To-do:

1. ~~Adjust the code for the clean "DOF" issue -- done~~
2. ~~Fix the issue -- adjust the isues with the at_risk date -- done~~
3. ~~Fix the "Free Time Issue" -- done~~
4. **Address issues with negative time to recidivate (current strategy may not be perfect) -- there are some rows where the next date of offense occurs BEFORE the at_risk_dt for the previous offense. How do we deal with these anomalies?**
5. ~~Make sure that the final exported dataframe is at the id_variable, dos level -- done~~
6. Document (in Word Doc or even at the end of this file the logic used at each step of the process and why decisions where made)

Ongoing Questions:

1. ~~Should we subset the data to where dof < OR equal to dos or JUST keep data that is strictly less then (dof < dos)?~~
2. What additional variables besides JP_MIN, OGS, need to be aggregated at the ID_VARIABLE, DOS level (perhaps PRS?() Is this something that Audrey has already done for the demographics dataset?
3. **There are some rows where the next date of offense occurs BEFORE the at_risk_dt for the previous offense. How do we deal with these anomalies?**

Load Data

In [ ]:
```
#this jupyter notebook is essentially the same as the "recidivism-check" notebook, just cleaned up a bit (hence the name)
#import required libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import sqlite3

#get the folder path for this data
pa_sentencing_path = os.path.dirname(os.path.dirname(os.path.dirname(os.getcwd())))

#read in the correct data file (need to read in this file because of the additional columns it has)
psc_trimmed = pd.read_csv(os.path.join(pa_sentencing_path, "Project", "data", "PSC_data_trimmed_v1.csv"))
```

```
/opt/anaconda3/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3063: DtypeWarning: Columns (19,29,45,46,48,49,50,51,52,53,59,63,64,66) have mixed types.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [ ]:
```
# copying the original loaded data to a working data frame to use and compare with later
#df = df_tbl_db.copy() #if accessing the database

df = psc_trimmed.copy() # if accessing the psc_trimmed file directly


#change column names to uppercase
df.columns = df.columns.str.upper()
```

In [ ]:
```
df.head() #inspect the dataset
```

Out[ ]:

	JPR_ID	OFF_SEX	OFF_RACE	DOFAGE	OTN	OFN_TITLE	OFN_COUNT	OFN_LABEL
0	640001	F	White	36.689938	H182628-5	18	1	Corruption of Minors - when of a sexual nature
1	642480	M	White	18.540726	0	75	1	DUI - M-2
2	660434	M	White	36.914442	H3618344	75	1	DUI - M-2
3	628940	M	Black	22.297057	G0816126	18	1	Simple Assault
4	594048	M	White	40.087611	H240127-6	75	1	DUI - M-1

5 rows × 69 columns

Convert Dates

In [ ]:
```
#convert date strings to datetime variable
df[['DOF','DOS']] = df[['DOF','DOS']].apply(pd.to_datetime,format="%d %b %y")
```

In [ ]:
```
# extracting out the just the year from the date to be used later
df['DOF_YEAR'] = pd.DatetimeIndex(df['DOF']).year
df['DOS_YEAR'] = pd.DatetimeIndex(df['DOS']).year
```

In [ ]:
```
#checking the range of values for the DOF and DOS variables
print("The minimum date of offense in the dataset is: {}".format(df[["DOF"]].min()[0]))
print("The maximum date of offense in the dataset is: {}".format(df[["DOF"]].max()[0]))
print("The minimum date of sentencing in the dataset is: {}".format(df[["DOS"]].min()[0]))
print("The maximum date of sentencing in the dataset is: {}".format(df[["DOS"]].max()[0]))
```

