CSCI82 - Adherence Survival Analysis Project

Pharmacy DEID Data - GLP-1s, Anti-tnfs

Package Imports

In [1]:
```
import pandas as pd
import datetime as dt
import numpy as np

# special matplotlib command for global plot configuration
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline

import seaborn as sns
from seaborn import regplot

import datetime

from lifelines import CoxPHFitter
from lifelines import KaplanMeierFitter

from seaborn import regplot

import warnings
warnings.filterwarnings("ignore")
```

Import Data

In [2]:
```
df = pd.read_csv('adherence_df.csv')
df = df.drop (labels='Unnamed: 0',axis=1)
```

In [3]:
```
df.ndc_group.unique()
```

Out[3]:
```
array(['GLP1', 'Other', 'anti-tnf'], dtype=object)
```

In [4]:
```
#Reformat date as date time
df.date = pd.to_datetime(df.date,format="%Y-%d-%m",errors='ignore')
```

In [5]:
```
#remove all NA date vaules
df = df[~df.date.isna()]
```

In [6]:
```
#filter for only 2023 data
df = df[df.date >= '2023-01-01']
len(df)
```

Out[6]:
```
61054
```

Treatment Category: Exploratory Analysis:

Note: This Section was used to select one of three treatment categories to use in our initial model. The results showed that the best data coverage was with the GLP-1 data set

Loading [MathJax]/extensions/Safe.js

```
In [7]: df.date.min(),df.date.max()
```

```
Out[7]: ('2023-01-01', '2023-12-01')
```

```
In [8]: #Unique GLP1 patients
        len(df[df.ndc_group =='GLP1'].patient_id.unique())
```

```
Out[8]: 4241
```

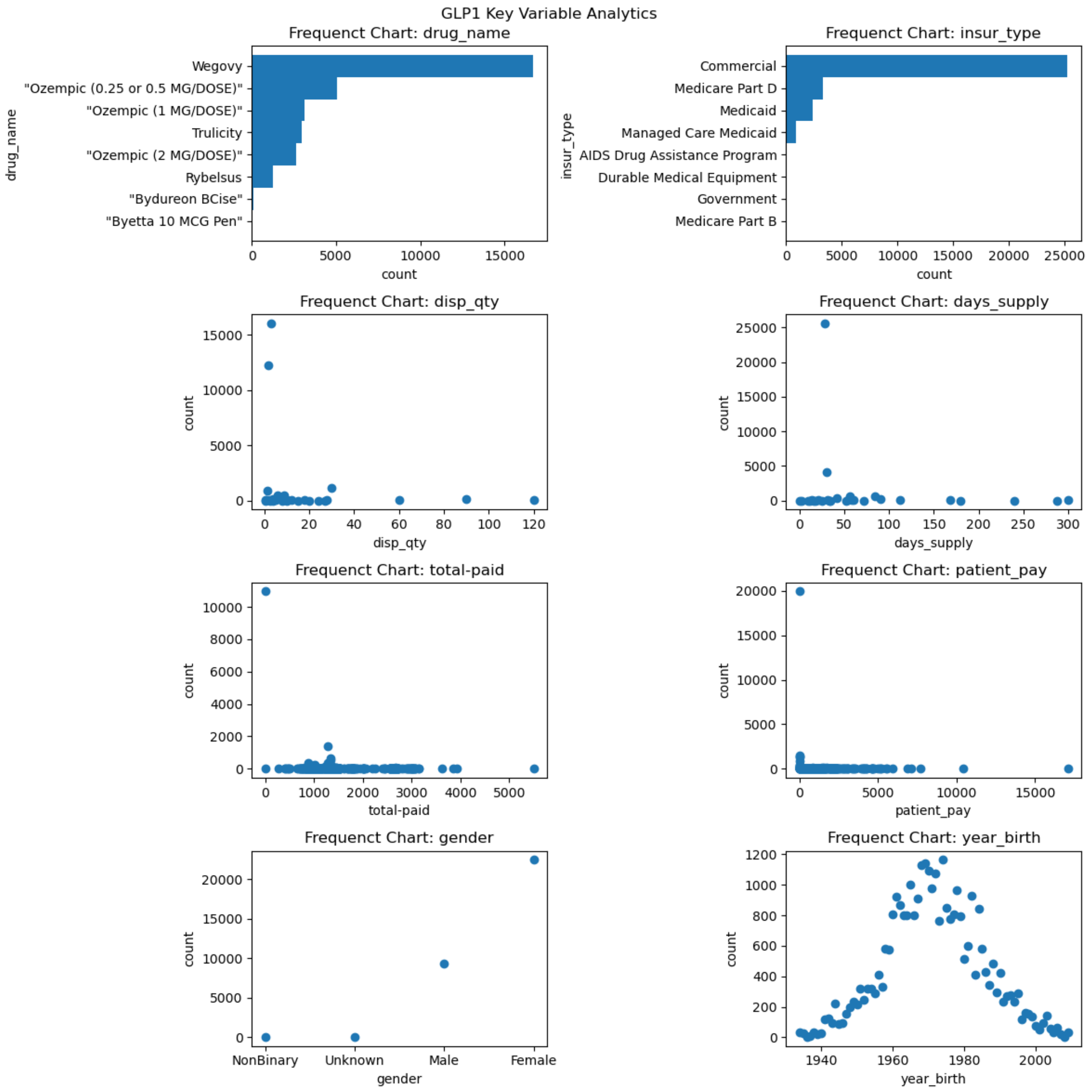
```
In [9]: #Unique GLP1 rx's
        len(df[df.ndc_group =='GLP1'].rx_id.unique())
```

```
Out[9]: 8858
```

GLP-1 Statistics

```
In [10]: var = ['drug_name','insur_type','disp_qty','days_supply','total-paid','patient_pay','gender','
         plot_df = df[df.ndc_group =='GLP1']

         fig, axes = plt.subplots(4, 2, figsize=(12, 12))

         ax = axes.flatten()

         for i in range(len(var)):
             if (var[i] == 'insur_type') or (var[i] == 'drug_name'):
                 ax[i].barh(plot_df[var[i]].value_counts().sort_values().index,plot_df[var[i]].value_co
                 ax[i].set_title(f'Frequent Chart: {var[i]}')
                 ax[i].set_xlabel(f'count')
                 ax[i].set_ylabel(f'{var[i]}')
             else:
                 ax[i].plot(plot_df[var[i]].value_counts().sort_values(),linestyle="",marker="o")
                 ax[i].set_title(f'Frequent Chart: {var[i]}')
                 ax[i].set_ylabel(f'count')
                 ax[i].set_xlabel(f'{var[i]}')

         plt.suptitle('GLP1 Key Variable Analytics')
         plt.tight_layout()
```

Loading [MathJax]/extensions/Safe.js



Anti-tnf Statistics

In [11]:
```
#Unique anti-tnf patients
len(df[df.ndc_group =='anti-tnf'].patient_id.unique())
```

Out[11]:
```
154
```

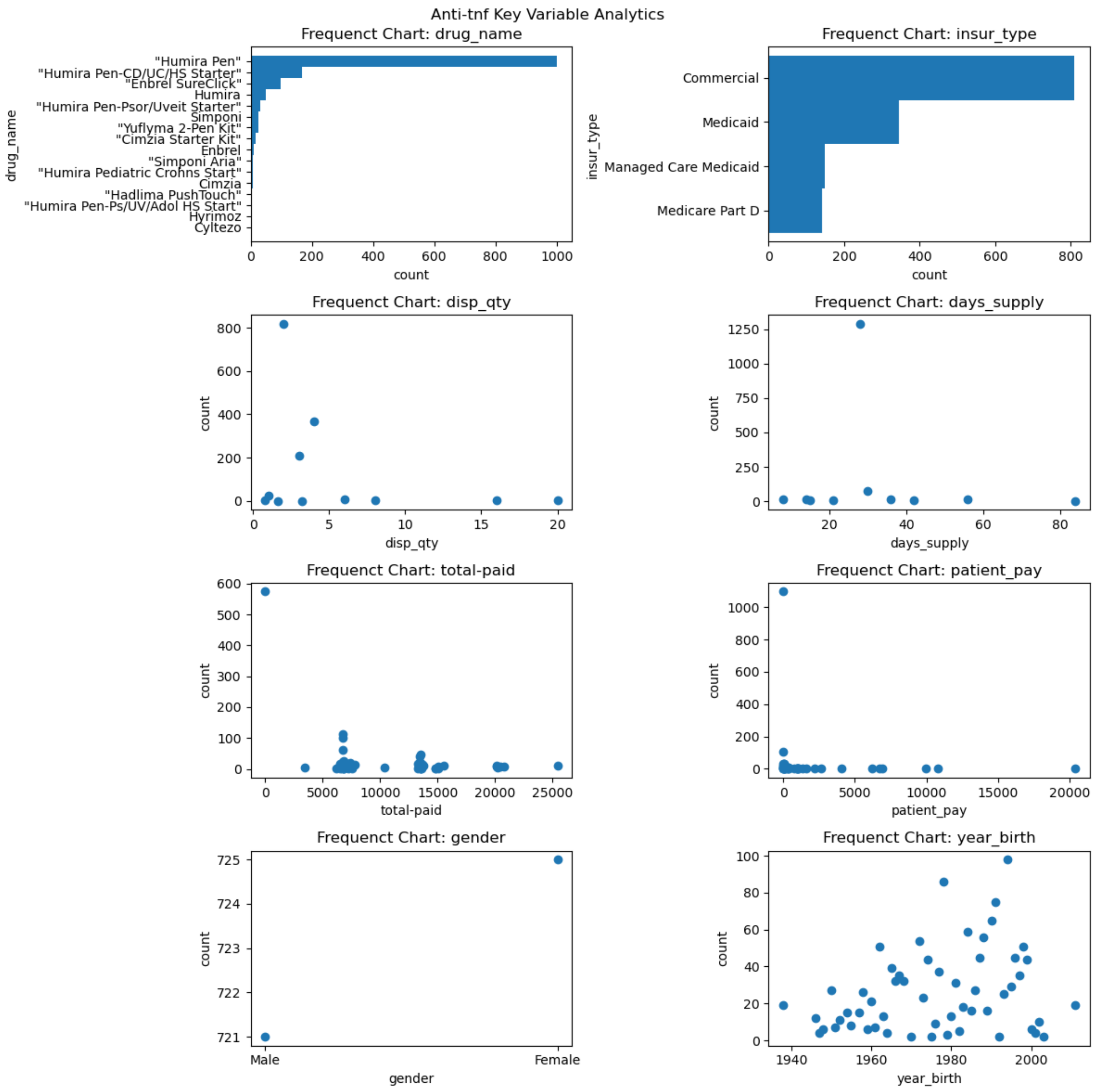
In [12]:
```
#Unique GLP1 rx's
len(df[df.ndc_group =='anti-tnf'].rx_id.unique())
```

