Package Imports

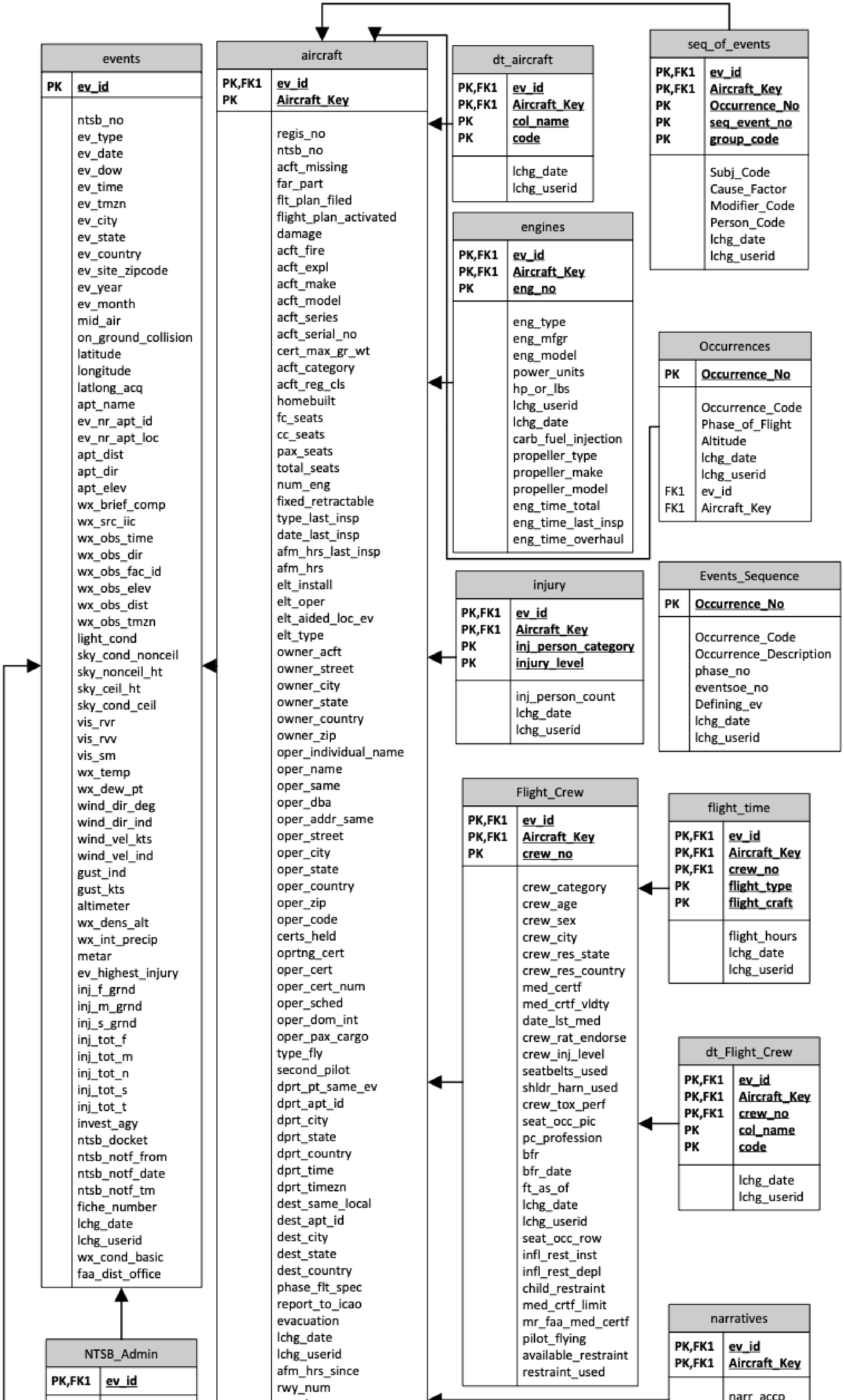
In [1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import mysql.connector as sql
import scipy.stats
from scipy.stats import chi2
import statsmodels.api
import category_encoders as ce
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

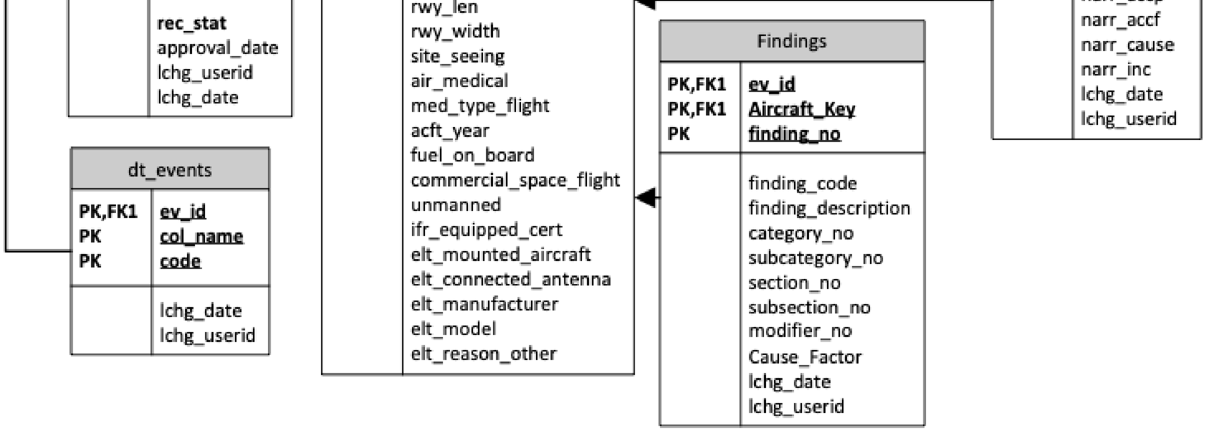
Dataset

This report utilizes the NTSB Census of US Civil Aviation Accidents. This dataset contains extensive information on the vast majority of aviation accidents and incidents since 2008. This includes everything from the date and location of the accident to the flight hours of each crew member and much, much more. This report uses data from April 23rd, 2024. The most recent data can be found at http://data.ntsb.gov/avdata.

Data Collection

Data was downloaded from http://data.ntsb.gov/avdata. The NTSB distributes the data in an MDB file that must be loaded into Microsoft Access. I then exported the tables that I would be using as Excel files, which I finally converted into CSV files that were loaded into a MySQL server. Once in the MySQL server, I dropped columns of entirely null values and columns that consisted of unimportant data for this report such as the NTSB employee that input the data.


events
PK ev_id
ntsb_no
ev_type
ev_date
ev_dow
ev_time
ev_tmzn
ev_city
ev_state
ev_country
ev_site_zipcode
ev_year
ev_month
mid_air
on_ground_collision
latitude
longitude
latlong_acq
apt_name
ev_nr_apt_id
ev_nr_apt_loc
apt_dist
apt_dir
apt_elev
wx_brief_comp
wx_src_iic
wx_obs_time
wx_obs_dir
wx_obs_fac_id
wx_obs_elev
wx_obs_dist
wx_obs_tmzn
light_cond
sky_cond_nonceil
sky_nonceil_ht
sky_ceil_ht
sky_cond_ceil
vis_rvr
vis_rvv
vis_sm
wx_temp
wx_dew_pt
wind_dir_deg
wind_dir_ind
wind_vel_kts
wind_vel_ind
gust_ind
gust_kts
altimeter
wx_dens_alt
wx_int_precip
metar
ev_highest_injury
inj_f_grnd
inj_m_grnd
inj_s_grnd
inj_tot_f
inj_tot_m
inj_tot_n
inj_tot_s
inj_tot_t
invest_agy
ntsb_docket
ntsb_notf_from
ntsb_notf_date
ntsb_notf_tm
fiche_number
lchg_date
lchg_userid
wx_cond_basic
faa_dist_office
NTSB_Admin
PK,FK1 ev_id
aircraft
PK,FK1 ev_id
PK Aircraft_Key
regis_no
ntsb_no
acft_missing
far_part
flt_plan_filed
flight_plan_activated
damage
acft_fire
acft_expl
acft_make
acft_model
acft_series
acft_serial_no
cert_max_gr_wt
acft_category
acft_reg_cls
homebuilt
fc_seats
cc_seats
pax_seats
total_seats
num_eng
fixed_retractable
type_last_insp
date_last_insp
afm_hrs_last_insp
afm_hrs
elt_install
elt_oper
elt_aided_loc_ev
elt_type
owner_acft
owner_street
owner_city
owner_state
owner_country
owner_zip
oper_individual_name
oper_name
oper_same
oper_dba
oper_addr_same
oper_street
oper_city
oper_state
oper_country
oper_zip
oper_code
certs_held
oprtng_cert
oper_cert
oper_cert_num
oper_sched
oper_dom_int
oper_pax_cargo
type_fly
second_pilot
dprt_pt_same_ev
dprt_apt_id
dprt_city
dprt_state
dprt_country
dprt_time
dprt_timezn
dest_same_local
dest_apt_id
dest_city
dest_state
dest_country
phase_flt_spec
report_to_icao
evacuation
lchg_date
lchg_userid
afm_hrs_since
rwy_num
dt_aircraft
PK,FK1 ev_id
PK,FK1 Aircraft_Key
PK col_name
PK code
lchg_date
lchg_userid
seq_of_events
PK,FK1 ev_id
PK,FK1 Aircraft_Key
PK Occurrence_No
PK seq_event_no
PK group_code
Subj_Code
Cause_Factor
Modifier_Code
Person_Code
lchg_date
lchg_userid
engines
PK,FK1 ev_id
PK,FK1 Aircraft_Key
PK eng_no
eng_type
eng_mfgr
eng_model
power_units
hp_or_lbs
lchg_userid
lchg_date
carb_fuel_injection
propeller_type
propeller_make
propeller_model
eng_time_total
eng_time_last_insp
eng_time_overhaul
Occurrences
PK Occurrence_No
Occurrence_Code
Phase_of_Flight
Altitude
lchg_date
lchg_userid
FK1 ev_id
FK1 Aircraft_Key
injury
PK,FK1 ev_id
PK,FK1 Aircraft_Key
PK inj_person_category
PK injury_level
inj_person_count
lchg_date
lchg_userid
Events_Sequence
PK Occurrence_No
Occurrence_Code
Occurrence_Description
phase_no
eventsoe_no
Defining_ev
lchg_date
lchg_userid
Flight_Crew
PK,FK1 ev_id
PK,FK1 Aircraft_Key
PK crew_no
crew_category
crew_age
crew_sex
crew_city
crew_res_state
crew_res_country
med_certf
med_crtf_vldty
date_lst_med
crew_rat_endorse
crew_inj_level
seatbelts_used
shldr_harn_used
crew_tox_perf
seat_occ_pic
pc_profession
bfr
bfr_date
ft_as_of
lchg_date
lchg_userid
seat_occ_row
infl_rest_inst
infl_rest_depl
child_restraint
med_crtf_limit
mr_faa_med_certf
pilot_flying
available_restraint
restraint_used
flight_time
PK,FK1 ev_id
PK,FK1 Aircraft_Key
PK,FK1 crew_no
PK flight_type
PK flight_craft
flight_hours
lchg_date
lchg_userid
dt_Flight_Crew
PK,FK1 ev_id
PK,FK1 Aircraft_Key
PK,FK1 crew_no
PK col_name
PK code
lchg_date
lchg_userid
narratives
PK,FK1 ev_id
PK,FK1 Aircraft_Key
narr_accp



Background Information

This project is inspired by the work of Xiaoge Zhang, Prabhakar Srinivasan, and Sankaran Mahadevan in their paper "Sequential deep learning from NTSB reports for aviation safety prognosis" (https://doi.org/10.1016/j.ssci.2021.105390). During my reading of their report, I found their problem statement had the order of operations in reverse. In their paper, they began with the textual "Sequence of Events" and predicted the damage done to the aircraft, if fatalities occurred, and whether the event was categorized as an accident or incident. I believe that is backwards, as in an investigation, the sequence of events would be written last after all facts are known, and the predicted information would be some of the first information to be known.

Problem Statement

Accordingly, their research inspired two main questions that will be addressed in this report:

1. Can surface-level features (eg. data that can be immediately gathered at the beginning of an investigation) be used to predict the cause of the accident?

2. What does this tell us about the causes of accidents?

I hope that answering these two questions will add to the wealth of information surrounding the causality of aviation accidents and will help to reduce the amount of avoidable aviation accidents.

SQL Interactions

After my cursory cleaning in MySQL, I then imported the data into this Jupyter Notebook for further preparation and analysis.

Connecting to SQL Database

In [2]:
```
sql = sql.connect(
  host="localhost",
  user="root",
  password="hire72locate",
  database="ntsb"
)
```

In [3]:
```
cursor = sql.cursor(buffered=True)
```

In [4]:
```
cursor.execute("show tables")
for x in cursor:
    print(x)
```

```
('aircraft',)
('engines',)
('events',)
('Findings',)
('Flight_Crew',)
```

Pulling Data

In [5]:
```
query = "select * from aircraft"
aircraft = pd.read_sql(query, sql)

query = "select * from engines"
engines = pd.read_sql(query, sql)

query = "select * from events"
events = pd.read_sql(query, sql)

query = "select * from Findings"
Findings = pd.read_sql(query, sql)

query = "select * from Flight_Crew"
Flight_Crew = pd.read_sql(query, sql)
```

```
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/io/sql.py:761: Us
erWarning: pandas only support SQLAlchemy connectable(engine/connection) ordatabase stri
ng URI or sqlite3 DBAPI2 connectionother DBAPI2 objects are not tested, please consider
using SQLAlchemy
  warnings.warn(
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/io/sql.py:761: Us
erWarning: pandas only support SQLAlchemy connectable(engine/connection) ordatabase stri
ng URI or sqlite3 DBAPI2 connectionother DBAPI2 objects are not tested, please consider
using SQLAlchemy
  warnings.warn(
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/io/sql.py:761: Us
erWarning: pandas only support SQLAlchemy connectable(engine/connection) ordatabase stri
ng URI or sqlite3 DBAPI2 connectionother DBAPI2 objects are not tested, please consider
using SQLAlchemy
  warnings.warn(
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/io/sql.py:761: Us
erWarning: pandas only support SQLAlchemy connectable(engine/connection) ordatabase stri
ng URI or sqlite3 DBAPI2 connectionother DBAPI2 objects are not tested, please consider
using SQLAlchemy
  warnings.warn(
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/io/sql.py:761: Us
erWarning: pandas only support SQLAlchemy connectable(engine/connection) ordatabase stri
ng URI or sqlite3 DBAPI2 connectionother DBAPI2 objects are not tested, please consider
using SQLAlchemy
  warnings.warn(
```

Cleaning

In this cleaning stage, I started with general filters on the events table, then cleaned each column of the table. I then merged the child tables onto the cleaned events table to filter away data on the events that were dropped from the events table. I then went through each column of the child tables cleaning each. Finally, I began early EDA by showing details about each column. Exact steps are shown below.

events

In [6]:
```
events.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26988 entries, 0 to 26987
Data columns (total 62 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ev_id                26988 non-null  object 
 1   ntsb_no              26988 non-null  object 
 2   ev_type              26988 non-null  object 
 3   ev_date              26988 non-null  object 
 4   ev_dow               26988 non-null  object 
 5   ev_time              26988 non-null  int64  
 6   ev_tmzn              26988 non-null  object 
 7   ev_city              26988 non-null  object 
 8   ev_state             26988 non-null  object 
 9   ev_country           26988 non-null  object 
 10  ev_site_zipcode      26988 non-null  int64  
 11  ev_year              26988 non-null  int64  
 12  ev_month             26988 non-null  int64  
 13  mid_air              26988 non-null  object 
 14  on_ground_collision  26988 non-null  object 
 15  latitude             26988 non-null  object 
 16  longitude            26988 non-null  object 
 17  latlong_acq          26988 non-null  object 
 18  apt_name             26988 non-null  object 
 19  ev_nr_apt_id         26988 non-null  object 
 20  ev_nr_apt_loc        26988 non-null  object 
 21  apt_dist             26988 non-null  float64
 22  apt_dir              26988 non-null  int64  
 23  apt_elev             26988 non-null  int64  
 24  wx_brief_comp        26988 non-null  int64  
 25  wx_src_iic           26988 non-null  object 
 26  wx_obs_time          26988 non-null  int64  
 27  wx_obs_dir           26988 non-null  int64  
 28  wx_obs_fac_id        26988 non-null  object 
 29  wx_obs_elev          26988 non-null  int64  
 30  wx_obs_dist          26988 non-null  int64  
 31  wx_obs_tmzn          26988 non-null  object 
 32  light_cond           26988 non-null  object 
 33  sky_cond_nonceil     26988 non-null  object 
 34  sky_nonceil_ht       26988 non-null  int64  
 35  sky_ceil_ht          26988 non-null  int64  
 36  sky_cond_ceil        26988 non-null  object 
 37  vis_rvr              26988 non-null  int64  
 38  vis_sm               26988 non-null  float64
 39  wx_temp              26988 non-null  int64  
 40  wx_dew_pt            26988 non-null  int64  
 41  wind_dir_deg         26988 non-null  int64  
 42  wind_dir_ind         26988 non-null  object 
 43  wind_vel_kts         26988 non-null  int64  
 44  wind_vel_ind         26988 non-null  object 
 45  gust_ind             26988 non-null  object 
 46  gust_kts             26988 non-null  int64  
 47  altimeter            26988 non-null  float64
 48  metar                26988 non-null  object 
 49  ev_highest_injury    26988 non-null  object 
```

```
 50  inj_f_grnd           26988 non-null  int64
 51  inj_m_grnd           26988 non-null  int64
 52  inj_s_grnd           26988 non-null  int64
 53  inj_tot_f            26988 non-null  int64
 54  inj_tot_m            26988 non-null  int64
 55  inj_tot_n            26988 non-null  int64
 56  inj_tot_s            26988 non-null  int64
 57  inj_tot_t            26988 non-null  int64
 58  invest_agy           26988 non-null  object
 59  wx_cond_basic        26988 non-null  object
 60  dec_latitude         26988 non-null  float64
 61  dec_longitude        26987 non-null  float64
dtypes: float64(5), int64(27), object(30)
memory usage: 12.8+ MB
```

In [7]:
```
events = events.replace(r'^\s*$', np.nan, regex=True)
events.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26988 entries, 0 to 26987
Data columns (total 62 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ev_id                26988 non-null  object
 1   ntsb_no              26988 non-null  object
 2   ev_type              26988 non-null  object
 3   ev_date              26988 non-null  object
 4   ev_dow               26988 non-null  object
 5   ev_time              26988 non-null  int64
 6   ev_tmzn              26049 non-null  object
 7   ev_city              26972 non-null  object
 8   ev_state             22793 non-null  object
 9   ev_country           26986 non-null  object
 10  ev_site_zipcode      26988 non-null  int64
 11  ev_year              26988 non-null  int64
 12  ev_month             26988 non-null  int64
 13  mid_air              414 non-null    object
 14  on_ground_collision  414 non-null    object
 15  latitude             24361 non-null  object
 16  longitude            24362 non-null  object
 17  latlong_acq          20267 non-null  object
 18  apt_name             16824 non-null  object
 19  ev_nr_apt_id         16985 non-null  object
 20  ev_nr_apt_loc        23155 non-null  object
 21  apt_dist             26988 non-null  float64
 22  apt_dir              26988 non-null  int64
 23  apt_elev             26988 non-null  int64
 24  wx_brief_comp        26988 non-null  int64
 25  wx_src_iic           21406 non-null  object
 26  wx_obs_time          26988 non-null  int64
 27  wx_obs_dir           26988 non-null  int64
 28  wx_obs_fac_id        20492 non-null  object
 29  wx_obs_elev          26988 non-null  int64
 30  wx_obs_dist          26988 non-null  int64
 31  wx_obs_tmzn          15979 non-null  object
 32  light_cond           22489 non-null  object
 33  sky_cond_nonceil     18700 non-null  object
 34  sky_nonceil_ht       26988 non-null  int64
 35  sky_ceil_ht          26988 non-null  int64
 36  sky_cond_ceil        19737 non-null  object
 37  vis_rvr              26988 non-null  int64
 38  vis_sm               26988 non-null  float64
 39  wx_temp              26988 non-null  int64
 40  wx_dew_pt            26988 non-null  int64
 41  wind_dir_deg         26988 non-null  int64
```

```
 42  wind_dir_ind         26896 non-null  object
 43  wind_vel_kts         26988 non-null  int64
 44  wind_vel_ind         26896 non-null  object
 45  gust_ind             26896 non-null  object
 46  gust_kts             26988 non-null  int64
 47  altimeter            26988 non-null  float64
 48  metar                3518 non-null   object
 49  ev_highest_injury    25825 non-null  object
 50  inj_f_grnd           26988 non-null  int64
 51  inj_m_grnd           26988 non-null  int64
 52  inj_s_grnd           26988 non-null  int64
 53  inj_tot_f            26988 non-null  int64
 54  inj_tot_m            26988 non-null  int64
 55  inj_tot_n            26988 non-null  int64
 56  inj_tot_s            26988 non-null  int64
 57  inj_tot_t            26988 non-null  int64
 58  invest_agy           26759 non-null  object
 59  wx_cond_basic        22813 non-null  object
 60  dec_latitude         26988 non-null  float64
 61  dec_longitude        26987 non-null  float64
dtypes: float64(5), int64(27), object(30)
memory usage: 12.8+ MB
```

Limiting scope to USA

In [8]:
```
events = events[events['ev_country']=='USA']
```

In [9]:
```
events.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 22334 entries, 0 to 26987
Data columns (total 62 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   ev_id               22334 non-null  object
 1   ntsb_no             22334 non-null  object
 2   ev_type             22334 non-null  object
 3   ev_date             22334 non-null  object
 4   ev_dow              22334 non-null  object
 5   ev_time             22334 non-null  int64
 6   ev_tmzn             22246 non-null  object
 7   ev_city             22327 non-null  object
 8   ev_state            22281 non-null  object
 9   ev_country          22334 non-null  object
 10  ev_site_zipcode     22334 non-null  int64
 11  ev_year             22334 non-null  int64
 12  ev_month            22334 non-null  int64
 13  mid_air             355 non-null    object
 14  on_ground_collision 355 non-null    object
 15  latitude            22290 non-null  object
 16  longitude           22290 non-null  object
 17  latlong_acq         18794 non-null  object
 18  apt_name            15795 non-null  object
 19  ev_nr_apt_id        15942 non-null  object
 20  ev_nr_apt_loc       21542 non-null  object
 21  apt_dist            22334 non-null  float64
 22  apt_dir             22334 non-null  int64
 23  apt_elev            22334 non-null  int64
 24  wx_brief_comp       22334 non-null  int64
 25  wx_src_iic          21057 non-null  object
 26  wx_obs_time         22334 non-null  int64
 27  wx_obs_dir          22334 non-null  int64
 28  wx_obs_fac_id       20178 non-null  object
```

```
 29  wx_obs_elev          22334 non-null  int64
 30  wx_obs_dist          22334 non-null  int64
 31  wx_obs_tmzn          15756 non-null  object
 32  light_cond           22078 non-null  object
 33  sky_cond_nonceil     18432 non-null  object
 34  sky_nonceil_ht       22334 non-null  int64
 35  sky_ceil_ht          22334 non-null  int64
 36  sky_cond_ceil        19472 non-null  object
 37  vis_rvr              22334 non-null  int64
 38  vis_sm               22334 non-null  float64
 39  wx_temp              22334 non-null  int64
 40  wx_dew_pt            22334 non-null  int64
 41  wind_dir_deg         22334 non-null  int64
 42  wind_dir_ind         22307 non-null  object
 43  wind_vel_kts         22334 non-null  int64
 44  wind_vel_ind         22307 non-null  object
 45  gust_ind             22307 non-null  object
 46  gust_kts             22334 non-null  int64
 47  altimeter            22334 non-null  float64
 48  metar                3495 non-null   object
 49  ev_highest_injury    22220 non-null  object
 50  inj_f_grnd           22334 non-null  int64
 51  inj_m_grnd           22334 non-null  int64
 52  inj_s_grnd           22334 non-null  int64
 53  inj_tot_f            22334 non-null  int64
 54  inj_tot_m            22334 non-null  int64
 55  inj_tot_n            22334 non-null  int64
 56  inj_tot_s            22334 non-null  int64
 57  inj_tot_t            22334 non-null  int64
 58  invest_agy           22118 non-null  object
 59  wx_cond_basic        22103 non-null  object
 60  dec_latitude         22334 non-null  float64
 61  dec_longitude        22333 non-null  float64
dtypes: float64(5), int64(27), object(30)
memory usage: 10.7+ MB
```

Dropping unusable columns

In [10]:
```
#events[events['wind_vel_kts']==0].wind_vel_kts.count()/events.wind_vel_kts.count()
#events['ev_site_zipcode'].value_counts()
#events[events['wind_vel_kts']<25].wind_vel_kts.hist()
#events['wind_vel_kts'].hist()
```

In [11]:
```
events = events.drop(columns=['ntsb_no', 'ev_date', 'ev_time', 'ev_tmzn', 'ev_city', 'ev
```