```
The minimum date of offense in the dataset is: 1984-11-14 00:00:00
The maximum date of offense in the dataset is: 2020-05-08 00:00:00
The minimum date of sentencing in the dataset is: 2001-01-01 00:00:00
The maximum date of sentencing in the dataset is: 2019-12-31 00:00:00
```

Note: As shown in the above code chunk, there isn't anamolous behavior in the date ranges (i.e. a date in the year 1909 or 2090) for the date of offense (DOF) or date of sentence (DOS) variables -- therefore, an additional date correction was not applied in this case.

Clean DOS > DOF

Note: group offense by ID_VAR, JPR_ID, MIN(DOF) to get the first DOF associated for a single JPR_ID

In [ ]:
```
 #count how many values of DOF are missing in the original dataset
dof_missing = df[df['DOF'].isnull()]

print("There are {:,} rows with missing DOFs in the dataset.".format(len
(dof_missing)))
```

```
There are 15,965 rows with missing DOFs in the dataset.
```

Step 1: Make sure that we are only looking at the minimum value for the DOF across all of the charges associated with one JPR_ID. This is the procedure because we don't wan't to count a DOF as an instance of recidivism if it occurs BEFORE the date of sentencing.

In [ ]:
```
#at the JPR_ID level we only want ONE DOF because becuase we don't want
 to take into account DOF's that occur
#BEFORE the DOS (associated with the JPR_ID) as an instance of recidivis
m. -- each JPR_ID should have only ONE DOS

df["NEW_DOF"] = df.groupby(["JPR_ID"])["DOF"].transform("min")
```

In [ ]:
```
df.head()[["JPR_ID", "DOF", "NEW_DOF"]]
```

Out[ ]:

	JPR_ID	DOF	NEW_DOF
0	640001	2000-04-01	2000-04-01
1	642480	1999-12-31	1999-12-31
2	660434	2000-12-23	2000-12-23
3	628940	2000-06-26	2000-06-26
4	594048	2000-10-15	2000-10-15

In [ ]:
```
dof_missing = df[df['NEW_DOF'].isnull()]

percent_missing = len(dof_missing)/len(df)
print("After cleaning, there are {:,} ({:%}) rows with missing DOFs in t
he dataset.".format(len(dof_missing), percent_missing))
```

```
After cleaning, there are 11,785 (0.454381%) rows with missing DOFs in
the dataset.
```

Step 2: Subset the data to just include those rows where NEW_DOF <= DOS

In [ ]:
```
#make sure the sentencing
before_length = len(df)
df = df[df.NEW_DOF <= df.DOS] #should this be <= ?
after_length = len(df)

print("Before DOF <= DOS correction there were {:,} rows and after clean
ing there were {:,} rows. A change of {:,}.".format(before_length, after
_length, before_length - after_length))
```

```
Before DOF <= DOS correction there were 2,593,636 rows and after cleani
ng there were 2,581,813 rows. A change of 11,823.
```

Clean Missing PRS Score

In [ ]:
```
before_length = len(df)
#subset to just the id variables with a PRS score missing
id_varswith_prsmissing= set(df[df.PRS.isnull()].ID_VARIABLE)

#remove id vars with missing PRS
df_prs_notaffected = df[~df.ID_VARIABLE.isin(id_varswith_prsmissing)]

#reassign to working dataframe
df = df_prs_notaffected

after_length = len(df)
print("Before PRS correction there were {:,} rows and after cleaning the
re were {:,} rows. A change of {:,} rows and {} people.".format(before_l
ength, after_length, before_length - after_length, len(id_varswith_prsmi
ssing)))
```

```
Before PRS correction there were 2,581,813 rows and after cleaning ther
e were 2,581,750 rows. A change of 63 rows and 18 people.
```

Clean JP CC Bug

In [ ]:
```
# Obtaining the id variables with jp_bug
id_varswith_jpbug= set(df[df.JP_CC_BUG=='Y'].ID_VARIABLE)
```

In [ ]:
```
# assigning all the rows associated with the jp bugs to a seperate dataf
rame
df_with_jpbug=  df[df.ID_VARIABLE.isin(id_varswith_jpbug)]
```

In [ ]:
```
# Removing the rows of these that are beyond 2016(rows in the future tha
t are affected by the JP_CC_BUG)
df_jp_bug_cleaned = df_with_jpbug[df.DOS_YEAR<2016]
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: Use
rWarning: Boolean Series key will be reindexed to match DataFrame inde
x.
```

In [ ]:
```
# Isolating the rows associated with id_vars in the original dataframe t
hat is not associated with the bug
df_jpbug_notaffected = df[~df.ID_VARIABLE.isin(id_varswith_jpbug)]
```

In [ ]:
```
# Rejoining the rows affected by the JP_CC_bug after cleaning them to th
e rows not affected by the bug
df_cleaned_1 = pd.concat([df_jpbug_notaffected,df_jp_bug_cleaned])  #new
working df

