```
drive.mount('/content/drive')
pd.set_option('display.max_columns', None)
```

```
Mounted at /content/drive
```

```
!pip install tableone
```

```
Collecting tableone
  Downloading tableone-0.9.5-py3-none-any.whl.metadata (7.3 kB)
Requirement already satisfied: jinja2>=3.1.4 in /usr/local/lib/python3.11/dist-packages (from tableone) (3.1.6)
Requirement already satisfied: numpy>=1.19.1 in /usr/local/lib/python3.11/dist-packages (from tableone) (2.0.2)
Requirement already satisfied: openpyxl>=3.1.2 in /usr/local/lib/python3.11/dist-packages (from tableone) (3.1.5)
Requirement already satisfied: pandas>=2.0.3 in /usr/local/lib/python3.11/dist-packages (from tableone) (2.2.2)
Requirement already satisfied: scipy>=1.10.1 in /usr/local/lib/python3.11/dist-packages (from tableone) (1.15.3)
Requirement already satisfied: statsmodels>=0.14.1 in /usr/local/lib/python3.11/dist-packages (from tableone) (0.14.4)
Requirement already satisfied: tabulate>=0.9.0 in /usr/local/lib/python3.11/dist-packages (from tableone) (0.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2>=3.1.4->tableone) (3.
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.11/dist-packages (from openpyxl>=3.1.2->tableone) (2.0.0
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=2.0.3->tableo
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=2.0.3->tableone) (2025.
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=2.0.3->tableone) (202
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-packages (from statsmodels>=0.14.1->tableone)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels>=0.14.1->tableon
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>=2.0
Downloading tableone-0.9.5-py3-none-any.whl (43 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.3/43.3 kB 1.3 MB/s eta 0:00:00
Installing collected packages: tableone
Successfully installed tableone-0.9.5
```

```
import pandas as pd
import numpy as np
from google.colab import drive
from tableone import TableOne
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import StratifiedGroupKFold
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.linear_model import LogisticRegression
```

1. Data Cleaning

First, we take a very basic look at the first 20 rows of the table to get a sense of the data:

```
data = pd.read_csv('/content/drive/My Drive/data.csv')
print(data.shape)
print(data.head(20))
```

```
7         110.217357          78.235861       52.285790                NaN
```

```
8         101.217357          71.235861       52.852790                NaN
9         101.217357          66.235861       52.658790                NaN
10        122.217357          73.235861       53.477790          100.703444
11        111.217357          68.235861       52.652790                NaN
12        113.217357          71.235861       52.897790                NaN
13        111.217357          69.235861       52.390790                NaN
14        115.217357          72.235861       52.424790                NaN
15        103.217357          69.235861       52.545790                NaN
16        127.790457          70.037530       75.164936                NaN
17        117.790457          75.037530       83.635936                NaN
18        108.790457          68.037530       79.975936                NaN
19               NaN                NaN       80.409936                NaN

    ejection_fraction
0           65.661492
1           65.661492
2           65.661492
3           65.661492
4           65.661492
5           65.661492
6           65.661492
7           65.661492
8           65.661492
9           65.661492
10          65.661492
11          65.661492
12          65.661492
13          65.661492
14          65.661492
15          65.661492
16          60.129093
17          60.129093
18          60.129093
19          60.129093
```

The above told us that there are a LOT of columns. Let us describe all of them for a very quick eyeball of value distributions:

```
data_summary = data.describe(include='all')
data_summary
```

	subject_id	is_eligible	days_since_enrollment	cohort	brthdat_coded	age	sex	sex_coded	dmcbpot_coded	et
count	1549	1549	1399.000000	1368	0.0	1368.000000	1368	1368.000000	730.000000	
unique	72	2	NaN	4	NaN	NaN	2	NaN	NaN	
top	101-0067	True	NaN	Cohort 1	NaN	NaN	Female	NaN	NaN	Hi
freq	30	1323	NaN	484	NaN	NaN	730	NaN	NaN	
mean	NaN	NaN	1.938528	NaN	NaN	57.999269	NaN	1.533626	0.345205	
std	NaN	NaN	9.282897	NaN	NaN	16.913027	NaN	0.499050	0.475761	
min	NaN	NaN	-37.000000	NaN	NaN	18.000000	NaN	1.000000	0.000000	
25%	NaN	NaN	-3.000000	NaN	NaN	47.000000	NaN	1.000000	0.000000	
50%	NaN	NaN	3.000000	NaN	NaN	62.000000	NaN	2.000000	0.000000	
75%	NaN	NaN	9.000000	NaN	NaN	70.000000	NaN	2.000000	1.000000	
max	NaN	NaN	17.000000	NaN	NaN	93.000000	NaN	2.000000	1.000000	