In [12]:
```
events.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 22334 entries, 0 to 26987
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ev_id             22334 non-null  object
 1   ev_type           22334 non-null  object
 2   ev_dow            22334 non-null  object
 3   ev_state          22281 non-null  object
 4   ev_month          22334 non-null  int64
 5   ev_nr_apt_loc     21542 non-null  object
 6   wx_src_iic        21057 non-null  object
 7   wx_obs_time       22334 non-null  int64
 8   light_cond        22078 non-null  object
 9   vis_sm            22334 non-null  float64
 10  wx_temp           22334 non-null  int64
```

```
 11  wind_vel_kts        22334 non-null  int64
 12  ev_highest_injury   22220 non-null  object
 13  inj_f_grnd          22334 non-null  int64
 14  inj_m_grnd          22334 non-null  int64
 15  inj_s_grnd          22334 non-null  int64
 16  inj_tot_f           22334 non-null  int64
 17  inj_tot_m           22334 non-null  int64
 18  inj_tot_n           22334 non-null  int64
 19  inj_tot_s           22334 non-null  int64
 20  inj_tot_t           22334 non-null  int64
 21  wx_cond_basic       22103 non-null  object
dtypes: float64(1), int64(12), object(9)
memory usage: 3.9+ MB
```

Checking Quality of Remaining Columns

ev_id is PK

-ev_type is nominal categorical; imbalanced but usable

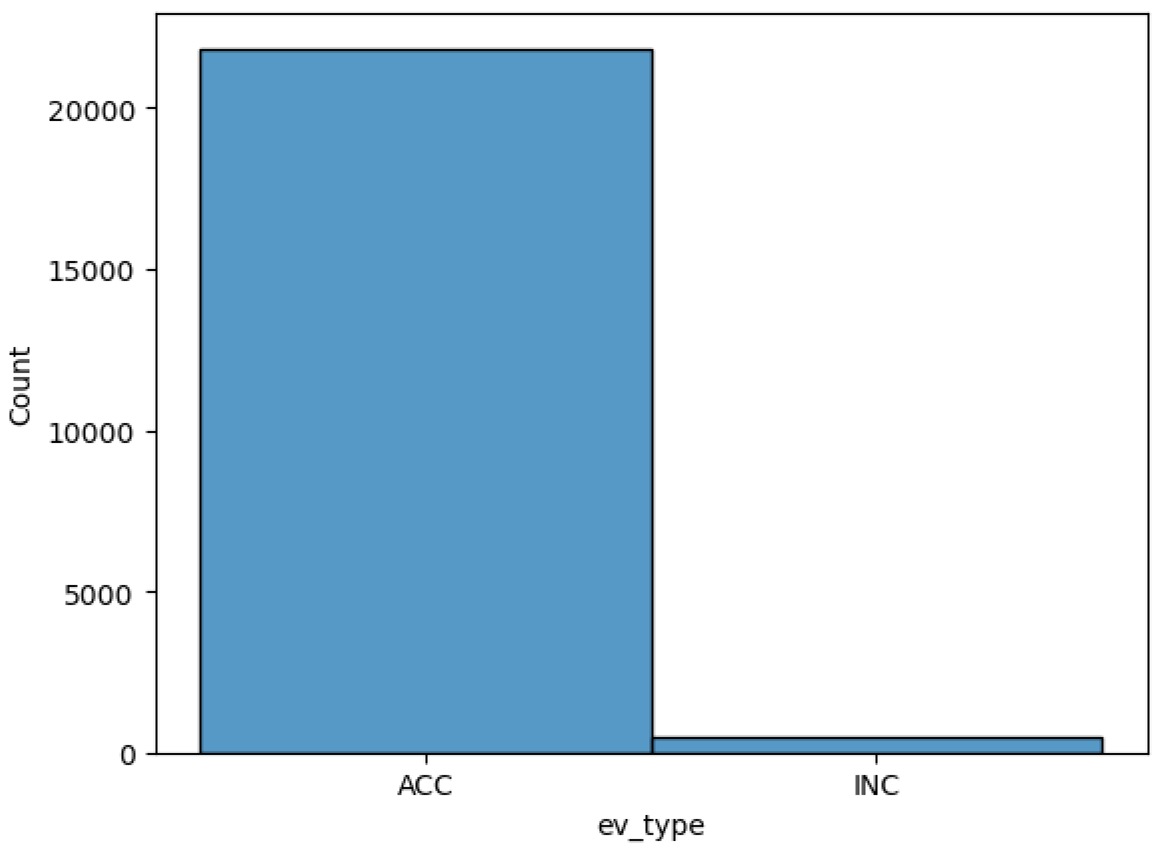
In [13]:

```
print("ACC: " + str(events[events['ev_type']=='ACC'].ev_type.count()))
print("INC: " + str(events[events['ev_type']=='INC'].ev_type.count()))
sns.histplot(data=events['ev_type'])
```

```
ACC: 21847
INC: 487
```

Out[13]:

```
<Axes: xlabel='ev_type', ylabel='Count'>
```



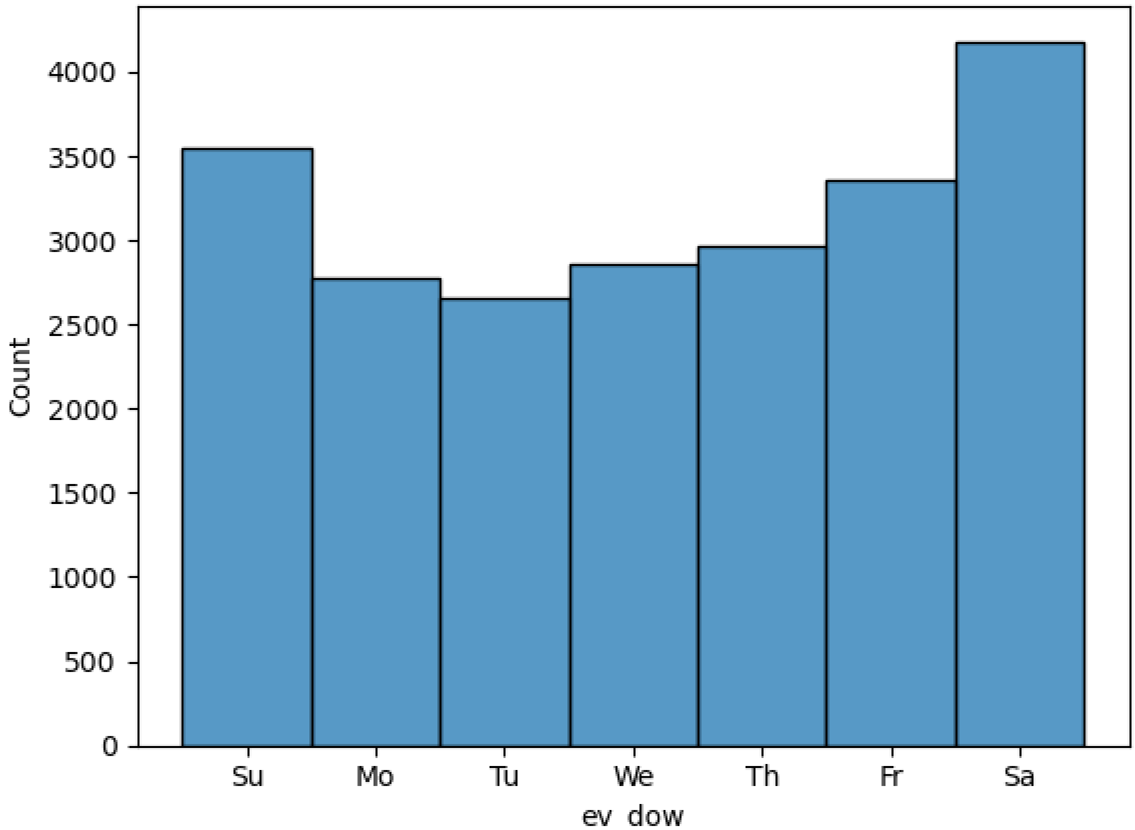
-ev_dow will be treated as nominal categorical

In [14]:

```
events['ev_dow'] = pd.Categorical(events['ev_dow'], ['Su','Mo','Tu','We','Th','Fr','Sa']
sns.histplot(data=events['ev_dow'])
```

Out[14]:

```
<Axes: xlabel='ev_dow', ylabel='Count'>
```



ev_state is nominal categorical

Dropped any non FAA recognized abbreviations (https://www.faa.gov/air_traffic/publications/atpubs/cnt_html/appendix_a.html)

In [15]:
```
notastate = ['OF', 'CB', 'GM', 'PO', 'AO']
for one in notastate:
    events.drop(events[events['ev_state'] == one].index, inplace=True)
```

In [16]:
```
events['ev_state'].value_counts()
```

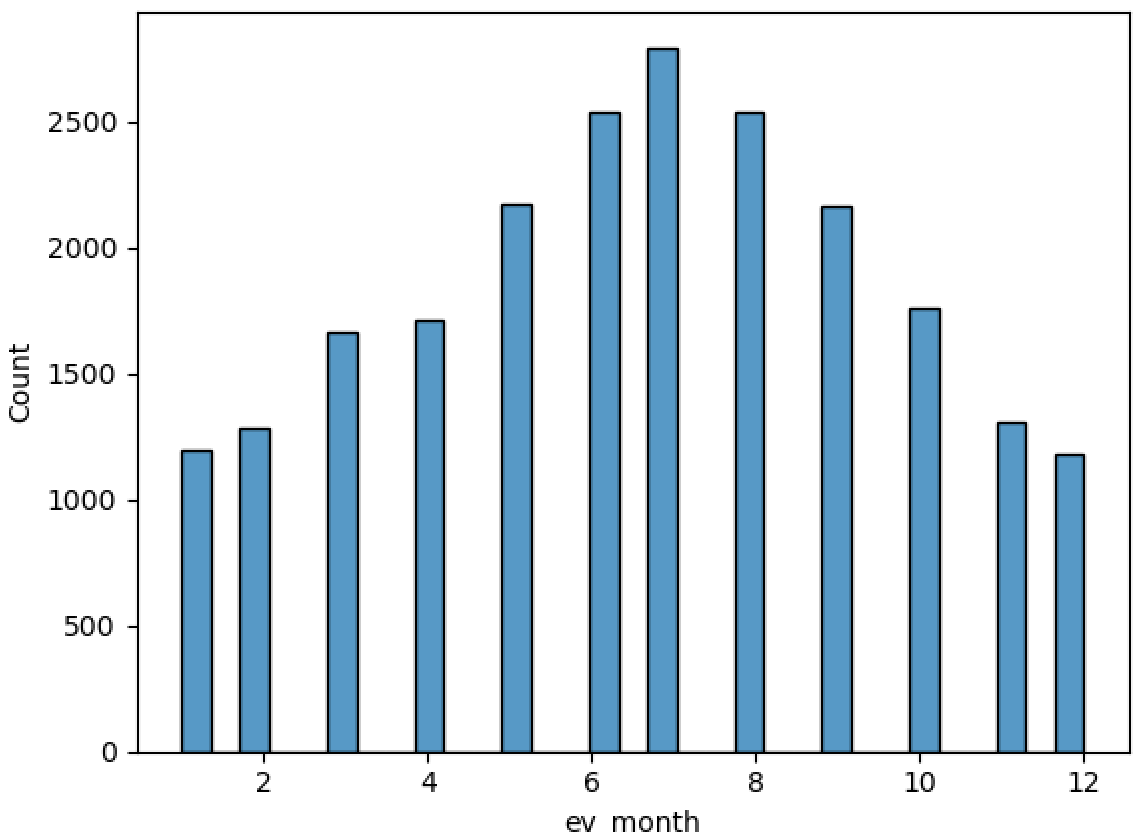
Out[16]:
```
CA    2084
TX    1863
FL    1641
AK    1455
AZ     878
CO     704
WA     699
GA     630
NC     521
ID     506
IL     505
NY     497
OR     488
OH     478
PA     455
UT     450
MI     445
WI     415
VA     404
NV     398
MO     396
MN     396
TN     372
NM     351
```

```
IN      351
AR      335
LA      318
OK      315
KS      300
SC      293
AL      288
MT      287
NJ      272
IA      253
MD      236
WY      194
KY      189
NE      188
MA      187
MS      187
HI      149
SD      133
ME      133
ND      123
CT      112
WV       91
NH       91
PR       69
VT       56
RI       29
DE       27
DC        2
Name: ev_state, dtype: int64
```

ev_month is nominal categorical

In [17]:
```python
sns.histplot(events['ev_month'])
```

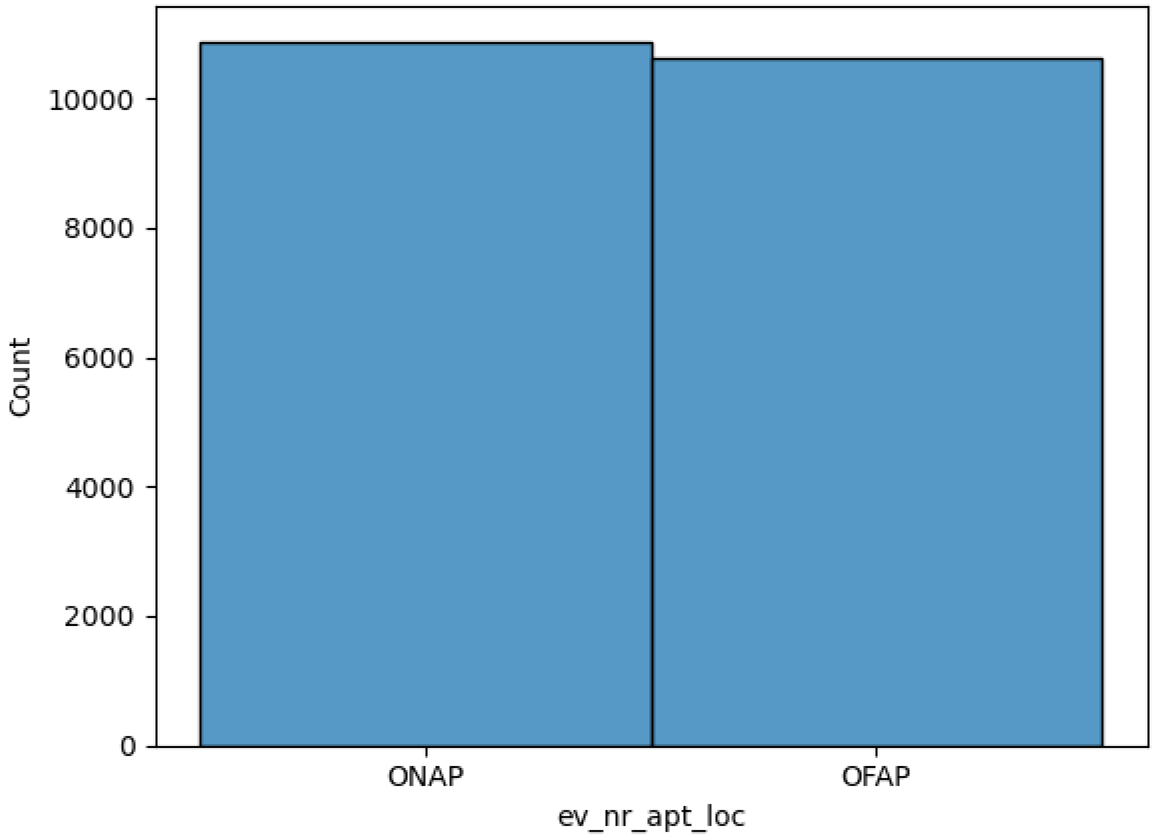
Out[17]:
```
<Axes: xlabel='ev_month', ylabel='Count'>
```



-ev_nr_apt_loc is nominal categorical

In [18]:
```
sns.histplot(events['ev_nr_apt_loc'])
```

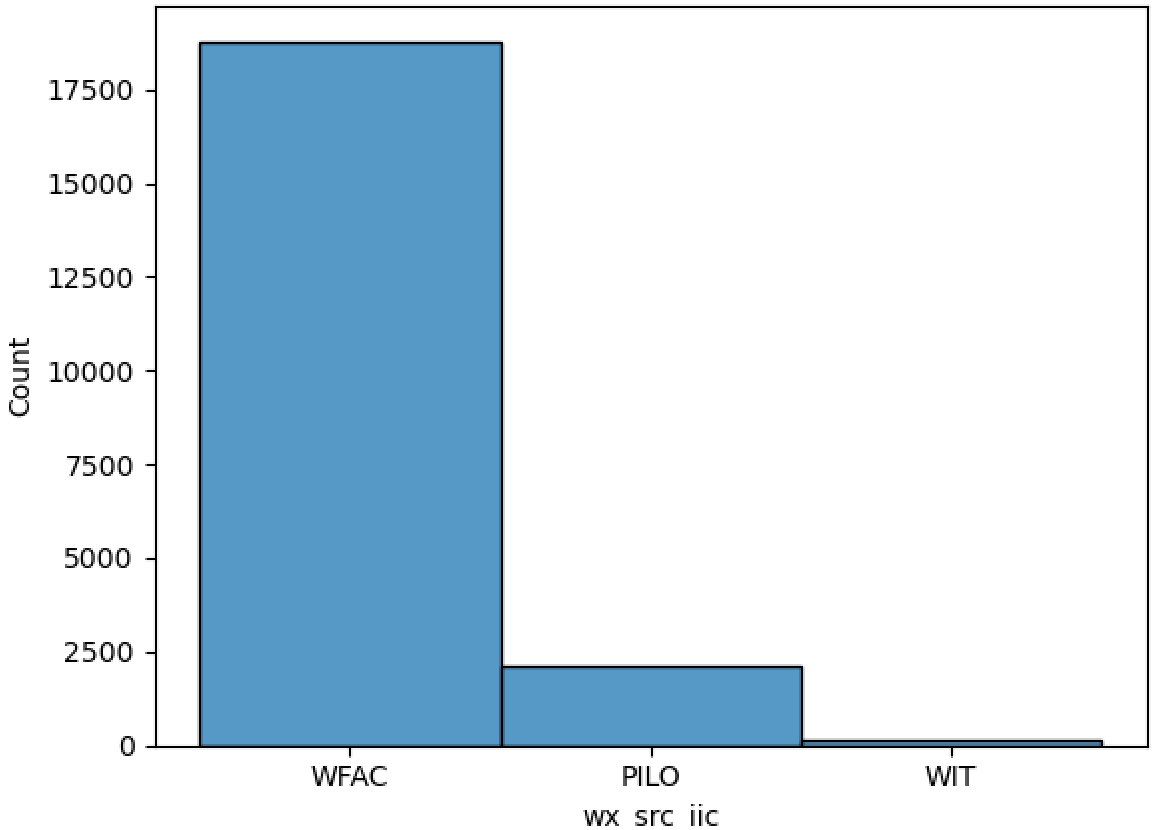
Out[18]:
```
<Axes: xlabel='ev_nr_apt_loc', ylabel='Count'>
```



wx_src_iic is nominal categorical

In [19]:
```
sns.histplot(events['wx_src_iic'])
```

Out[19]:
```
<Axes: xlabel='wx_src_iic', ylabel='Count'>
```



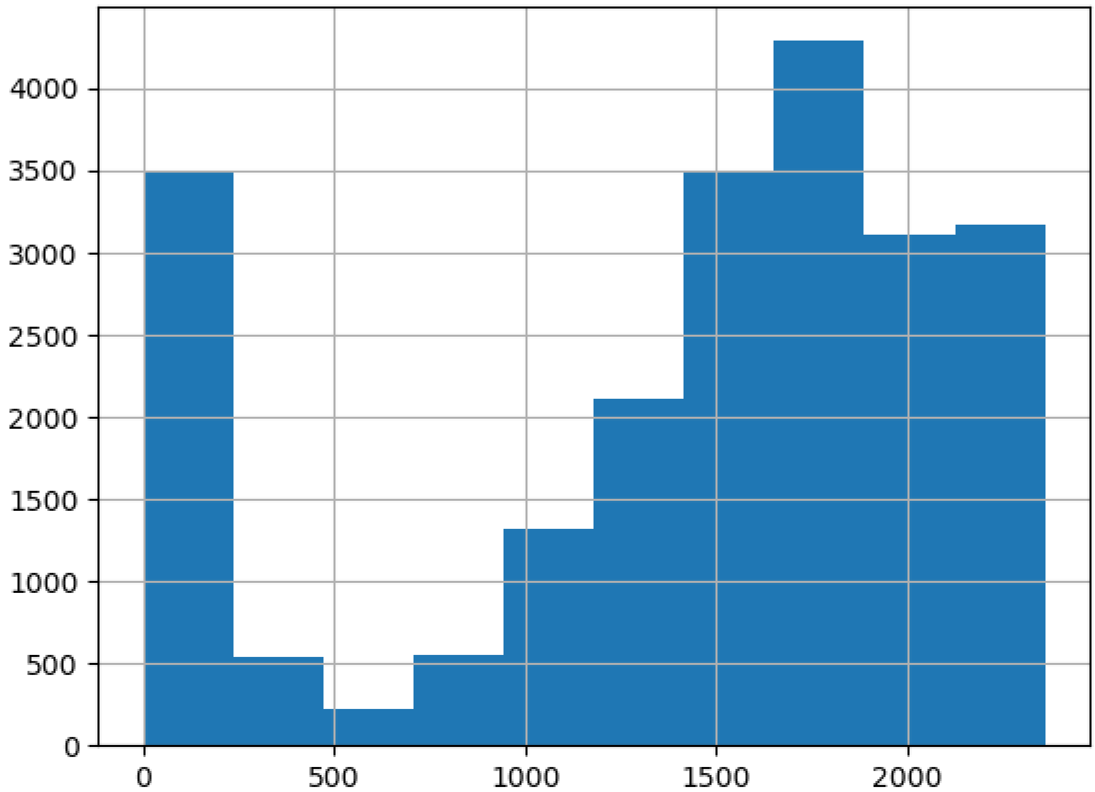
In [20]:
```
events.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 22292 entries, 0 to 26987
Data columns (total 22 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   ev_id              22292 non-null  object
 1   ev_type            22292 non-null  object
 2   ev_dow             22292 non-null  category
 3   ev_state           22239 non-null  object
 4   ev_month           22292 non-null  int64
 5   ev_nr_apt_loc      21505 non-null  object
 6   wx_src_iic         21022 non-null  object
 7   wx_obs_time        22292 non-null  int64
 8   light_cond         22040 non-null  object
 9   vis_sm             22292 non-null  float64
 10  wx_temp            22292 non-null  int64
 11  wind_vel_kts       22292 non-null  int64
 12  ev_highest_injury  22179 non-null  object
 13  inj_f_grnd         22292 non-null  int64
 14  inj_m_grnd         22292 non-null  int64
 15  inj_s_grnd         22292 non-null  int64
 16  inj_tot_f          22292 non-null  int64
 17  inj_tot_m          22292 non-null  int64
 18  inj_tot_n          22292 non-null  int64
 19  inj_tot_s          22292 non-null  int64
 20  inj_tot_t          22292 non-null  int64
 21  wx_cond_basic      22065 non-null  object
dtypes: category(1), float64(1), int64(12), object(8)
memory usage: 3.8+ MB
```

wx_obs_time is continuous

In [21]:
```
events.wx_obs_time.hist()
```

Out[21]: <Axes: >

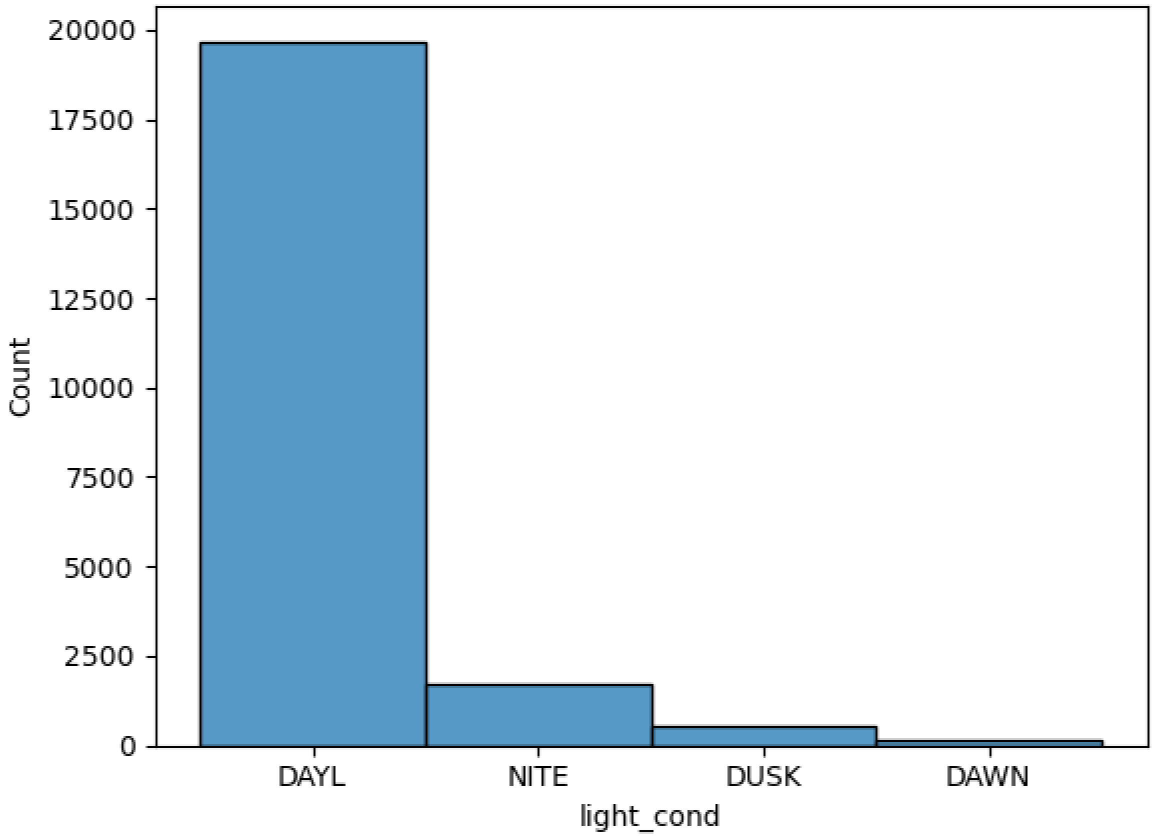


-light_cond is nomical categorical

Consolidated categories

In [22]:
```
events['light_cond'].replace('NDRK','NITE', inplace=True)
events['light_cond'].replace('NR','NITE', inplace=True)
events['light_cond'].replace('NBRT','NITE', inplace=True)

sns.histplot(events['light_cond'])
```