df = df_cleaned_1
```

In [ ]:
```
after_length = len(df)

print("After the JP_CC_BUG correction there are {:,} rows. ".format(afte
r_length))
```

```
After the JP_CC_BUG correction there are 2,562,389 rows.
```

Implement At Risk Date Calculation Logic

STEP 1: Create a New JP_MIN variable that takes the Max(JP_MIN) for a given JPR_ID

In [ ]:
```
#Fix Issues with the missing JP_MIN
num_missing_jp_min = len(df.loc[pd.isna(df["JP_MIN"])]) #[["JPR_ID", "JP
_MIN"]]
print("There are {:,} entries in the dataset missing a JP_MIN value.".fo
rmat(num_missing_jp_min))

df["ADJ_JPMIN"] = df.groupby(["JPR_ID"])["JP_MIN"].transform("max")

num_missing_jp_min = len(df.loc[pd.isna(df["ADJ_JPMIN"])]) #[["JPR_ID",
 "JP_MIN"]]
print("There are {:,} entries in the dataset missing a  ADJ_JPMIN valu
e.".format(num_missing_jp_min))
```

```
There are 338,881 entries in the dataset missing a JP_MIN value.
There are 338,807 entries in the dataset missing a  ADJ_JPMIN value.
```

STEP 2: Collapse the data at the ID_VARIABLE, DOS-LEVEL

In [ ]:
```
df_collapsed = df.copy()

# #get the max values of the OGS and JP_MIN values -- possibly further a
djustments need to be at this level
df_collapsed['OGS'] = df_collapsed.groupby(["ID_VARIABLE", "DOS"])["OGS"
].transform(max)

#df_collapsed["ADJ_JPMIN"] = df_collapsed.groupby(["ID_VARIABLE", "DO
S"])["ADJ_JPMIN"].transform(max)
#df_collapsed["PRS"] = df_collapsed.groupby(["ID_VARIABLE", "DOS"])["PR
S"].transform(max)


#collapse data to be at the id variable, DOS level (need to ungroup the
 data for the at_risk date calculation to work)
df_collapsed = df_collapsed.copy().groupby(["ID_VARIABLE", "DOS"]).first
().reset_index()

#inspect the results
df_collapsed[["ID_VARIABLE", "DOS", "NEW_DOF", "INC_SANCTION_EXISTS", "A
DJ_JPMIN"]] #, "OFN_LIFE_DEATH", "JP_LIFE_DEATH"]]
```

Out[ ]:

	ID_VARIABLE	DOS	NEW_DOF	INC_SANCTION_EXISTS	ADJ_JPMIN
0	1000001	2010-02-18	2009-06-25	N	16.0
1	1000002	2017-01-31	2015-09-01	Y	120.0
2	1000003	2002-05-08	2001-09-07	N	0.0
3	1000003	2009-03-04	2009-03-04	Y	92.0
4	1000004	2013-12-10	2013-09-19	N	0.0
...	...	...	...	...	...
1481188	1916193	2002-01-07	2001-05-03	N	0.0
1481189	1916194	2016-11-14	2015-03-30	N	0.0
1481190	1916195	2009-06-04	2009-05-16	N	0.0
1481191	1916196	2014-03-03	2013-07-05	Y	31.0
1481192	1916197	2018-08-30	2017-05-21	N	0.0

1481193 rows × 5 columns

STEP 3: Calculate the AT_RISK_DT using the following logic

In [ ]:
```
def create_at_risk_date(row):
    #need to account for REALLY large JP_MIN values