Out[12]:
```
281
```

In [13]:
```
var = ['drug_name','insur_type','disp_qty','days_supply','total-paid','patient_pay','gender','
plot_df = df[df.ndc_group =='anti-tnf']

fig, axes = plt.subplots(4, 2, figsize=(12, 12))

ax = axes.flatten()

for i in range(len(var)):
```

Loading [MathJax]/extensions/Safe.js

```
    if (var[i] == 'insur_type') or (var[i] == 'drug_name'):
        ax[i].barh(plot_df[var[i]].value_counts().sort_values().index,plot_df[var[i]].value_co
        ax[i].set_title(f'Frequent Chart: {var[i]}')
        ax[i].set_xlabel(f'count')
        ax[i].set_ylabel(f'{var[i]}')
    else:
        ax[i].plot(plot_df[var[i]].value_counts().sort_values(),linestyle="",marker="o")
        ax[i].set_title(f'Frequent Chart: {var[i]}')
        ax[i].set_ylabel(f'count')
        ax[i].set_xlabel(f'{var[i]}')

plt.suptitle('Anti-tnf Key Variable Analytics')
plt.tight_layout()
```



DPP/Other Drug Statistics

```
In [14]: df.ndc_group.unique()

Out[14]: array(['GLP1', 'Other', 'anti-tnf'], dtype=object)
```

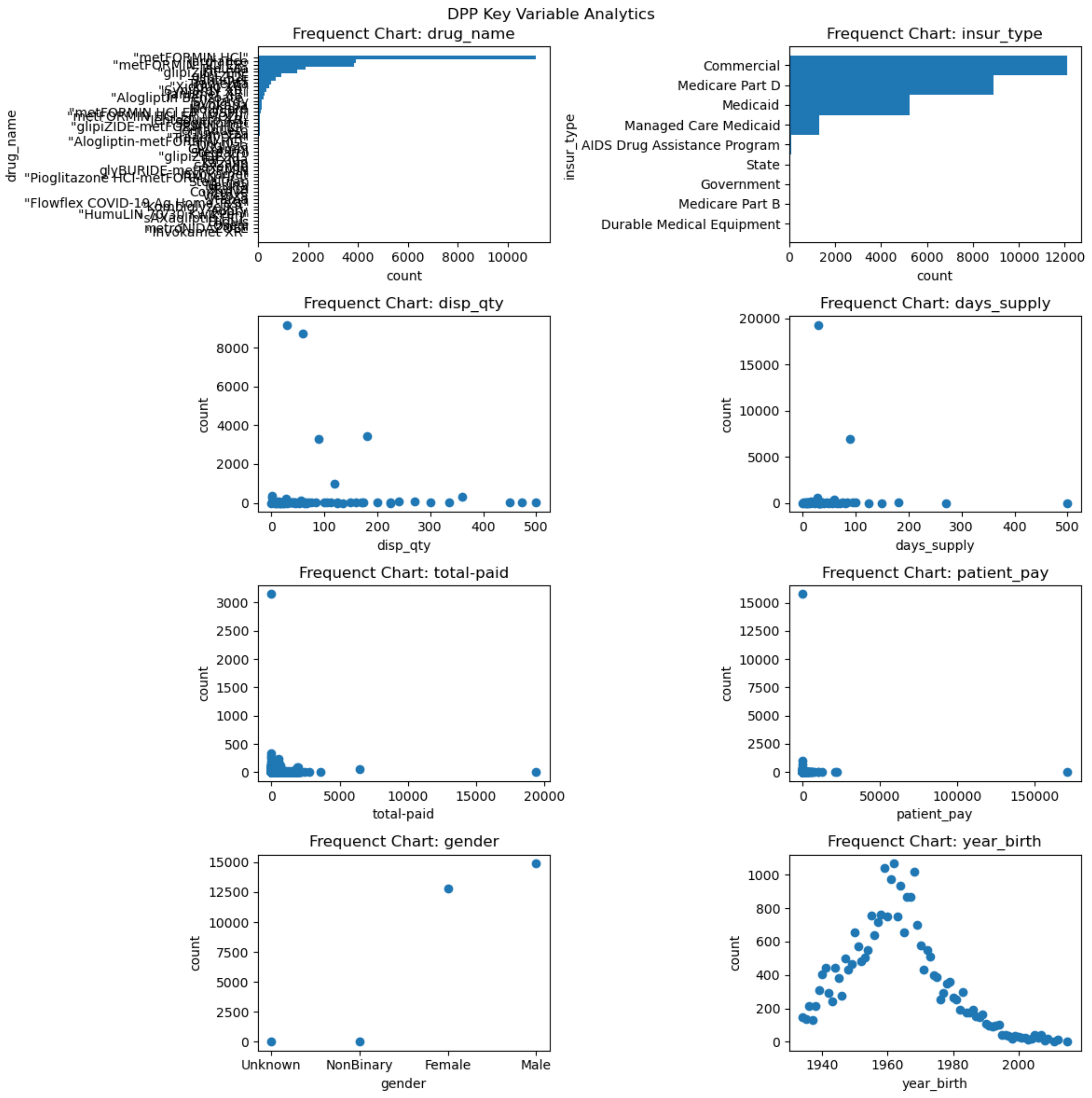
Loading [MathJax]/extensions/Safe.js

In [15]:
```
#Unique DPP patients
len(df[df.ndc_group =='Other'].patient_id.unique())
```

Out[15]:
```
3630
```

In [16]:
```
var = ['drug_name','insur_type','disp_qty','days_supply','total-paid','patient_pay','gender','
plot_df = df[df.ndc_group =='Other']

fig, axes = plt.subplots(4, 2, figsize=(12, 12))

ax = axes.flatten()

for i in range(len(var)):
    if (var[i] == 'insur_type') or (var[i] == 'drug_name'):
        ax[i].barh(plot_df[var[i]].value_counts().sort_values().index,plot_df[var[i]].value_cc
        ax[i].set_title(f'Frequenct Chart: {var[i]}')
        ax[i].set_xlabel(f'count')
        ax[i].set_ylabel(f'{var[i]}')
    else:
        ax[i].plot(plot_df[var[i]].value_counts().sort_values(),linestyle="",marker="o")
        ax[i].set_title(f'Frequenct Chart: {var[i]}')
        ax[i].set_ylabel(f'count')
        ax[i].set_xlabel(f'{var[i]}')


plt.suptitle('DPP Key Variable Analytics')
plt.tight_layout()
```

Loading [MathJax]/extensions/Safe.js



GLP-1 Adherence

Create a patient specific fill data set

In [17]:
```
#https://medium.com/algorexhealth/prescription-days-and-medication-management-d0219c5b828f

#narrow our data set to only glp-1 s
glp_df = df[df.ndc_group =='GLP1']

#create a column with only drug field
glp_df['drug'] = glp_df.drug_name.str.strip(r'"')
glp_df['drug'] = glp_df.drug.str.split(' ')
glp_df['drug'] = [i[0] for i in glp_df['drug']]

#Remove rejected claims:
glp_df =  glp_df[glp_df.claim_resp == 'Paid']

#Create a patient_txn unique field to help with deduplication
glp_df['patient_txn'] = glp_df['patient_id']+ glp_df['rx_txn_id']
```

Loading [MathJax]/extensions/Safe.js

```
uni_txn_pat = glp_df['pat_txn'].unique()

#Drop duplicate rows, keeping the first row
glp_df= glp_df.drop_duplicates(['pat_txn'], keep="last", inplace=False)
```

Ozempic and Wegovy Alone

In [18]:
```
#Unique patients
len(glp_df[(glp_df.drug == 'Ozempic')|(glp_df.drug == 'Wegovy')].patient_id.unique())
```

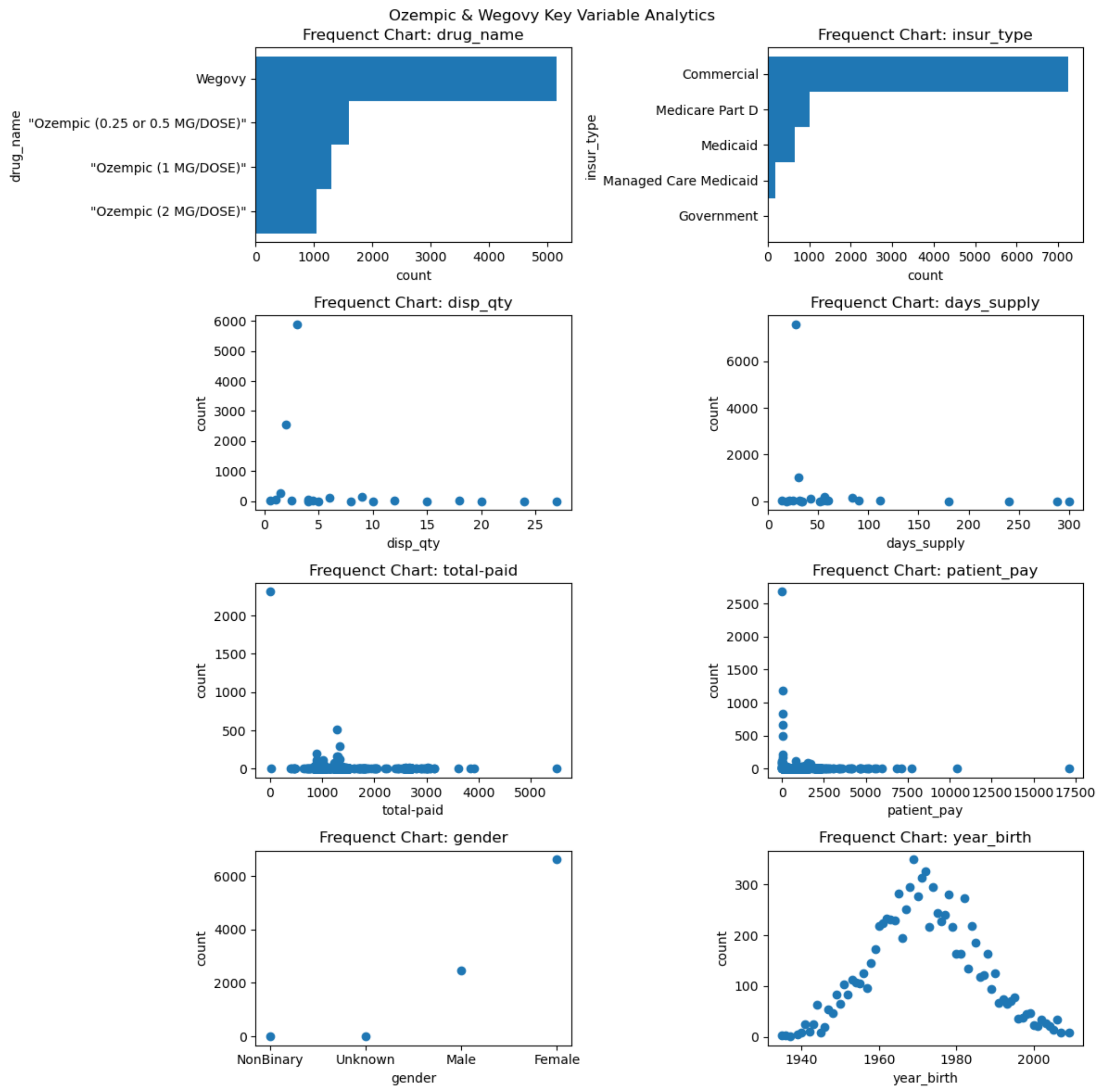
Out[18]:
```
3162
```

In [19]:
```
#Unique transactions
brand_df = glp_df[(glp_df.drug == 'Ozempic')|(glp_df.drug == 'Wegovy')]
len(brand_df)
```