After a quick look above, seems like there are a lot of empty columns. So I am going to choose to remove them to slim down the dataset, bringing it to 76 columns instead of 90

```
non_empty_cols = data_summary.loc['count'] > 0
print(non_empty_cols)
data_no_empty_cols = data.loc[:, non_empty_cols]


print("number of columns after removing empties: " + str(data_no_empty_cols.shape[1]))
```

```
subject_id               True
is_eligible              True
days_since_enrollment    True
```

```
cohort                   True
brthdat_coded           False
                        ...
mean_systolic_bp         True
mean_diastolic_bp        True
mean_weight_kg           True
kccq_summary_score       True
ejection_fraction        True
Name: count, Length: 90, dtype: bool
number of columns after removing empties: 76
```

```
numeric_df = data_no_empty_cols.select_dtypes(include=[np.number])
categorical_df = data_no_empty_cols.select_dtypes(exclude=[np.number])

print(f"Numeric columns: {numeric_df.shape[1]}")
print(f"Categorical columns: {categorical_df.shape[1]}")
```

```
Numeric columns: 57
Categorical columns: 19
```

I want to get a closer look at values to see if there are any obvious coding issues or strange values. So I split the dataset into categorical and numeric variables so that the describe is a little cleaner:

```
numeric_df.describe()
```

	days_since_enrollment	age	sex_coded	dmcbpot_coded	race_coded	vshtu_coded	vswt	vswtu_coded	baseline_
count	1399.000000	1368.000000	1368.000000	730.000000	1368.000000	1368.0	1368.000000	1368.0	1368.00
mean	1.938528	57.999269	1.533626	0.345205	4.361111	2.0	195.688138	2.0	30.70
std	9.282897	16.913027	0.499050	0.475761	1.038967	0.0	49.734863	0.0	7.73
min	-37.000000	18.000000	1.000000	0.000000	1.000000	2.0	109.104768	2.0	19.60
25%	-3.000000	47.000000	1.000000	0.000000	3.000000	2.0	148.128527	2.0	25.20
50%	3.000000	62.000000	2.000000	0.000000	5.000000	2.0	190.556253	2.0	28.90
75%	9.000000	70.000000	2.000000	1.000000	5.000000	2.0	225.828214	2.0	34.10
max	17.000000	93.000000	2.000000	1.000000	5.000000	2.0	317.080020	2.0	57.90

```
categorical_df.describe()
```

	subject_id	is_eligible	cohort	sex	ethnicity	race	race_ethnicity	vsdat	vstim	vshtu	vswtu	vsbpnd	prnd	rrn
count	1549	1549	1368	1368	1368	1368	1368	1368	1368	1368	1368	1368	1368	136
unique	72	2	4	2	2	4	5	54	55	1	1	1	1	
top	101-0067	True	Cohort 1	Female	Not Hispanic or Latino	White	White	3/27/23	11:44:00	in	lb	No	No	N
freq	30	1323	484	730	1265	971	868	58	60	1368	1368	1368	1368	136

Splitting the dataset made a few things more obvious:

- Several categorical variables have single values all the way through. Many variables that fall into this category are related to units
- Several variables have a coded and non-coded version, which is somewhat redundant.
- Days since enrollment can be negative, which is important context when defining "baseline"
- There are some negative values in noisy_hours, activity_hours, lead_on_hours, hrv, and a few columns that relate to num_events. I can't think of a real world scenario in which these values would be negative, but I am going to keep them in the dataset for this model, biasing towards real world conditions as opposed to perfectly clean dataset.
- EF is a numeric value. Our task is to predict EF < 50, so it will need to be coded into binary categories

Knowing that our analysis involves regression influences what variables we keep in our analytical dataset. I am choosing to remove:

- any row where is_eligible = False (appears to be assigned at the patient level)
- cohort variable-- since we have no information on the different cohorts, I am choosing to treat population as one cohort
- Any variable related to units, as they are uniform for all records in this dataset

- Any variable that has a "coded" version (keep only the coded version)
- All categorical variables related to date and time: days_since_enrollment provides us relative day information
- Since this is a simple, initial model, I am going to keep in race_coded and exclude categorical ethnicity. ethnicity can be added in later for more granularity if desired.