Out[22]: <Axes: xlabel='light_cond', ylabel='Count'>



vis_sm is continuous

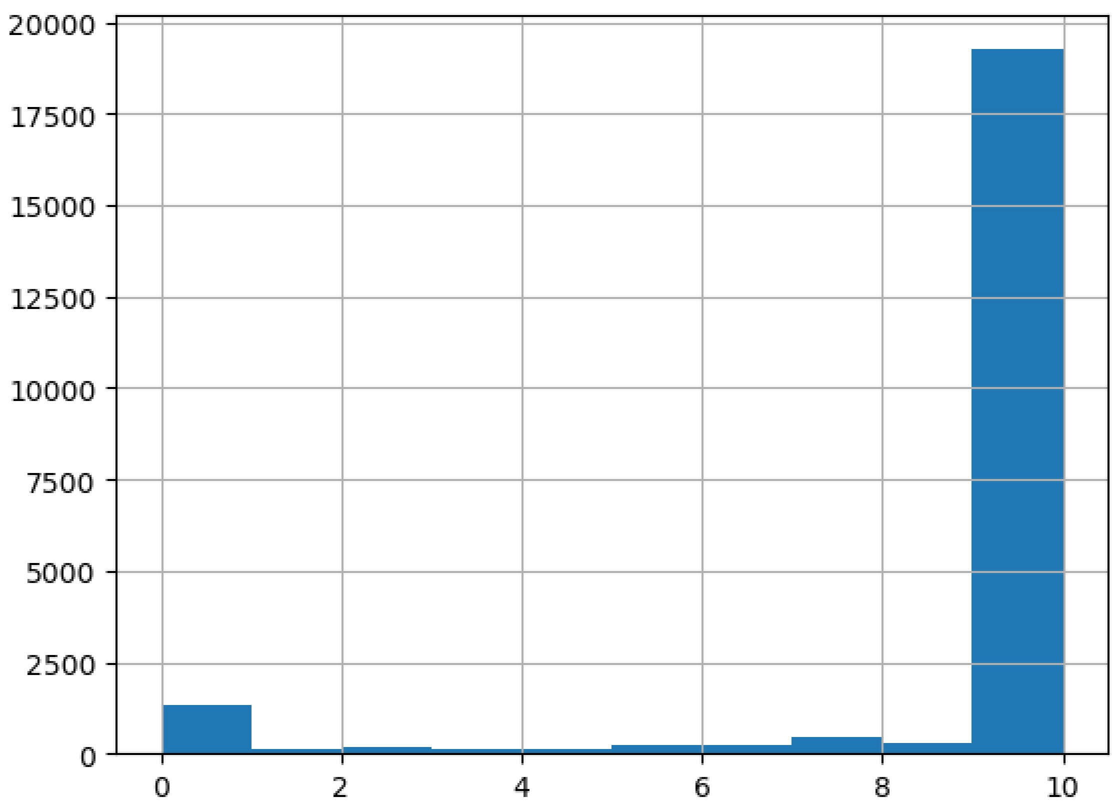
Replaced all observations > 10 with 10, because that is the conventional maximum visiblity

In [23]:

```
events.loc[events['vis_sm']>10, ['vis_sm']] = 10

events['vis_sm'].hist()
```

Out[23]:

```
<Axes: >
```



wx_temp is continuous

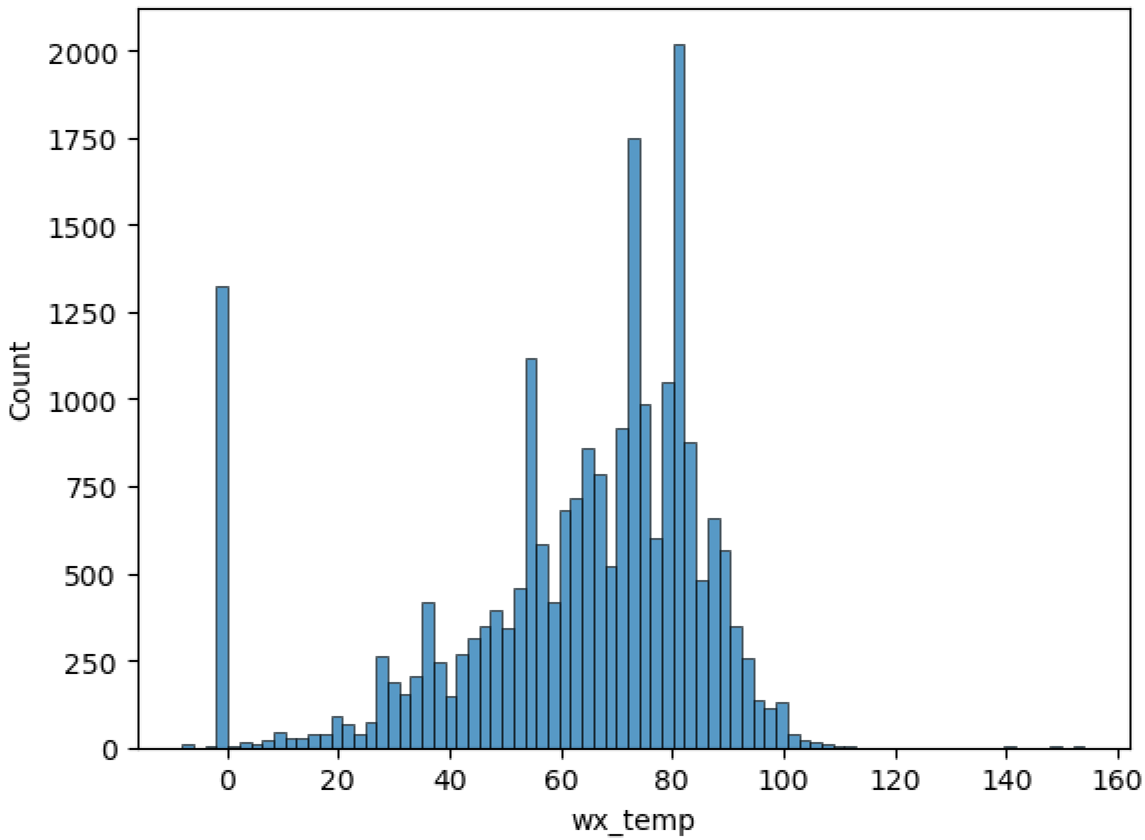
0 (potential nulls) are ~5.5% of data

Dropped ±3 standard deviations

In [24]:
```
print('Percent 0s: ' + str(events[events['wx_temp']==0].wx_temp.count()/events.wx_temp.c
events.drop(index=events[events['wx_temp']>(events['wx_temp'].mean() + 3*events['wx_temp
events.drop(index=events[events['wx_temp']<(events['wx_temp'].mean() - 3*events['wx_temp
sns.histplot(events['wx_temp'])
```

```
Percent 0s: 0.059303786111609545
```

Out[24]:
```
&lt;Axes: xlabel='wx_temp', ylabel='Count'&gt;
```



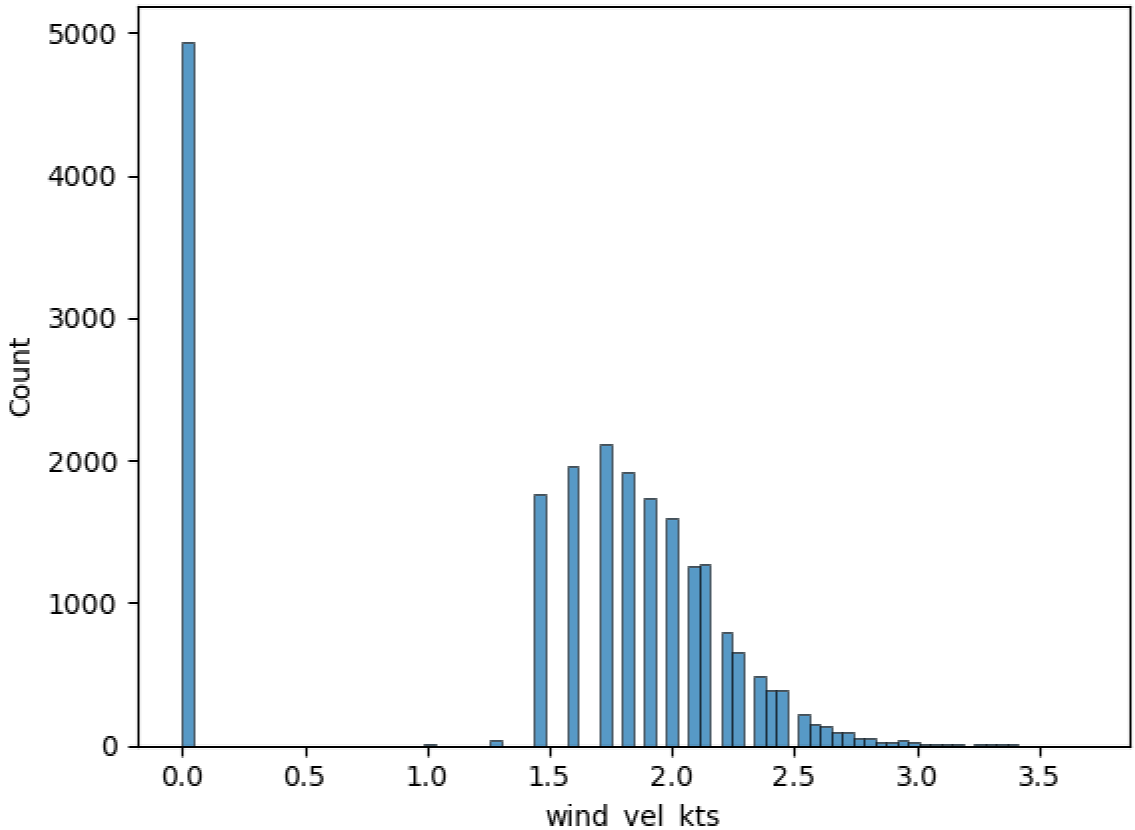
wind_vel_kts is continuous

Could be useful, but can't tell what 0s are nulls and which are real. Using cube root to help account for right-tail and drop ±3 std of outliers.

In [25]:
```
print('Percent 0s: ' + str(events[events['wind_vel_kts']==0].wind_vel_kts.count()/events

events['wind_vel_kts'] = np.cbrt(events['wind_vel_kts'])

events.drop(index=events[events['wind_vel_kts']>(events['wind_vel_kts'].mean() + 3*event
events.drop(index=events[events['wind_vel_kts']>(events['wind_vel_kts'].mean() + 3*event

sns.histplot(events['wind_vel_kts'])
```

```
Percent 0s: 0.22208235928789785
```

Out[25]:
```
&lt;Axes: xlabel='wind_vel_kts', ylabel='Count'&gt;
```

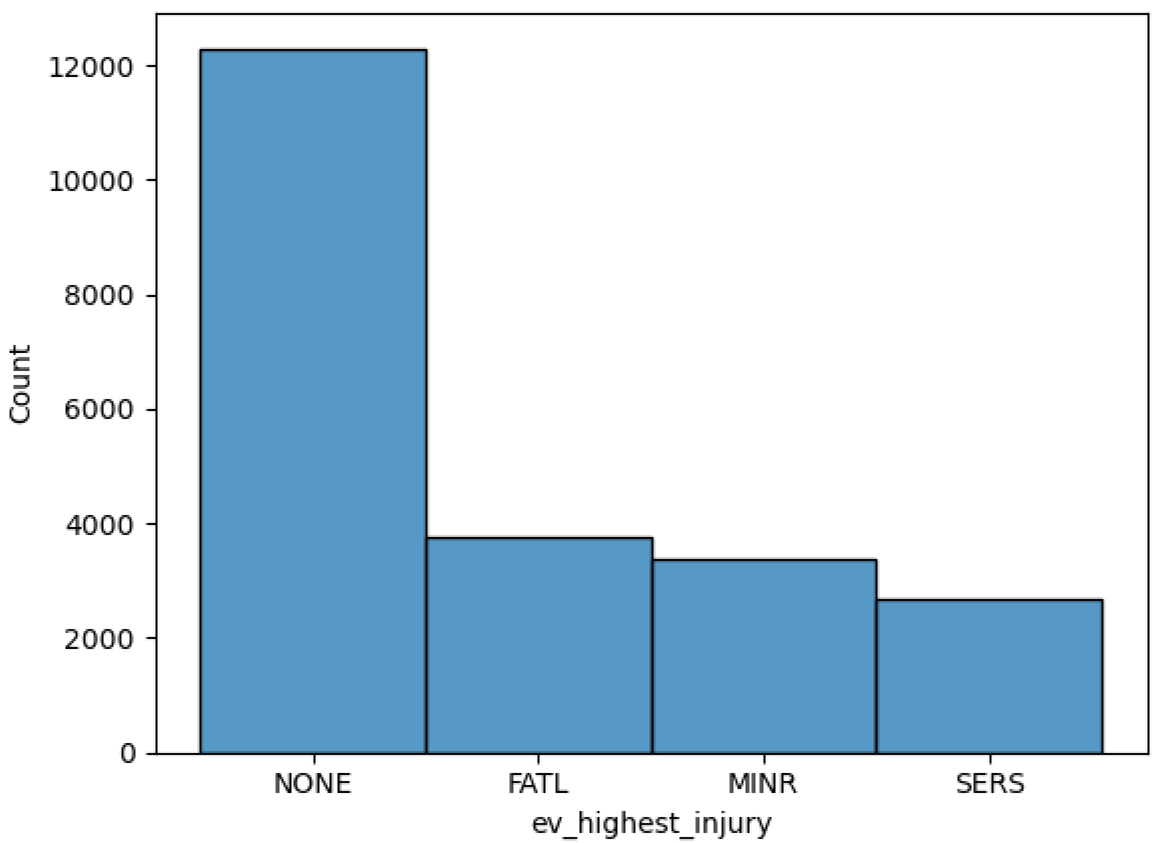


ev_highest_injury is nominal categorical

In [26]:
```
print('Percent NONE: '+str(events[events['ev_highest_injury']=='NONE'].ev_highest_injury
sns.histplot(events['ev_highest_injury'])
```

```
Percent NONE: 0.5557514124293785
```

Out[26]:
```
<Axes: xlabel='ev_highest_injury', ylabel='Count'>
```



Injuries is continuous but massively imbalanced

Appear balanced due as per the check against total As total is composed of the other columns, it has been dropped

In [27]:
```
print(events['inj_f_grnd'].value_counts())
print(events['inj_s_grnd'].value_counts())
print(events['inj_m_grnd'].value_counts())
print(events['inj_tot_f'].value_counts())
print(events['inj_tot_s'].value_counts())
print(events['inj_tot_m'].value_counts())
```

```
0     22195
1        33
2         5
3         3
10        1
4         1
Name: inj_f_grnd, dtype: int64
0     22171
1        57
2         7
3         2
66        1
Name: inj_s_grnd, dtype: int64
0    22169
1       48
2       13
3        4
4        2
5        2
Name: inj_m_grnd, dtype: int64
0     18491
1      2169
2      1052
3       271
4       156
5        53
6        20
7         8
9         6
8         4
10        4
49        1
14        1
16        1
11        1
Name: inj_tot_f, dtype: int64
0     19124
1      2407
2       549
3        98
4        39
5         8
6         7
7         3
50        1
8         1
9         1
Name: inj_tot_s, dtype: int64
0      18144
1       2807
2        976
```

```
3         184
4          68
5          15
6          12
7           8
8           4
9           3
10          2
19          2
41          1
27          1
22          1
43          1
125         1
20          1
12          1
21          1
88          1
137         1
25          1
11          1
13          1
Name: inj_tot_m, dtype: int64
```

In [28]:
```
cols = ['inj_f_grnd', 'inj_s_grnd', 'inj_m_grnd', 'inj_tot_f', 'inj_tot_s', 'inj_tot_m',
sum = 0

for col in cols:
    sum += events[events[col] != 0][col].sum()

print(sum)
print(events['inj_tot_t'].sum())
```

```
107744
107742
```

In [29]:
```
events.drop(columns=['inj_tot_t'], inplace = True)
```

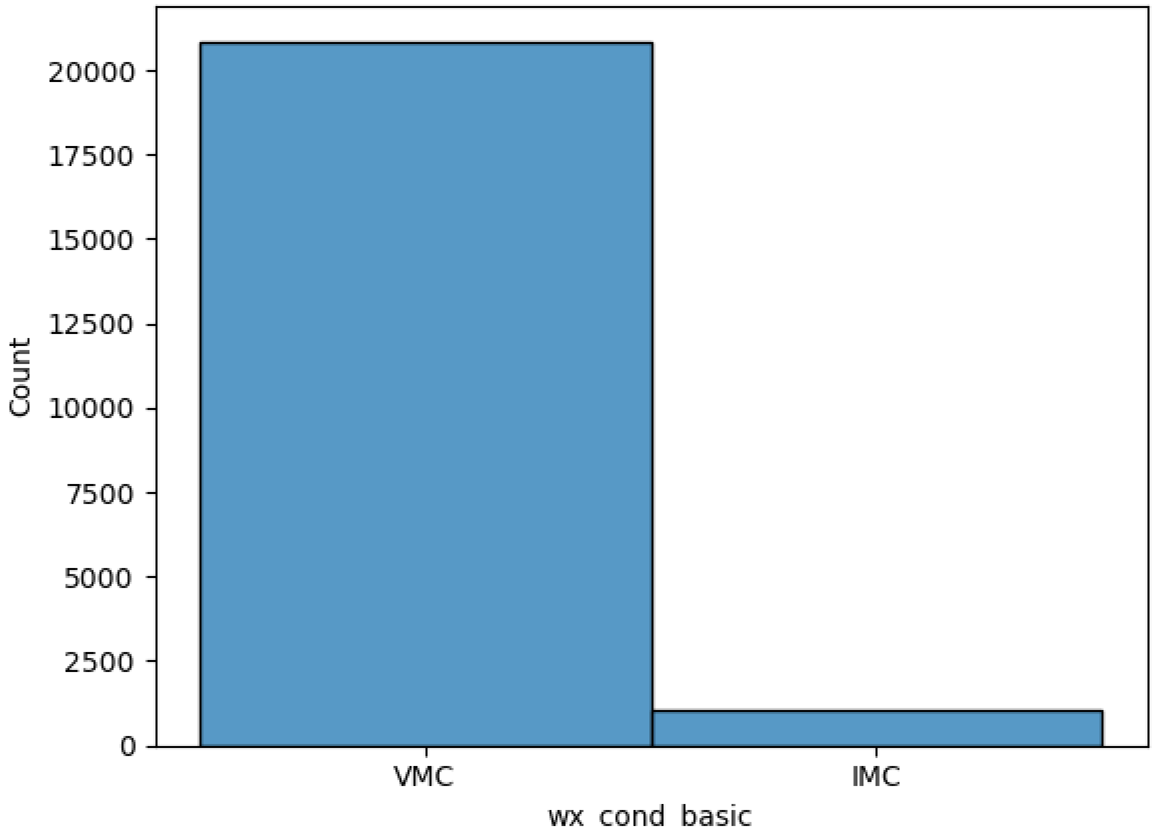
wx_cond_basic is nominal categorical

Heavily imbalanced, dropping records with "Unk"

In [30]:
```
print(events['wx_cond_basic'].value_counts())
events.drop(events[events['wx_cond_basic'] == 'Unk'].index, inplace=True)
print("Percent VMC: " + str(events[events['wx_cond_basic']=='VMC'].wx_cond_basic.count()
sns.histplot(events['wx_cond_basic'])
```

```
VMC    20855
IMC     1021
Unk      135
Name: wx_cond_basic, dtype: int64
Percent VMC: 0.9533278478698116
```

Out[30]:
```
<Axes: xlabel='wx_cond_basic', ylabel='Count'>
```



Checking number of records left in events if all nulls are dropped

Still resonable, so dropping null records

In [31]:
```
events.dropna().info()
events.dropna(inplace = True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20194 entries, 0 to 26966
Data columns (total 21 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   ev_id              20194 non-null  object
 1   ev_type            20194 non-null  object
 2   ev_dow             20194 non-null  category
 3   ev_state           20194 non-null  object
 4   ev_month           20194 non-null  int64
 5   ev_nr_apt_loc      20194 non-null  object
 6   wx_src_iic         20194 non-null  object
 7   wx_obs_time        20194 non-null  int64
 8   light_cond         20194 non-null  object
 9   vis_sm             20194 non-null  float64
 10  wx_temp            20194 non-null  int64
 11  wind_vel_kts       20194 non-null  float64
 12  ev_highest_injury  20194 non-null  object
 13  inj_f_grnd         20194 non-null  int64
 14  inj_m_grnd         20194 non-null  int64
 15  inj_s_grnd         20194 non-null  int64
 16  inj_tot_f          20194 non-null  int64
 17  inj_tot_m          20194 non-null  int64
 18  inj_tot_n          20194 non-null  int64
 19  inj_tot_s          20194 non-null  int64
 20  wx_cond_basic      20194 non-null  object
dtypes: category(1), float64(2), int64(10), object(8)
memory usage: 3.3+ MB
```

aircraft

In [32]:
```
query = "select * from aircraft"
aircraft = pd.read_sql(query, sql)
```

```
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/io/sql.py:761: UserWarning: pandas only support SQLAlchemy connectable(engine/connection) ordatabase string URI or sqlite3 DBAPI2 connectionother DBAPI2 objects are not tested, please consider using SQLAlchemy
  warnings.warn(
```

In [33]:
```
aircraft = aircraft.replace(r'^\s*$', np.nan, regex=True)
aircraft.shape
```

Out[33]:
```
(27412, 72)
```

In [34]:
```
aircraft.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27412 entries, 0 to 27411
Data columns (total 72 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   ev_id                  27412 non-null  object 
 1   Aircraft_Key           27412 non-null  int64  
 2   regis_no               27368 non-null  object 
 3   ntsb_no                27412 non-null  object 
 4   acft_missing           27412 non-null  object 
 5   far_part               26735 non-null  object 
 6   flt_plan_filed         23324 non-null  object 
 7   flight_plan_activated  16762 non-null  object 
 8   damage                 25684 non-null  object 
 9   acft_fire              27211 non-null  object 
 10  acft_expl              26386 non-null  object 
 11  acft_make              27370 non-null  object 
 12  acft_model             27359 non-null  object 
 13  acft_series            10371 non-null  object 
 14  acft_serial_no         25144 non-null  object 
 15  cert_max_gr_wt         27412 non-null  int64  
 16  acft_category          27066 non-null  object 
 17  homebuilt              27412 non-null  object 
 18  fc_seats               27412 non-null  int64  
 19  cc_seats               27412 non-null  int64  
 20  pax_seats              27412 non-null  int64  
 21  total_seats            27412 non-null  int64  
 22  num_eng                27412 non-null  int64  
 23  fixed_retractable      27412 non-null  object 
 24  type_last_insp         20462 non-null  object 
 25  date_last_insp         18159 non-null  object 
 26  afm_hrs_last_insp      27412 non-null  float64
 27  afm_hrs                27412 non-null  float64
 28  elt_install            19623 non-null  object 
 29  elt_oper               15394 non-null  object 
 30  elt_aided_loc_ev       9066 non-null   object 
 31  elt_type               12465 non-null  object 
 32  owner_acft             15122 non-null  object 
 33  owner_street           8460 non-null   object 
 34  owner_city             23304 non-null  object 
 35  owner_state            22766 non-null  object 
 36  owner_country          25698 non-null  object 
 37  owner_zip              22109 non-null  object 
 38  oper_individual_name   27412 non-null  object 
```

```
 39   oper_name              14565 non-null   object
 40   oper_dba               1418 non-null    object
 41   oper_street            6618 non-null    object
 42   oper_city              22631 non-null   object
 43   oper_state             22095 non-null   object
 44   oper_country           25491 non-null   object
 45   oper_zip               20750 non-null   object
 46   oper_code              1964 non-null    object
 47   certs_held             24477 non-null   object
 48   oper_sched             3213 non-null    object
 49   oper_dom_int           2948 non-null    object
 50   oper_pax_cargo         2765 non-null    object
 51   type_fly               22250 non-null   object
 52   second_pilot           22420 non-null   object
 53   dprt_pt_same_ev        3431 non-null    object
 54   dprt_apt_id            20231 non-null   object
 55   dprt_city              22611 non-null   object
 56   dprt_state             20803 non-null   object
 57   dprt_country           23184 non-null   object
 58   dprt_time              27412 non-null   int64
 59   dest_apt_id            18226 non-null   object
 60   dest_city              20790 non-null   object
 61   dest_state             19101 non-null   object
 62   dest_country           21654 non-null   object
 63   afm_hrs_since          27412 non-null   object
 64   rwy_num                12555 non-null   object
 65   rwy_len                27412 non-null   int64
 66   rwy_width              27412 non-null   int64
 67   site_seeing            24410 non-null   object
 68   air_medical            24434 non-null   object
 69   med_type_flight        122 non-null     object
 70   acft_year              27412 non-null   int64
 71   fuel_on_board          27412 non-null   float64
dtypes: float64(3), int64(11), object(58)
memory usage: 15.1+ MB
```

