,
    "PMI: Skip" : -1,
    'NA': -1
}

health = health.replace({'Smoking: Serious Quit Attempt': xdict1})

#DELETES
del health[ 'Past 3 Month Use Frequency: Cocaine 3 Month Use']
del health[ 'Past 3 Month Use Frequency: Inhalant 3 Month Use']
del health[ 'Past 3 Month Use Frequency: Marijuana 3 Month Use']
del health[ 'Past 3 Month Use Frequency: Other 3 Month Use']
del health["Past 3 Month Use Frequency: Other Stimulant 3 Month Use"]
del health["Past 3 Month Use Frequency: Prescription Opioid 3 Month Use"]
del health[ 'Pregnancy: 1 Pregnancy Status']
del health['Smokeless Tobacco: Smokeless Tobacco Frequency']
del health['Smoking: Current Daily Cigarette Number']
```

In [321…

```
health.reset_index(inplace=True)
```

In [323…

```
# Convert all -1, and Null to np.nan
health = health.replace({-1: np.nan, 'Null': np.nan})
h_col = list(health.columns)
```

In [325…

```
# import numpy as np
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
imp = IterativeImputer(max_iter=1000, random_state=0)
imp.fit(health.drop(columns = 'Marital Status: Current Marital Status'))
IterativeImputer(random_state=0)
health_np = np.round(imp.transform(health.drop(columns = 'Marital Status: Curren
```

In [327…

```
h_col.remove('Marital Status: Current Marital Status')
```

```
hlth_filled = pd.DataFrame(health_np, columns = h_col)
```

In [329…

```
# for i in hlth_filled.columns:
#     print(sum(hlth_filled[i].isnull()))
```

In [331…

```
hlth_filled.person_id = hlth_filled.person_id.astype(int)
```

Machine Learning

In [333…

```
ade2_min = overlap_df2.reset_index()[overlap_df2.reset_index().columns[:39]]
clopidogrel_3['ADE_Classification'] = pd.Series(['No']*1907)
ade2_min['ADE_Classification'] = pd.Series(['Yes']*overlap_df2.shape[0])

clop_ade = pd.concat([clopidogrel_3, ade2_min], ignore_index=True).groupby(
    'person_id', as_index = False).apply(lambda x: x.sort_values(
    'ADE_Classification', ascending = False).head(1)).reset_index(drop = True)
```

In [335…

```
clop_ade.ADE_Classification = clop_ade.ADE_Classification.replace({'Yes': 1, 'No
```

In [337…

```
clop_ade['age'] = pd.DatetimeIndex(clop_ade.index_date).year - clop_ade.year_of_
```

In [339…

```
# follow_up_period,
# avg_days_bw_records,
# dispensed_count,
# avg_days_supply,
# race,
# ethnicity,
# sex_at_birth,
# ADE_Classification,
# age
```

In [341…

```
clop_ade.sex_at_birth = clop_ade.sex_at_birth.replace({'Male': 'Male', 'Female':
                                                       'Not male, not female, pr
                                                       'No matching concept': 'O
```

In [343…

```
clop_ade.race = clop_ade.race.replace({'White': 'White', 'Asian': 'Asian',
                                                       'Black or African America
                                                       'PMI: Skip': 'Unknown',
                                       'None of these': 'Other',
                                       'I prefer not to answer': 'Unknown',
                                       'None Indicated': 'Unknown',
                                       'More than one population': 'More than on
                                       'No matching concept': 'Unknown',
                                       'Another single population': 'Another sin
```

In [345…

```
clop_ade.ethnicity = clop_ade.ethnicity.replace({'Not Hispanic or Latino': 'Not
                                       'Hispanic or Latino': 'Hispanic or Latino
```