    # Because of this error message OverflowError: Python int too large
 to convert to C long
    # 25 is more years than we have in our data, so their at_risk date a
lso get set to some value far in the future
    upper_limit = 25.0 * 365.0

    num_days_in_month = 30.0

    #if offense has a life or death flag, set their at_risk_date abritar
ily large
    if row['OFN_LIFE_DEATH'] == "Y":
        at_risk_date = pd.to_datetime('2035-12-31')

    if row['JP_LIFE_DEATH'] == "Y":
        at_risk_date = pd.to_datetime('2035-12-31')

    #if they were not incarcerated, then their at risk date is just thei
r date of offense
    if row["INC_SANCTION_EXISTS"] == "N":
        at_risk_date = row['NEW_DOF']

    #if they were incarcerated, look at the below logic to determine the
ir at-risk date
    else:

        if row["ADJ_JPMIN"] < upper_limit:

            if row["INC_SANCTION_EXISTS"] == "Y" and pd.notna(row['ADJ_J
PMIN']):
                at_risk_date = row['DOS'] + pd.Timedelta(days = row['ADJ
_JPMIN'])

            elif row["INC_SANCTION_EXISTS"] == "Y" and pd.notna(row['INC
MIN']):
                at_risk_date = row['DOS'] + pd.Timedelta(days = row['INC
MIN'] * num_days_in_month)

            else:
                at_risk_date = row['INC_END']

        else:
            at_risk_date = pd.to_datetime('2035-12-31')

    return at_risk_date


# df["AT_RISK_DT"] = np.where(
#     df['INC_SANCTION_EXISTS'] == "Y" and pd.notna(df['JP_MIN']), 1, 0)

# test = df[:2000]
# #apply the function to the data (row by row)
# test["AT_RISK_DT"] = test.apply(create_at_risk_date, axis = 1)
```

```
#  #adjust so that the times do not include minutes and seconds
# test["AT_RISK_DT"] = pd.to_datetime(test["AT_RISK_DT"]).dt.date

# # #inspect the results
# test[['ID_VARIABLE', 'JPR_ID',"JP_MIN", "INCMIN", "INC_END", "ADJ_JPMI
N", "INC_SANCTION_EXISTS", "DOS", "NEW_DOF", "AT_RISK_DT"]]

#test = df[:2000]
#apply the function to the data (row by row)
df_collapsed["AT_RISK_DT"] = df_collapsed.apply(create_at_risk_date, axi
s = 1)

 #adjust so that the times do not include minutes and seconds
df_collapsed["AT_RISK_DT"] = pd.to_datetime(df_collapsed["AT_RISK_DT"]).
dt.date

# #inspect the results
df_collapsed[['ID_VARIABLE', 'JPR_ID',"JP_MIN", "INCMIN", "INC_END", "AD
J_JPMIN", "INC_SANCTION_EXISTS", "DOS", "NEW_DOF", "AT_RISK_DT"]]
```

Out[ ]:

	ID_VARIABLE	JPR_ID	JP_MIN	INCMIN	INC_END	ADJ_JPMIN	INC_SANCTION_EXIST
0	1000001	4915383	16.0	0.526316	17 Jan 12	16.0	
1	1000002	5678165	120.0	4.000000	30 Jan 19	120.0	
2	1000003	1070248	0.0	NaN	NaN	0.0	
3	1000003	4902797	92.0	3.000000	03 Sep 09	92.0	
4	1000004	5318013	0.0	NaN	NaN	0.0	
...	...	...	...	...	...	...	
1481188	1916193	425719	0.0	NaN	NaN	0.0	
1481189	1916194	5557602	0.0	NaN	NaN	0.0	
1481190	1916195	3884756	0.0	NaN	NaN	0.0	
1481191	1916196	5351194	31.0	1.000000	02 Mar 15	31.0	
1481192	1916197	5830496	0.0	NaN	NaN	0.0	

1481193 rows × 10 columns

Note: In the above at_risk_date calculation code, there is an "upper_limit" because the largest JP_MIN value is 230,000+ days, which is the equivalent of about 631 years. This person would not recidivate in our dataset and Python throws a "OverflowError: Python int too large to convert to C long" for these individuals. So, in order to allow the code to run, those with jp_min values equivalent to more days than we have data for, will just get an at-risk date very far into the future.

In [ ]:
```
#OverflowError: Python int too large to convert to C long

largest_jpmin =  df_collapsed["ADJ_JPMIN"].max()
largest_jpmin_in_years = largest_jpmin/365.0
print("The largest JP_MIN value is {:,} days, which is {} years. This ca
uses Python to throw the following error: OverflowError: Python int too
 large to convert to C long.".format(largest_jpmin, largest_jpmin_in_yea
rs))
```