Filtering to rows that align with cleaned events table

In [35]:
```
aircraft = aircraft.merge(events, on='ev_id', how='right')
aircraft.drop(columns=['ev_type', 'ev_dow', 'ev_state', 'ev_month',
       'ev_nr_apt_loc', 'wx_src_iic', 'wx_obs_time', 'light_cond',
       'vis_sm', 'wx_temp', 'wind_vel_kts', 'ev_highest_injury', 'inj_f_grnd',
       'inj_m_grnd', 'inj_s_grnd', 'inj_tot_f', 'inj_tot_m', 'inj_tot_n',
       'inj_tot_s', 'wx_cond_basic'], inplace=True)
```

In [36]:
```
aircraft.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20461 entries, 0 to 20460
Data columns (total 72 columns):
 #    Column                 Non-Null Count   Dtype
---   ------                 --------------   -----
 0    ev_id                  20461 non-null   object
 1    Aircraft_Key           20461 non-null   int64
 2    regis_no               20458 non-null   object
 3    ntsb_no                20461 non-null   object
 4    acft_missing           20461 non-null   object
 5    far_part               20461 non-null   object
 6    flt_plan_filed         20115 non-null   object
 7    flight_plan_activated  14558 non-null   object
 8    damage                 20109 non-null   object
 9    acft_fire              20454 non-null   object
 10   acft_expl              20107 non-null   object
 11   acft_make              20459 non-null   object
```

```
 12  acft_model           20460 non-null  object
 13  acft_series          7219 non-null   object
 14  acft_serial_no       20424 non-null  object
 15  cert_max_gr_wt       20461 non-null  int64
 16  acft_category        20458 non-null  object
 17  homebuilt            20461 non-null  object
 18  fc_seats             20461 non-null  int64
 19  cc_seats             20461 non-null  int64
 20  pax_seats            20461 non-null  int64
 21  total_seats          20461 non-null  int64
 22  num_eng              20461 non-null  int64
 23  fixed_retractable    20461 non-null  object
 24  type_last_insp       18779 non-null  object
 25  date_last_insp       16954 non-null  object
 26  afm_hrs_last_insp    20461 non-null  float64
 27  afm_hrs              20461 non-null  float64
 28  elt_install          18153 non-null  object
 29  elt_oper             14299 non-null  object
 30  elt_aided_loc_ev     8539 non-null   object
 31  elt_type             11566 non-null  object
 32  owner_acft           11583 non-null  object
 33  owner_street         6753 non-null   object
 34  owner_city           20324 non-null  object
 35  owner_state          20212 non-null  object
 36  owner_country        20353 non-null  object
 37  owner_zip            19703 non-null  object
 38  oper_individual_name   20461 non-null  object
 39  oper_name            10888 non-null  object
 40  oper_dba             1233 non-null   object
 41  oper_street          5373 non-null   object
 42  oper_city            19982 non-null  object
 43  oper_state           19874 non-null  object
 44  oper_country         20129 non-null  object
 45  oper_zip             18854 non-null  object
 46  oper_code            1451 non-null   object
 47  certs_held           20088 non-null  object
 48  oper_sched           1527 non-null   object
 49  oper_dom_int         1376 non-null   object
 50  oper_pax_cargo       1184 non-null   object
 51  type_fly             19534 non-null  object
 52  second_pilot         19211 non-null  object
 53  dprt_pt_same_ev      2864 non-null   object
 54  dprt_apt_id          17227 non-null  object
 55  dprt_city            18899 non-null  object
 56  dprt_state           18820 non-null  object
 57  dprt_country         18985 non-null  object
 58  dprt_time            20461 non-null  int64
 59  dest_apt_id          15516 non-null  object
 60  dest_city            17403 non-null  object
 61  dest_state           17349 non-null  object
 62  dest_country         17700 non-null  object
 63  afm_hrs_since        20461 non-null  object
 64  rwy_num              11420 non-null  object
 65  rwy_len              20461 non-null  int64
 66  rwy_width            20461 non-null  int64
 67  site_seeing          20393 non-null  object
 68  air_medical          20371 non-null  object
 69  med_type_flight      104 non-null    object
 70  acft_year            20461 non-null  int64
 71  fuel_on_board        20461 non-null  float64
dtypes: float64(3), int64(11), object(58)
memory usage: 11.4+ MB
```

Dropping columns with missing/poisoned data

In [37]:
```
aircraft.drop(columns=['regis_no', 'ntsb_no', 'flight_plan_activated', 'acft_model', 'ac
```

acft_missing is nominal categorical

Heavily imbalanced

In [38]:
```
aircraft['acft_missing'].value_counts()
```

Out[38]:
```
N    20443
Y       18
Name: acft_missing, dtype: int64
```

-far_part is nominal categorical

Combining Part 91, 91K, and NUSN into '091'; Part 121 and NUSC into '121' Dropping 'UNK'

The applicable regulation part (14 CFR) or authority the aircraft was operating under at the time of the accident. (ARMF=armed forces, NUSC=non-US commercial, NUSN=non-US non-commercial, PUBF=public use - federal, PUBS=public use - state, PUBL=public use-local, PUBU=public use, UNK=unknown

91=general aviation, 137=agricultural, 135=charter and regional, 121=commercial and large cargo, 133=helicopter load, 129=foreign air carriers, 107=unmanned, 103=ultralight, 125=corporate

In [39]:
```
aircraft['far_part'].replace('091K', '091', inplace= True)
aircraft['far_part'].replace('NUSN', '091', inplace= True)
aircraft['far_part'].replace('NUSC', '121', inplace= True)
aircraft.drop(aircraft[aircraft['far_part']=='UNK'].index, inplace = True)
aircraft['far_part'].value_counts()
```

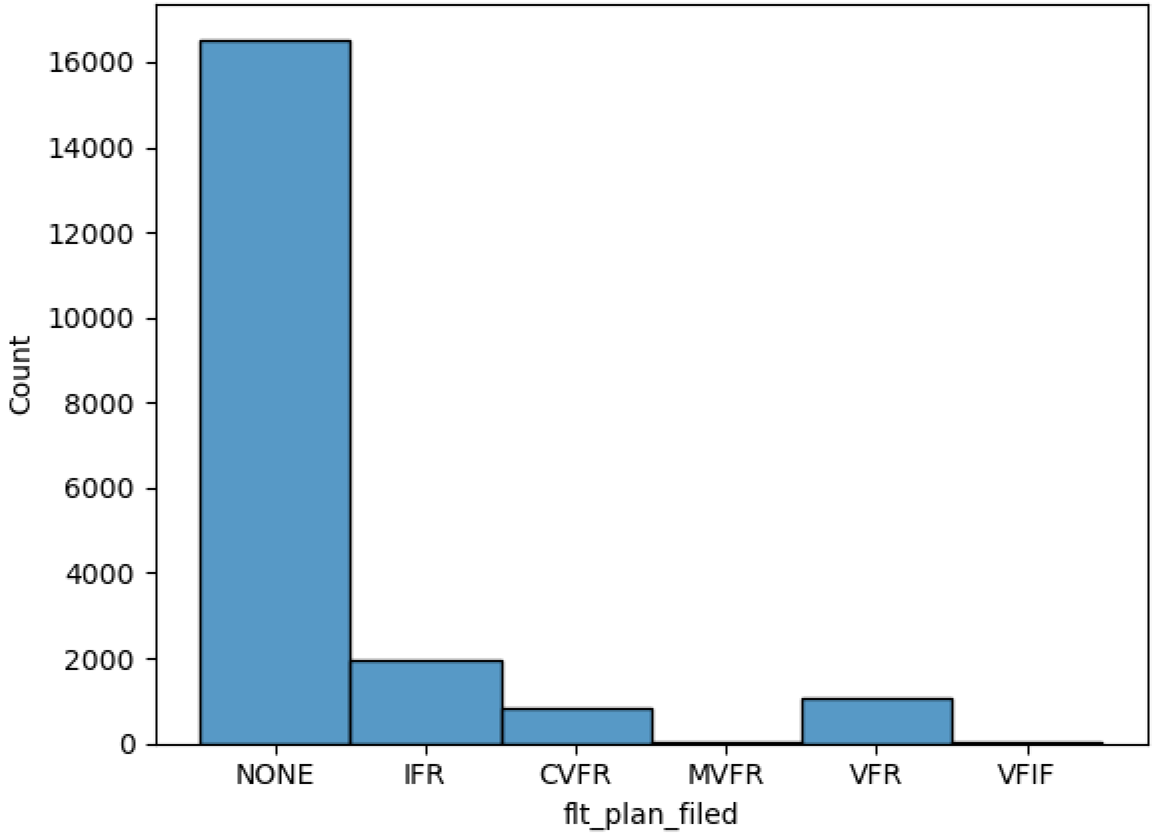
Out[39]:
```
091     18106
137       981
135       691
121       283
PUBU      246
133       106
129        19
107         7
103         5
ARMF        4
125         3
Name: far_part, dtype: int64
```

flt_plan_filed is nominal categorical

Imputing UNK with NONE

In [40]:
```
aircraft['flt_plan_filed'].replace('UNK', 'NONE', inplace=True)
aircraft['flt_plan_filed'].fillna('NONE', inplace=True)
sns.histplot(aircraft['flt_plan_filed'])
```

Out[40]:
```
<Axes: xlabel='flt_plan_filed', ylabel='Count'>
```



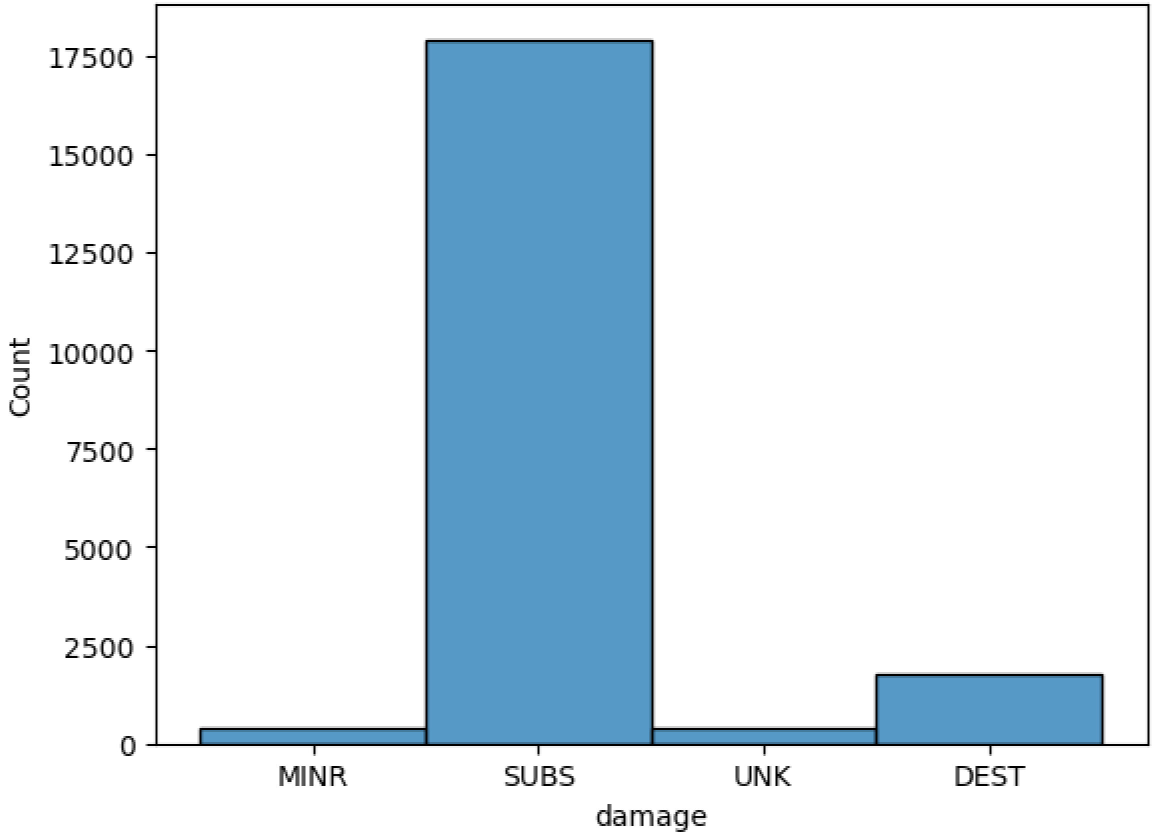
damage is nominal categorical

Combined null and UNK

In [41]:
```
aircraft.damage.replace(np.nan, 'UNK', inplace=True)
print(aircraft['damage'].value_counts())
sns.histplot(aircraft.damage)
```

```
SUBS    17914
DEST     1772
UNK       386
MINR      379
Name: damage, dtype: int64
```

Out[41]:
```
<Axes: xlabel='damage', ylabel='Count'>
```

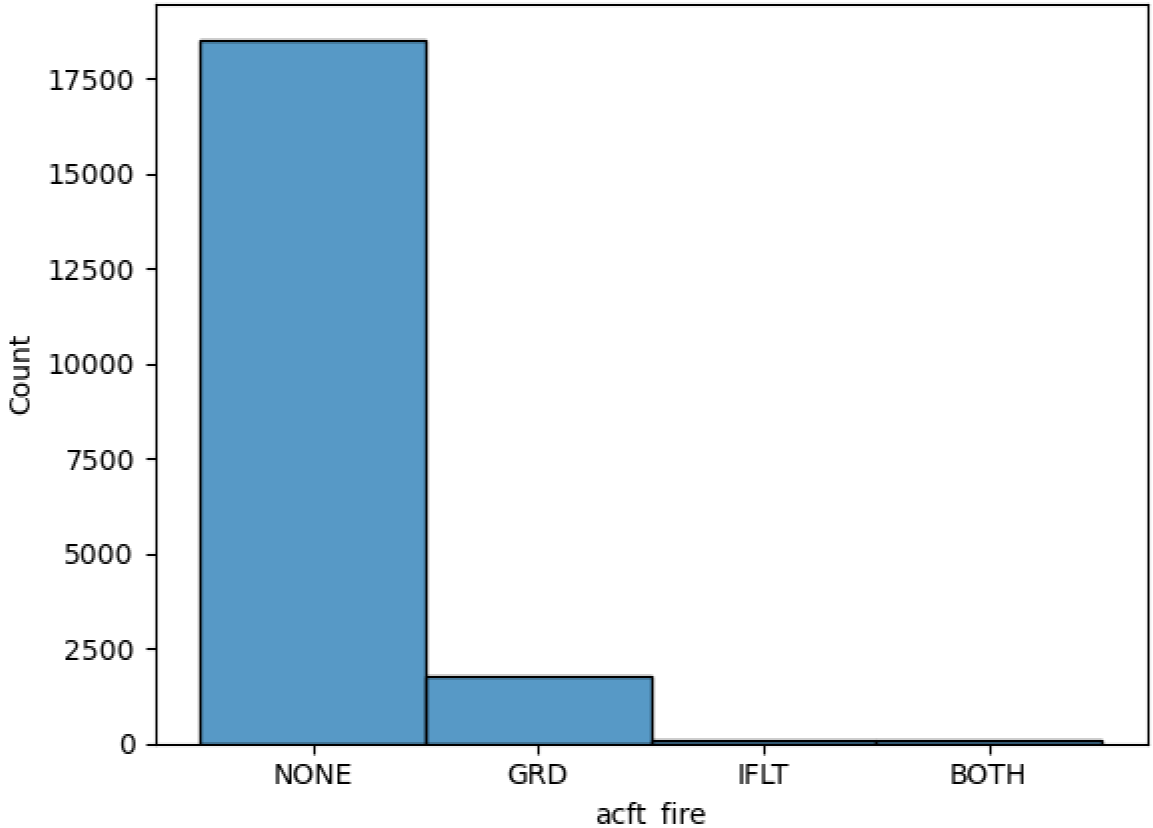


acft_fire is nominal categorical

Imputing NONE into unknowns

In [42]:
```
aircraft['acft_fire'].value_counts()
aircraft['acft_fire'].replace('UNK', 'NONE', inplace=True)
aircraft['acft_fire'].replace('UNKT', 'NONE', inplace=True)
aircraft['acft_fire'].fillna('NONE', inplace=True)
sns.histplot(aircraft.acft_fire)
```

Out[42]:
```
<Axes: xlabel='acft_fire', ylabel='Count'>
```

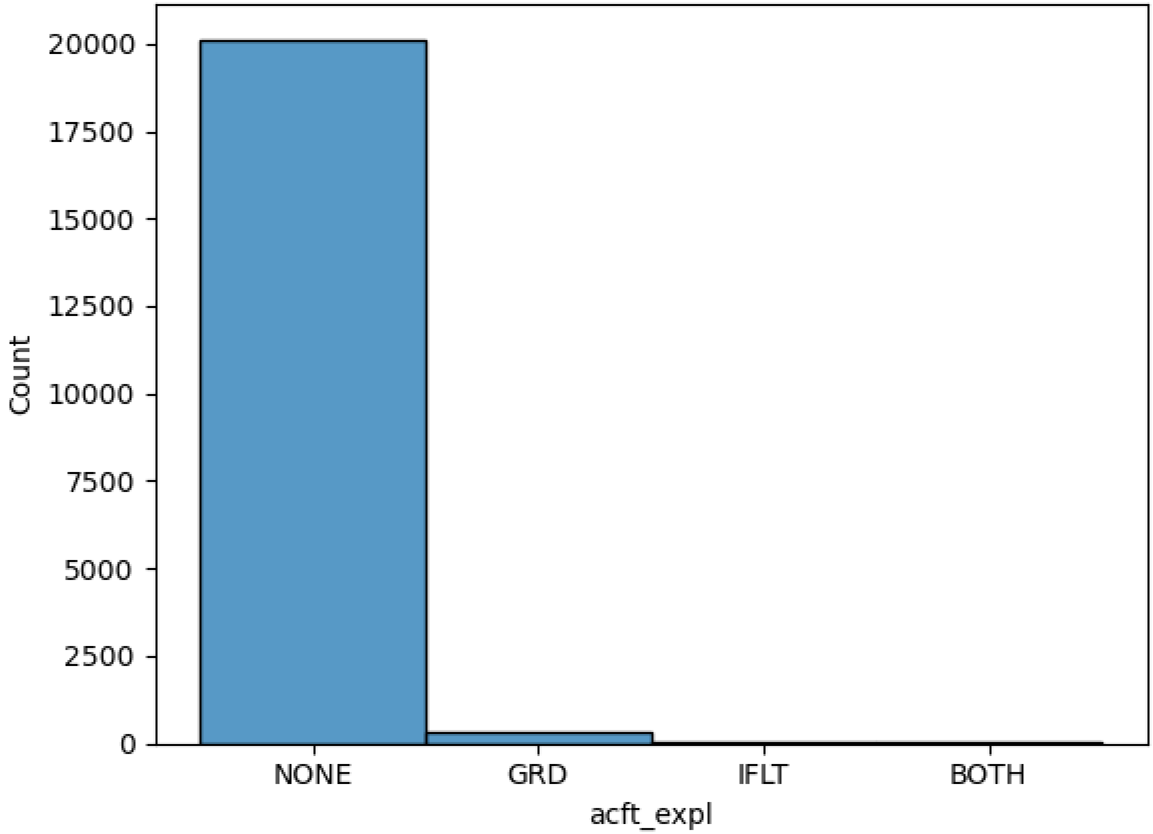


acft_expl is nominal categorical

Imputing NONE into unknowns

In [43]:
```
aircraft['acft_expl'].value_counts()
aircraft['acft_expl'].replace('UNK', 'NONE', inplace=True)
aircraft['acft_expl'].replace('UNKT', 'NONE', inplace=True)
aircraft['acft_expl'].fillna('NONE', inplace=True)
sns.histplot(aircraft.acft_expl)
```

Out[43]:
```
<Axes: xlabel='acft_expl', ylabel='Count'>
```



acft_make is nominal categorical

Combined same brands manually down to value_count of 20, then grouped rest into group OTHER
Imputed nulls into OTHER

In [44]:
```
aircraft['acft_make'] = aircraft.acft_make.str.upper()
aircraft.groupby('acft_make').filter(lambda x : len(x)>20).acft_make.value_counts()
rep = {'CIRRUS DESIGN CORP': 'CIRRUS', 'ROBINSON HELICOPTER COMPANY': 'ROBINSON HELICOPT
aircraft.acft_make.replace(rep, inplace=True)

aircraft.loc[aircraft.groupby('acft_make').acft_make.transform('count').lt(100), 'acft_m
aircraft.acft_make.fillna('OTHER', inplace=True)
print(aircraft.acft_make.value_counts())
```

```
OTHER                  7177
CESSNA                 5200
PIPER                  3105
BEECH                  1111
ROBINSON HELICOPTER     573
BELL                    492
AIR TRACTOR INC         335
MOONEY                  310
CIRRUS                  300
BOEING                  222
GRUMMAN                 204
CHAMPION                175
MAULE                   172
BELLANCA                168
AERONCA                 162
SCHWEIZER               160
HUGHES                  128
VANS                    120
LUSCOMBE                117
STINSON                 115
EUROCOPTER              105
Name: acft_make, dtype: int64
```

acft_category is nominal categorical

Condensed UNK, null, and PLFT into AIR

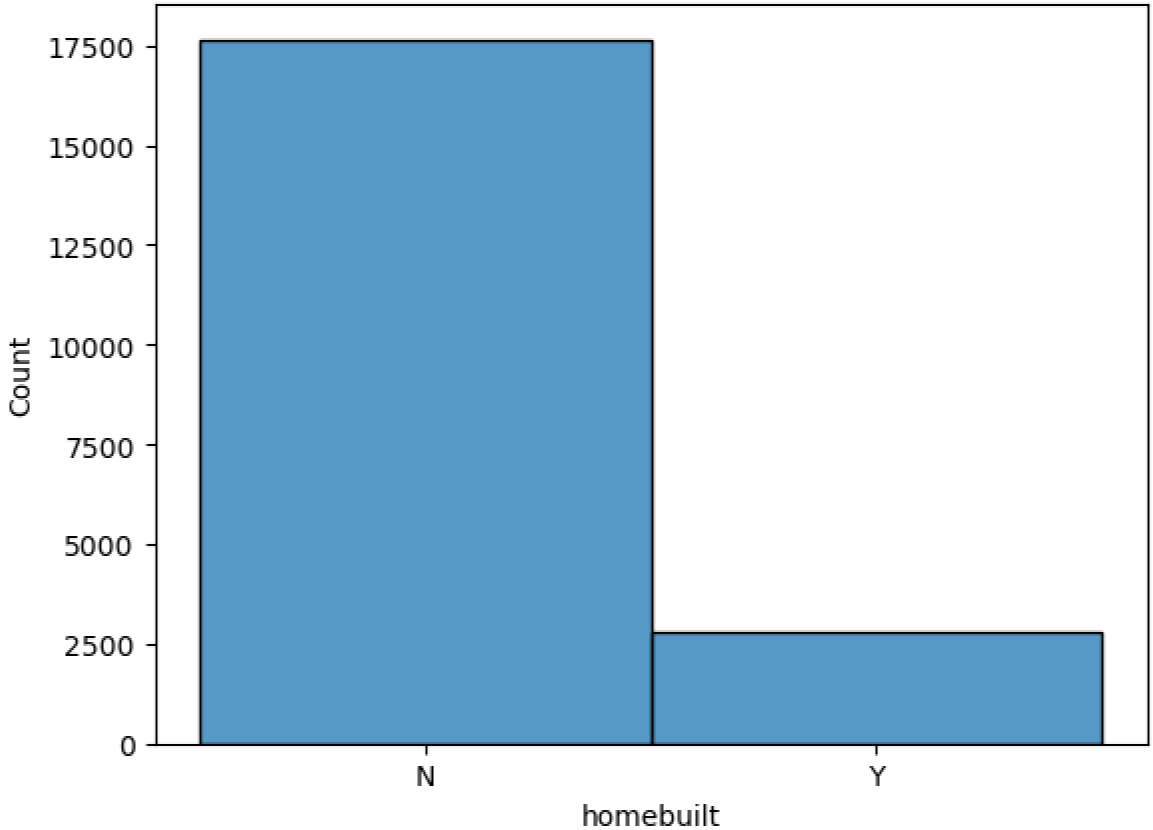
In [45]:
```
aircraft.acft_category.fillna('AIR', inplace=True)
aircraft.acft_category.replace('UNK', 'AIR', inplace=True)
aircraft.acft_category.replace('PLFT', 'AIR', inplace=True)
aircraft['acft_category'].value_counts()
```