Out[19]:
```
9102
```

In [20]:
```
var = ['drug_name','insur_type','disp_qty','days_supply','total-paid','patient_pay','gender','
plot_df = brand_df

fig, axes = plt.subplots(4, 2, figsize=(12, 12))

ax = axes.flatten()

for i in range(len(var)):
    if (var[i] == 'insur_type') or (var[i] == 'drug_name'):
        ax[i].barh(plot_df[var[i]].value_counts().sort_values().index,plot_df[var[i]].value_co
        ax[i].set_title(f'Frequent Chart: {var[i]}')
        ax[i].set_xlabel(f'count')
        ax[i].set_ylabel(f'{var[i]}')
    else:
        ax[i].plot(plot_df[var[i]].value_counts().sort_values(),linestyle="",marker="o")
        ax[i].set_title(f'Frequent Chart: {var[i]}')
        ax[i].set_ylabel(f'count')
        ax[i].set_xlabel(f'{var[i]}')


plt.suptitle('Ozempic & Wegovy Key Variable Analytics')
plt.tight_layout()
```

Loading [MathJax]/extensions/Safe.js



Censor, Duration, PDC and Other Patient Features

In [21]:
```
#Define a set of unique patients in data set of interest
unique_patient = brand_df.patient_id.unique()
```

Test Code that Steps Through a Single Patient

In [22]:
```
pdc_ratio = ['']

yes=True
p=85

if yes==True:
    test_patient = brand_df[brand_df.patient_id==unique_patient[p]]

    start_date = test_patient.date.min()
    end_date = df.date.max()
                 = test_patient.date
    days_supply  = test_patient.days_supply
```

Loading [MathJax]/extensions/Safe.js

```
    current_drug = test_patient.drug

    index_range = pd.date_range(start=start_date, end=end_date, freq='D')
    #calculate covered perieod for each fill
    supply_deltas = pd.Series([pd.to_timedelta(days, unit='D') for days in days_supply], index

    supply_enddates = (pd.to_datetime(fill_dates) + pd.to_timedelta(days_supply, unit='D'))

    #filter values where covered period is after study end data 12/01/2023
    new_df = pd.DataFrame({'drug':current_drug,'fill_date': supply_deltas.index,'end_date':sup
                           'days_covered':days_supply})
    new_df = new_df.sort_values('fill_date').reset_index(drop=True)
    new_df = new_df[new_df.end_date < '2023-12-01']

    #calculate PDC
    covered_days =new_df.days_covered.sum()
    total_days =  pd.to_datetime(end_date) - pd.to_datetime(new_df.fill_date.min())

    pdc = covered_days/total_days.days

    #certain cases when you titrate up to next ndc will have multiple fills and pdc will be gr
    if pdc > 1:
        pdc = 1

    #append to tracking list of patients
    pdc_ratio.append(pdc)

    print("Covered Days: ",covered_days)
    print("Total Days: ",total_days.days)
    print(f"Proportion of Days Covered (PDC): {pdc*100: .1f}%")
```

```
Covered Days:  196
Total Days:  317
Proportion of Days Covered (PDC):  61.8%
```

In [23]:
```
new_df
```

Out[23]:

	drug	fill_date	end_date	days_covered
0	Ozempic	2023-01-18	2023-03-15	56
1	Ozempic	2023-03-23	2023-04-20	28
2	Ozempic	2023-05-17	2023-06-14	28
3	Ozempic	2023-08-14	2023-09-11	28
4	Ozempic	2023-09-08	2023-10-06	28
5	Ozempic	2023-10-05	2023-11-02	28

In [24]:
```
## Censoring and Duration Logic
cen = []
dur = []
dur_helper = ''
grace_period = 30 #days
censor=[]
duration = []  #duration for non-censored should be study end_date - first fill date
               #duration for a censored event should be the last_fill_date+days_supply+grace_p


#calculate a grace period date
new_df['grace_date'] = new_df['end_date']+datetime.timedelta(grace_period)
#reformat to datetime
new_df['fill_date'] = pd.to_datetime(new_df['fill_date'])
```

Loading [MathJax]/extensions/Safe.js

```
#set first fill censor value to zero
cen.append(0)

#set first dur as the first days_covered/supply
dur.append(new_df.days_covered[0])


#logic to check fills for whehter the patient refills within the grace window
#if not, then censored. Records duration of therapy in each case:

for i in range(1,len(new_df),1): #skip the first row of df
    #if last fill available, check to see if the grace_date
    #is less then our study period end date. if yes, then censor and record duration
    #if not, don't censor, but record days on therapy
    if i == len(new_df)-1:
        if new_df.loc[i,'grace_date'] < pd.to_datetime(end_date):
            cen.append(1)
            dur_helper = (new_df.end_date.max()-new_df.fill_date.min()).days
            dur.append(dur_helper)
        else:
            cen.append(0)
            dur_helper = (pd.to_datetime(end_date)-new_df.fill_date.min()).days
            dur.append(dur_helper)
    else:
        #check to see if end date current is less than grace date previous,
        #if yes censor = 1 and record days on therapy
        if new_df.loc[i,'fill_date'] < new_df.loc[i-1,'grace_date']:
            cen.append(0)
            dur_helper = (new_df.loc[i,'grace_date']-new_df.fill_date.min()).days
            dur.append(dur_helper)
        else:
            cen.append(1)
            dur_helper = (new_df.loc[i-1,'end_date']-new_df.fill_date.min()).days
            dur.append(dur_helper)
    print (f'Censor fill {i+1}: ',cen)
    print (f'Duration fill {i+1}: ',dur)


# Calculate final censor and duration values:

# special case for when all censor values are 0
if all([ v == 0 for v in cen ]):
    censor = 0
    duration = dur[len(new_df)-1]

# else, flag the first one that turns 1
else:
    censor = np.max(cen)
    dur_index = np.argmax(censor)
    duration = dur[np.argmax(cen)]

#record final censor and duration for patient:
print('Final censor value: ',censor)
print('Final duration: ',duration)
```

Loading [MathJax]/extensions/Safe.js

```
Censor fill 2:  [0, 0]
Duration fill 2:  [56, 122]
Censor fill 3:  [0, 0, 0]
Duration fill 3:  [56, 122, 177]
Censor fill 4:  [0, 0, 0, 1]
Duration fill 4:  [56, 122, 177, 147]
Censor fill 5:  [0, 0, 0, 1, 0]
Duration fill 5:  [56, 122, 177, 147, 291]
Censor fill 6:  [0, 0, 0, 1, 0, 0]
Duration fill 6:  [56, 122, 177, 147, 291, 317]
Final censor value:  1
Final duration:  147
```

In [25]:
```
new_df
```

Out[25]:

	drug	fill_date	end_date	days_covered	grace_date
0	Ozempic	2023-01-18	2023-03-15	56	2023-04-14
1	Ozempic	2023-03-23	2023-04-20	28	2023-05-20
2	Ozempic	2023-05-17	2023-06-14	28	2023-07-14
3	Ozempic	2023-08-14	2023-09-11	28	2023-10-11
4	Ozempic	2023-09-08	2023-10-06	28	2023-11-05
5	Ozempic	2023-10-05	2023-11-02	28	2023-12-02

In [26]:
```
test_patient.columns
```

Out[26]:
```
Index(['date', 'patient_id', 'claim_id', 'pharmacy_id', 'rx_id', 'rx_txn_id',
       'new_claim_id', 'patient_pay_amt', 'refill_allowed', 'refill_num',
       'refill_due', 'refill_past_due_days', 'drug_info', 'disp_qty',
       'days_supply', 'insur_type', 'year_birth', 'gender', 'patient_pay',
       'claim_resp', 'total-paid', 'payor_pay', 'ndc', 'drug_name',
       'dose_form', 'manuf', 'ta_1', 'ta_2', 'other_id', 'rxgx', 'ndc_group',
       'drug', 'pat_txn'],
      dtype='object')
```

In [27]:
```
##For censored patients
covered_days= new_df.loc[:np.argmax(cen)-1,'days_covered'].sum()
total_days =  (pd.to_datetime(new_df.loc[np.argmax(cen)-1,'end_date'])
                 - pd.to_datetime(new_df.fill_date.min()))
pdc = covered_days/total_days.days

print("Covered Days: ",covered_days)
print("Total Days: ",total_days.days)
print(f"Proportion of Days Covered (PDC): {pdc*100: .1f}%")
```

```
Covered Days:  112
Total Days:  147
Proportion of Days Covered (PDC):  76.2%
```

Create a Final Patient Data Set with Censor Information

In [28]:
```
####  VERSION 2: PDC for Censored Patients Calculated at point of censoring
# Calculate overall PDC, Censor and Duration ##################################################
# SOURCE:   PDC: https://medium.com/algorexhealth/prescription-days-and-medication-management-c
# SOURCE: Date Time: https://blog.finxter.com/how-to-add-days-to-a-pandas-date-column/

#variables to start
grace_period = 30 #days
```