```
The largest JP_MIN value is 230,468.0 days, which is 631.4191780821918
years. This causes Python to throw the following error: OverflowError:
Python int too large to convert to C long.
```

Populate Next DOF

In [ ]:
```
#sort the data
df_collapsed = df_collapsed.sort_values(by = ["ID_VARIABLE", "NEW_DOF"])

#shift the data up by one to create the new vaariable "NEXT_DOF"
df_collapsed['NEXT_DOF'] = df_collapsed.groupby(['ID_VARIABLE'])['NEW_DO
F'].shift(-1).dt.date

df_collapsed[:20][["ID_VARIABLE", "JPR_ID", "DOS", "NEW_DOF", "NEXT_DOF"
, "AT_RISK_DT", "INC_SANCTION_EXISTS"]]
```

Out[ ]:

	ID_VARIABLE	JPR_ID	DOS	NEW_DOF	NEXT_DOF	AT_RISK_DT	INC_SANCTION_EXISTS
0	1000001	4915383	2010-02-18	2009-06-25	NaT	2009-06-25	N
1	1000002	5678165	2017-01-31	2015-09-01	NaT	2017-05-31	Y
2	1000003	1070248	2002-05-08	2001-09-07	2009-03-04	2001-09-07	N
3	1000003	4902797	2009-03-04	2009-03-04	NaT	2009-06-04	Y
4	1000004	5318013	2013-12-10	2013-09-19	2018-07-09	2013-09-19	N
5	1000004	5922309	2018-09-26	2018-07-09	NaT	2018-07-09	N
6	1000005	1203958	2008-08-11	2006-08-14	NaT	2009-02-10	Y
7	1000006	378762	2006-08-30	2005-10-08	NaT	2007-11-30	Y
8	1000007	43891	2004-03-02	2003-04-18	NaT	2003-04-18	N
9	1000008	4992054	2011-05-13	2011-01-16	NaT	2011-11-13	Y
10	1000009	2609876	2007-12-19	2007-04-23	NaT	2007-04-23	N
11	1000010	3031754	2010-06-24	2008-09-28	2010-04-16	2008-09-28	N
13	1000010	4933898	2010-11-22	2010-04-16	2010-04-18	2010-04-16	N
12	1000010	4928562	2010-08-31	2010-04-18	2016-01-11	2010-09-04	Y
14	1000010	5675362	2016-12-01	2016-01-11	2016-07-05	2016-01-11	N
15	1000010	5672811	2016-12-05	2016-07-05	2018-02-09	2016-07-05	N
16	1000010	5912924	2018-08-08	2018-02-09	2018-03-03	2018-02-09	N
17	1000010	5881555	2018-08-14	2018-03-03	NaT	2018-10-14	Y
18	1000011	4926698	2003-12-08	2003-03-22	2005-04-21	2003-03-22	N
19	1000011	357944	2006-10-03	2005-04-21	2006-01-17	2005-04-21	N

Check for "Free Time"

(i.e.: Do we have enough data for an individual to see if they recidivated in 3 years or not?)

Procedure Below:

1. Subset just to those whose at_risk date < max DOS df[["DOS"]].max()
2. Then, we also want to remove those whose last next_dof is null and whose last dof > 2017
3. Essentially, we want to subset (whatever grouping variable we're using) to just those entries where next_dof is null and FOR THIS SAME ROW, if the dof >= pd.todatetime("2017-01-01") -- remove these entries

In [ ]:
```
#subset to those whose at_risk_date < the largest sentencing date that w
e have

before_length = len(df_collapsed)