Out[45]:
```
AIR     17541
HELI     1980
GLI       361
WSFT      163
BALL      162
GYRO      135
PPAR       89
ULTR       20
Name: acft_category, dtype: int64
```

homebuilt is nominal categorical

In [46]:
```
sns.histplot(aircraft['homebuilt'])
```

Out[46]:
```
<Axes: xlabel='homebuilt', ylabel='Count'>
```

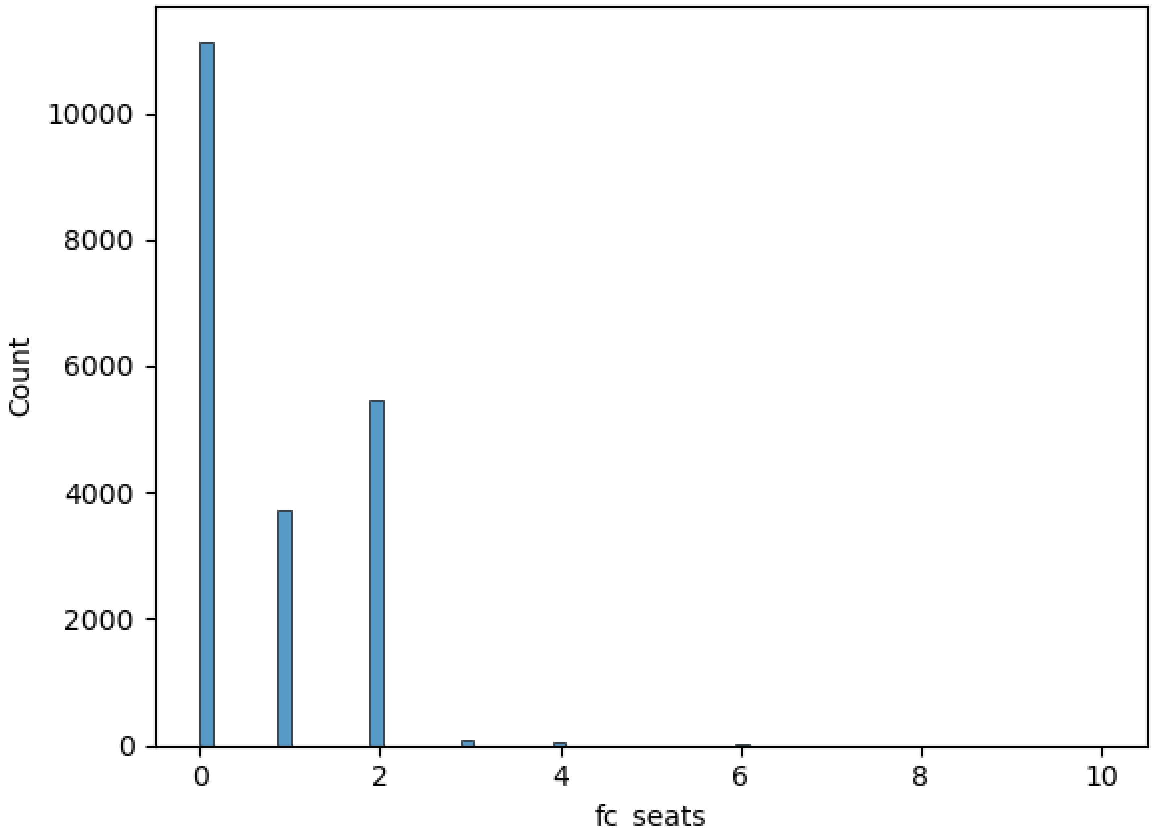


fc_seats is continuous

In [47]:
```
print(aircraft['fc_seats'].value_counts())
sns.histplot(aircraft['fc_seats'])
```

```
0     11137
2      5452
1      3718
3        83
4        54
6         3
5         2
7         1
10        1
Name: fc_seats, dtype: int64
```

Out[47]: <Axes: xlabel='fc_seats', ylabel='Count'>



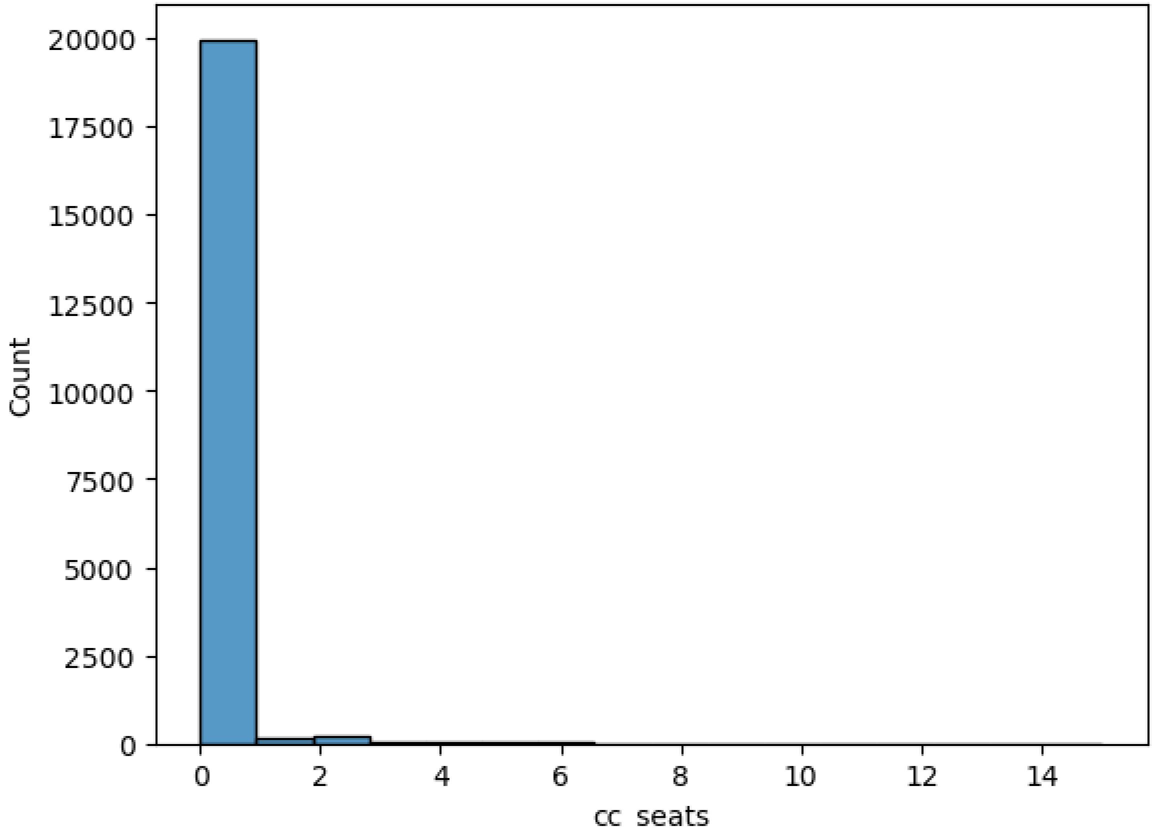
cc_seat is continuous

Replacing outlier with mean

In [48]:
```
aircraft['cc_seats'].replace(66, round(aircraft['cc_seats'].mean()), inplace=True)
print(aircraft['cc_seats'].value_counts())
sns.histplot(aircraft['cc_seats'])
```

```
0     19950
2       187
1       141
4        57
3        39
5        27
6        25
7         7
8         5
9         4
12        3
13        3
15        1
10        1
14        1
Name: cc_seats, dtype: int64
```

Out[48]:
```
<Axes: xlabel='cc_seats', ylabel='Count'>
```



pax_seats is continuous

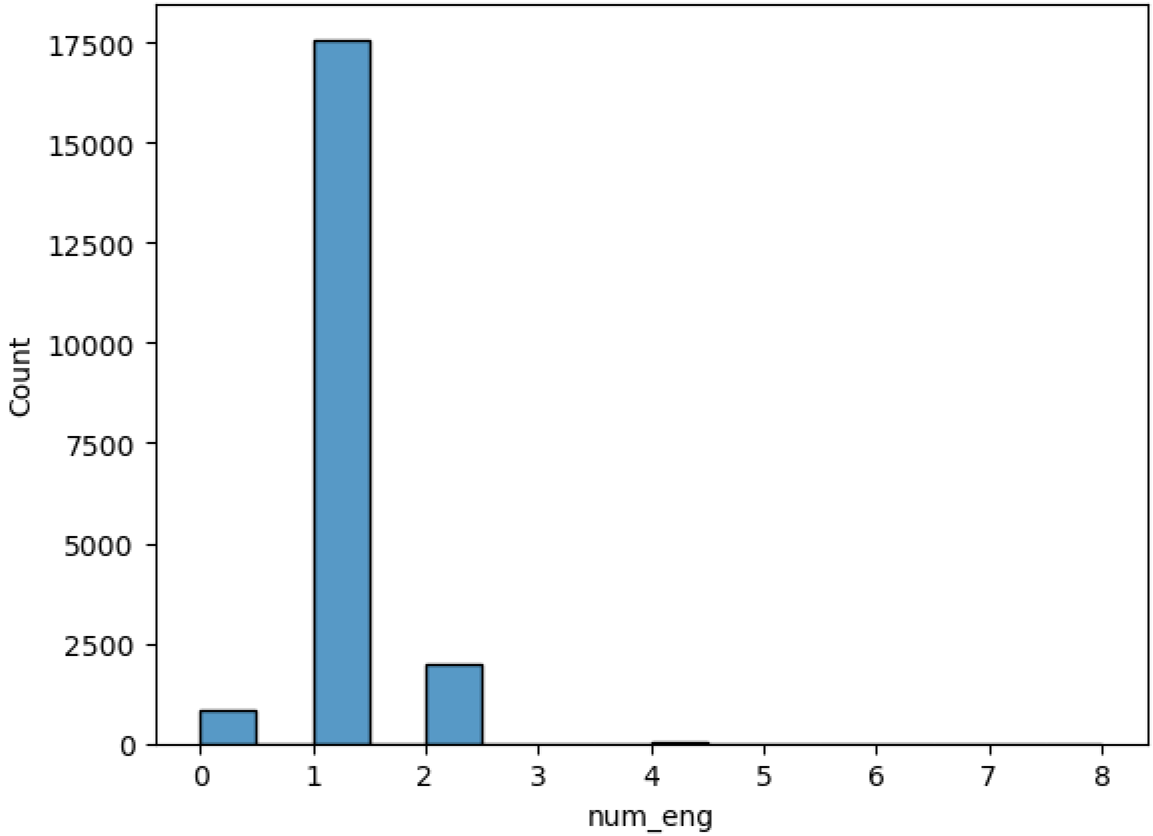
In [49]:
```
print(aircraft['pax_seats'].value_counts())
print(aircraft['pax_seats'].min())
print(aircraft['pax_seats'].max())
```

```
0      12298
2       2825
1       1971
4       1243
3        949
       ...
187        1
295        1
239        1
293        1
176        1
Name: pax_seats, Length: 99, dtype: int64
0
364
```

num_eng is continuous

In [50]:
```
sns.histplot(aircraft['num_eng'])
```

Out[50]:
```
<Axes: xlabel='num_eng', ylabel='Count'>
```



fixed_retractable is nominal categorical

In [51]:
```
aircraft['fixed_retractable'].value_counts()
```

Out[51]:
```
FIXD    15736
RETR     4715
Name: fixed_retractable, dtype: int64
```

type_last_insp is nominal categorical

In [52]:
```
aircraft['type_last_insp'].fillna('UNK', inplace=True)
aircraft['type_last_insp'].value_counts()
```

Out[52]:
```
ANNL    11180
UNK      2868
100H     2768
COND     2249
COAW      805
AAIP      581
Name: type_last_insp, dtype: int64
```

type_fly is nominal categorical

Combining all public use and other use.

Imputing UNK and null to personal

AAPL=aerial application, ADRP=air drop, AOBV=aerial observation, ASHO=air race/show, BANT=banner tow, BUS=business, EXEC=executive/corporate, FERY=ferry, FLTS=flight test, EXLD=external load, FIRF=fire fighting, GLDT=glider tow, INST=instructional, OTH=other, OWRK=other work use, PERS=personal, POSI=positioning, PUBU=public use, UNK=unknown

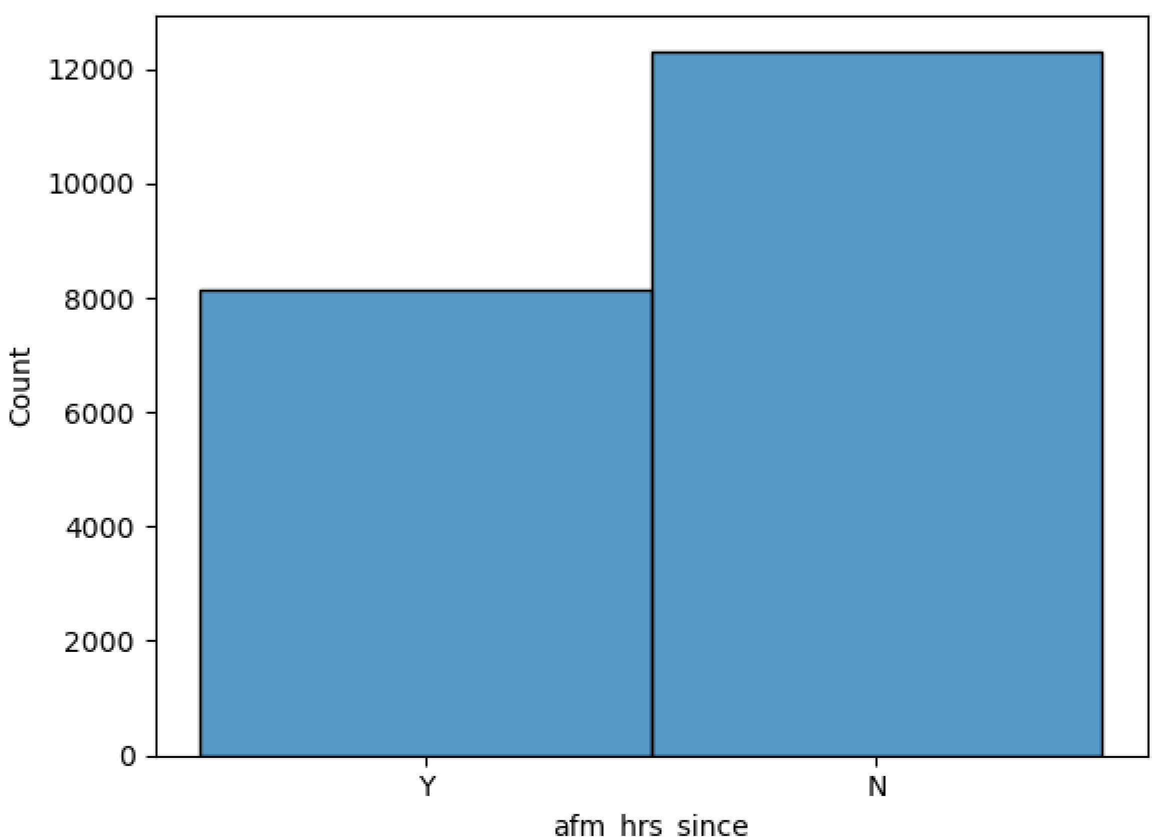
In [53]:
```
rep = {'BUS ': 'BUS', 'OWRK': 'OTH', 'PUBF': 'PUBU', 'PUBS': 'PUBU', 'PUBL': 'PUBU', 'UN
aircraft['type_fly'].replace(rep, inplace=True)
aircraft['type_fly'].fillna('PERS', inplace=True)
aircraft['type_fly'].value_counts()
```

Out[53]:
```
PERS    14063
INST     2845
AAPL      983
BUS       493
POSI      405
OTH       310
AOBV      263
FLTS      240
PUBU      227
FERY      104
SKYD       95
EXLD       94
ASHO       94
EXEC       85
BANT       74
GLDT       42
FIRF       28
ADRP        6
Name: type_fly, dtype: int64
```

afm_hrs_since is nominal categorical

In [54]:
```
sns.histplot(aircraft['afm_hrs_since'])
```

Out[54]:
```
<Axes: xlabel='afm_hrs_since', ylabel='Count'>
```



site_seeing is nominal categorical

Imputed nulls with N

In [55]:
```
aircraft['site_seeing'].fillna('N', inplace=True)
aircraft['site_seeing'].value_counts()
```

Out[55]:
```
N    20214
Y      237
Name: site_seeing, dtype: int64
```

air_medical is nominal categorical

Imputed nulls with N

In [56]:
```
aircraft['air_medical'].fillna('N', inplace=True)
aircraft['air_medical'].value_counts()
```

Out[56]:
```
N    20296
Y      155
Name: air_medical, dtype: int64
```

In [57]:
```
aircraft.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20451 entries, 0 to 20460
Data columns (total 21 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   ev_id              20451 non-null  object
 1   Aircraft_Key       20451 non-null  int64 
 2   acft_missing       20451 non-null  object
 3   far_part           20451 non-null  object
 4   flt_plan_filed     20451 non-null  object
 5   damage             20451 non-null  object
 6   acft_fire          20451 non-null  object
 7   acft_expl          20451 non-null  object
 8   acft_make          20451 non-null  object
 9   acft_category      20451 non-null  object
 10  homebuilt          20451 non-null  object
 11  fc_seats           20451 non-null  int64 
 12  cc_seats           20451 non-null  int64 
 13  pax_seats          20451 non-null  int64 
 14  num_eng            20451 non-null  int64 
 15  fixed_retractable  20451 non-null  object
 16  type_last_insp     20451 non-null  object
 17  type_fly           20451 non-null  object
 18  afm_hrs_since      20451 non-null  object
 19  site_seeing        20451 non-null  object
 20  air_medical        20451 non-null  object
dtypes: int64(5), object(16)
memory usage: 3.4+ MB
```

engines

In [58]:
```
query = "select * from engines"
engines = pd.read_sql(query, sql)
```

```
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/io/sql.py:761: UserWarning: pandas only support SQLAlchemy connectable(engine/connection) ordatabase string URI or sqlite3 DBAPI2 connectionother DBAPI2 objects are not tested, please consider using SQLAlchemy
  warnings.warn(
```

In [59]:
```
engines = engines.replace(r'^\s*$', np.nan, regex=True)
engines.shape
```

Out[59]: (24852, 15)

In [60]:
```
engines = engines.merge(events, on='ev_id', how='right')
engines.drop(columns=['ev_type', 'ev_dow', 'ev_state', 'ev_month',
       'ev_nr_apt_loc', 'wx_src_iic', 'wx_obs_time', 'light_cond',
       'vis_sm', 'wx_temp', 'wind_vel_kts', 'ev_highest_injury', 'inj_f_grnd',
       'inj_m_grnd', 'inj_s_grnd', 'inj_tot_f', 'inj_tot_m', 'inj_tot_n',
       'inj_tot_s', 'wx_cond_basic'], inplace=True)
```

In [61]:
```
engines.drop(columns=['eng_no', 'eng_model', 'power_units', 'hp_or_lbs', 'carb_fuel_inje
```

eng_type is nominal categorical

Grouped bottom categories together

Imputing nulls into REC

In [62]:
```
rep = {'UNK':'NONE', 'ELEC': 'NONE', 'LR': 'NONE', 'GTFN': 'NONE'}
engines['eng_type'].replace(rep, inplace=True)
engines['eng_type'].fillna('REC', inplace=True)
engines.eng_type.value_counts()
```

Out[62]:
```
REC     18824
TP       1355
TS       1046
TF        875
TJ        176
NONE       83
Name: eng_type, dtype: int64
```

eng_mfgr is nominal categorical

Grouped the brands, then the rest into OTHER

In [63]:
```
engines['eng_mfgr'] = engines.eng_mfgr.str.upper()
engines.groupby('eng_mfgr').filter(lambda x : len(x)>20).eng_mfgr.value_counts()
rep = {'CONT MOTOR': 'CONTINENTAL', 'CONTINENTAL MOTORS': 'CONTINENTAL', 'P&W CANADA': '
engines.eng_mfgr.replace(rep, inplace=True)

engines.loc[engines.groupby('eng_mfgr').eng_mfgr.transform('count').lt(1000), 'eng_mfgr'
engines.eng_mfgr.fillna('OTHER', inplace=True)
print(engines.eng_mfgr.value_counts())
```

```
LYCOMING            8321
CONTINENTAL         5499
OTHER               5388
PRATT & WHITNEY     1752
ROTAX               1399
Name: eng_mfgr, dtype: int64
```

In [64]:
```
engines.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 22359 entries, 0 to 22358
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   ev_id         22359 non-null  object
 1   Aircraft_Key  21108 non-null  float64
 2   eng_type      22359 non-null  object
 3   eng_mfgr      22359 non-null  object
```

```
dtypes: float64(1), object(3)
memory usage: 873.4+ KB
```

Flight_Crew

In [65]:

```
query = "select * from Flight_Crew"
Flight_Crew = pd.read_sql(query, sql)
```

```
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/io/sql.py:761: UserWarning: pandas only support SQLAlchemy connectable(engine/connection) ordatabase string URI or sqlite3 DBAPI2 connectionother DBAPI2 objects are not tested, please consider using SQLAlchemy
  warnings.warn(
```

In [66]:

```
Flight_Crew = Flight_Crew.replace(r'^\s*$', np.nan, regex=True)
Flight_Crew.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27853 entries, 0 to 27852
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   ev_id             27853 non-null  object
 1   Aircraft_Key      27853 non-null  int64 
 2   crew_no           27853 non-null  int64 
 3   crew_category     27543 non-null  object
 4   crew_age          27853 non-null  int64 
 5   crew_sex          22333 non-null  object
 6   crew_city         25853 non-null  object
 7   crew_res_state    25524 non-null  object
 8   crew_res_country  27525 non-null  object
 9   med_certf         25024 non-null  object
 10  med_crtf_vldty    22858 non-null  object
 11  date_lst_med      22621 non-null  object
 12  crew_inj_level    27058 non-null  object
 13  crew_tox_perf     23742 non-null  object
 14  seat_occ_pic      26525 non-null  object
 15  pc_profession     26065 non-null  object
 16  bfr_date          16713 non-null  object
 17  ft_as_of          10374 non-null  object
 18  mr_faa_med_certf  311 non-null    object
dtypes: int64(3), object(16)
memory usage: 4.0+ MB
```

In [67]:

```
Flight_Crew = Flight_Crew.merge(aircraft, on=['ev_id','Aircraft_Key'], how='right')
Flight_Crew.drop(columns=['acft_missing', 'far_part',
       'flt_plan_filed', 'damage', 'acft_fire', 'acft_expl', 'acft_make',
       'acft_category', 'homebuilt', 'fc_seats', 'cc_seats', 'pax_seats',
       'num_eng', 'fixed_retractable', 'type_last_insp', 'type_fly',
       'afm_hrs_since', 'site_seeing', 'air_medical'], inplace=True)
Flight_Crew.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25487 entries, 0 to 25486
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ev_id             25487 non-null  object 
 1   Aircraft_Key      25487 non-null  int64  
 2   crew_no           24682 non-null  float64
 3   crew_category     24578 non-null  object 
 4   crew_age          24682 non-null  float64
 5   crew_sex          20350 non-null  object 
 6   crew_city         23721 non-null  object 
```

```
 7   crew_res_state    23471 non-null  object
 8   crew_res_country  24488 non-null  object
 9   med_certf         22805 non-null  object
 10  med_crtf_vldty    21020 non-null  object
 11  date_lst_med      20737 non-null  object
 12  crew_inj_level    24298 non-null  object
 13  crew_tox_perf     20899 non-null  object
 14  seat_occ_pic      24175 non-null  object
 15  pc_profession     23817 non-null  object
 16  bfr_date          15522 non-null  object
 17  ft_as_of          9906 non-null   object
 18  mr_faa_med_certf  292 non-null    object
dtypes: float64(2), int64(1), object(16)
memory usage: 3.9+ MB
```

In [68]:

```
Flight_Crew.drop(columns=['crew_no',
       'med_crtf_vldty', 'date_lst_med',
       'crew_tox_perf', 'pc_profession', 'bfr_date',
       'ft_as_of', 'mr_faa_med_certf', 'crew_sex', 'crew_city',
       'crew_res_state', 'crew_res_country'], inplace=True)
```

crew_category is nominal categorical

Imputing nulls to OTHR

In [69]:

```
Flight_Crew.crew_category.fillna('OTHR', inplace=True)
Flight_Crew['crew_category'].value_counts()
```

Out[69]:

```
PLT     18110
DSTU     1843
FLTI     1662
PASS     1638
OTHR     1077
CPLT      643
PRPS      376
KPLT       85
CABN       44
FENG        9
Name: crew_category, dtype: int64
```

crew_age is continuous

Rounding all values and imputing 0s to mean Imputing outliers (±3 std) to mean

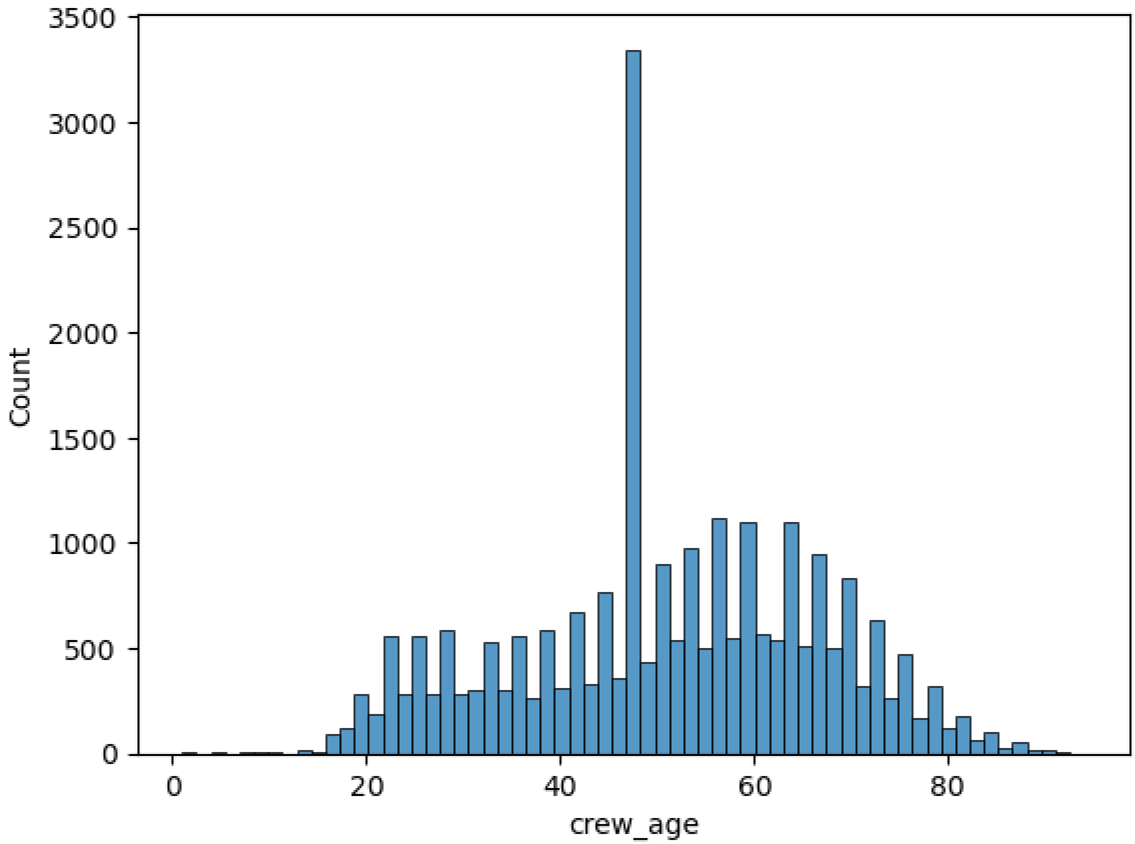
In [70]:

```
print('Percent null/0s: '+str((Flight_Crew[Flight_Crew['crew_age'] == 0].crew_age.count(

Flight_Crew['crew_age'].replace(Flight_Crew[Flight_Crew['crew_age']> Flight_Crew['crew_a
Flight_Crew['crew_age'].replace(Flight_Crew[Flight_Crew['crew_age']< Flight_Crew['crew_a
Flight_Crew['crew_age'].fillna(Flight_Crew['crew_age'].mean(), inplace=True)
Flight_Crew['crew_age'].replace(0, Flight_Crew['crew_age'].mean(), inplace=True)
Flight_Crew['crew_age'] = round(Flight_Crew['crew_age'])

sns.histplot(Flight_Crew['crew_age'])
```

```
Percent null/0s: 0.10173405720768171
```

Out[70]:

```
<Axes: xlabel='crew_age', ylabel='Count'>
```



med_certf is nominal categorical

Imputed nulls with UNK

In [71]:
```
Flight_Crew['med_certf'].fillna('UNK', inplace=True)
Flight_Crew['med_certf'].value_counts()
```

Out[71]:
```
CL3     9259
CL2     6520
CL1     4226
UNK     2876
NONE    1145
SPRT     833
BASC     628
Name: med_certf, dtype: int64
```

crew_inj_level is nominal categorical

Imputed nulls with UNKN

In [72]:
```
Flight_Crew['crew_inj_level'].fillna('UNKN', inplace=True)
Flight_Crew['crew_inj_level'].value_counts()
```

Out[72]:
```
NONE    14774
FATL     3702
MINR     3482
SERS     2330
UNKN     1199
Name: crew_inj_level, dtype: int64
```

seat_occ_pic is nominal categorical

Imputed nulls with UNK

In [73]:
```
Flight_Crew['seat_occ_pic'].fillna('UNK', inplace=True)
Flight_Crew['seat_occ_pic'].value_counts()
```

Out[73]:
```
LEFT    13136
RGT      4752
FRT      2557
UNK      2149
REAR     1139
SNGL     1107
CTR       436
NONE      211
Name: seat_occ_pic, dtype: int64
```

In [74]:
```
Flight_Crew.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25487 entries, 0 to 25486
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   ev_id           25487 non-null  object 
 1   Aircraft_Key    25487 non-null  int64  
 2   crew_category   25487 non-null  object 
 3   crew_age        25487 non-null  float64
 4   med_certf       25487 non-null  object 
 5   crew_inj_level  25487 non-null  object 
 6   seat_occ_pic    25487 non-null  object 
dtypes: float64(1), int64(1), object(5)
memory usage: 1.6+ MB
```

Findings

As part of the cleaning process for the findings table, I dropped any of the findings beyond the 5th finding. This was to reduce the repetition in the training data to a manageable amount while preserving the quality of multiple findings.