Loading [MathJax]/extensions/Safe.js

```
censor=[]
duration = []  #duration for non-censored should be study end_date - first fill date
            #duration for a censored event should be the last_fill_date+days_supply+grace_peri

total_fills=[]
fills_wegovy = []
fills_ozempic = []
drug = []
gender= []
avg_patient_pay_fill = []
avg_total_pay_fill = []
days_covered = []
days_total = []
avg_age = []
fills_commercial = []
fills_medicare = []
fills_medicaid= []
fills_govt= []
payor= []
payor_num= []
helper_p = ""


#Cycle through each unique patient to calculate PDC, Censor, and Duration Values
for p in range(len(unique_patient)):

    #Isolate data for each patient:
    test_patient = brand_df[brand_df.patient_id==unique_patient[p]].reset_index(drop=True)

    ### Creating a Fill Level Df for Each Patient   ############################################

    start_date = test_patient.date.min()
    end_date = df.date.max()
    fill_dates = test_patient.date
    days_supply = test_patient.days_supply
    current_drug = test_patient.drug

    index_range = pd.date_range(start=start_date, end=end_date, freq='D')
    #calculate covered perieod for each fill
    supply_deltas = pd.Series([pd.to_timedelta(days, unit='D') for days in days_supply], index

    supply_enddates = (pd.to_datetime(fill_dates) + pd.to_timedelta(days_supply, unit='D'))

    #create a data frame for the pateint for key dates for the patient
    new_df = pd.DataFrame({'drug':current_drug,'fill_date': supply_deltas.index,'end_date':sup
                           'days_covered':days_supply})
    new_df = new_df.sort_values('fill_date').reset_index(drop=True)  #sort in order of occurar

    #calculate a grace period date
    new_df['grace_date'] = new_df['end_date']+datetime.timedelta(grace_period)
    #reformat to_datetime
    new_df['fill_date'] = pd.to_datetime(new_df['fill_date'])


    ### Censor and Duraton Calc Logic ########################################################

    ## Censoring and variables to track each fill
    cen = []
    dur = []
    dur_helper = ''

    #set first fill censor value to zero for this patient
    cen.append(0)
Loading [MathJax]/extensions/Safe.js dur as the first days_covered/supply for patient
```

```
dur.append(new_df.days_covered[0])

#logic to check fills for whehter the patient refills within the grace window
#if not, then censored. Records duration of therapy in each case:

for i in range(1,len(new_df),1): #skip the first row of df
    #if last fill available, check to see if the grace_date
    #is less then our study period end date. if yes, then censor and record duration
    #if not, don't censor, but record days on therapy
    if i == len(new_df)-1:
        if new_df.loc[i,'grace_date'] < pd.to_datetime(end_date):
            cen.append(1)
            dur_helper = (new_df.end_date.max()-new_df.fill_date.min()).days
            dur.append(dur_helper)
        else:
            cen.append(0)
            dur_helper = (pd.to_datetime(end_date)-new_df.fill_date.min()).days
            dur.append(dur_helper)
    else:
        #check to see if end date current is less than grace date previous,
        #if yes censor = 1 and record days on therapy
        if new_df.loc[i,'fill_date'] < new_df.loc[i-1,'grace_date']:
            cen.append(0)
            dur_helper = (new_df.loc[i,'grace_date']-new_df.fill_date.min()).days
            dur.append(dur_helper)
        else:
            cen.append(1)
            dur_helper = (new_df.loc[i-1,'end_date']-new_df.fill_date.min()).days
            dur.append(dur_helper)

# Calculate final censor and duration values:

# special case for when all censor values are 0
if all([ v == 0 for v in cen ]):
    censor.append(0)
    duration.append(dur[len(new_df)-1])

# else, flag the first one that turns 1
else:
    censor.append(np.max(cen))
    duration.append(dur[np.argmax(cen)])

### Calculating PDC #######################################################################
new_df = new_df[new_df.end_date < '2023-12-01'].reset_index(drop=True)

if np.argmax(cen) == 1:
    covered_days= new_df.loc[:np.argmax(cen)-1,'days_covered'].sum()
    total_days =  (pd.to_datetime(new_df.loc[np.argmax(cen)-1,'end_date'])
                   - pd.to_datetime(new_df.fill_date.min()))
    pdc = covered_days/total_days.days


else:
    #calculate PDC
    covered_days =new_df.days_covered.sum()
    total_days =  pd.to_datetime(end_date) - pd.to_datetime(new_df.fill_date.min())

    pdc = covered_days/total_days.days

#certain cases when you titrate up to next ndc will have multiple fills and pdc will be gr
if pdc >1:
    pdc = 1

          tracking list of patients
```

Loading [MathJax]/extensions/Safe.js

```
    pdc_ratio.append(pdc)


    ### Calculating Other Patient-Level Feautres   ###########################################
    total_fills.append(len(test_patient))
    fills_wegovy.append(len(test_patient[test_patient.drug == 'Wegovy']))
    fills_ozempic.append(len(test_patient[test_patient.drug == 'Ozempic']))
    if len(test_patient[test_patient.drug == 'Ozempic']) > len(test_patient[test_patient.drug
        drug.append('Ozempic')
    else:
        drug.append('Wegovy')

    gender.append(test_patient.gender[0])
    avg_patient_pay_fill.append(test_patient['patient_pay'].mean())
    avg_total_pay_fill.append(test_patient['total-paid'].mean())
    days_covered.append(covered_days)
    days_total.append(total_days)
    avg_age.append(2023-test_patient.year_birth[0])
    fills_commercial.append(len(test_patient[test_patient.insur_type == 'Commercial']))
    fills_medicare.append(len(test_patient[test_patient.insur_type == 'Medicare Part D']))
    fills_medicaid.append(len(test_patient[(test_patient.insur_type == 'Managed Care Medicaid'
                                           (test_patient.insur_type == 'Medicaid')]))
    fills_govt.append(len(test_patient[test_patient.insur_type == 'Government']))

    #create an array to determine payor
    payor_array = [len(test_patient[test_patient.insur_type == 'Commercial']),
                   len(test_patient[test_patient.insur_type == 'Medicare Part D']),
                   len(test_patient[(test_patient.insur_type == 'Managed Care Medicaid')|
                                    (test_patient.insur_type == 'Medicaid')]),
                   len(test_patient[test_patient.insur_type == 'Government'])]
    payor_choice = ['Commercial','Medicare','Medicaid','Govenrment']

    payor_num.append(np.argmax(payor_array))
    payor.append(payor_choice[np.argmax(payor_array)])
```

In [29]:

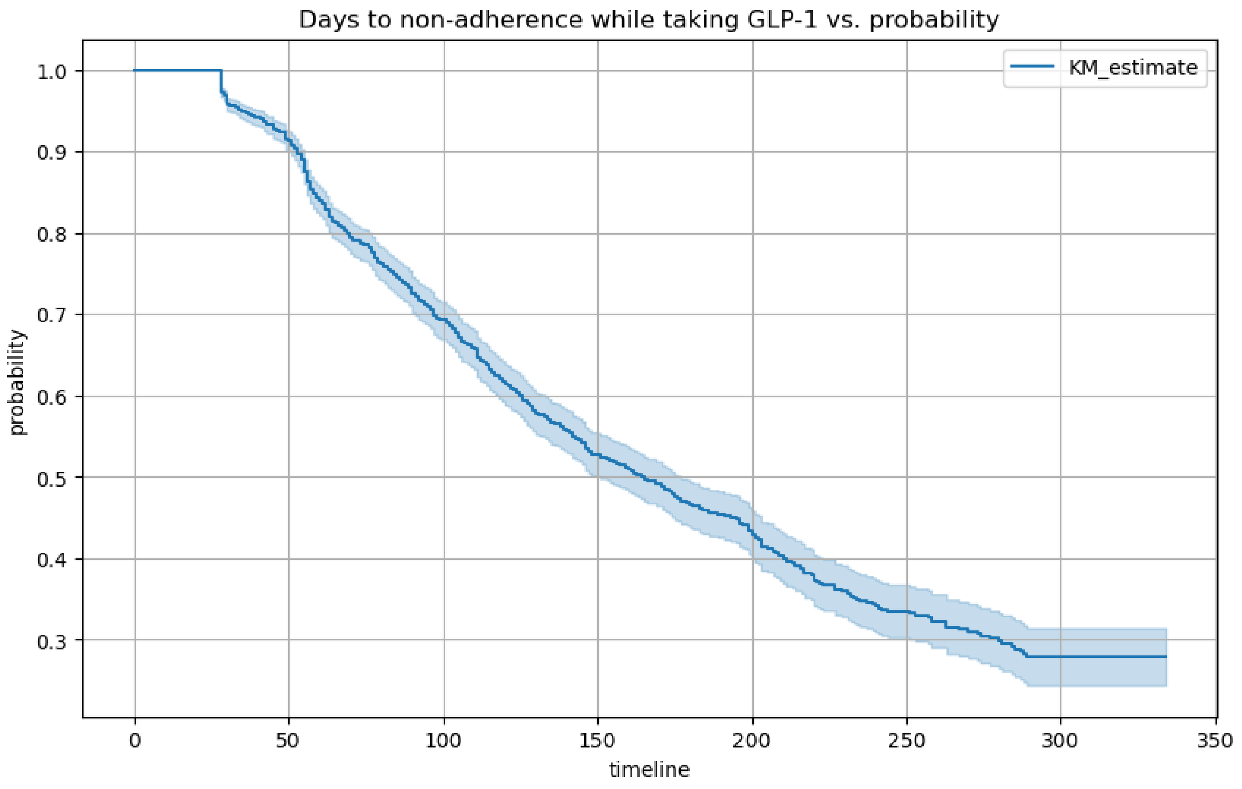
```
#Start a patient dataframe
hazard_df = pd.DataFrame({'patient_id':unique_patient,'pdc': pdc_ratio,'censor':censor,'durati
                           'total_fills':total_fills,'fills_wegovy':fills_wegovy,'fills_ozempi
                           'drug':drug,'gender':gender,'avg_patient_pay_fill':avg_patient_pay_
                           'avg_total_pay_fill': avg_total_pay_fill,'days_covered':days_covere
                           'avg_age':avg_age,'fills_commercial':fills_commercial,'fills_medica
                           'fills_medicaid':fills_medicaid,'fills_govt':fills_govt,'payor':pay
                         'payor_num':payor_num})
```

My hard fought Kaplan Meiers Curves

In [30]:

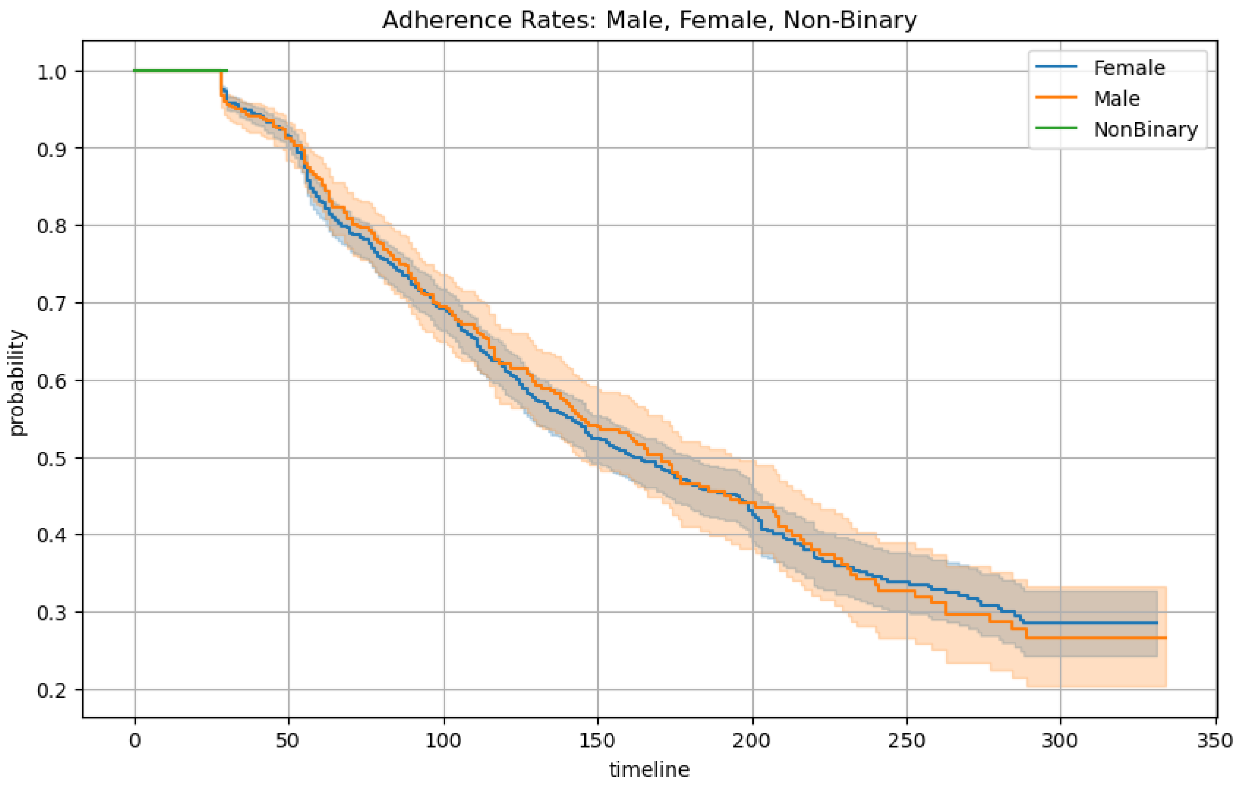
```
kmf = KaplanMeierFitter()
T = hazard_df['duration']
C = hazard_df['censor']
kmf.fit(T,C);

fig, ax = plt.subplots(figsize=(10,6));
plt.title('Days to non-adherence while taking GLP-1 vs. probability')
plt.ylabel('probability')
plt.xlabel('days')
kmf.plot();
ax.grid();
```

Loading [MathJax]/extensions/Safe.js



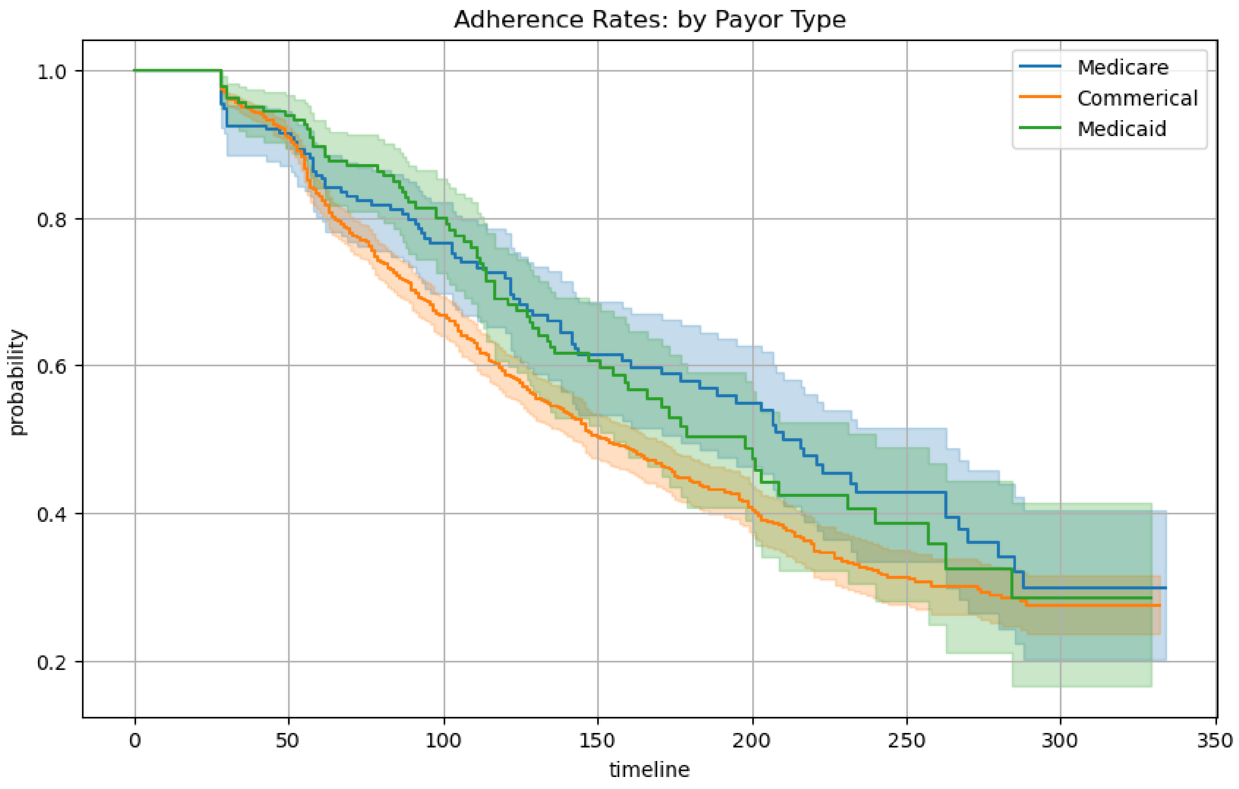
In [31]:
```python
#Create separate dfs for male/female/non-binary
men_df = hazard_df[hazard_df.gender == 'Male']
women_df =hazard_df[hazard_df.gender == 'Female']
nonbin_df =hazard_df[hazard_df.gender == 'NonBinary']

#Create separate models
kmf_m = KaplanMeierFitter()
T_m = men_df.duration
C_m = men_df.censor
kmf_m.fit(T_m,C_m);

kmf_w = KaplanMeierFitter()
T_w = women_df.duration
C_w = women_df.censor
kmf_w.fit(T_w,C_w)

kmf_nb = KaplanMeierFitter()
T_nb = nonbin_df.duration
C_nb = nonbin_df.censor
kmf_nb.fit(T_nb,C_nb)


#Plot curve
fig, ax = plt.subplots(figsize=(10,6));
plt.title('Adherence Rates: Male, Female, Non-Binary')
plt.ylabel('probability')
kmf_w.plot(label='Female');
kmf_m.plot(label='Male');
kmf_nb.plot(label='NonBinary');
ax.grid();
```

Loading [MathJax]/extensions/Safe.js



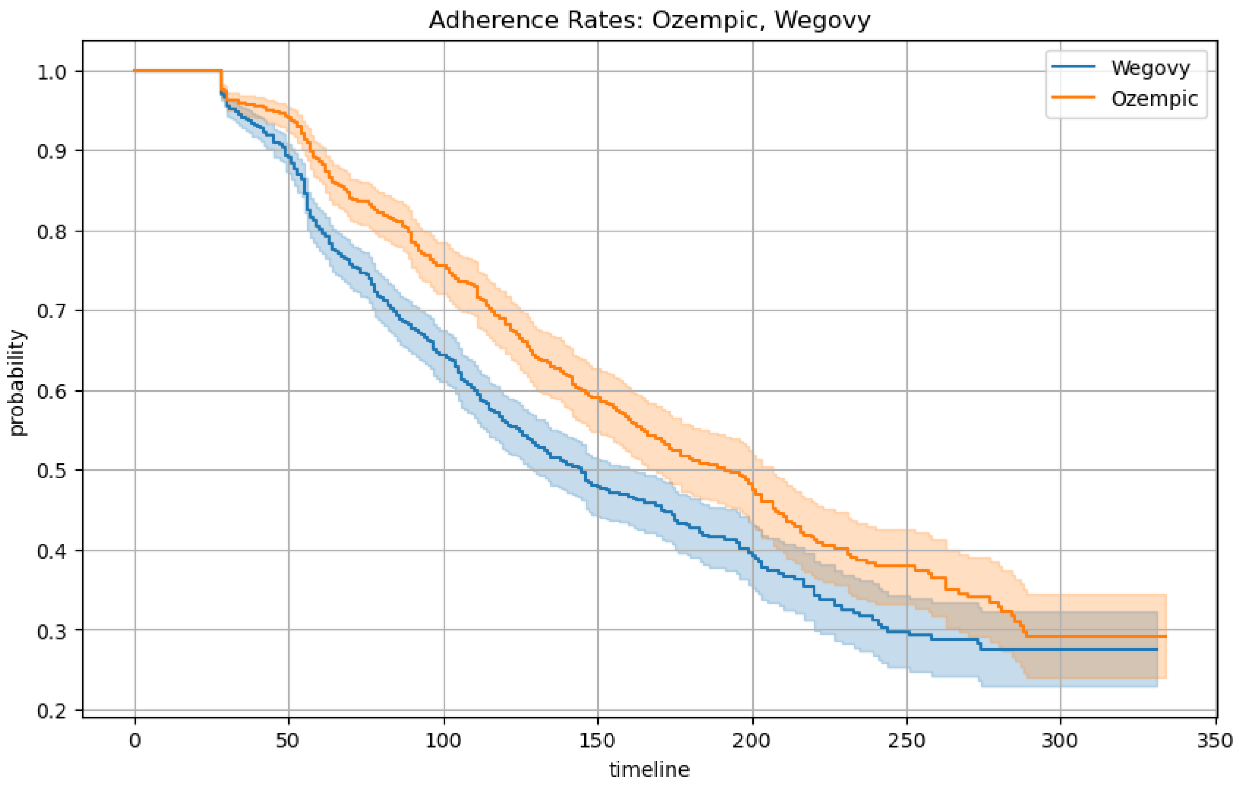
In [32]:
```
#Create separate dfs by payor type
commercial_df = hazard_df[hazard_df.payor == 'Commercial']
medicare_df =hazard_df[hazard_df.payor == 'Medicare']
medicaid_df =hazard_df[hazard_df.payor == 'Medicaid']

#Create separate models
kmf_m = KaplanMeierFitter()
T_m = commercial_df.duration
C_m = commercial_df.censor
kmf_m.fit(T_m,C_m);

kmf_w = KaplanMeierFitter()
T_w = medicare_df.duration
C_w = medicare_df.censor
kmf_w.fit(T_w,C_w)

kmf_nb = KaplanMeierFitter()
T_nb = medicaid_df.duration
C_nb = medicaid_df.censor
kmf_nb.fit(T_nb,C_nb)


#Plot
fig, ax = plt.subplots(figsize=(10,6));
plt.title('Adherence Rates: by Payor Type')
plt.ylabel('probability')
kmf_w.plot(label='Medicare');
kmf_m.plot(label='Commerical');
kmf_nb.plot(label='Medicaid');
ax.grid();
```

