CA4

April 18, 2023

1 DAT200 CA4 2023

Kaggle username:

1.0.1 Imports

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.impute import SimpleImputer
import csv
```

1.0.2 Reading data

[ ]:
```
train = pd.read_csv('data/train.csv', index_col=0)
test = pd.read_csv('data/test.csv', index_col=0)
```

1.0.3 Data exploration and visualisation

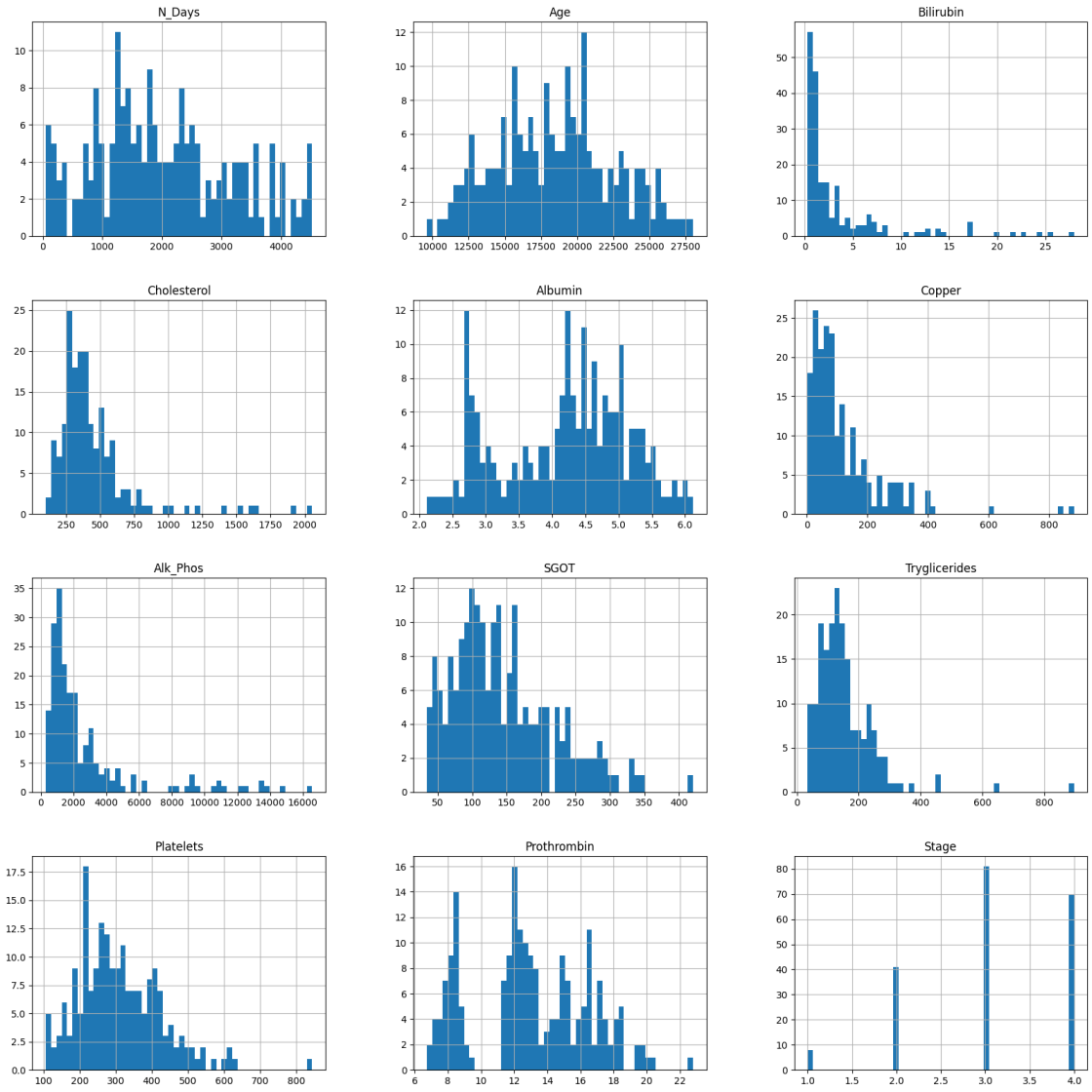
[ ]:
```
train.head()
```

[ ]:
```
       N_Days Status             Drug    Age Sex Ascites Hepatomegaly Spiders  \
index
0         980      D  D-penicillamine  18713   F       N            Y       Y
1        1455      C          Placebo  12398   F       N            Y       N
2         216      D          Placebo  19246   F       Y            Y       Y
3        2216      C          Placebo  19221   F       N            Y       Y
4        1701      C  D-penicillamine  11485   F       N            N       N