Taking all of this together, we retain most of the numerical df and discard the categorical df except for subject_id and is_eligible

```
# remove unwanted columns
selected_numeric_cols = numeric_df.drop(columns = ['vshtu_coded', 'vswtu_coded', 'vstempu_coded'])
selected_categorical_cols = categorical_df[['subject_id', 'is_eligible']]

#rejoin into single dataframe, pre-emptively name _57_cols because we're adding a binary col right after
data_57_cols = pd.concat([selected_numeric_cols, selected_categorical_cols], axis=1)
print(data_57_cols.shape)
```

```
(1549, 56)
```

```
#create new variable binary_ef_less_than_50
data_57_cols['binary_ef_less_than_50'] = np.where(data_57_cols['ejection_fraction'] < 50, 1, 0)
print(data_57_cols['subject_id'].nunique())
print(data_57_cols.shape)
```

```
72
(1549, 57)
```

```
#exclude rows that happened before study start and patients who are ineligible
data_at_or_after_study_start = data_57_cols[
    (data_57_cols['days_since_enrollment'] > 0) &
    (data_57_cols['is_eligible'] == True)
]

#check shape after removing negative enrollment days and ineligible patients
print(data_at_or_after_study_start.shape)
```

```
(825, 57)
```

This is the point at which I am choosing to create my analytical dataset. I want to use un-imputed data in my xgboost model, so I am not including the imputation transformations to the saved dataset. My analytical dataset ends up being 825 rows and 57 columns.

```
#save data to csv
data_at_or_after_study_start.to_csv('/content/drive/My Drive/analytic_dataset.csv', index=True)
```

2. Table 1

Below I create the subset of data used for the Table1. I defined "baseline" as the earliest day after day 0 (days_since_enrollment >=0) , and excluded any measurements taken prior to that. I also removed any subject_ids with is_eligible = False, assuming this means they are not eligible for this study.

```
#check num of unique patients
print(data_at_or_after_study_start['subject_id'].nunique())

#index of minimum date per patient
idx = data_at_or_after_study_start.groupby('subject_id')['days_since_enrollment'].idxmin()

#use indices from above to select row with minimum days_since_enrollment >=0 as baseline
df_one_row_per_patient = data_at_or_after_study_start.loc[idx].reset_index(drop=True)

#check shape. n=58 indicates 14 patients were ineligible and/or had only negative days_since_enrollment
print(df_one_row_per_patient.shape)
```

```
58
(58, 57)
```

```
#rejoin in sex and race for tableone purposes
demographics = categorical_df[['subject_id', 'race', 'sex']].drop_duplicates(subset='subject_id')
```

```
tableone_table = df_one_row_per_patient.merge(demographics, on='subject_id', how='left')

tableone_columns = ['age', 'sex', 'race', 'baseline_bmi',  'oxsat', 'vssysbp', 'vsdiabp', 'mean_heart_rate_variability_ms', 'lea
categorical = ['sex', 'race']
rename = {'baseline_bmi': 'bmi', 'oxsat': 'oxygen saturation', 'vssysbp': 'systolic bp', 'vsdiabp':'diastolic bp', 'mean_heart_r


mytableone = TableOne(tableone_table, columns=tableone_columns, categorical=categorical, rename=rename, groupby= 'binary_ef_less
print(mytableone.tabulate(tablefmt = "fancy_grid"))
```