In [75]:
```
Findings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64006 entries, 0 to 64005
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   ev_id           64006 non-null  object
 1   Aircraft_Key    64006 non-null  int64 
 2   finding_no      64006 non-null  int64 
 3   category_no     64006 non-null  int64 
 4   subcategory_no  64006 non-null  int64 
dtypes: int64(4), object(1)
memory usage: 2.4+ MB
```

Dropping any 5th or higher ranking findings

In [76]:
```
Findings.drop(Findings[Findings['finding_no'] > 4].index, inplace = True)
```

Dropping "Organizational issues" and "Not determined" categories to limit number of classes

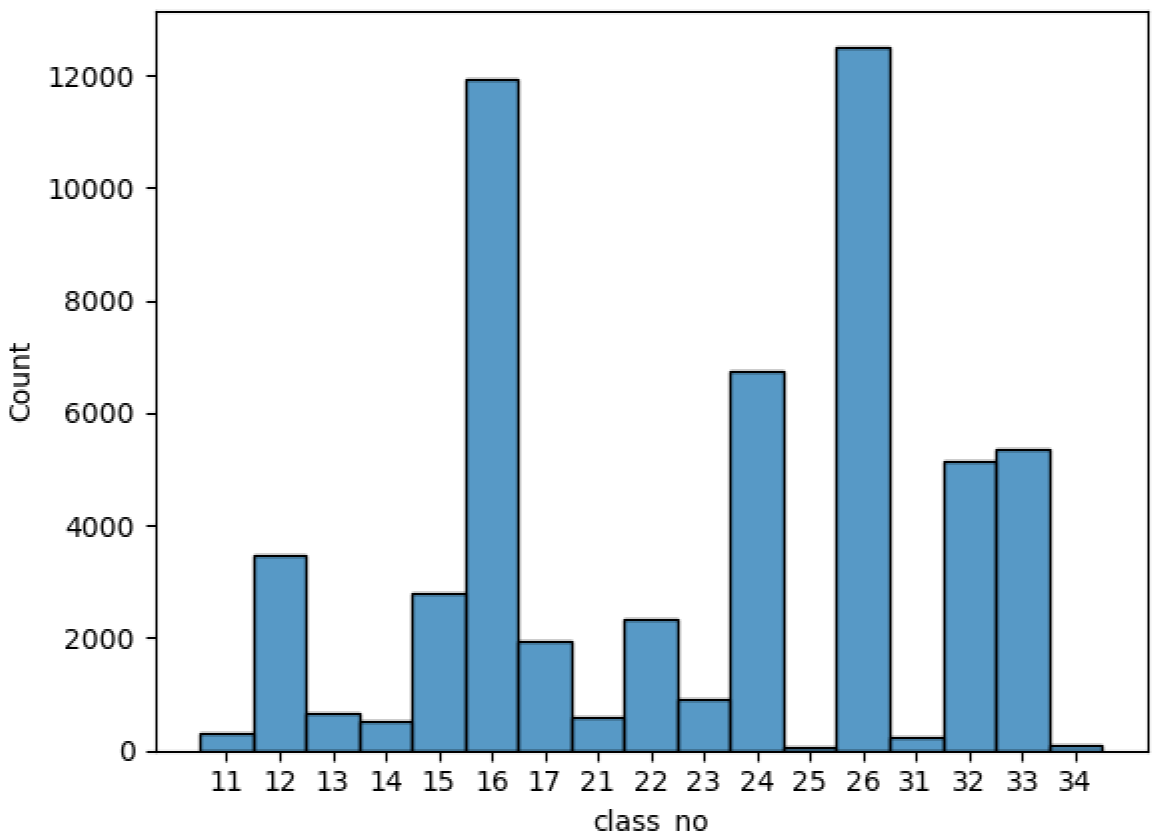
In [77]:
```
Findings.drop(Findings[Findings['category_no'] > 3].index, inplace = True)
```

In [78]:
```
Findings['class_no'] = Findings['category_no'].astype(str) + Findings['subcategory_no'].
Findings['class_no'] = pd.Categorical(Findings['class_no'])
Findings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 55556 entries, 0 to 64005
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   ev_id           55556 non-null  object
 1   Aircraft_Key    55556 non-null  int64
 2   finding_no      55556 non-null  int64
 3   category_no     55556 non-null  int64
 4   subcategory_no  55556 non-null  int64
 5   class_no        55556 non-null  category
dtypes: category(1), int64(4), object(1)
memory usage: 2.6+ MB
```

In [79]:
```
sns.histplot(Findings['class_no'])
```

Out[79]:
```
<Axes: xlabel='class_no', ylabel='Count'>
```



EDA

*Beyond the analysis shown during the cleaning stage

For my EDA, I merged all of the tables into one. While the above visuals show the distibution for each column, I needed to know the correlation and association of the variables. To resolve this, I performed a Chi^2 Test for Association between all of the categorical variables, a Correlation Test between all of the continuous variables, and a One-Way-ANOVA Test between the categorical and continuous variables.

Merging Tables

In [80]:
```
AcFc = aircraft.merge(Flight_Crew, on=['ev_id', 'Aircraft_Key'], how='right')
AcFcE = AcFc.merge(engines, on=['ev_id', 'Aircraft_Key'], how='right')
AcFcEF = AcFcE.merge(Findings, on=['ev_id', 'Aircraft_Key'], how='right')
full = events.merge(AcFcEF, on=['ev_id'], how='right')
full.dropna(inplace=True)
full.info()
```

```
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/core/reshape/merg
e.py:1214: RuntimeWarning: invalid value encountered in cast
  if not (rk == rk.astype(lk.dtype))[~np.isnan(rk)].all():
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/core/dtypes/cast.
py:2221: RuntimeWarning: invalid value encountered in cast
  casted = element.astype(dtype)
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/core/reshape/merg
e.py:1204: RuntimeWarning: invalid value encountered in cast
  if not (lk == lk.astype(rk.dtype))[~np.isnan(lk)].all():
<class 'pandas.core.frame.DataFrame'>
Int64Index: 70309 entries, 0 to 76078
Data columns (total 52 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   ev_id               70309 non-null  object
 1   ev_type             70309 non-null  object
 2   ev_dow              70309 non-null  category
 3   ev_state            70309 non-null  object
 4   ev_month            70309 non-null  float64
 5   ev_nr_apt_loc       70309 non-null  object
 6   wx_src_iic          70309 non-null  object
 7   wx_obs_time         70309 non-null  float64
 8   light_cond          70309 non-null  object
 9   vis_sm              70309 non-null  float64
 10  wx_temp             70309 non-null  float64
 11  wind_vel_kts        70309 non-null  float64
 12  ev_highest_injury   70309 non-null  object
 13  inj_f_grnd          70309 non-null  float64
 14  inj_m_grnd          70309 non-null  float64
 15  inj_s_grnd          70309 non-null  float64
 16  inj_tot_f           70309 non-null  float64
 17  inj_tot_m           70309 non-null  float64
 18  inj_tot_n           70309 non-null  float64
 19  inj_tot_s           70309 non-null  float64
 20  wx_cond_basic       70309 non-null  object
 21  Aircraft_Key        70309 non-null  float64
 22  acft_missing        70309 non-null  object
 23  far_part            70309 non-null  object
 24  flt_plan_filed      70309 non-null  object
 25  damage              70309 non-null  object
 26  acft_fire           70309 non-null  object
 27  acft_expl           70309 non-null  object
 28  acft_make           70309 non-null  object
 29  acft_category       70309 non-null  object
 30  homebuilt           70309 non-null  object
 31  fc_seats            70309 non-null  float64
 32  cc_seats            70309 non-null  float64
 33  pax_seats           70309 non-null  float64
 34  num_eng             70309 non-null  float64
 35  fixed_retractable   70309 non-null  object
 36  type_last_insp      70309 non-null  object
 37  type_fly            70309 non-null  object
 38  afm_hrs_since       70309 non-null  object
 39  site_seeing         70309 non-null  object
```

```
 40  air_medical         70309 non-null  object
 41  crew_category       70309 non-null  object
 42  crew_age            70309 non-null  float64
 43  med_certf           70309 non-null  object
 44  crew_inj_level      70309 non-null  object
 45  seat_occ_pic        70309 non-null  object
 46  eng_type            70309 non-null  object
 47  eng_mfgr            70309 non-null  object
 48  finding_no          70309 non-null  int64
 49  category_no         70309 non-null  int64
 50  subcategory_no      70309 non-null  int64
 51  class_no            70309 non-null  category
dtypes: category(2), float64(18), int64(3), object(29)
memory usage: 27.5+ MB
```

In [150…:

```
full.shape
```

Out[150]:

```
(70309, 52)
```

In [81]:

```
full.columns
conts = ['cc_seats', 'crew_age',
       'fc_seats', 'inj_f_grnd', 'inj_m_grnd', 'inj_s_grnd',
       'inj_tot_f', 'inj_tot_m', 'inj_tot_n', 'inj_tot_s', 'num_eng',
       'pax_seats', 'vis_sm', 'wind_vel_kts', 'wx_temp']
cats = ['ev_type', 'ev_dow', 'ev_state', 'ev_month',
       'ev_nr_apt_loc', 'wx_src_iic', 'wx_obs_time', 'light_cond',
     'ev_highest_injury', 'wx_cond_basic', 'acft_missing',
       'far_part', 'flt_plan_filed', 'damage', 'acft_fire', 'acft_expl',
       'acft_make', 'acft_category', 'homebuilt', 'fixed_retractable', 'type_last_insp',
       'type_fly', 'afm_hrs_since', 'site_seeing', 'air_medical',
       'crew_category', 'med_certf', 'crew_inj_level',
       'seat_occ_pic', 'eng_type', 'eng_mfgr', 'class_no']
```

Chi^2 Test for Association on Categorical Variables

In [82]:

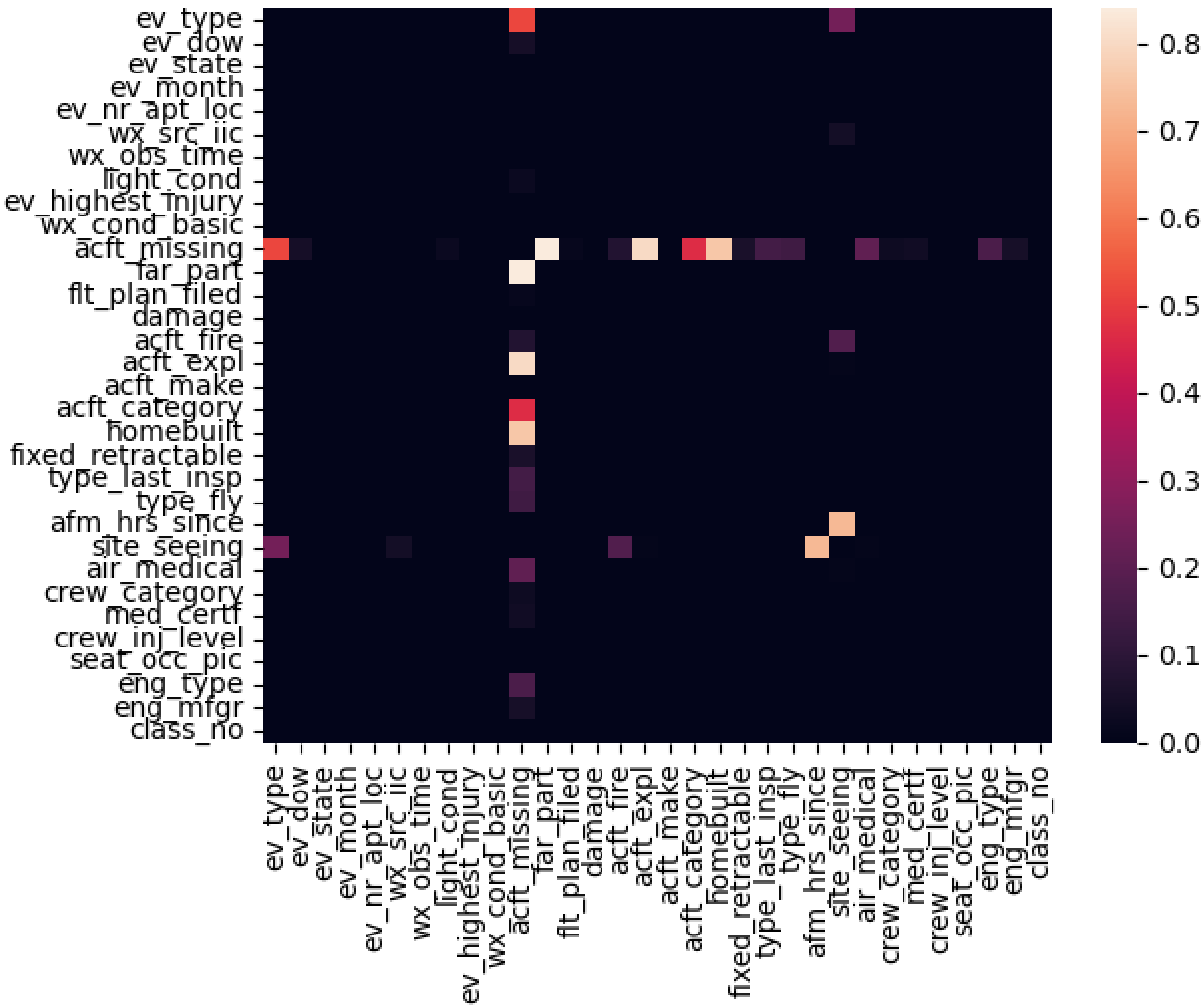
```
length = len(cats)
chi = np.zeros((length, length))
p_val = np.zeros((length, length))
chi[chi == 0] = np.nan
p_val[p_val == 0] = np.nan
for i in range(length):
    for j in range(length):
        ct_table_ind = pd.crosstab(full[cats[i]], full[cats[j]])
        chi2_stat, p, dof, expected= scipy.stats.chi2_contingency(ct_table_ind)
        chi[i,j] = chi2_stat
        p_val[i,j] = p
```

In [83]:

```
sns.heatmap(p_val, xticklabels = cats, yticklabels = cats)
```

Out[83]:

```
<Axes: >
```

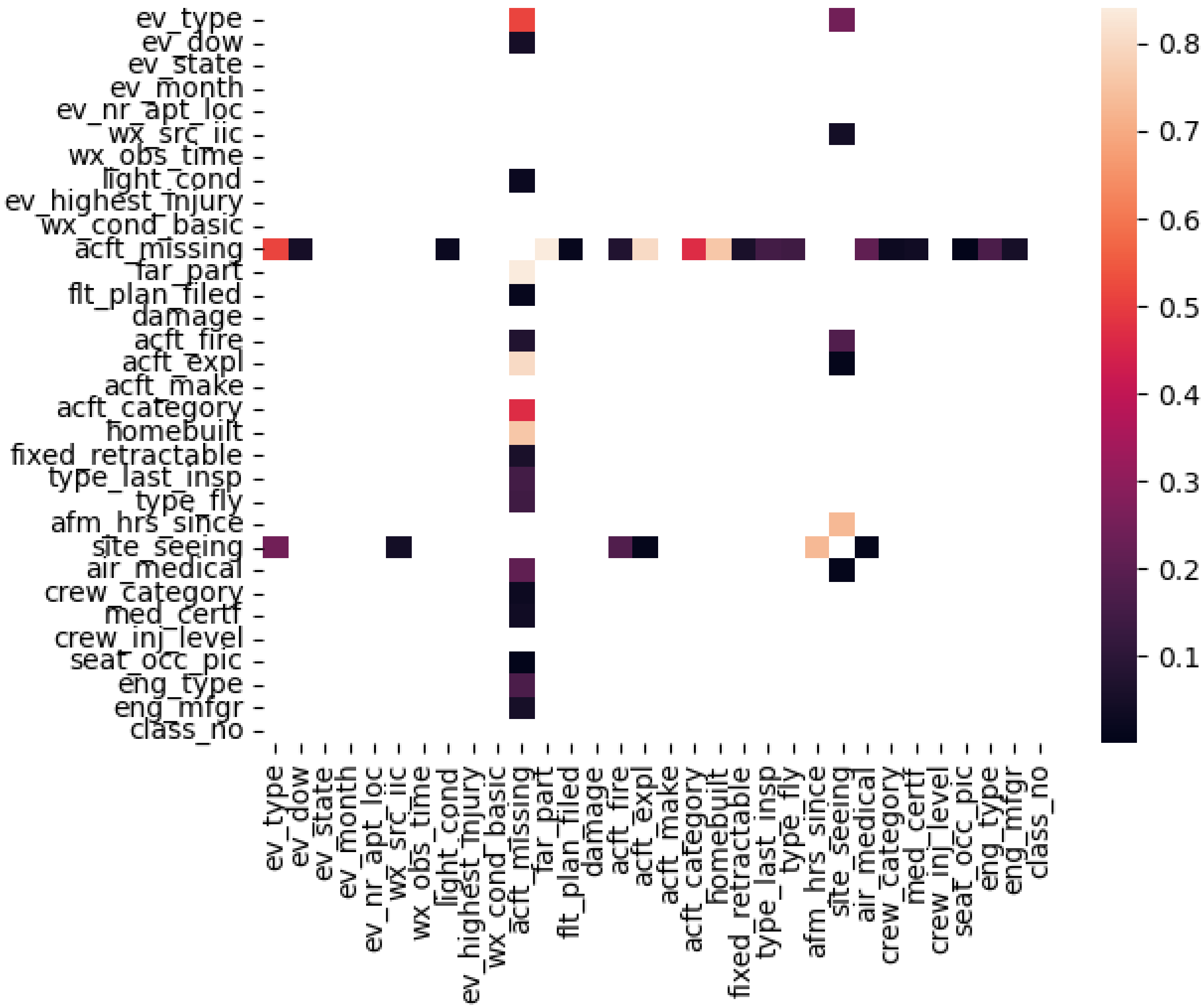


For the above image, the values are P-Values for the Chi^2 Test. For these tests, higher P-Value = more likely to be associated.

In [84]:
```
p_high = np.copy(p_val)
p_high[p_high <= 0.001] = np.nan
```

In [85]:
```
sns.heatmap(p_high, xticklabels = cats, yticklabels = cats)
```

Out[85]: <Axes: >



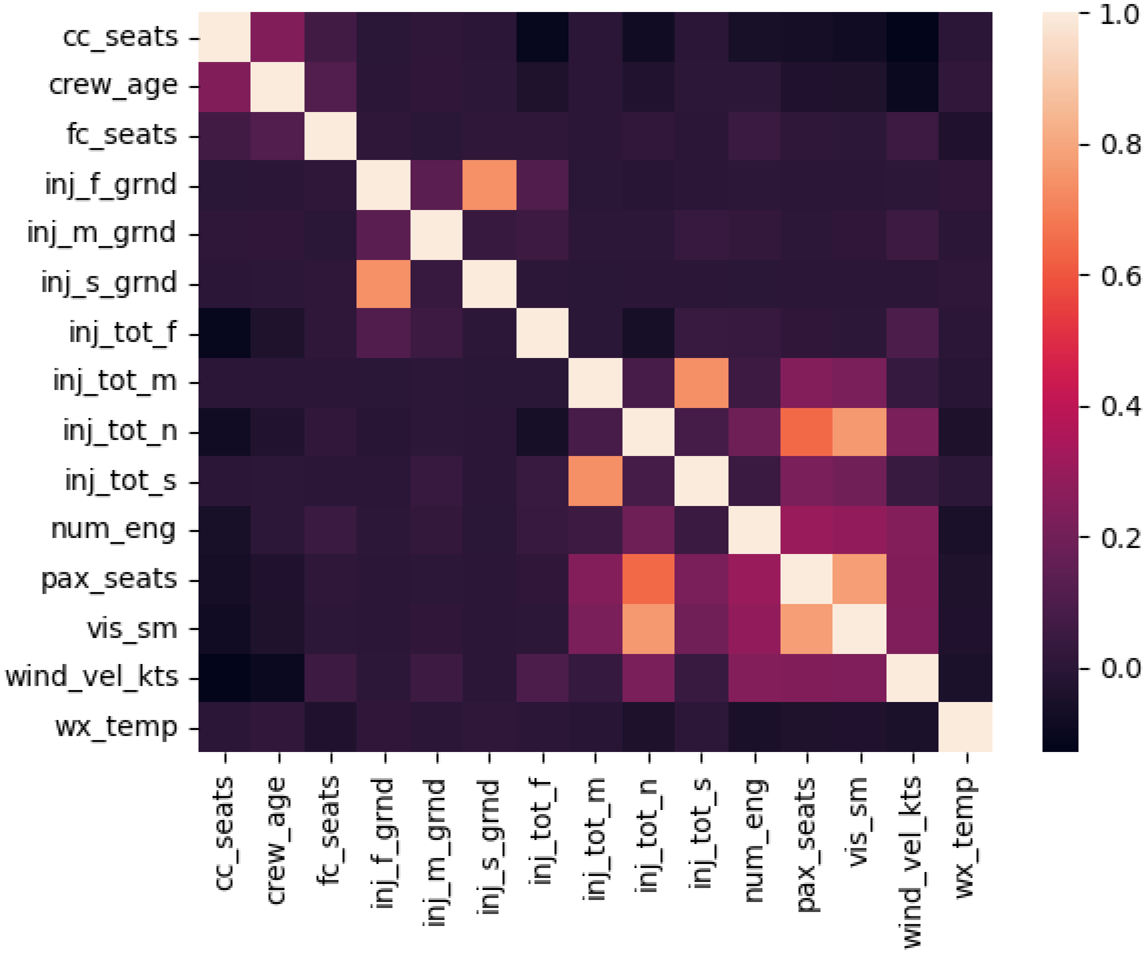
The above image is the same Chi^2 Test with low P-Values dropped for better visibility. As a result of the high association of acft_missing and many other categories, it was dropped.