      Edema  Bilirubin  Cholesterol  Albumin  Copper  Alk_Phos     SGOT  \
```

index							
0	N	6.7	561.0	5.610	154.5	1468.5	192.975
1	N	1.3	456.0	5.280	145.5	2433.0	106.500
2	N	24.5	1638.0	5.025	349.5	5610.0	220.875
3	N	0.7	201.6	3.208	8.8	968.0	58.280
4	N	1.1	403.2	4.488	57.6	987.6	100.800

index	Tryglicerides	Platelets	Prothrombin	Stage
0	150.0	399.0	16.65	4
1	253.5	382.5	14.25	4
2	648.0	598.5	22.80	4
3	46.4	247.2	7.60	2
4	129.6	290.4	11.64	3

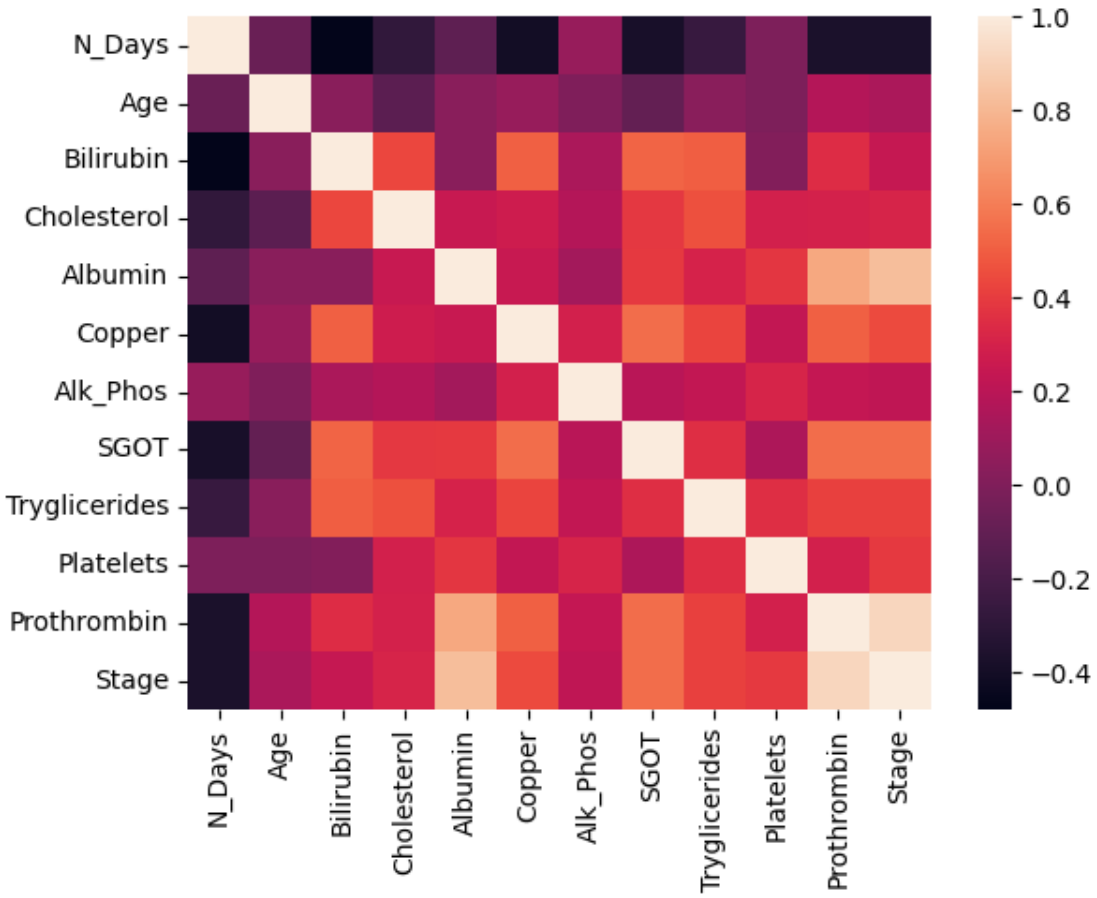
[ ]:
```
train.hist(figsize=(20,20), bins=50)
plt.show()
```



Here you can see that there is many in stage 3 and, and not that many in stage 1.

[ ]:
```
sns.heatmap(train.corr())
plt.show()
```