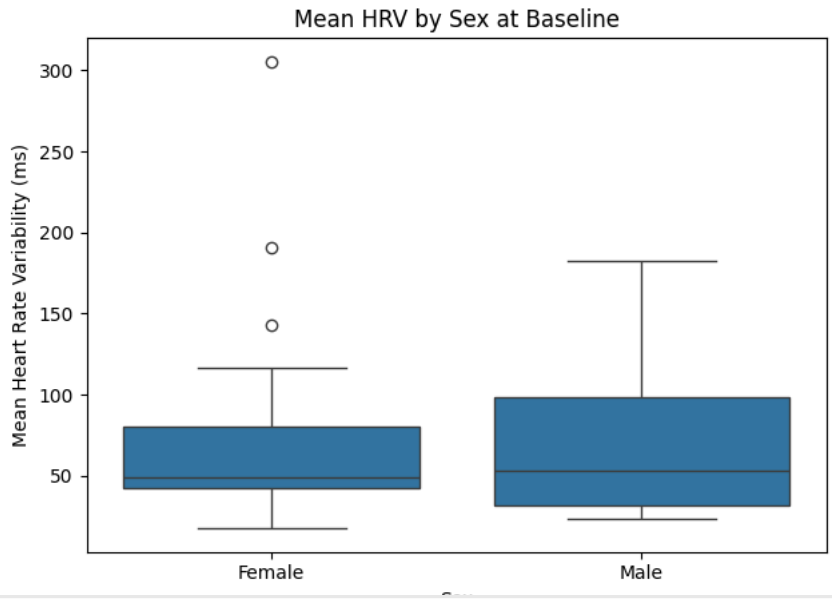
		Missing	Overall	0	1
n			58	51	7
age, mean (SD)		0	57.2 (18.4)	57.0 (18.6)	58.6 (
sex, n (%)	Female		31 (53.4)	29 (56.9)	2 (28.
	Male		27 (46.6)	22 (43.1)	5 (71.
race, n (%)	American Indian or Alaska Native		1 (1.7)	0 (0.0)	1 (14.
	Asian		2 (3.4)	2 (3.9)	0 (0.0
	Black or African American		12 (20.7)	10 (19.6)	2 (28.
	White		43 (74.1)	39 (76.5)	4 (57.
bmi, mean (SD)		0	30.5 (7.8)	31.0 (8.0)	27.3 (
oxygen saturation, mean (SD)		1	97.9 (1.3)	97.9 (1.3)	97.6 (
systolic bp, mean (SD)		0	130.3 (21.7)	130.3 (21.7)	130.5
diastolic bp, mean (SD)		0	79.8 (11.9)	80.6 (10.9)	74.0 (
mean hrv, mean (SD)		0	70.7 (53.4)	70.0 (54.9)	76.3 (
num of hours with lead on, mean (SD)		0	22.4 (5.0)	22.6 (4.5)	20.8 (

This table shows the makeup of the two groups of patients, the 1 group being those who have ef < 50 at baseline, and 0 being those who have ef >= 50 at baseline. While there are visual differences in some of the variables, such as mean hrv and bmi p-values indicate the differences are not significant. Important to note the imbalanced groups, this affects performance later when we are modeling the data, there are many more patients with ef >= 50 than patients with ef < 50 at baseline. These values look reasonable and appropriate given what I know about real world values of these metrics.

3. Data Viz

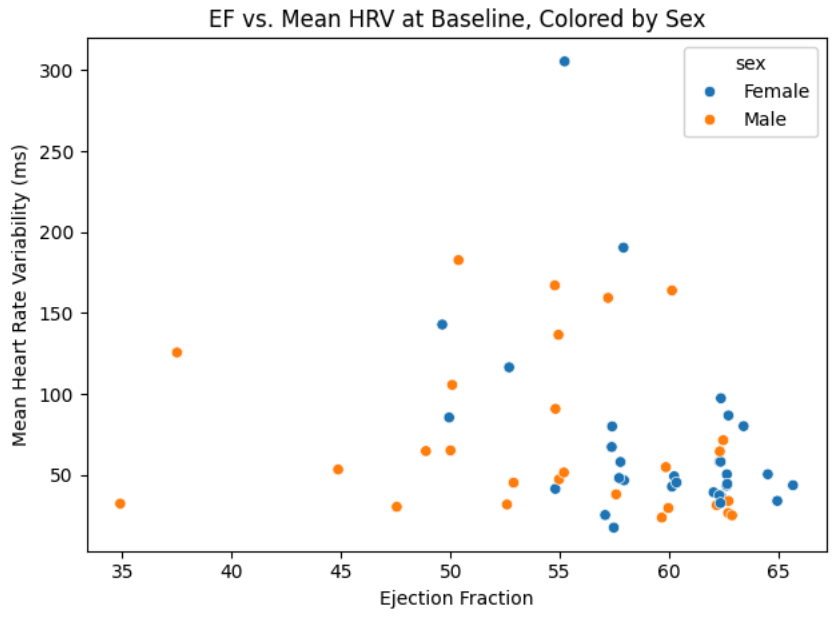
I'm really interested in potential relationships between hrv and EF because I did a project investigating hrv and EEG relationships in schizophrenic patients way back in undergrad. I want to see the variability of mean hrv across patients in this cohort, and also want to see if there is an obvious visual relationship between mean hrv and EF. For simplicity, I am just looking at baseline at the moment since that dataframe is already available to me. I also am choosing to split out men and women to highlight any relationships related to sex.

```
#boxplot by sex of mean hrv at baseline. reuse tableone table because it has everything we need

sns.boxplot(data=tableone_table, x='sex', y='mean_heart_rate_variability_ms')
plt.title('Mean HRV by Sex at Baseline')
plt.xlabel('Sex')
plt.ylabel('Mean Heart Rate Variability (ms)')
plt.tight_layout()
plt.show()
```