#what is the maximum sentence date?
last_day = pd.to_datetime(df_collapsed[["DOS"]].max())[0]
df_collapsed = df_collapsed[df_collapsed["AT_RISK_DT"] <= last_day]

after_length = len(df_collapsed)

print("There are {:,} id_var, dos combos where the at risk date is after
the last date of sentence available.".format(before_length - after_lengt
h))
```

```
There are 31,613 id_var, dos combos where the at risk date is after the
last date of sentence available.
```

Here, I calculate a "LAST_DOF" variable, which will then be used to subset the data to only those whose latest offense was before 2017

In [ ]:
```
df_collapsed["LAST_DOF"] = df_collapsed.loc[df_collapsed["NEXT_DOF"].isnull(), "NEW_DOF"]

df_collapsed[["ID_VARIABLE", "DOS", "NEW_DOF", "NEXT_DOF", "LAST_DOF"]]
```

Out[ ]:

	ID_VARIABLE	DOS	NEW_DOF	NEXT_DOF	LAST_DOF
0	1000001	2010-02-18	2009-06-25	NaT	2009-06-25
1	1000002	2017-01-31	2015-09-01	NaT	2015-09-01
2	1000003	2002-05-08	2001-09-07	2009-03-04	NaT
3	1000003	2009-03-04	2009-03-04	NaT	2009-03-04
4	1000004	2013-12-10	2013-09-19	2018-07-09	NaT
...	...	...	...	...	...
1481188	1916193	2002-01-07	2001-05-03	NaN	2001-05-03
1481189	1916194	2016-11-14	2015-03-30	NaN	2015-03-30
1481190	1916195	2009-06-04	2009-05-16	NaN	2009-05-16
1481191	1916196	2014-03-03	2013-07-05	NaN	2013-07-05
1481192	1916197	2018-08-30	2017-05-21	NaN	2017-05-21

1449580 rows × 5 columns

In [ ]:
```
#subset the data to only those whose last_dof is before 2017
#before_length = len(df_collapsed)

last_day = pd.to_datetime("2017-01-01")

#subset the dataset to either where the LAST_DOF is null OR LAST_DOF < l
ast_day
df_collapsed = df_collapsed.loc[(df_collapsed["LAST_DOF"].isnull()) | (d
f_collapsed["LAST_DOF"] < last_day)]

# after_length = len(df_collapsed)
# print("There are {:,} id_var, dos combos whose's last dof is not in sc
ope.".format(before_length - after_length))

df_collapsed[["ID_VARIABLE", "DOS", "NEW_DOF", "NEXT_DOF", "LAST_DOF"]]
```

Out[ ]:

	ID_VARIABLE	DOS	NEW_DOF	NEXT_DOF	LAST_DOF
0	1000001	2010-02-18	2009-06-25	NaT	2009-06-25
1	1000002	2017-01-31	2015-09-01	NaT	2015-09-01
2	1000003	2002-05-08	2001-09-07	2009-03-04	NaT
3	1000003	2009-03-04	2009-03-04	NaT	2009-03-04
4	1000004	2013-12-10	2013-09-19	2018-07-09	NaT
...	...	...	...	...	...
1481187	1916192	2015-10-20	2014-05-06	NaN	2014-05-06
1481188	1916193	2002-01-07	2001-05-03	NaN	2001-05-03
1481189	1916194	2016-11-14	2015-03-30	NaN	2015-03-30
1481190	1916195	2009-06-04	2009-05-16	NaN	2009-05-16
1481191	1916196	2014-03-03	2013-07-05	NaN	2013-07-05

1323978 rows × 5 columns

CREATE TIME TO RECIDIVATE AND RECIDIVSM VARIABLES

Time to Recidivate -- Currently there are issues where NEXT_DOF happens BEFORE the AT_RISK_DT

In [ ]:
```
#inspect the results -- an issue where the next_dof is less than the nex
t_dof
#print(df_collapsed["TIME_TO_RECIDIVATE"].min(), df_collapsed["TIME_TO_R
ECIDIVATE"].max())