```
                                        'PMI: Skip': 'Unknown',
                                        'What Race Ethnicity: Race Ethnicity None
                                        'PMI: Prefer Not To Answer': 'Unknown'})
```

In [347…
```
X = clop_ade[['person_id', 'follow_up_period', 'avg_days_bw_records', 'dispensed
          'race', 'ethnicity', 'sex_at_birth', 'age']]
```

In [351…
```
survey_clop = clop_ade[['person_id', 'ADE_Classification', 'follow_up_period', '
          'race', 'ethnicity', 'sex_at_birth', 'age']]
```

In [352…
```
survey_clop = survey_clop.merge(survey_filled, on='person_id')
```

In [353…
```
X = X.merge(hlth_filled, on='person_id')
```

In [354…
```
hlth_filled.columns
```

Out[354…
```
Index(['person_id', 'Alcohol: Alcohol Participant',
       'Alcohol: Drink Frequency Past Year',
       'Cigar Smoking: Cigar Smoke Participant',
       'Education Level: Highest Grade',
       'Electronic Smoking: Electric Smoke Participant',
       'Hookah Smoking: Hookah Smoke Participant',
       'Insurance: Health Insurance', 'Overall Health: Average Fatigue 7 Days',
       'Overall Health: Average Pain 7 Days',
       'Overall Health: Emotional Problem 7 Days',
       'Overall Health: Everyday Activities', 'Overall Health: General Health',
       'Overall Health: General Mental Health',
       'Overall Health: General Physical Health',
       'Overall Health: General Quality', 'Overall Health: Organ Transplant',
       'Overall Health: Social Satisfaction',
       'Smokeless Tobacco: Smokeless Tobacco Participant',
       'Smoking: 100 Cigs Lifetime', 'Smoking: Serious Quit Attempt',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Blood Vessels',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Bone',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Cornea',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Heart',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Kidney',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Liver',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Lung',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Other Tissue',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Skin',
       'Organ Transplant: Organ Transplant Description = Organ Transplant Descri
ption: Valve',
       'Organ Transplant: Organ Transplant Description = PMI: Skip',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Cocaine Us
e',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Hallucinogen
```

```
s Use',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Inhalants Us
e',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Marijuana Us
e',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Methamphetam
ine Use',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: None Of Thes
e Drugs',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Other Specif
y',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Prescription
Opioids Use',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Prescription
Stimulants Use',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Sedatives Us
e',
       'Recreational Drug Use: Which Drugs Used = Which Drugs Used: Street Opioi
ds Use',
       'Recreational Drug Use: Which Drugs Used = Null'],
      dtype='object')
```

In [355…

```
clop_r = clop_ade[['person_id', 'ADE_Classification', 'follow_up_period', 'avg_d
          'race', 'ethnicity', 'sex_at_birth', 'age']]
```

In [356…

```
clop_r = clop_r.merge(hlth_filled, on='person_id')
```

In [357…

```
# define save_df that saves into csv
def save_df(df, name):
    # This code saves your dataframe into a csv file in a "data" folder in Googl

    # Replace df with THE NAME OF YOUR DATAFRAME
    my_dataframe = df

#     # Replace 'test.csv' with THE NAME of the file you're going to store in th
    destination_filename = f'{name}.csv'
    print(destination_filename)

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

    # save dataframe in a csv file in the same workspace as the notebook
    my_dataframe.to_csv(destination_filename, index=False)

#     # get the bucket name
    my_bucket = os.getenv('WORKSPACE_BUCKET')

#     # copy csv file to the bucket
    os.system(f"gsutil cp './{destination_filename}' '{my_bucket}/data/'")
    print(f'[INFO] {destination_filename} is successfully uploaded in your bucke
```

In [358…

```
save_df(clop_r, 'clop')
```

```
clop.csv
```

```
[INFO] clop.csv is successfully uploaded in your bucket.
```

In [359…

```
save_df(survey_clop, 'categorical_clop')
```

```
categorical_clop.csv
[INFO] categorical_clop.csv is successfully uploaded in your bucket.
```

In [360…

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

# This code lists objects in your Google Bucket

# Get the bucket name
my_bucket = os.getenv('WORKSPACE_BUCKET')

# List objects in the bucket
print(subprocess.check_output(f"gsutil ls -r {my_bucket}", shell=True).decode('u
```

```
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/:
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/categorical_clop.csv
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/clop.csv
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/clop_cond1
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/clop_cond2
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/clop_cond3
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/clop_cond4
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/clop_cond5
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/clop_cond_all1
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/clop_cond_all2
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/clop_cond_all3
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/clop_cond_all4
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/clop_cond_all5
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/data/clopidogrel

gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/:
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/Azathioprine.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/Clop - LR.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/Clop - ML.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/Clopidogrel - Data
Collection and Wrangling.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/Clopidogrel - Surv
ey Questions.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/Clopidogrel - Visu
alizations.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/Clopidogrel.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/Duplicate of Clop
- LR.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/Quinton.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/SPADE EDA 2.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/SPADE EDA.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/Survey Data.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/Warf-RF.ipynb
gs://fc-secure-0617591c-0448-460c-9405-9f5784007562/notebooks/Warfarin.ipynb
```