Loading [MathJax]/extensions/Safe.js



In [33]:
```
#Create separate dfs for two drugs
o_df = hazard_df[hazard_df.drug == 'Ozempic']
w_df =hazard_df[hazard_df.drug == 'Wegovy']

#Create separate models
kmf_m = KaplanMeierFitter()
T_m = o_df.duration
C_m = o_df.censor
kmf_m.fit(T_m,C_m);

kmf_w = KaplanMeierFitter()
T_w = w_df.duration
C_w = w_df.censor
kmf_w.fit(T_w,C_w)

#Plot kaplan meier curve
fig, ax = plt.subplots(figsize=(10,6));
plt.title('Adherence Rates: Ozempic, Wegovy')
plt.ylabel('probability')
kmf_w.plot(label='Wegovy');
kmf_m.plot(label='Ozempic');
ax.grid();
```

Loading [MathJax]/extensions/Safe.js



Cox Proportional Hazards

In [34]:
```
#Modify data frame to be numeric for the date set
#Create new version of df for this problem
cox_df = hazard_df.loc[:,['censor','duration','avg_age','gender','payor_num','drug','avg_patie

cox_df.gender = [1 if i=='Female' else 0 for i in cox_df.gender]
cox_df.drug = [1 if i=='Wegovy' else 0 for i in cox_df.drug]
```

In [35]:
```
#Create new version of df for this problem
sig_df = cox_df.dropna()

#count number of features to iterate (subtract duration/event cols)
feat_num = len(sig_df.columns) - 2

for i in range(feat_num):
    #instantiate a model
    cph = CoxPHFitter()
    cph.fit(sig_df, duration_col='duration', event_col='censor')

    #create a dataframe with summary
    cph_df = cph.summary

    #check to see if no coef are >0.05, and if so pass
    if len(cph_df[cph_df.p>0.05]) == 0:
        pass
    #check to see what the highest p-value is and drop that coef's column from dataframe
    else:
        sort = cph_df.p.sort_values(ascending=False)
        drop_column = sort.index[0]
        #drop column from data frame
        sig_df.drop(drop_column,axis=1, inplace=True)

#Print resulting model summary:
```

Loading [MathJax]/extensions/Safe.js

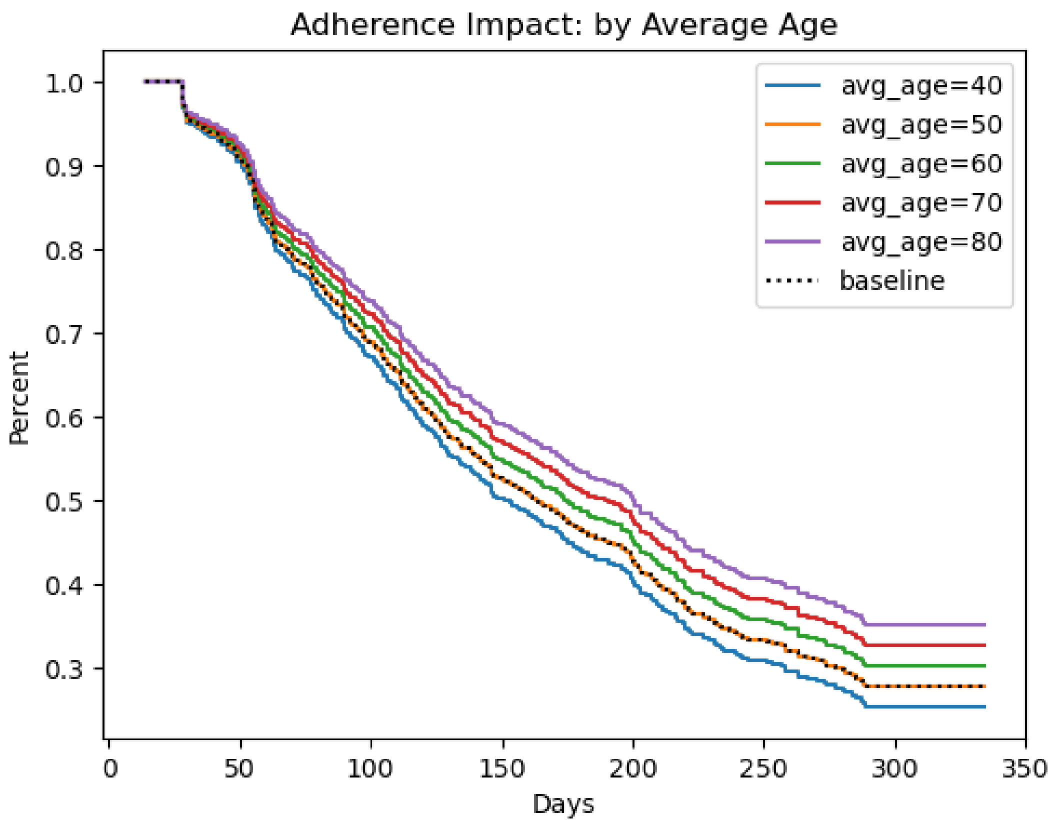
Out[35]:

covariate	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	cmp to	z	p
avg_age	-0.006729	0.993293	0.002772	-0.012163	-0.001296	0.987911	0.998705	0.0	-2.427376	0.015208
drug	0.213751	1.238314	0.072302	0.072042	0.355460	1.074701	1.426837	0.0	2.956378	0.003113
pdc	0.455690	1.577261	0.158144	0.145734	0.765646	1.156888	2.150384	0.0	2.881489	0.003958

In [36]:
```
baseline_ci = cph.concordance_index_
print('Baseline CPH Concordance Index: ',baseline_ci)
```

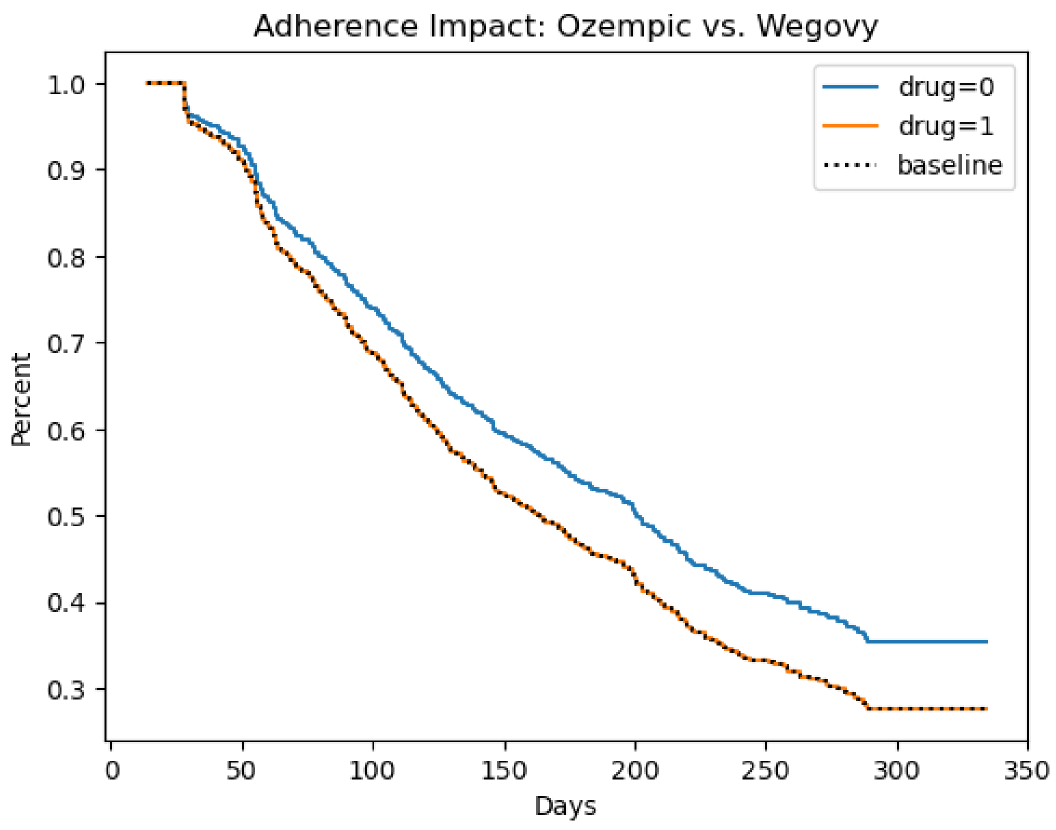
```
Baseline CPH Concordance Index:  0.6199891515635461
```

In [37]:
```
cph.plot_partial_effects_on_outcome(covariates=['avg_age'],values= np.arange(40, 90, 10))
plt.title('Adherence Impact: by Average Age')
plt.xlabel('Days')
plt.ylabel('Percent');
```