From the above boxplot, we can see that the mean baseline HRV is close in both males and females, but males in our cohort have a wider range of mean hrv. The outliers in the female side of the boxplot are pretty high, but still technically physologically possible.

```
#scatterplot of mean hrv and EF, color coded by sex

sns.scatterplot(
    data=tableone_table,
    x= 'ejection_fraction',
    y= 'mean_heart_rate_variability_ms',
    hue='sex'
)
plt.title('EF vs. Mean HRV at Baseline, Colored by Sex')
plt.xlabel('Ejection Fraction')
plt.ylabel('Mean Heart Rate Variability (ms)')
plt.tight_layout()
plt.show()
```



This scatterplot tells a similar visual story to the boxplot: EF and HRV coupled seem to cluster more for females than males. Future investigation would include a line of best fit and a p-value to see if there is any significant linear relationship between these two variables for males or females.

4a. Regression

The below code helps me choose an imputation strategy for variables for my regression model.

- Since kccq score is missing a ton of data, I am going to drop it.
- While max ventricular rate bpm is useful, we have a lot of related rate data and at 64% missingness I am not particularly worried about data loss from dropping this variable.
- I am not particularly interested in childbearing potential for this model, so I will also drop dmcbpot_coded
- The rest of the data has moderate to low missingness, upt to 36%, I am choosing to impute with the per patient mean so that this data can fit into regression model

```
na_summary = data_at_or_after_study_start.isnull().sum().to_frame(name='n_missing')
na_summary['pct_missing'] = data_at_or_after_study_start.isnull().mean() * 100
na_summary = na_summary.sort_values(by='pct_missing', ascending=False)

na_summary
```

	n_missing	pct_missing
kccq_summary_score	795	96.363636
max_ventricular_rate_bpm	529	64.121212
dmcbpot_coded	381	46.181818
sd_abs_max_displacement_uv	303	36.727273
mean_abs_max_displacement_uv	249	30.181818
max_max_displacement_uv	249	30.181818
min_max_displacement_uv	249	30.181818
sd_sleep_hr	131	15.878788
mean_sleep_hr	131	15.878788
min_mean_1min_night_hr	131	15.878788
max_mean_5min_night_hr	131	15.878788
max_mean_1min_night_hr	131	15.878788
min_mean_5min_night_hr	131	15.878788
mean_sleep_rr	130	15.757576
sd_sleep_rr	130	15.757576
mean_systolic_bp	127	15.393939
mean_diastolic_bp	127	15.393939
mean_weight_kg	122	14.787879
max_mean_1min_hr	99	12.000000
max_resting_hr	99	12.000000
noisy_hours	99	12.000000
lead_on_hours	99	12.000000
activity_hours	99	12.000000
num_st_changes	99	12.000000
num_st_change_events	99	12.000000
sd_heart_rate_variability_ms	99	12.000000
mean_heart_rate_variability_ms	99	12.000000
num_nsvt_events	99	12.000000
af_burden_pct	99	12.000000
min_mean_5min_hr	99	12.000000
max_active_hr	99	12.000000
min_mean_1min_hr	99	12.000000
max_mean_5min_hr	99	12.000000
sd_temperature_c	99	12.000000
mean_temperature_c	99	12.000000
oxsat	14	1.696970
height_m	0	0.000000
vswt	0	0.000000
baseline_bmi	0	0.000000
vssysbp	0	0.000000
vsdiabp	0	0.000000
days_since_enrollment	0	0.000000
tempnd_coded	0	0.000000
prnd_coded	0	0.000000
vspulse	0	0.000000
vsrr	0	0.000000

rrnd_coded	0	0.000000
osnd_coded	0	0.000000
vstemp	0	0.000000
vsbpnd_coded	0	0.000000
age	0	0.000000
sex_coded	0	0.000000
race_coded	0	0.000000
ejection_fraction	0	0.000000
subject_id	0	0.000000
is_eligible	0	0.000000