In [361…

```
del X['person_id']
```

In [362…

```
Y = clop_ade[['person_id', 'ADE_Classification']]
Y = Y.merge(hlth_filled, on = 'person_id')
```

In [363…

```
Y = Y['ADE_Classification']
```

In [364…
```
X = pd.get_dummies(X, columns=['race', 'ethnicity', 'sex_at_birth'])
```

In [365…
```
create_download_link(clop_r, "Download CLOP", "clop.csv")
```

Out[365… Download CLOP

In [ ]:

Logistic Regression

In [366…
```
#splits = [0.25, 0.33, 0.4]
thresh = [0,.1,.2,.3,.4,.5,.6,.7,.8,.9,1]
beta = [0.1,0.25,0.5,1,1.25,1.5,2,5]
fbeta = [[0]*len(thresh)]*len(beta)
fbeta_df = pd.DataFrame(columns = ['Beta', 'Thresh', 'fbeta'])
```

In [373…
```
Xtr, Xte, Ytr, Yte = train_test_split(X, Y, test_size=0.33, random_state=0)
#w = {1:(len(Ytr)-sum(Ytr)), 0:sum(Ytr)}
clf = LogisticRegression(solver='lbfgs', max_iter=1000, random_state=0, multi_cl
                         ).fit(Xtr, Ytr)
Yhat = clf.predict_proba(Xte)

Ypred = (Yhat[:,1] > 0.15).astype(int)

print(classification_report(Yte.to_numpy(), Ypred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.83      0.88       570
           1       0.22      0.47      0.30        59

    accuracy                           0.79       629
   macro avg       0.58      0.65      0.59       629
weighted avg       0.87      0.79      0.82       629

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:765: Co
nvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [374…
```
# sum((Yte.to_numpy()) == Ypred)/len(Ypred)
```

In [375…
```
# ## Representation of Confusion Matrix:
# [TN, FP]
# [FN, TP]
```