list_neg_time = list(df_collapsed.loc[df_collapsed['NEXT_DOF'] < df_coll
apsed['AT_RISK_DT']]["ID_VARIABLE"])
neg_time_analysis = df_collapsed.loc[df_collapsed["ID_VARIABLE"].isin(li
st_neg_time)]

neg_time_analysis["TIME_TO_RECIDIVATE"] = neg_time_analysis["NEXT_DOF"]
- neg_time_analysis["AT_RISK_DT"]
neg_time_analysis['TIME_TO_RECIDIVATE'] = neg_time_analysis['TIME_TO_REC
IDIVATE'].dt.days



#they committed another offense before their previous at risk date
# df_collapsed['NEXT_DOF'] = df_collapsed.groupby(['ID_VARIABLE'])['NEW_
DOF'].shift(-1).dt.date

# df_collapsed["PREVIOUS_AT_RISK_DT"] = df_collapsed.groupby(['ID_VARIAB
LE'])['AT_RISK_DT'].shift(1) #.dt.date
#df_collapsed[["ID_VARIABLE", "DOS", "ADJ_JPMIN", "NEW_DOF", "NEXT_DOF",
"AT_RISK_DT", "PREVIOUS_AT_RISK_DT"]]

# test = df_collapsed.loc[df_collapsed["NEXT_DOF"] <df_collapsed["PREVIO
US_AT_RISK_DT"] ]
# test[["ID_VARIABLE", "DOS", "ADJ_JPMIN", "NEW_DOF", "NEXT_DOF", "AT_RI
SK_DT", "PREVIOUS_AT_RISK_DT"]]

print("There are {:,} ({:%}) id_var, dos combos where the next date of o
ffense occurs before at risk date.".format(len(list_neg_time),len(list_n
eg_time)/len(df_collapsed) ))
neg_time_analysis[["ID_VARIABLE", "DOS", "ADJ_JPMIN", "NEW_DOF", "NEXT_D
OF", "AT_RISK_DT", "TIME_TO_RECIDIVATE"]]
```

```
There are 77,818 (5.877590%) id_var, dos combos where the next date of
offense occurs before at risk date.

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

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

Out[ ]:

	ID_VARIABLE	DOS	ADJ_JPMIN	NEW_DOF	NEXT_DOF	AT_RISK_DT	TIME_TO_RECID
27	1000016	2003-01-10	1096.0	2001-04-20	2006-04-15	2006-01-10	
28	1000016	2006-09-18	5.0	2006-04-15	2006-04-16	2006-09-23	
29	1000016	2006-10-16	1004.0	2006-04-16	NaT	2009-07-16	
30	1000017	2013-03-05	0.0	2012-10-12	2015-12-05	2012-10-12	1
32	1000017	2016-06-10	44.0	2015-12-05	2016-02-07	2016-07-24	
...	...	...	...	...	...	...	
1481143	1916166	2005-04-07	364.0	2004-02-07	2006-03-04	2006-04-06	
1481146	1916166	2009-06-01	1096.0	2006-03-04	2006-03-04	2012-06-01	-2
1481147	1916166	2009-06-15	0.0	2006-03-04	2008-01-23	2006-03-04	
1481145	1916166	2009-04-17	183.0	2008-01-23	2008-10-10	2008-01-23	
1481144	1916166	2009-03-26	160.0	2008-10-10	NaT	2009-09-02	

226826 rows × 7 columns

In [ ]:
```
#subtract the next_dof and at_risk_dt variables
df_collapsed['TIME_TO_RECIDIVATE'] = np.where(
    df_collapsed['NEXT_DOF'] > df_collapsed['AT_RISK_DT'],

    pd.to_datetime(df_collapsed['NEXT_DOF']) - pd.to_datetime(df_collaps
ed['AT_RISK_DT']),

    #DOUBLE CHECK THIS -- ATTEMPT TO DEAL WITH NEGATIVE TIME_TO_RECIDIVA
TE
    pd.to_datetime(df_collapsed['NEXT_DOF']) - pd.to_datetime(df_collaps
ed['NEW_DOF'])
    )

#update the time to recidivate column to JUST be the number of days as a
n integer/float
df_collapsed['TIME_TO_RECIDIVATE'] = df_collapsed['TIME_TO_RECIDIVATE'].
dt.days

df_collapsed[["ID_VARIABLE", "DOS", "NEW_DOF", "NEXT_DOF", "TIME_TO_RECI
DIVATE"]]
```