Next steps: Generate code with na_summary | View recommended plots | New interactive sheet

```
modeling_data_imputed = data_at_or_after_study_start.drop(columns=['kccq_summary_score', 'max_ventricular_rate_bpm', 'dmcbpot_cc

vars_to_impute = na_summary[
    (na_summary['pct_missing'] < 37) &
    (na_summary['pct_missing'] > 1)
].index.tolist()

for col in vars_to_impute:
    modeling_data_imputed[col] = modeling_data_imputed.groupby('subject_id')[col].transform(lambda x: x.fillna(x.mean()))
    modeling_data_imputed[col] = modeling_data_imputed[col].fillna(modeling_data_imputed[col].mean())  #option: fallback to over
```

We are predicting binary outcomes at each timepoint for this analysis, EF <50 (1) or EF >= 50 (0), so we use logistic regression. Several caveats here:

- I suspect the dataset is highly imbalanced, given what we saw above in the baseline data visualizations.
- I also chose to keep most variables in the dataset, so there will likely be a lot of noise given that I haven't identified any irrelevant variables.
- This is longitudinal patient data, and patients will have multiple timepoints to predict ef at--this means there are going to be relationships between variables.

For time purposes, and since this is an initial model of the data, I am going to see how models perform on all the data as a learning excercise to give a starting point for theoretical future iterations on the data and the model. I am not expecting great performance at this point.

```
#create groups to avoid data leakage. We are predictig ef for each timepoint
#in the dataset, so patients will have multiple predictions, but this should ensure all of a patient's data is in either test or
groups = data_at_or_after_study_start['subject_id']

#since we want to use imputed data for regression and non-imputed for xgboost, X is different from both. y will be the same
X_reg = modeling_data_imputed.drop(columns=['subject_id', 'binary_ef_less_than_50', 'ejection_fraction'])
X_xgb = data_at_or_after_study_start.drop(columns=['subject_id', 'binary_ef_less_than_50', 'ejection_fraction'])

y = data_at_or_after_study_start['binary_ef_less_than_50']

#create stratified folds so that the few ef <50 occurences are distributed across folds
gkf = StratifiedGroupKFold(n_splits=5)

splits = list(gkf.split(X=data_at_or_after_study_start, y=y, groups=groups))
```

I am evaluating this and the next model for AUC and accuracy. Accuracy is not the best metric to use when you know the dataset is imbalanced, but I was curious regardless. To use logistic regression on this imbalanced dataset, I am using the class_weight parameter, an l2 penalty, and the liblinear solver. L2 (ridge regression) helps with convergence in small, noisy datasets with multicollinearity--all features our data has. class_weight=balanced gives more weight to the minority class, encouraging learning from them. The liblinear solver is more stable than other choices in small, noisy datasets and works well with l2 and binary classification problems.

```
auc_scores = []
acc_scores = []
```

```
#loop for 5 fold cross validation of this model, evaluating accuracy and AUC
for train_idx, test_idx in splits:
    X_train, X_test = X_reg.iloc[train_idx], X_reg.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]


    model = LogisticRegression(penalty = 'l2', solver='liblinear', max_iter=5000, class_weight='balanced'
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    auc = roc_auc_score(y_test, y_prob)
    acc = accuracy_score(y_test, y_pred)

    auc_scores.append(auc)
    acc_scores.append(acc)

    print(f"Fold AUC: {auc:.3f}, Accuracy: {acc:.3f}")

print(f"\nMean AUC: {np.mean(auc_scores):.3f}")
```

```
Fold AUC: 0.204, Accuracy: 0.701
Fold AUC: 0.536, Accuracy: 0.701
Fold AUC: 0.513, Accuracy: 0.778
Fold AUC: 0.447, Accuracy: 0.586
Fold AUC: 0.468, Accuracy: 0.760

Mean AUC: 0.434
Mean Accuracy: 0.705
```

Accuracy and AUC show that while the model is fairly accurate (expected due to imbalance), AUC indicates the model is worse at predicting outcomes than random choice.

4b. Xgboost

Xgboost may be a better approach for this problem because it allows missing data. I am going to calculate the appropriate weighting for the majority and minority classes and use that in our model to address imbalance.

```
#calculate negative samples / postive samples
n_pos = sum(y == 1)
n_neg = sum(y == 0)
scale_pos_weight = n_neg / n_pos


auc_scores = []
acc_scores = []

#loop for cross validation of this model, evaluating auc and accuracy

for train_idx, test_idx in splits:
    X_train, X_test = X_xgb.iloc[train_idx], X_xgb.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
```