```
# ([[416, 158], thresh = 0.1
#     [ 23,  33]])
Ypred = (Yhat[:,1] > 0.15).astype(int)
confusion_matrix(Yte.to_numpy(), Ypred)
```

Out[375…
```
array([[471,  99],
       [ 31,  28]])
```

In [376…
```
fig, ax = plt.subplots(1, 1, figsize = (8, 6))
fpr, tpr, thresholds = metrics.roc_curve(Yte.to_numpy(), Yhat[:, 1])
x = np.linspace(0,1,100)
ax.plot(fpr, tpr, marker='.')
ax.plot(x, x)
ax.set_xlabel('False Positive Rate', fontsize = 15)
ax.set_ylabel('True Positive Rate', fontsize = 15)
ax.tick_params(labelsize = 13)
ax.set_title('ROC - Logistic Regression', fontsize = 17)
plt.show()
```

In [377…
```
metrics.auc(fpr, tpr)
```

Out[377…
```
0.7236990782039845
```

In [378…
```
X[['follow_up_period', 'avg_days_bw_records', 'dispensed_count', 'avg_days_suppl
```

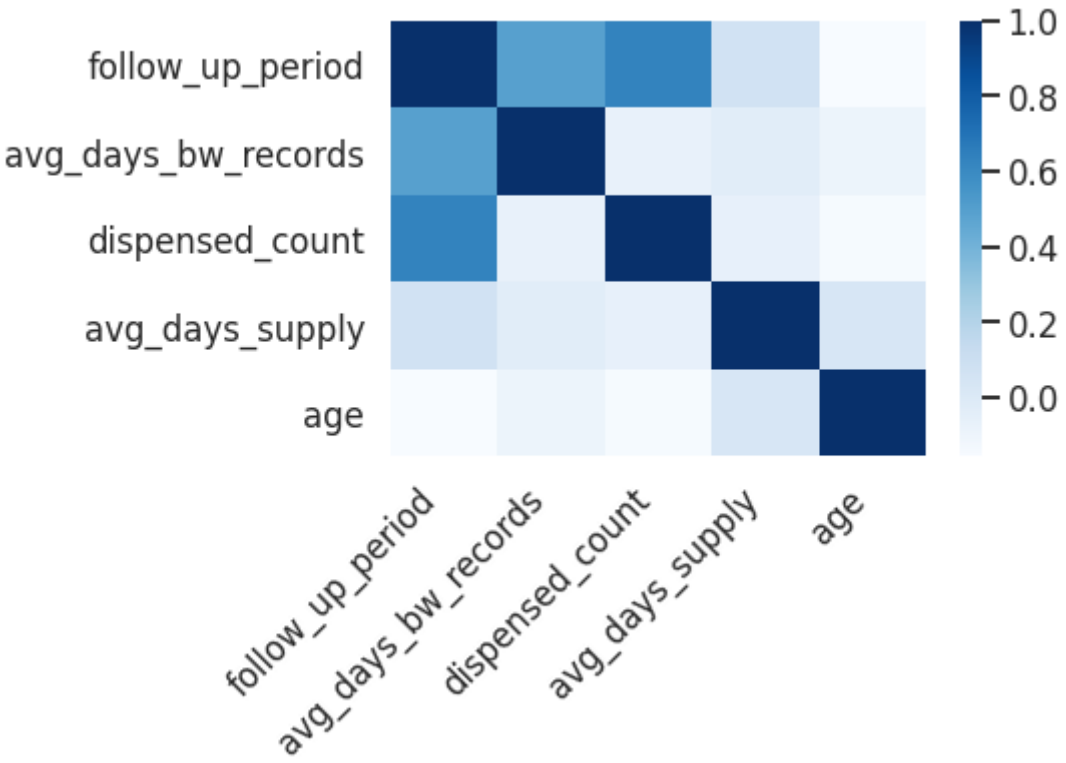
Out[378…

	follow_up_period	avg_days_bw_records	dispensed_count	avg_days_suppl
follow_up_period	1.000000	0.493421	0.631454	0.06682
avg_days_bw_records	0.493421	1.000000	-0.068269	-0.02561

	follow_up_period	avg_days_bw_records	dispensed_count	avg_days_suppl
dispensed_count	0.631454	-0.068269	1.000000	-0.05925
avg_days_supply	0.066825	-0.025619	-0.059255	1.00000
age	-0.156626	-0.091721	-0.144170	0.03258

In [379…

```
chart = sns.heatmap(X[['follow_up_period', 'avg_days_bw_records', 'dispensed_cou
            annot=False, cmap = "Blues")
chart.set_xticklabels(chart.get_xticklabels(), rotation=45, horizontalalignment=
plt.show()
```



In [380…

```
# import statsmodels.api as sm

# model = sm.Logit(Ytr, Xtr)
# result = model.fit(method='newton', maxiter=10000)
# result.predict(Xte)
```

In [381…

```
# result.summary2()
```

In [382…

```
# X.dtypes
```

Cross-Valid

In [383…

```
Xtr, Xte, Ytr, Yte = train_test_split(X, Y, test_size=0.33, random_state=0)
#w = {1:(len(Ytr)-sum(Ytr)), 0:sum(Ytr)}
clf = LogisticRegression(penalty='l1',solver='liblinear', max_iter=10000, random
                         )
```