Out[ ]:

	ID_VARIABLE	DOS	NEW_DOF	NEXT_DOF	TIME_TO_RECIDIVATE
0	1000001	2010-02-18	2009-06-25	NaT	NaN
1	1000002	2017-01-31	2015-09-01	NaT	NaN
2	1000003	2002-05-08	2001-09-07	2009-03-04	2735.0
3	1000003	2009-03-04	2009-03-04	NaT	NaN
4	1000004	2013-12-10	2013-09-19	2018-07-09	1754.0
...	...	...	...	...	...
1481187	1916192	2015-10-20	2014-05-06	NaN	NaN
1481188	1916193	2002-01-07	2001-05-03	NaN	NaN
1481189	1916194	2016-11-14	2015-03-30	NaN	NaN
1481190	1916195	2009-06-04	2009-05-16	NaN	NaN
1481191	1916196	2014-03-03	2013-07-05	NaN	NaN

1323978 rows × 5 columns

In [ ]:
```
#number of days in  years
three_years_in_days = float(3) * 365.0
five_years_in_days = float(5) * 365.0

#JUDICIAL-PROCEEDING LEVEL RECIDIVISM
#final_next_dof["RECIDIVISM_3Y"] = final_next_dof.apply(create_recidivism_var, years = 3, axis = 1)

df_collapsed["RECIDIVISM_3Y"] = np.where(
    (df_collapsed['TIME_TO_RECIDIVATE'] > 0) & (df_collapsed['TIME_TO_RECIDIVATE'] <= three_years_in_days), 1, 0)

df_collapsed["RECIDIVISM_5Y"] = np.where(
    (df_collapsed['TIME_TO_RECIDIVATE'] > 0) & (df_collapsed['TIME_TO_RECIDIVATE'] <= five_years_in_days), 1, 0)

df_collapsed[["ID_VARIABLE", "DOS", "ADJ_JPMIN", "NEW_DOF", "NEXT_DOF", "AT_RISK_DT", "TIME_TO_RECIDIVATE", "RECIDIVISM_3Y", "RECIDIVISM_5Y"]]
```

Out[ ]:

	ID_VARIABLE	DOS	ADJ_JPMIN	NEW_DOF	NEXT_DOF	AT_RISK_DT	TIME_TO_RECID
0	1000001	2010-02-18	16.0	2009-06-25	NaT	2009-06-25	
1	1000002	2017-01-31	120.0	2015-09-01	NaT	2017-05-31	
2	1000003	2002-05-08	0.0	2001-09-07	2009-03-04	2001-09-07	2
3	1000003	2009-03-04	92.0	2009-03-04	NaT	2009-06-04	
4	1000004	2013-12-10	0.0	2013-09-19	2018-07-09	2013-09-19	1
...	...	...	...	...	...	...	
1481187	1916192	2015-10-20	7.0	2014-05-06	NaT	2015-10-27	
1481188	1916193	2002-01-07	0.0	2001-05-03	NaT	2001-05-03	
1481189	1916194	2016-11-14	0.0	2015-03-30	NaT	2015-03-30	
1481190	1916195	2009-06-04	0.0	2009-05-16	NaT	2009-05-16	
1481191	1916196	2014-03-03	31.0	2013-07-05	NaT	2014-04-03	

1323978 rows × 9 columns

Export The Results to CSV

(PA_SENTENCING/Project/data/recidivism_dataset.csv)

In [ ]:
```
#Export the Results to a CSV


#export the dataframe with the recidivism variables to a new dataframe
output_path = os.path.join(pa_sentencing_path, "Project", "data", "recid
ivism_dataset.csv")

df_collapsed.to_csv(output_path) #export the final results
```