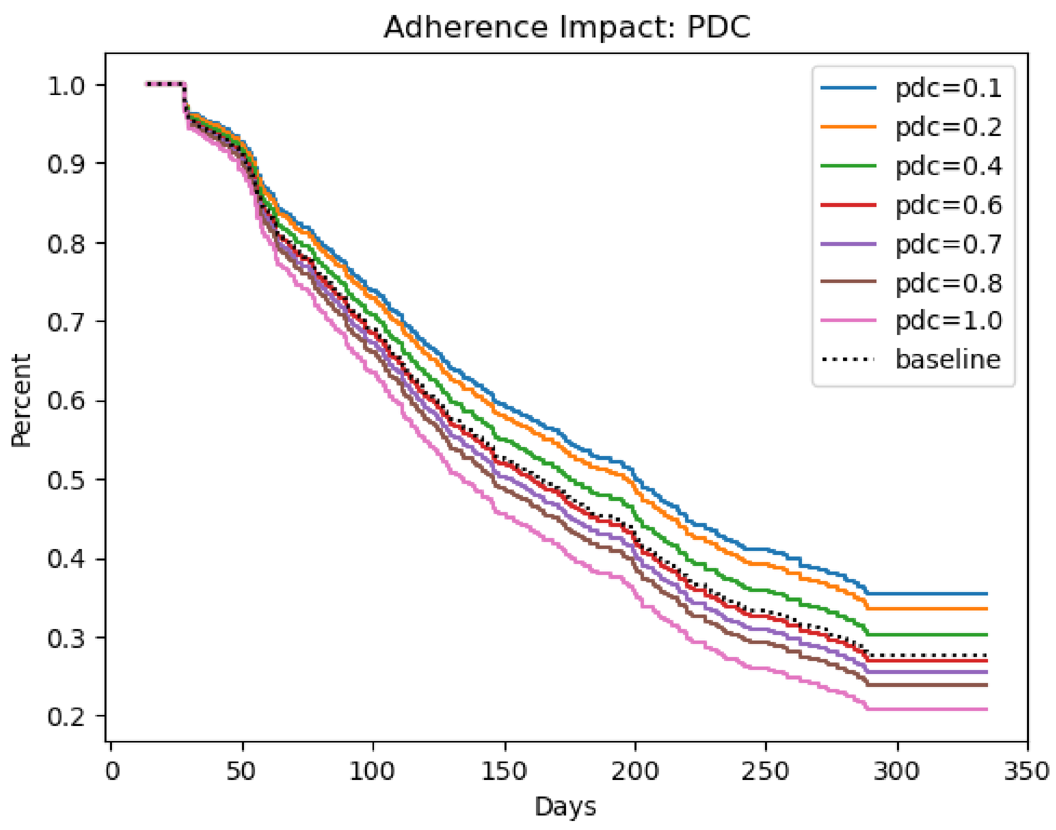


In [38]:
```
cph.plot_partial_effects_on_outcome(covariates=['drug'],values=[0,1])
plt.title('Adherence Impact: Ozempic vs. Wegovy')
plt.xlabel('Days')
plt.ylabel('Percent');
```

Loading [MathJax]/extensions/Safe.js



In [39]:
```
cph.plot_partial_effects_on_outcome(covariates=['pdc'],values=[0.1,0.2,0.4,0.6,0.7,0.8,1])
plt.title('Adherence Impact: PDC')
plt.xlabel('Days')
plt.ylabel('Percent');
```



Loading [MathJax]/extensions/Safe.js

Predicting Survival

REFERENCES

General

https://humboldt-wi.github.io/blog/research/information_systems_1920/group2_survivalanalysis/

Random Forrest https://scikit-survival.readthedocs.io/en/stable/api/generated/sksurv.ensemble.RandomSurvivalForest.html

https://square.github.io/pysurvival/models/random_survival_forest.html

https://notebook.community/sebp/scikit-survival/examples/00-introduction

Evaluating Survival Models https://scikit-survival.readthedocs.io/en/stable/user_guide/evaluating-survival-models.html

https://scikit-survival.readthedocs.io/en/latest/api/generated/sksurv.metrics.integrated_brier_score.html

https://scikit-survival.readthedocs.io/en/latest/api/generated/sksurv.metrics.concordance_index_censored.html

Package Imports

In [40]:
```
# Install packages
# pip install scikit-survival
# pip install random-survival-forest
```

In [41]:
```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline

from sksurv.linear_model import CoxPHSurvivalAnalysis, CoxnetSurvivalAnalysis
from sksurv.metrics import (
    concordance_index_censored,
    concordance_index_ipcw,
    cumulative_dynamic_auc,
    integrated_brier_score,
)
from sksurv.preprocessing import OneHotEncoder
from sksurv.ensemble import RandomSurvivalForest
from sksurv.metrics import concordance_index_censored
from sksurv.metrics import integrated_brier_score


rstate = 82
```

In [42]:
```
#https://github.com/sebp/scikit-survival/blob/master/sksurv/datasets/base.py

def _get_x_y_survival(dataset, col_event, col_time, val_outcome):
    if col_event is None or col_time is None:
        y = None
        x_frame = dataset
```

Loading [MathJax]/extensions/Safe.js

```
        y = np.empty(dtype=[(col_event, bool), (col_time, np.float64)], shape=dataset.shape[0
        y[col_event] = (dataset[col_event] == val_outcome).values
        y[col_time] = dataset[col_time].values

        x_frame = dataset.drop([col_event, col_time], axis=1)

    return x_frame, y
```

Prepare Data

In [43]:
```
cox_new_df = hazard_df

#Get dummies/One Hot Encode categorical variables
ohe_cols = ['drug','gender','payor']
cox_new_df = pd.get_dummies(cox_new_df, columns = ohe_cols)

#change type of the get dummies variables to int
cox_new_df.iloc[:,16:25]= cox_new_df.iloc[:,16:25].astype(int)
#drop NAs
cox_new_df = cox_new_df.dropna()
#drop patient id field
cox_new_df = cox_new_df.drop('patient_id',axis=1)
```

In [44]:
```
#Information on variables used in model
cox_new_df.info()
```

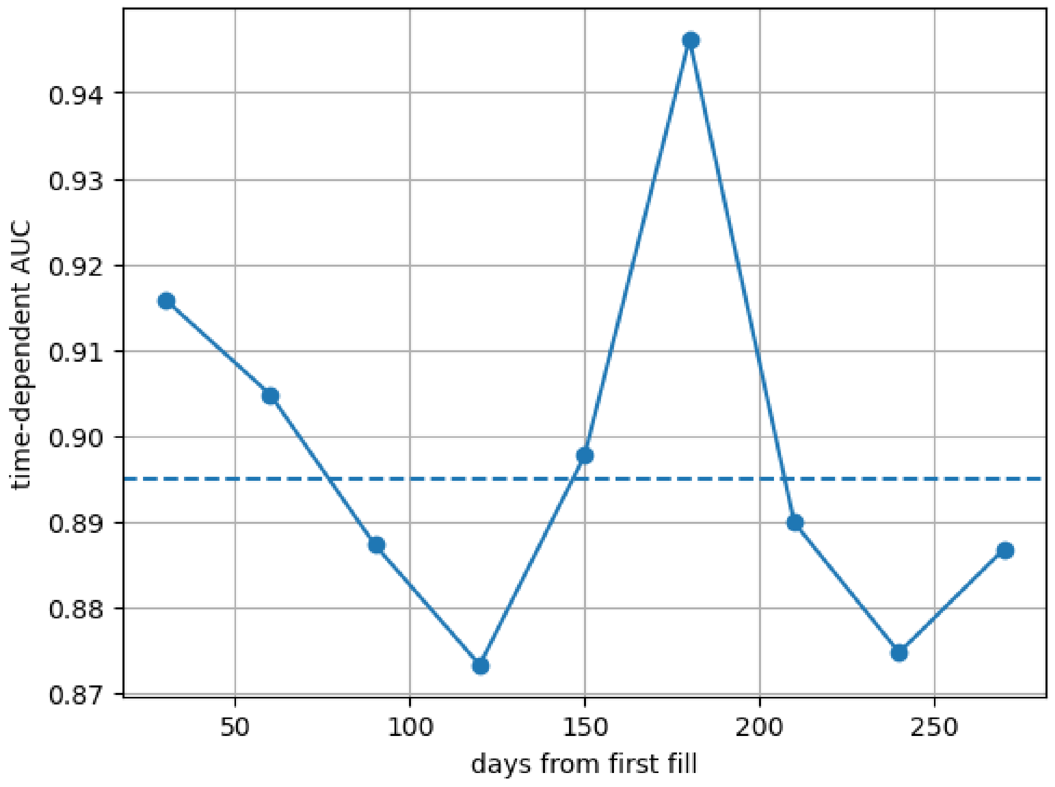
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2908 entries, 0 to 3161
Data columns (total 25 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   pdc                   2908 non-null   float64
 1   censor                2908 non-null   int64
 2   duration              2908 non-null   int64
 3   total_fills           2908 non-null   int64
 4   fills_wegovy          2908 non-null   int64
 5   fills_ozempic         2908 non-null   int64
 6   avg_patient_pay_fill  2908 non-null   float64
 7   avg_total_pay_fill    2908 non-null   float64
 8   days_covered          2908 non-null   int64
 9   avg_age               2908 non-null   float64
 10  fills_commercial      2908 non-null   int64
 11  fills_medicare        2908 non-null   int64
 12  fills_medicaid        2908 non-null   int64
 13  fills_govt            2908 non-null   int64
 14  payor_num             2908 non-null   int64
 15  drug_Ozempic          2908 non-null   int32
 16  drug_Wegovy           2908 non-null   int32
 17  gender_Female         2908 non-null   int32
 18  gender_Male           2908 non-null   int32
 19  gender_NonBinary      2908 non-null   int32
 20  gender_Unknown        2908 non-null   int32
 21  payor_Commercial      2908 non-null   int32
 22  payor_Govenrment      2908 non-null   int32
 23  payor_Medicaid        2908 non-null   int32
 24  payor_Medicare        2908 non-null   uint8
dtypes: float64(4), int32(9), int64(11), uint8(1)
memory usage: 468.6 KB
```

In [45]:
```
# Split the data into train/test subsets
X, y = _get_x_y_survival(cox_new_df, 'censor', 'duration', 1)
Loading [MathJax]/extensions/Safe.js , y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=rstate
In [46]: #Save indices
train_index = X_train.index.tolist()
test_index = X_test.index.tolist()
```

CPH Prediction

```
In [47]: # Split the data into train/test subsets
         cph = CoxPHSurvivalAnalysis()
         cph.fit(X_train, y_train)

Out[47]: ▾ CoxPHSurvivalAnalysis
         CoxPHSurvivalAnalysis()
```

```
In [48]: times = np.arange(30, 300, 30)
         cph_risk_scores = cph.predict(X_test)
         cph_auc, cph_mean_auc = cumulative_dynamic_auc(y_train, y_test, cph_risk_scores, times)

         plt.plot(times, cph_auc, marker="o")
         plt.axhline(cph_mean_auc, linestyle="--")
         plt.xlabel("days from first fill")
         plt.ylabel("time-dependent AUC")
         plt.grid(True)
```



```
In [49]: cph_ci = cph.score(X_test, y_test)
         print("C-index", cph_ci)

C-index 0.8278483831459066
```

Random Forest Prediction

```
                  vivalForest(n_estimators=50,
                              min_samples_split=7,
```

Loading [MathJax]/extensions/Safe.js

```
                           min_samples_leaf=10,
                           max_features="sqrt",
                           n_jobs=-1,
                           random_state=rstate,
                           verbose=1)

rsf.fit(X_train, y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  3.1min finished
```

Out[50]:

```
▾                       RandomSurvivalForest
RandomSurvivalForest(min_samples_leaf=10, min_samples_split=7, n_estimators=50,
                     n_jobs=-1, random_state=82, verbose=1)
```

In [51]:

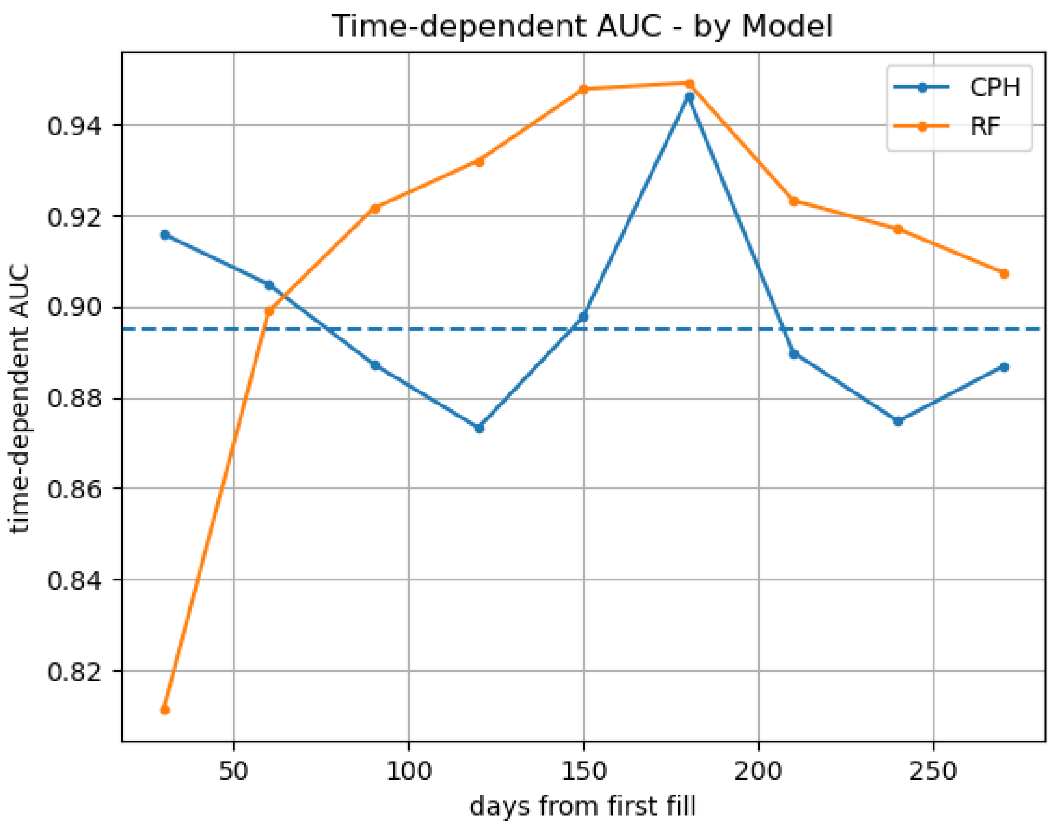
```
y_pred = rsf.predict(X_test)

rsf_ci = rsf.score(X_test, y_test)
print("C-index", rsf_ci)
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done  50 out of  50 | elapsed:    0.0s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
C-index 0.8678442475623916
[Parallel(n_jobs=8)]: Done  50 out of  50 | elapsed:    0.0s finished
```

In [52]:

```
#### Plot of Time Dependent AUC
cph_auc, cph_mean_auc = cumulative_dynamic_auc(y_train, y_test, cph_risk_scores, times)
rf_auc, rf_mean_auc = cumulative_dynamic_auc(y_train, y_test, y_pred, times)

plt.plot(times, cph_auc, marker=".",label='CPH')
plt.plot(times, rf_auc, marker=".",label='RF')
plt.axhline(cph_mean_auc, linestyle="--")
plt.title("Time-dependent AUC - by Model")
plt.xlabel("days from first fill")
plt.ylabel("time-dependent AUC")
plt.legend()
plt.grid(True)
```

Loading [MathJax]/extensions/Safe.js



Gradient Boost Prediction

References:

Boostinghttps://scikit-survival.readthedocs.io/en/stable/user_guide/boosting.html

XGBSE Not used but for future reference https://towardsdatascience.com/xgbse-improving-xgboost-for-survival-analysis-393d47f1384a

In [53]:
```
from sksurv.ensemble import ComponentwiseGradientBoostingSurvivalAnalysis
from sksurv.ensemble import GradientBoostingSurvivalAnalysis
```

In [54]:
```
gb_cph_tree = GradientBoostingSurvivalAnalysis(n_estimators=100, learning_rate=1.0, max_depth=
gb_cph_tree.fit(X_train, y_train)
ci_gb = gb_cph_tree.score(X_test, y_test)
print('C-Index: ',ci_gb)
```

```
C-Index:  0.883774197140085
```

In [55]:
```
concordance_df = pd.DataFrame({"Model": ['CPH','Random Forest','GB'], "C-index": [cph_ci,rsf_c
concordance_df
```

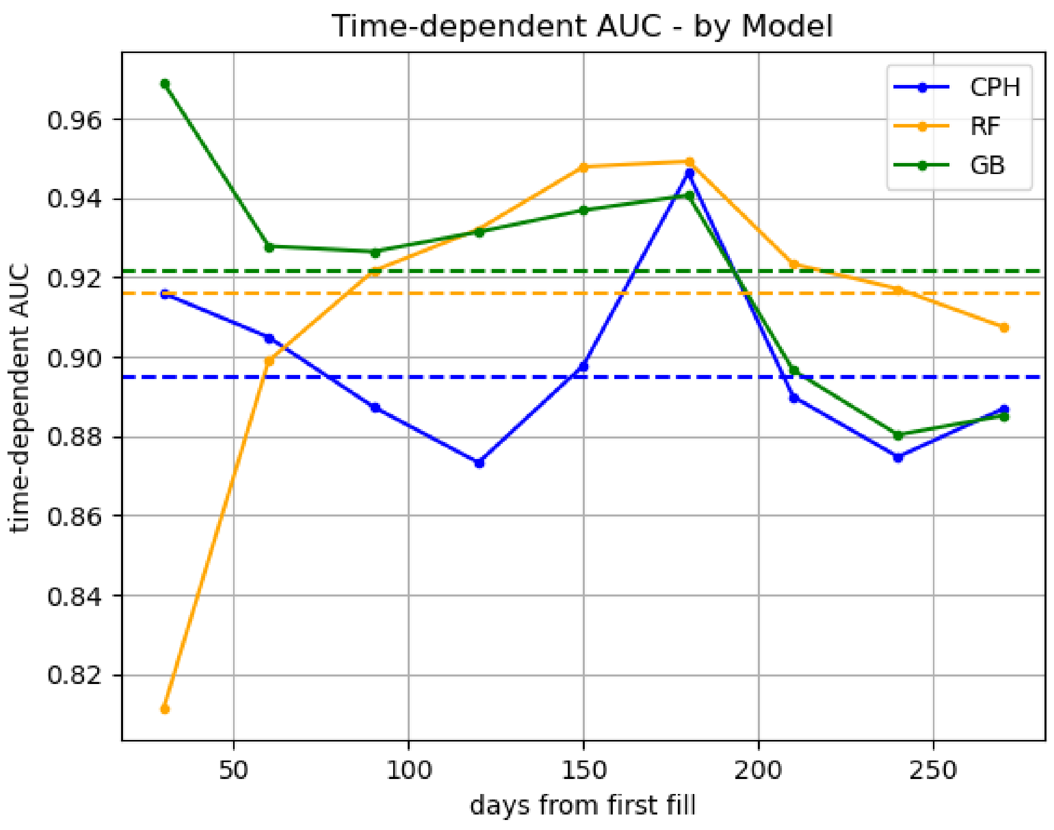
Out[55]:

	Model	C-index
0	CPH	0.827848
1	Random Forest	0.867844
2	GB	0.883774

Loading [MathJax]/extensions/Safe.js

```
me Dependent AUC
gb_pred = gb_cph_tree.predict(X_test)
```

```
gb_auc, gb_mean_auc = cumulative_dynamic_auc(y_train, y_test, gb_pred, times)

plt.plot(times, cph_auc, marker=".",label='CPH',color='blue')
plt.plot(times, rf_auc, marker=".",label='RF',color='orange')
plt.plot(times, gb_auc, marker=".",label='GB',color='green')
plt.axhline(cph_mean_auc, linestyle="--",color='blue')
plt.axhline(rf_mean_auc, linestyle="--",color='orange')
plt.axhline(gb_mean_auc, linestyle="--",color='green')
plt.title("Time-dependent AUC - by Model")
plt.xlabel("days from first fill")
plt.ylabel("time-dependent AUC")
plt.legend()
plt.grid(True)
```



Integrated Brier Score

Loading [MathJax]/extensions/Safe.js

Time-dependent Brier Score

The time-dependent Brier score is an extension of the mean squared error to right censored data. Given a time point t, it is defined as:

$$\mathrm{BS}^c(t) = \frac{1}{n} \sum_{i=1}^{n} I(y_i \leq t \land \delta_i = 1) \frac{(0 - \hat{\pi}(t|\mathbf{x}_i))^2}{\hat{G}(y_i)} + I(y_i > t) \frac{(1 - \hat{\pi}(t|\mathbf{x}_i))^2}{\hat{G}(t)},$$

where $\hat{\pi}(t|\mathbf{x})$ is a model's predicted probability of remaining event-free up to time point t for feature vector $\mathbf{x}$, and $1/\hat{G}(t)$ is an inverse probability of censoring weight.

Note that the time-dependent Brier score is only applicable for models that are able to estimate a survival function. For instance, it cannot be used with Survival Support Vector Machines.

The Brier score is often used to assess calibration. If a model predicts a 10% risk of experiencing an event at time t, the observed frequency in the data should match this percentage for a well calibrated model. In addition, the Brier score is also a measure of discrimination: whether a model is able to predict risk scores that allow us to correctly determine the order of events. The concordance index is probably the most common measure of discrimination. However, the concordance index disregards the actual values of predicted risk scores – it is a ranking metric – and is unable to tell us anything about calibration.

In [57]:
```
rsf_surv_prob = np.row_stack([fn(times) for fn in rsf.predict_survival_function(X_test)])
cph_surv_prob = np.row_stack([fn(times) for fn in cph.predict_survival_function(X_test)])
gb_surv_prob = np.row_stack([fn(times) for fn in gb_cph_tree.predict_survival_function(X_test)
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done  50 out of  50 | elapsed:    0.0s finished
```

In [58]:
```
score_brier = pd.Series(
    [
        integrated_brier_score(y, y_test, prob, times)
        for prob in (rsf_surv_prob, cph_surv_prob,gb_surv_prob)
    ],
    index=[0,1,2],
    name="IBS",
)

pd.concat((concordance_df, score_brier), axis=1).round(3)
```

Out[58]:

	Model	C-index	IBS
0	CPH	0.828	0.099
1	Random Forest	0.868	0.587
2	GB	0.884	0.110

In [ ]:

Loading [MathJax]/extensions/Safe.js