```
scores = cross_val_score(clf, X, Y, scoring="roc_auc", cv=10)
```

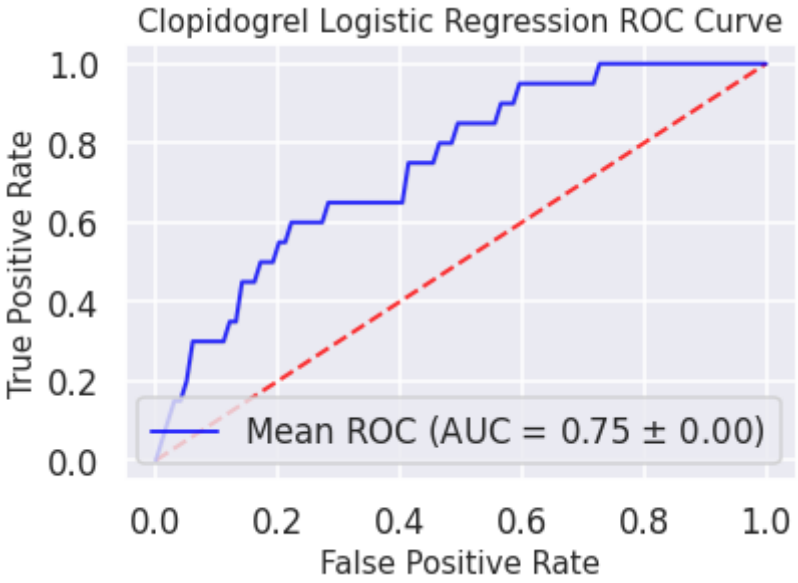
In [384…

```
scores.mean()
```

Out[384… 0.746626129306755

In [387…

```
clf = LogisticRegression(penalty='l1',solver='liblinear', max_iter=10000, random

cv = StratifiedKFold(n_splits=10)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, test) in enumerate(cv.split(X, Y)):
    clf.fit(X.loc[train], Y.loc[train])
#     viz = plot_roc_curve(clf, X.loc[test], Y.loc[test], alpha=0.3, lw=1, ax=ax
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),lw=2,

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2)
ax.legend(loc="lower right")
ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05])
ax.set_xlabel('False Positive Rate', size=15)
ax.set_ylabel('True Positive Rate', size=15)
ax.set_title("Clopidogrel Logistic Regression ROC Curve", size=15)
plt.show()
```



Random Forest

In [388…
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random
```

In [389…
```
regressor = RandomForestClassifier(n_estimators=20, random_state=1)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

regressor.fit(X_train, y_train)

# class_weight = dict({0:0.52, 1:4.27})
rdf = RandomForestClassifier(bootstrap=True,
            class_weight='balanced_subsample',
            criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=4, min_samples_split=10,
            min_weight_fraction_leaf=0.0, n_estimators=500,
            oob_score=False,
            random_state=175,
            verbose=0, warm_start=False)
rdf.fit(X_train, y_train)
y_pred = rdf.predict_proba(X_test)
```

In [390…
```
# [TN, FP]
# [FN, TP]



print(confusion_matrix(y_test,y_pred[:, 1] > .325))
```

```
[[334  99]
 [ 19  25]]
```

In [391…
```
scores = cross_val_score(rdf, X, Y, cv=10, scoring = 'roc_auc')
scores.mean()
```

Out[391… 0.7601213948183151

In [392…
```
# y_test["ADE_Classification"]
y_test_df = pd.DataFrame(y_test)
y_test_df.shape
```

Out[392… (477, 1)

In [393…
```
y_pred.shape
```

Out[393… (477, 2)

In [394…
```
print(accuracy_score(y_test,y_pred[:, 1] > .4))
```

0.8197064989517819

In [395…
```
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred[:, 1]>0.325)


metrics.auc(fpr, tpr)
```

Out[395… 0.6697722023934495

In [397…
```
X1 = X.reset_index(drop=True)
```

In [398…
```
Y1 = pd.DataFrame(Y)
Y1.reset_index(drop = True)
```

Out[398…

	ADE_Classification
0	0
1	0
2	0
3	0
4	0
...	...
1901	0
1902	0
1903	0
1904	0
1905	0