In [86]:
```
aircraft.drop(columns=['acft_missing'], inplace = True)
```

Correlation Matrix between Continuous Variables

In [89]:
```
con = full.drop(columns = full.columns.difference(conts))
sns.heatmap(con.corr(), xticklabels = conts, yticklabels = conts)
```

Out[89]:
```
<Axes: >
```



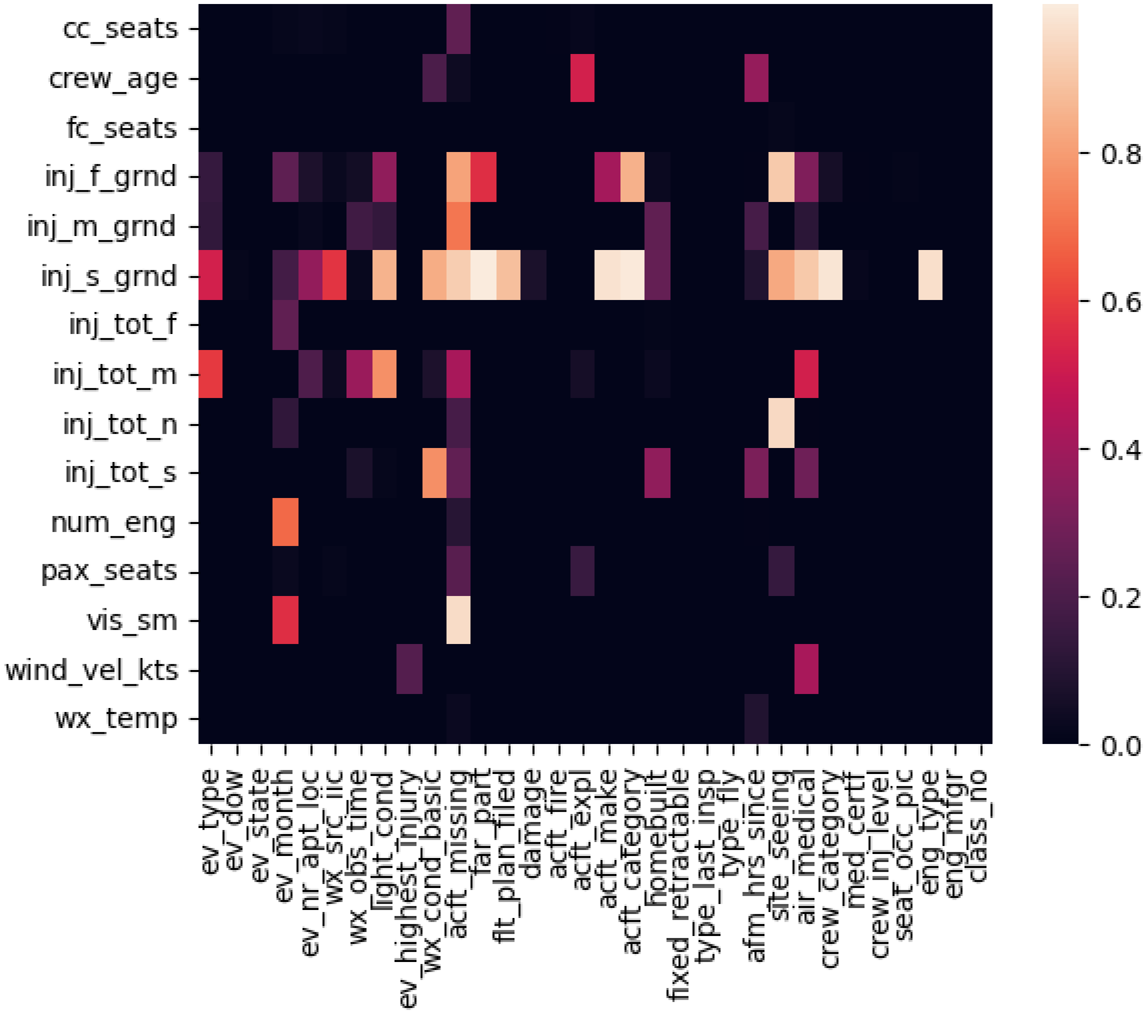
The above image is a Correlation Matrix between all of the continuous variables. For this test, higher value = more likely to be correlated.

One-Way-ANOVA between Categorical and Contiuous Variables

In [90]:
```
ANOVAp = np.zeros((len(conts), len(cats)))
for i in range(len(conts)):
    for j in range(len(cats)):
        formula = conts[i] + ' ~ ' + cats[j]
        result = statsmodels.formula.api.ols(formula, data = full).fit()
        table = statsmodels.api.stats.anova_lm(result)
        ANOVAp[i,j] = table.iat[0, 4]
```

In [91]:
```
sns.heatmap(ANOVAp, xticklabels = cats, yticklabels = conts)
```

Out[91]: <Axes: >

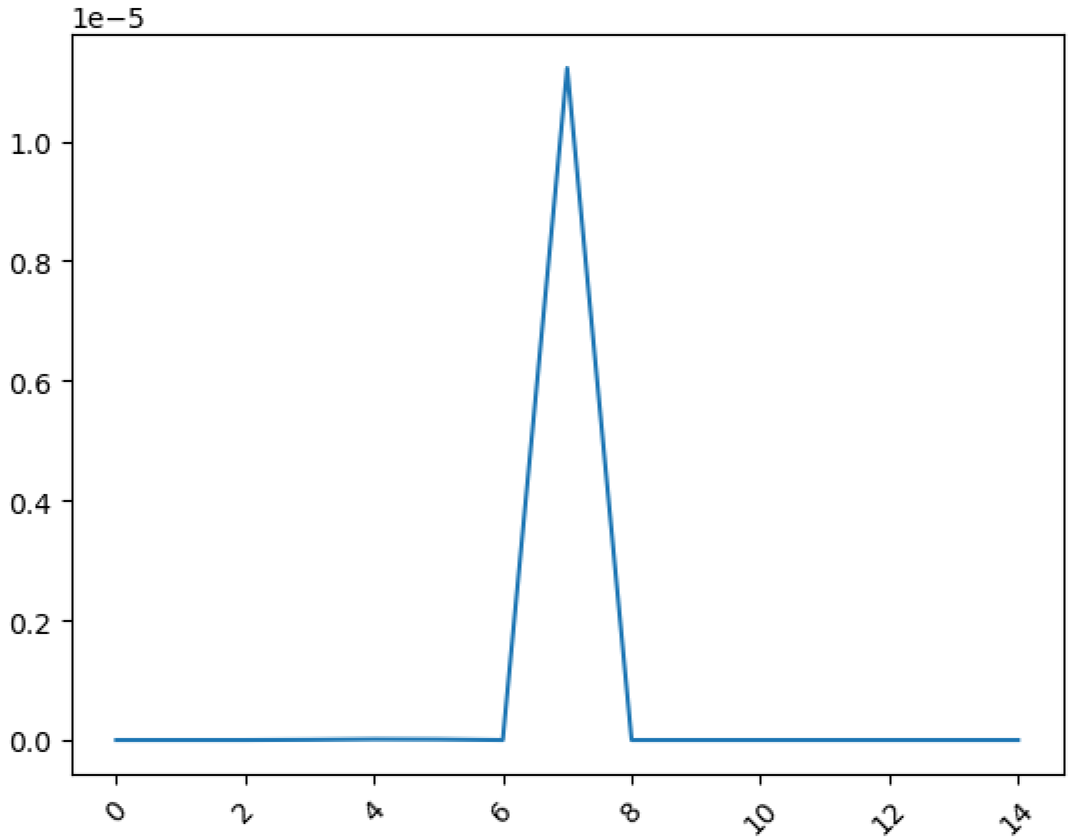


The above image is the One-Way-ANOVA Test between the categorical values and the continuous values. For these tests, lower P-Value means the categorical variable is more likely to influence the mean of the continuous variable.

The below images focus in on the class_no column of the ANOVA. In particular, these low values suggest all of the continuous variables are likely to have an effect on the causes of the aviation accident. The two below plots visualize the likihoods.

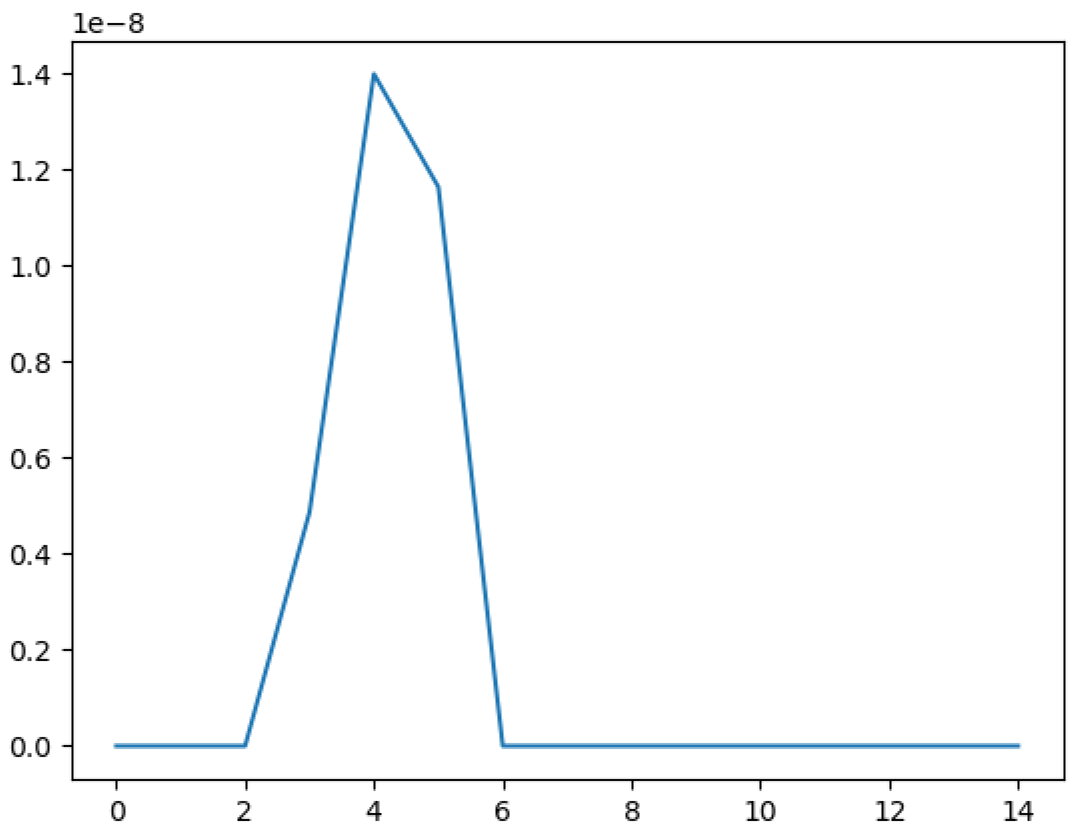
In [94]:
```
plt.xticks(rotation=45)
sns.lineplot(pd.Series(ANOVAp[:,-1]))
```

Out[94]: <Axes: >



In [95]:
```
sns.lineplot(pd.Series(ANOVAp[:,-1]).drop(7))
```

Out[95]: <Axes: >



In [96]:
```
pd.DataFrame(ANOVAp[:,-1], index=conts, columns=['P-Value']).sort_values(by='P-Value', a
```

Out[96]:

	P-Value
inj_tot_m	1.122160e-05

inj_m_grnd	1.397567e-08
inj_s_grnd	1.162576e-08
inj_f_grnd	4.862531e-09
inj_tot_s	5.782207e-23
crew_age	4.389297e-33
fc_seats	3.039993e-42
wind_vel_kts	3.730228e-118
wx_temp	1.092099e-127
vis_sm	7.958926e-188
cc_seats	1.440096e-195
inj_tot_f	3.234089e-206
inj_tot_n	3.745182e-222
pax_seats	6.193768e-241
num_eng	0.000000e+00

Encoding Categoricals

In my research, I found conflicting and confusing information on the many different strategies for encoding high cardinality nominal categorical features. Three, however, were regularly recommended: One-Hot-Encoding, Binary Encoding, and Feature Hash Encoding. I chose to test all three and pick the strategy that gave the best results. For the features that only had two options (Y/N | 1/0 | T/F | etc.), I binary encoded them in all three instances.

Main resources:

https://www.kdnuggets.com/2021/05/deal-with-categorical-data-machine-learning.html

https://kantschants.com/complete-guide-to-encoding-categorical-features#heading-nominal-categorical-features

Making lists of columns by categorical type

In [97]:
```
events.columns
```

Out[97]:
```
Index(['ev_id', 'ev_type', 'ev_dow', 'ev_state', 'ev_month', 'ev_nr_apt_loc',
       'wx_src_iic', 'wx_obs_time', 'light_cond', 'vis_sm', 'wx_temp',
       'wind_vel_kts', 'ev_highest_injury', 'inj_f_grnd', 'inj_m_grnd',
       'inj_s_grnd', 'inj_tot_f', 'inj_tot_m', 'inj_tot_n', 'inj_tot_s',
       'wx_cond_basic'],
      dtype='object')
```

In [98]:
```
evBi = ['ev_type', 'ev_nr_apt_loc', 'wx_cond_basic']
evNom = ['ev_dow', 'ev_state', 'ev_month', 'wx_src_iic', 'light_cond', 'ev_highest_injur
```

In [99]:
```
aircraft.columns
```

Out[99]:
```
Index(['ev_id', 'Aircraft_Key', 'far_part', 'flt_plan_filed', 'damage',
       'acft_fire', 'acft_expl', 'acft_make', 'acft_category', 'homebuilt',
       'fc_seats', 'cc_seats', 'pax_seats', 'num_eng', 'fixed_retractable',
```

```
       'type_last_insp', 'type_fly', 'afm_hrs_since', 'site_seeing',
       'air_medical'],
      dtype='object')
```

In [100…
```
acBi = ['homebuilt', 'fixed_retractable', 'afm_hrs_since', 'site_seeing', 'air_medical']
acNom = ['far_part', 'flt_plan_filed', 'damage', 'acft_fire', 'acft_expl', 'acft_make',
```

In [101…
```
engines.columns
```

Out[101]:
```
Index(['ev_id', 'Aircraft_Key', 'eng_type', 'eng_mfgr'], dtype='object')
```

In [102…
```
engBi = []
engNom = ['eng_type', 'eng_mfgr']
```

In [103…
```
Flight_Crew.columns
```

Out[103]:
```
Index(['ev_id', 'Aircraft_Key', 'crew_category', 'crew_age', 'med_certf',
       'crew_inj_level', 'seat_occ_pic'],
      dtype='object')
```

In [104…
```
fcBi = []
fcNom = ['crew_category', 'med_certf', 'crew_inj_level', 'seat_occ_pic']
```

Findings

In [105…
```
FindE = Findings
```

One-Hot-Encoding (dropping first)

In [106…
```
evOH = pd.get_dummies(events, columns = (evBi + evNom), drop_first = True)
acOH = pd.get_dummies(aircraft, columns = (acBi + acNom), drop_first = True)
engOH = pd.get_dummies(engines, columns = (engBi + engNom), drop_first = True)
fcOH = pd.get_dummies(Flight_Crew, columns = (fcBi + fcNom), drop_first = True)
```

In [107…
```
AcFcOH = acOH.merge(fcOH, on=['ev_id', 'Aircraft_Key'], how='inner')
AcFcEOH = AcFcOH.merge(engOH, on=['ev_id', 'Aircraft_Key'], how='inner')
AcFcEFOH = AcFcEOH.merge(FindE, on=['ev_id', 'Aircraft_Key'], how='inner')
```

```
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/core/reshape/merg
e.py:1214: RuntimeWarning: invalid value encountered in cast
  if not (rk == rk.astype(lk.dtype))[~np.isnan(rk)].all():
```

In [151…
```
AcFcEFOH.shape
```

Out[151]:
```
(70309, 124)
```

Binary Encoding

In [108…
```
evBiE = ce.binary.BinaryEncoder(cols=(evBi+evNom)).fit_transform(events)
acBiE = ce.binary.BinaryEncoder(cols=(acBi+acNom)).fit_transform(aircraft)
engBiE = ce.binary.BinaryEncoder(cols=(engBi+engNom)).fit_transform(engines)
fcBiE = ce.binary.BinaryEncoder(cols=(fcBi+fcNom)).fit_transform(Flight_Crew)
```

In [109…
```
AcFcBi = acBiE.merge(fcBiE, on=['ev_id', 'Aircraft_Key'], how='inner')
AcFcEBi = AcFcBi.merge(engBiE, on=['ev_id', 'Aircraft_Key'], how='inner')
AcFcEFBi = AcFcEBi.merge(FindE, on=['ev_id', 'Aircraft_Key'], how='inner')
```

```
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/core/reshape/merg
e.py:1214: RuntimeWarning: invalid value encountered in cast
  if not (rk == rk.astype(lk.dtype))[~np.isnan(rk)].all():
```

In [152…
```
AcFcEFBi.shape
```

Out[152]:
```
(70309, 74)
```

Feature Hashing (leaving binary categoricals binary encoded)

In [110…
```
evHaE = ce.binary.BinaryEncoder(cols=(evBi)).fit_transform(events)
acHaE = ce.binary.BinaryEncoder(cols=(acBi)).fit_transform(aircraft)
engHaE = ce.binary.BinaryEncoder(cols=(engBi)).fit_transform(engines)
fcHaE = ce.binary.BinaryEncoder(cols=(fcBi)).fit_transform(Flight_Crew)
```

In [111…
```
evHaE = ce.hashing.HashingEncoder(cols=(evNom)).fit_transform(evHaE)
acHaE = ce.hashing.HashingEncoder(cols=(acNom)).fit_transform(acHaE)
engHaE = ce.hashing.HashingEncoder(cols=(engNom)).fit_transform(engHaE)
fcHaE = ce.hashing.HashingEncoder(cols=(fcNom)).fit_transform(fcHaE)
```

In [112…
```
AcFcHa = acHaE.merge(fcHaE, on=['ev_id', 'Aircraft_Key'], how='inner')
AcFcEHa = AcFcHa.merge(engHaE, on=['ev_id', 'Aircraft_Key'], how='inner')
AcFcEFHa = AcFcEHa.merge(FindE, on=['ev_id', 'Aircraft_Key'], how='inner')
```

```
/Users/aidencamilleri/opt/anaconda3/lib/python3.9/site-packages/pandas/core/reshape/merg
e.py:1214: RuntimeWarning: invalid value encountered in cast
  if not (rk == rk.astype(lk.dtype))[~np.isnan(rk)].all():
```

In [153…
```
AcFcEFHa.shape
```

Out[153]:
```
(70309, 45)
```

Modelling (using Random Forest Classifier)

For modelling, I chose an 85%, 15% train-test-split to maintain a large training set, while still having plenty of records for validation. With the 85% test set, I then performed 10-fold cross validation to train a sklearn Random Forest Classifier for 17-class classification. Finally, the top performing model -the Binary Encoded model- predicted the top 4 causes for the accident.

Train-Test-Split

In [113…
```
evOH_train, evOH_test = train_test_split(evOH, test_size = 0.15)
evBi_train, evBi_test = train_test_split(evBiE, test_size = 0.15)
evHa_train, evHa_test = train_test_split(evHaE, test_size = 0.15)
```

Merging tables

In [114…
```
OH_train = evOH_train.merge(AcFcEFOH, on=['ev_id'], how='inner')
OH_test = evOH_test.merge(AcFcEFOH, on=['ev_id'], how='inner')
Bi_train = evBi_train.merge(AcFcEFBi, on=['ev_id'], how='inner')
Bi_test = evBi_test.merge(AcFcEFBi, on=['ev_id'], how='inner')
Ha_train = evHa_train.merge(AcFcEFHa, on=['ev_id'], how='inner')
Ha_test = evHa_test.merge(AcFcEFHa, on=['ev_id'], how='inner')
```

```
/var/folders/48/qxtv7xwj5dv_k99_l9rccfh40000gn/T/ipykernel_50500/29571477.py:5: FutureWa
rning: Passing 'suffixes' which cause duplicate columns {'col_6_y', 'col_2_y', 'col_7_
y', 'col_5_y', 'col_1_y', 'col_3_y', 'col_0_y', 'col_4_y'} in the result is deprecated a
nd will raise a MergeError in a future version.
  Ha_train = evHa_train.merge(AcFcEFHa, on=['ev_id'], how='inner')
/var/folders/48/qxtv7xwj5dv_k99_l9rccfh40000gn/T/ipykernel_50500/29571477.py:6: FutureWa
rning: Passing 'suffixes' which cause duplicate columns {'col_6_y', 'col_2_y', 'col_7_
y', 'col_5_y', 'col_1_y', 'col_3_y', 'col_0_y', 'col_4_y'} in the result is deprecated a
nd will raise a MergeError in a future version.
  Ha_test = evHa_test.merge(AcFcEFHa, on=['ev_id'], how='inner')
```

Creating X and y

In [115…

```
extras = ['ev_id', 'Aircraft_Key', 'finding_no', 'category_no', 'subcategory_no', 'class
ex = ['ev_id', 'Aircraft_Key', 'finding_no', 'category_no', 'subcategory_no']
OHX = OH_train[OH_train.columns.difference(extras)].to_numpy()
OHy = OH_train['class_no'].to_numpy()
BiX = Bi_train[Bi_train.columns.difference(extras)].to_numpy()
Biy = Bi_train['class_no'].to_numpy()
HaX = Ha_train[Ha_train.columns.difference(extras)].to_numpy()
Hay = Ha_train['class_no'].to_numpy()
```

Creating X for test sets

In [116…

```
OHXtest = OH_test[OH_test.columns.difference(extras)].to_numpy()
BiXtest = Bi_test[Bi_test.columns.difference(extras)].to_numpy()
HaXtest = Ha_test[Ha_test.columns.difference(extras)].to_numpy()
```

Creating Random Forests

In [117…

```
RfOH = RandomForestClassifier()
RfBi = RandomForestClassifier()
RfHa = RandomForestClassifier()
```

Setting up 10-Fold Cross Validation and Training

In [118…

```
OHScores = []
kFoldOH = KFold(n_splits=10)
for train_index,test_index in kFoldOH.split(OHX):
    print("Train Index: ", train_index, "\n")
    print("Test Index: ", test_index)

    OHX_train, OHX_test, OHy_train, OHy_test = OHX[train_index], OHX[test_index], OHy[tr
    RfOH.fit(OHX_train, OHy_train)
    OHScores.append(RfOH.score(OHX_test, OHy_test))
RfOH.fit(OHX_train,OHy_train)
OHScores.append(RfOH.score(OHX_test,OHy_test))
print(np.mean(OHScores))
cross_val_score(RfOH, OHX, OHy, cv=10)
```

```
Train Index:  [ 5973  5974  5975 ... 59724 59725 59726]

Test Index:  [    0     1     2 ...  5970  5971  5972]
Train Index:  [    0     1     2 ... 59724 59725 59726]

Test Index:  [ 5973  5974  5975 ... 11943 11944 11945]
Train Index:  [    0     1     2 ... 59724 59725 59726]

Test Index:  [11946 11947 11948 ... 17916 17917 17918]
```

```
Train Index:  [    0     1     2 ... 59724 59725 59726]

Test Index:  [17919 17920 17921 ... 23889 23890 23891]
Train Index:  [    0     1     2 ... 59724 59725 59726]

Test Index:  [23892 23893 23894 ... 29862 29863 29864]
Train Index:  [    0     1     2 ... 59724 59725 59726]

Test Index:  [29865 29866 29867 ... 35835 35836 35837]
Train Index:  [    0     1     2 ... 59724 59725 59726]

Test Index:  [35838 35839 35840 ... 41808 41809 41810]
Train Index:  [    0     1     2 ... 59724 59725 59726]

Test Index:  [41811 41812 41813 ... 47780 47781 47782]
Train Index:  [    0     1     2 ... 59724 59725 59726]

Test Index:  [47783 47784 47785 ... 53752 53753 53754]
Train Index:  [    0     1     2 ... 53752 53753 53754]

Test Index:  [53755 53756 53757 ... 59724 59725 59726]
0.25391558589593416
```

Out[118]:
```
array([0.23673196, 0.21865059, 0.21463251, 0.20575925, 0.20425247,
       0.21312573, 0.21446509, 0.20445412, 0.21701273, 0.24012056])
```

In [119…

```
BiScores = []
kFoldBi = KFold(n_splits=10)
for train_index,test_index in kFoldOH.split(BiX):
    print("Train Index: ", train_index, "\n")
    print("Test Index: ", test_index)

    BiX_train, BiX_test, Biy_train, Biy_test = BiX[train_index], BiX[test_index], Biy[tr
    RfBi.fit(BiX_train, Biy_train)
    BiScores.append(RfBi.score(BiX_test, Biy_test))
RfBi.fit(BiX_train,Biy_train)
BiScores.append(RfBi.score(BiX_test,Biy_test))
print(np.mean(BiScores))
cross_val_score(RfBi, BiX, Biy, cv=10)
```

```
Train Index:  [ 5970  5971  5972 ... 59695 59696 59697]

Test Index:  [   0     1     2 ... 5967 5968 5969]
Train Index:  [    0     1     2 ... 59695 59696 59697]

Test Index:  [ 5970  5971  5972 ... 11937 11938 11939]
Train Index:  [    0     1     2 ... 59695 59696 59697]

Test Index:  [11940 11941 11942 ... 17907 17908 17909]
Train Index:  [    0     1     2 ... 59695 59696 59697]

Test Index:  [17910 17911 17912 ... 23877 23878 23879]
Train Index:  [    0     1     2 ... 59695 59696 59697]

Test Index:  [23880 23881 23882 ... 29847 29848 29849]
Train Index:  [    0     1     2 ... 59695 59696 59697]

Test Index:  [29850 29851 29852 ... 35817 35818 35819]
Train Index:  [    0     1     2 ... 59695 59696 59697]

Test Index:  [35820 35821 35822 ... 41787 41788 41789]
Train Index:  [    0     1     2 ... 59695 59696 59697]

Test Index:  [41790 41791 41792 ... 47757 47758 47759]
Train Index:  [    0     1     2 ... 59695 59696 59697]
```

```
Test Index:  [47760 47761 47762 ... 53726 53727 53728]
Train Index:  [    0     1     2 ... 53726 53727 53728]

Test Index:  [53729 53730 53731 ... 59695 59696 59697]
0.247079628681235
```

Out[119]:
```
array([0.24723618, 0.22847571, 0.22378559, 0.22144054, 0.21624791,
       0.23500838, 0.21356784, 0.21959799, 0.22080751, 0.24426202])
```

In [120…
```
HaScores = []
kFoldHa = KFold(n_splits=10)
for train_index,test_index in kFoldHa.split(HaX):
    print("Train Index: ", train_index, "\n")
    print("Test Index: ", test_index)

    HaX_train, HaX_test, Hay_train, Hay_test = HaX[train_index], HaX[test_index], Hay[tr
    RfHa.fit(HaX_train, Hay_train)
    HaScores.append(RfHa.score(HaX_test, Hay_test))
RfHa.fit(HaX_train,Hay_train)
HaScores.append(RfHa.score(HaX_test,Hay_test))
print(np.mean(HaScores))
cross_val_score(RfHa, HaX, Hay, cv=10)
```

```
Train Index:  [ 5963  5964  5965 ... 59618 59619 59620]

Test Index:  [   0    1    2 ... 5960 5961 5962]
Train Index:  [    0     1     2 ... 59618 59619 59620]

Test Index:  [ 5963  5964  5965 ... 11922 11923 11924]
Train Index:  [    0     1     2 ... 59618 59619 59620]

Test Index:  [11925 11926 11927 ... 17884 17885 17886]
Train Index:  [    0     1     2 ... 59618 59619 59620]

Test Index:  [17887 17888 17889 ... 23846 23847 23848]
Train Index:  [    0     1     2 ... 59618 59619 59620]

Test Index:  [23849 23850 23851 ... 29808 29809 29810]
Train Index:  [    0     1     2 ... 59618 59619 59620]

Test Index:  [29811 29812 29813 ... 35770 35771 35772]
Train Index:  [    0     1     2 ... 59618 59619 59620]

Test Index:  [35773 35774 35775 ... 41732 41733 41734]
Train Index:  [    0     1     2 ... 59618 59619 59620]

Test Index:  [41735 41736 41737 ... 47694 47695 47696]
Train Index:  [    0     1     2 ... 59618 59619 59620]

Test Index:  [47697 47698 47699 ... 53656 53657 53658]
Train Index:  [    0     1     2 ... 53656 53657 53658]

Test Index:  [53659 53660 53661 ... 59618 59619 59620]
0.24701514042664152
```

Out[120]:
```
array([0.23629046, 0.21066756, 0.20580342, 0.227105  , 0.21335122,
       0.21938947, 0.21586716, 0.21502851, 0.22324723, 0.24136196])
```

Gathering One Hot Predictions

In [121…
```
OHpredicted = RfOH.predict_proba(OHXtest)
```

In [122…
```
bigArr = []
for line in OHpredicted:
    arr = []
```

```
    arr.append(np.argsort(line)[-1])
    arr.append(np.argsort(line)[-2])
    arr.append(np.argsort(line)[-3])
    arr.append(np.argsort(line)[-4])
    bigArr.append(arr)
```

In [123…

```
classes = {0:'11' , 1:'12' , 2:'13' , 3:'14' , 4:'15' , 5:'16' , 6:'17' , 7:'21' , 8:'22
OHpredicts = pd.DataFrame(bigArr, columns = ['PFinding1', 'PFinding2', 'PFinding3', 'PFi
OHpredicts = OHpredicts.to_numpy()
```

In [124…

```
OHtest = []
for line in range(round(OHpredicts.size/4)):
    OHtest.append(np.isin(OH_test['class_no'].values[line], OHpredicts[line]))
OH_test['CorPredict'] = pd.DataFrame(OHtest)
```

Gathering Binary Predictions

In [125…

```
Bipredicted = RfBi.predict_proba(BiXtest)
```

In [126…

```
bigArr = []
for line in Bipredicted:
    arr = []
    arr.append(np.argsort(line)[-1])
    arr.append(np.argsort(line)[-2])
    arr.append(np.argsort(line)[-3])
    arr.append(np.argsort(line)[-4])
    bigArr.append(arr)
```

In [127…

```
classes = {0:'11' , 1:'12' , 2:'13' , 3:'14' , 4:'15' , 5:'16' , 6:'17' , 7:'21' , 8:'22
Bipredicts = pd.DataFrame(bigArr, columns = ['PFinding1', 'PFinding2', 'PFinding3', 'PFi
Bipredicts = Bipredicts.to_numpy()
```

In [128…

```
Bitest = []
for line in range(round(Bipredicts.size/4)):
    Bitest.append(np.isin(Bi_test['class_no'].values[line], Bipredicts[line]))
Bi_test['CorPredict'] = pd.DataFrame(Bitest)
```

Gathering Feature Hashing Predictions

In [129…

```
Hapredicted = RfHa.predict_proba(HaXtest)
```

In [130…

```
bigArr = []
for line in Hapredicted:
    arr = []
    arr.append(np.argsort(line)[-1])
    arr.append(np.argsort(line)[-2])
    arr.append(np.argsort(line)[-3])
    arr.append(np.argsort(line)[-4])
    bigArr.append(arr)
```

In [131…

```
classes = {0:'11' , 1:'12' , 2:'13' , 3:'14' , 4:'15' , 5:'16' , 6:'17' , 7:'21' , 8:'22
Hapredicts = pd.DataFrame(bigArr, columns = ['PFinding1', 'PFinding2', 'PFinding3', 'PFi
Hapredicts = Hapredicts.to_numpy()
```

In [132…

```
Hatest = []
for line in range(round(Hapredicts.size/4)):
    Hatest.append(np.isin(Ha_test['class_no'].values[line], Hapredicts[line]))
Ha_test['CorPredict'] = pd.DataFrame(Hatest)
```

Results

The results of the model validation against the test set are below.

Accuracy: 82.94%

Recall: 71.05%

Precision (Low-Bound): 46.43%

Precision (High-Bound): 71.05%

In [133…

```
print('One-Hot Accuracy: ' + str(OH_test['CorPredict'].sum() / OH_test['CorPredict'].cou
print('Binary Accuracy: ' + str(Bi_test['CorPredict'].sum() / Bi_test['CorPredict'].coun
print('Feature Hashing Accuracy: ' + str(Ha_test['CorPredict'].sum() / Ha_test['CorPredi
```

```
One-Hot Accuracy: 0.7084672084672085
Binary Accuracy: 0.7132221279803977
Feature Hashing Accuracy: 0.7076160179640718
```

Calculating TP, TN, FP, FN

In [134…

```
Bi_test['pred1'] = Bipredicts[:,0]
Bi_test['pred2'] = Bipredicts[:,1]
Bi_test['pred3'] = Bipredicts[:,2]
Bi_test['pred4'] = Bipredicts[:,3]
```

In [156…

```
Bi_test['TP'] = np.nan
Bi_test['FP'] = np.nan
Bi_test['FN'] = np.nan
Bi_test['TN'] = np.nan
for group in Bi_test.groupby(by=['ev_id', 'Aircraft_Key']).groups:
    df = Bi_test.groupby(by=['ev_id', 'Aircraft_Key']).get_group(group)
    index = Bi_test[(Bi_test['ev_id'] == group[0]) & (Bi_test['Aircraft_Key'] == group[1
    predn = set(df[['pred1', 'pred2', 'pred3', 'pred4']].iloc[0])
    Bi_test.loc[index, 'TP'] = len(set(df.class_no).intersection(predn))
    Bi_test.loc[index, 'FP'] = len(predn.difference(set(df.class_no))) #- (len(predn) -
    Bi_test.loc[index, 'FN'] = len(set(df.class_no).difference(predn))
    Bi_test.loc[index, 'TN'] = len((set(classes.values()).difference(set(df.class_no))).
```

Accuracy, Recall, and Precision

In [157…

```
print("Accuracy: " + str((Bi_test.TP.sum() + Bi_test.TN.sum())/(17*Bi_test.TP.count())))
print("Recall: " + str(Bi_test.TP.sum() / (Bi_test.TP.sum() + Bi_test.FN.sum())))
print("Precision: " + str(Bi_test.TP.sum() / (Bi_test.TP.sum() + Bi_test.FP.sum())))
```

```
Accuracy: 0.829433384889155
Recall: 0.7104845687914624
Precision: 0.4642823485062671
```

Class Accuracies

In [139…

```
clsAcc = (Bi_test.groupby(by='class_no').TP.sum() + Bi_test.groupby(by='class_no').TN.su
```

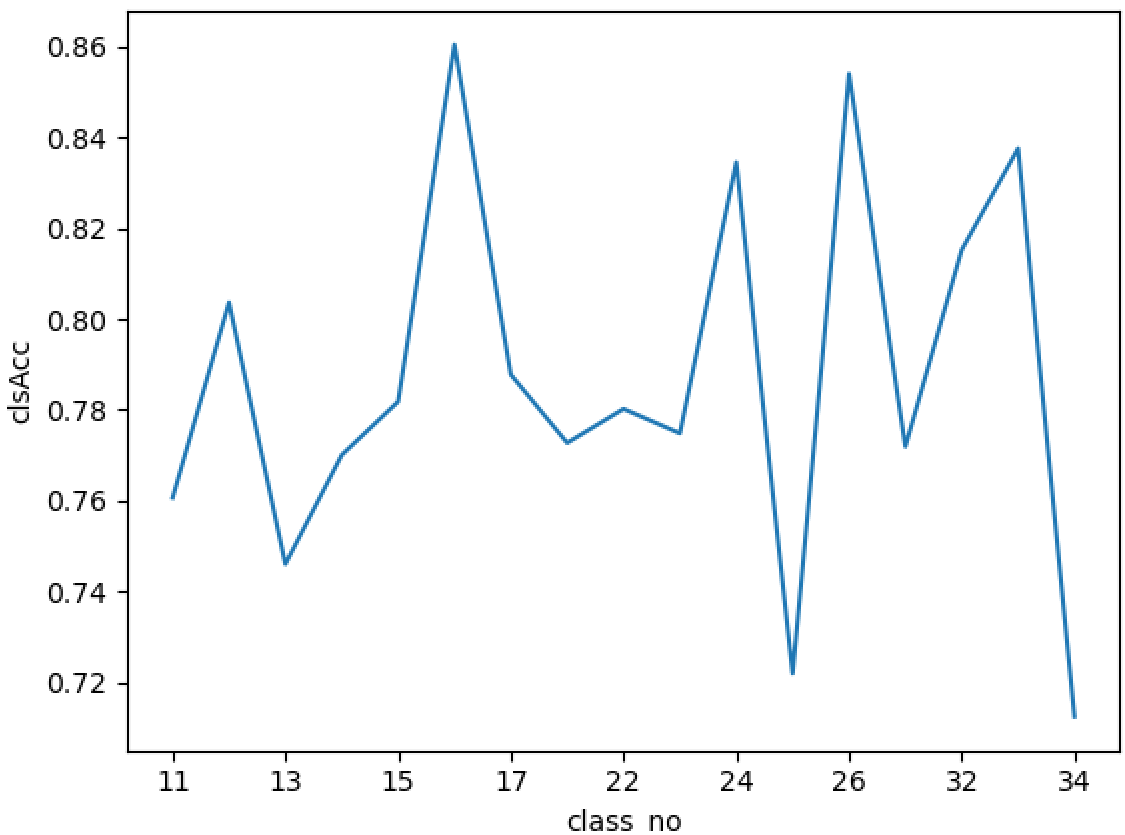
In [140…

```
clsAcc
```

Out[140]:
```
class_no
11    0.760649
12    0.803666
13    0.746014
14    0.770053
15    0.781810
16    0.860501
17    0.787784
21    0.772727
22    0.780211
23    0.774837
24    0.834554
25    0.721925
26    0.854059
31    0.771909
32    0.815260
33    0.837617
34    0.712418
dtype: float64
```

In [158…
```
plt.ylabel('clsAcc')
clsAcc.plot()
```

Out[158]: <Axes: xlabel='class_no', ylabel='clsAcc'>



Class Recalls

In [142…
```
clsRec = Bi_test.groupby(by='class_no').TP.sum() / (Bi_test.groupby(by='class_no').TP.su
```

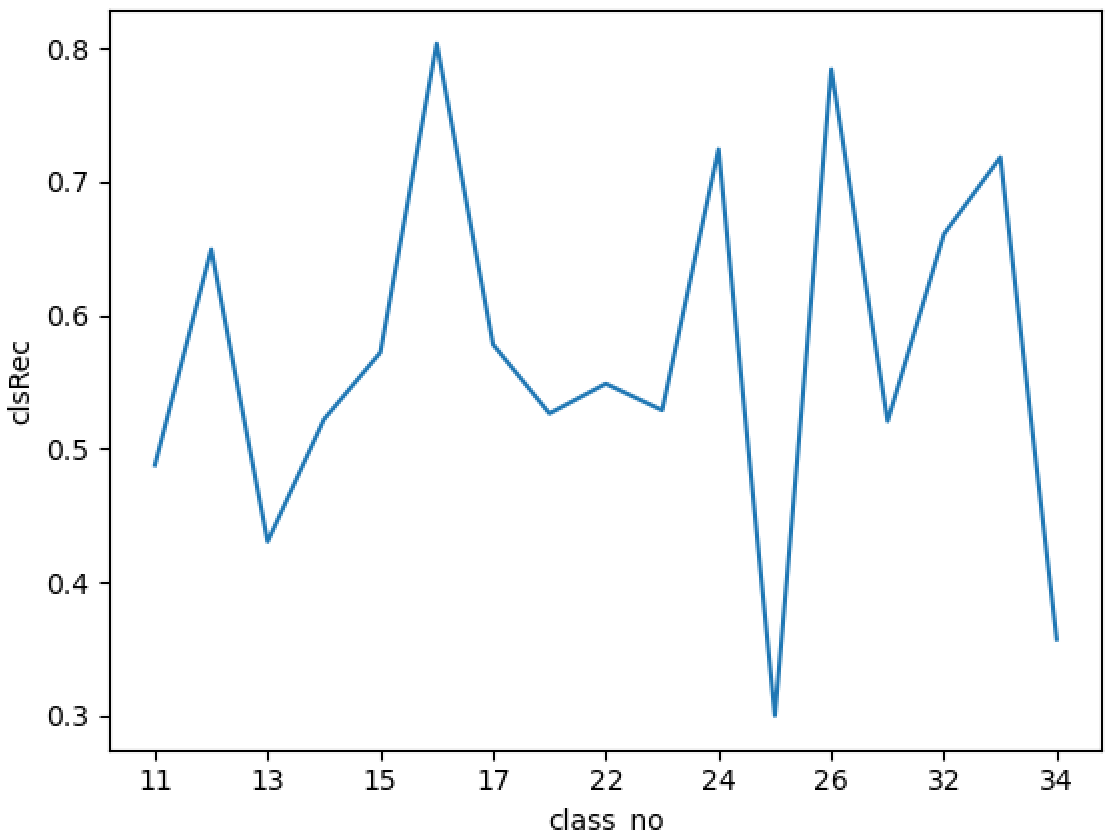
In [143…
```
clsRec
```

Out[143]:
```
class_no
11    0.487654
12    0.649226
13    0.430328
14    0.521930
```

```
15    0.571984
16    0.803384
17    0.577960
21    0.526316
22    0.548554
23    0.528757
24    0.724166
25    0.300000
26    0.784058
31    0.520548
32    0.660693
33    0.718169
34    0.357143
dtype: float64
```

In [159…
```
plt.ylabel('clsRec')
clsRec.plot()
```

Out[159]:
```
<Axes: xlabel='class_no', ylabel='clsRec'>
```



Class Precisions

In [145…
```
clsPrec = Bi_test.groupby(by='class_no').TP.sum() / (Bi_test.groupby(by='class_no').TP.s
```

In [146…
```
clsPrec
```

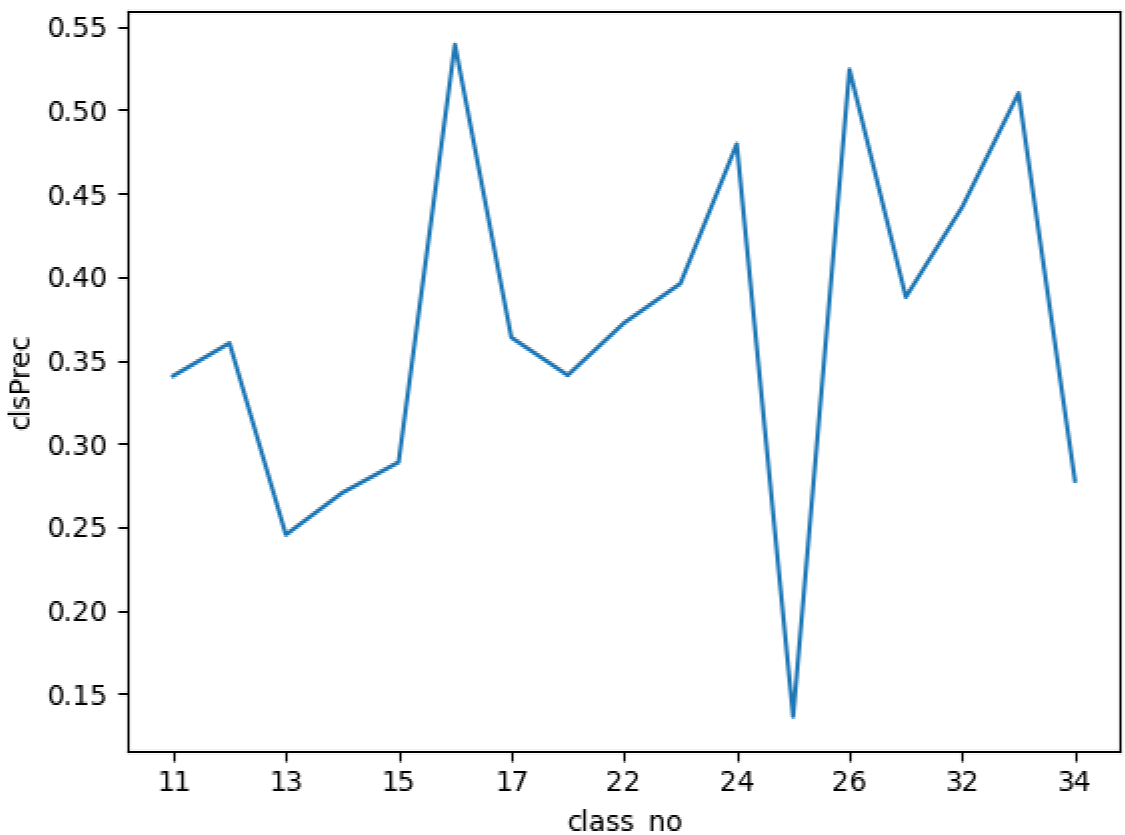
Out[146]:
```
class_no
11    0.340517
12    0.360190
13    0.245327
14    0.270455
15    0.288802
16    0.539055
17    0.363569
21    0.340909
22    0.372233
```

```
23    0.395833
24    0.479491
25    0.136364
26    0.524095
31    0.387755
32    0.441693
33    0.510020
34    0.277778
dtype: float64
```

In [160…

```python
plt.ylabel('clsPrec')
clsPrec.plot()
```

Out[160]: <Axes: xlabel='class_no', ylabel='clsPrec'>

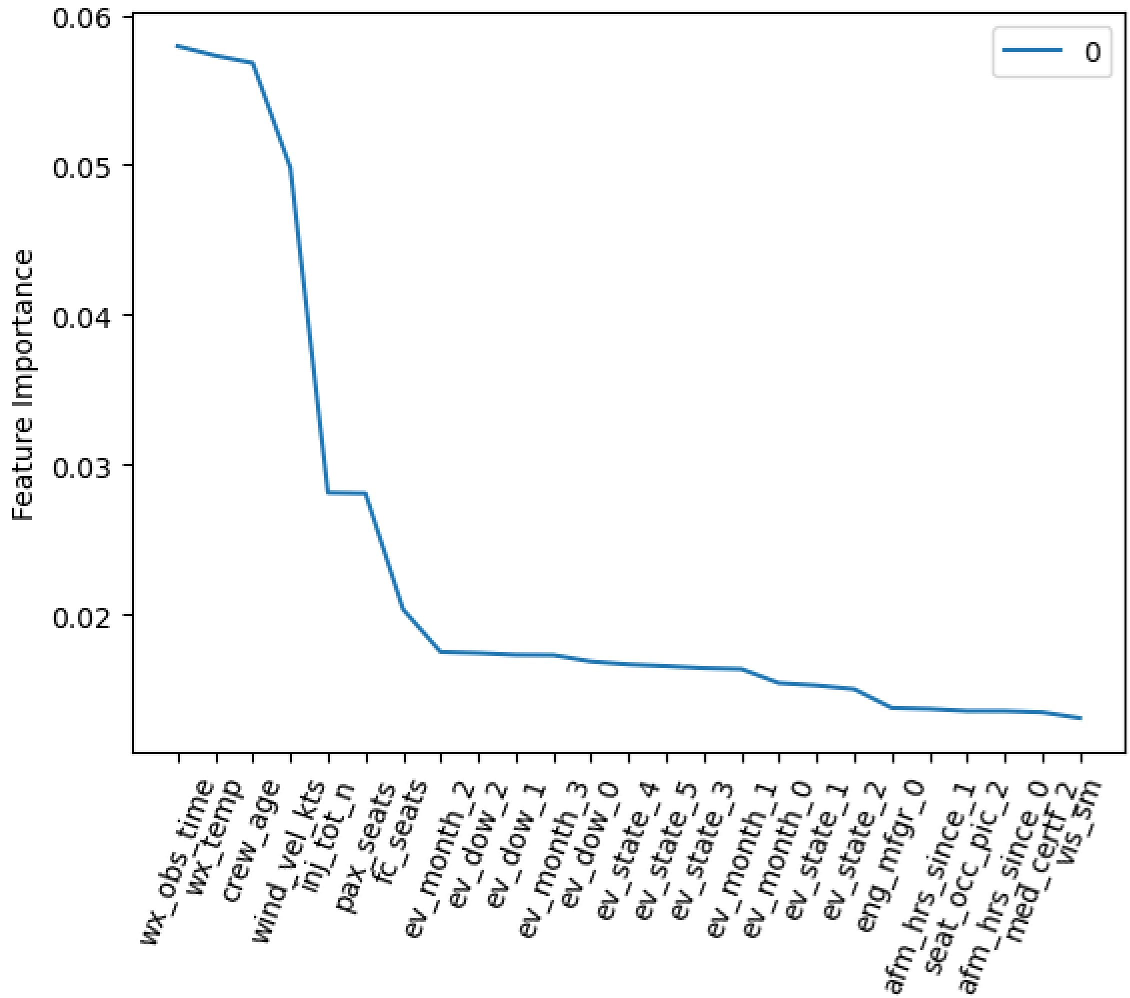


Feature Importances

In [161…

```python
plt.ylabel("Feature Importance")
plt.xticks(rotation=70)
sns.lineplot(pd.DataFrame(RfBi.feature_importances_, index = Bi_train[Bi_train.columns.d
```

Out[161]: <Axes: ylabel='Feature Importance'>



Category Performance

The model was most accurate at predicting classes 12, 16, 24, 32, 33.

The model had it's highest recall predicting classes 16, 24, 26, 32, 33.

Finally, the model was most precise on classes 16, 24, 26, 32, 33.

- 12 - Aircraft-Aircraft Systems
- 16 - Aircraft-Aircraft Oper/Perf/Capability
- 24 - Personnel-Action/Decision
- 26 - Personnel-Task Performance
- 32 - Environmental-Physical Environment
- 33 - Environmental-Conditions/Weather/Phenomena

Interpretation

To address my two main questions:

1. Can surface-level features be used to predict the cause of the accident?

In general, yes. The accuracy above 80% shows that the surface level features do hold information that indicates a potential cause for an accident. In addition, the continuous features seem to show more about the particular causes, so the model may be more accurate with features that I had to drop due to data cleanliness, like air pressure.

2. What does this tell us about the causes of accidents?

There are interactions between temperature, age of crew, wind velocity, total non-injuries, number of passenger seats, number of flight crew seats, day of week of the flight, and month of the flight that influence the cause of the accident. More research must be done on the particulars of the accident, but it is not unreasonable to assume that a summer month, weekend day of week, high age, and low number of flight crew will result in a higher chance of an accident caused by human error. Conversely, high temperatures reduce air density and lower overall aircraft performance, and high wind velocities can overspeed the aircraft. Therefore, those may point to environmental factors as the cause of the accident. Adding strength to these hypotheses, the model has the highest scores in predicting the causes that would be most closely associated with each.

Model Issues

The model was best at predicting the most common causes of the aviation accidents. While, the model still may be useful, this suggests it may have trouble identifying the more obsure causes for accidents. Possible future solutions could include upsampling the underrepresented causes, downsampling the most common causes, or using a model that allows for class weighting.

Acknowledgements

Finally, I would like to thank Dr. Shashi Jha (College of Charleston) for his guidance on this report; Jess Thomas (NTSB) for sharing the causes corresponding to the finding codes; and Xiaoge Zhang, Prabhakar Srinivasan, and Sankaran Mahadevan for inspiring this report.