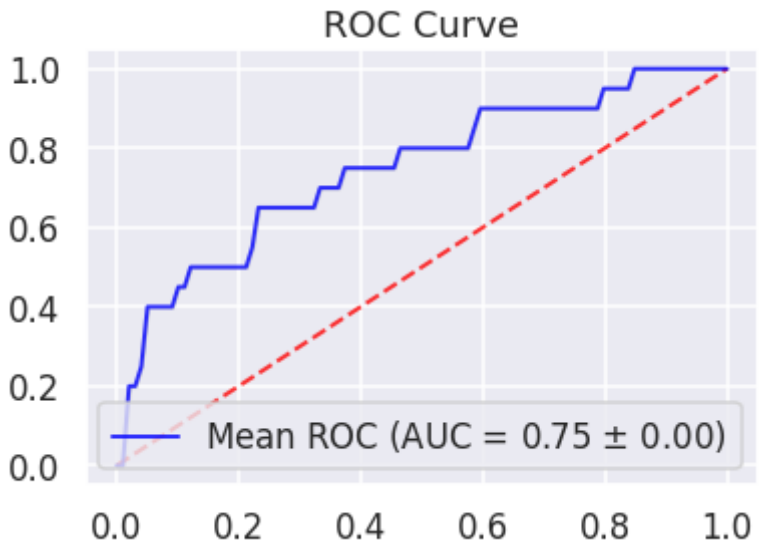
1906 rows × 1 columns

In [407…

```
cv = StratifiedKFold(n_splits=10)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, test) in enumerate(cv.split(X1, Y1)):
    rdf.fit(X1.loc[train], Y1.loc[train])
#     viz = plot_roc_curve(rdf, X1.loc[test], Y1.loc[test],

#                          alpha=0.3, lw=1, ax=ax)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),lw=2,

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2)
ax.legend(loc="lower right")
ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
       title="ROC Curve")
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
```



In [410…

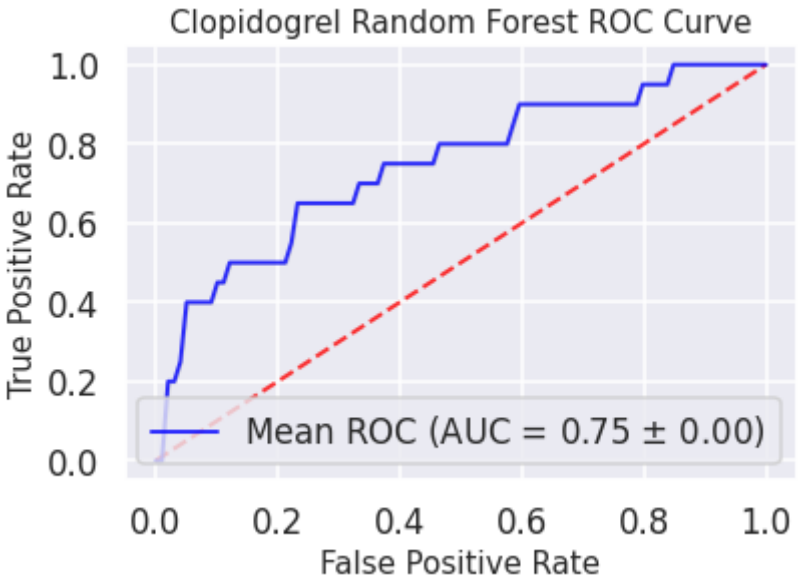
```
np.mean(aucs)
```

Out[410… 0.7470588235294118

In [409…

```
rdf = RandomForestClassifier(bootstrap=True,
            class_weight='balanced_subsample',
            criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=4, min_samples_split=10,
            min_weight_fraction_leaf=0.0, n_estimators=500,
            oob_score=False,
            random_state=175,
            verbose=0, warm_start=False)

cv = StratifiedKFold(n_splits=10)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, test) in enumerate(cv.split(X, Y)):
    rdf.fit(X.loc[train], Y.loc[train])
#     viz = plot_roc_curve(rdf, X1.loc[test], Y1.loc[test],alpha=0.3, lw=1, ax=a
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)
```

```
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),lw=2,

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2)
ax.legend(loc="lower right")
ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05])
ax.set_xlabel("False Positive Rate", size = 15)
ax.set_ylabel("True Positive Rate", size = 15)
ax.set_title("Clopidogrel Random Forest ROC Curve", size=15)
plt.show()
```



Noiseless Log Reg

In [402…
```
# follow up period, sex at birth, overal health general quality
```

In [403…
```
X_noiseless = X[['follow_up_period', 'sex_at_birth_Male', 'sex_at_birth_Female',
```

In [404…
```
clf = LogisticRegression(penalty='l1', solver='liblinear', max_iter=100, random_

scores = cross_val_score(clf, X_noiseless, Y, scoring="roc_auc", cv=10)
```

In [405…
```
scores.mean()
```

Out[405… 0.7448595857548928