```
/var/folders/54/3p1f2msx4rnd8668t_pcxddw0000gn/T/ipykernel_17867/4174621242.py:1
: FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only valid
columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(train.corr())
```



The plots won't really give much information before changing the categorical data to numerical data. It will also give a warning due to this.

1.0.4 Data cleaning

[ ]:
```
# Changing the categorical data to numerical values
class_labels = ['Status', 'Drug', 'Sex', 'Ascites', 'Hepatomegaly', 'Spiders',␣
 ↪'Edema']

for cl in class_labels:
    class_le = LabelEncoder()
    Y_le  = class_le.fit_transform(train[cl].values)
    train[cl]= Y_le

train.head()  # Printing the top of the dataframe to see if the categorical␣
 ↪values are changed
```

[ ]:

index	N_Days	Status	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders
0	980	2	0	18713	0	0	1	1
1	1455	0	1	12398	0	0	1	0
2	216	2	1	19246	0	1	1	1
3	2216	0	1	19221	0	0	1	1
4	1701	0	0	11485	0	0	0	0

index	Edema	Bilirubin	Cholesterol	Albumin	Copper	Alk_Phos	SGOT
0	0	6.7	561.0	5.610	154.5	1468.5	192.975
1	0	1.3	456.0	5.280	145.5	2433.0	106.500
2	0	24.5	1638.0	5.025	349.5	5610.0	220.875
3	0	0.7	201.6	3.208	8.8	968.0	58.280
4	0	1.1	403.2	4.488	57.6	987.6	100.800

index	Tryglicerides	Platelets	Prothrombin	Stage
0	150.0	399.0	16.65	4
1	253.5	382.5	14.25	4
2	648.0	598.5	22.80	4
3	46.4	247.2	7.60	2
4	129.6	290.4	11.64	3

[ ]:
```
# repeat for test
class_labels = ['Status', 'Drug', 'Sex', 'Ascites', 'Hepatomegaly', 'Spiders',␣
 ↪'Edema']

for cl in class_labels:
    class_le = LabelEncoder()
    Y_le  = class_le.fit_transform(test[cl].values)
    test[cl]= Y_le

test.head()  # Printing the top of the dataframe to see if the categorical␣
 ↪values are changed
```

[ ]:

index	N_Days	Status	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders
0	2576	0	1	17323	0	0	0	0
1	3445	0	1	23445	1	0	1	1
2	1690	2	0	16374	0	0	0	1
3	2221	0	1	13535	0	0	1	0
4	2178	0	0	18337	0	0	0	1

index	Edema	Bilirubin	Cholesterol	Albumin	Copper	Alk_Phos	SGOT
0	0	0.5	379.2	4.380	81.6	2059.2	225.06

```
1          0        0.6        378.0    5.745    61.5    1264.5   97.65
2          0        3.9        420.0    3.864   145.2    1521.6  327.36
3          0        0.5        119.2    3.232   181.6     478.4   42.16
4          0        0.5        480.0    4.080    10.8    1360.8  115.32

       Tryglicerides  Platelets  Prothrombin
index
0               85.2      427.2        11.76
1              124.5      504.0        17.10
2              277.2      324.0        11.52
3               45.6      132.8         7.92
4               66.0      427.2        12.24
```

[ ]:
```
# checking for NaN values
print(f'Column Number of missing values ')
for c in train.columns:
    n_NaN = train[c].isnull().sum()
    print(f'{c:<32} {n_NaN}')

train.shape
```

```
Column Number of missing values
N_Days                           0
Status                           0
Drug                             0
Age                              0
Sex                              0
Ascites                          0
Hepatomegaly                     0
Spiders                          0
Edema                            0
Bilirubin                        0
Cholesterol                     16
Albumin                          0
Copper                           1
Alk_Phos                         0
SGOT                             0
Tryglicerides                   16
Platelets                        4
Prothrombin                      0
Stage                            0
```

[ ]:
```
(200, 19)
```

[ ]:
```
# check the test data for NaN
print(f'Column Number of missing values ')
for c in test.columns:
    n_NaN = test[c].isnull().sum()
```

```
    print(f'{c:<32} {n_NaN}')

test.shape
```

```
Column Number of missing values
N_Days                           0
Status                           0
Drug                             0
Age                              0
Sex                              0
Ascites                          0
Hepatomegaly                     0
Spiders                          0
Edema                            0
Bilirubin                        0
Cholesterol                      9
Albumin                          0
Copper                           1
Alk_Phos                         0
SGOT                             0
Tryglicerides                   11
Platelets                        0
Prothrombin                      0
```

[ ]:
```
(109, 18)
```

[ ]:
```
# There NaN values in several of the categories which needs to be removed

imr = SimpleImputer(missing_values=np.nan, strategy='mean')
imr = imr.fit(train.values)
imputed_data = imr.transform(train.values)

train = pd.DataFrame(imputed_data, columns=train.columns)

# Printing to see if the NaN values are gone
print(f'Column Number of missing values ')
for c in train.columns:
    n_NaN = train[c].isnull().sum()
    print(f'{c:<32} {n_NaN}')
```

```
Column Number of missing values
N_Days                           0
Status                           0
Drug                             0
Age                              0
Sex                              0
Ascites                          0
Hepatomegaly                     0
```

```
Spiders                         0
Edema                           0
Bilirubin                       0
Cholesterol                     0
Albumin                         0
Copper                          0
Alk_Phos                        0
SGOT                            0
Tryglicerides                   0
Platelets                       0
Prothrombin                     0
Stage                           0
```

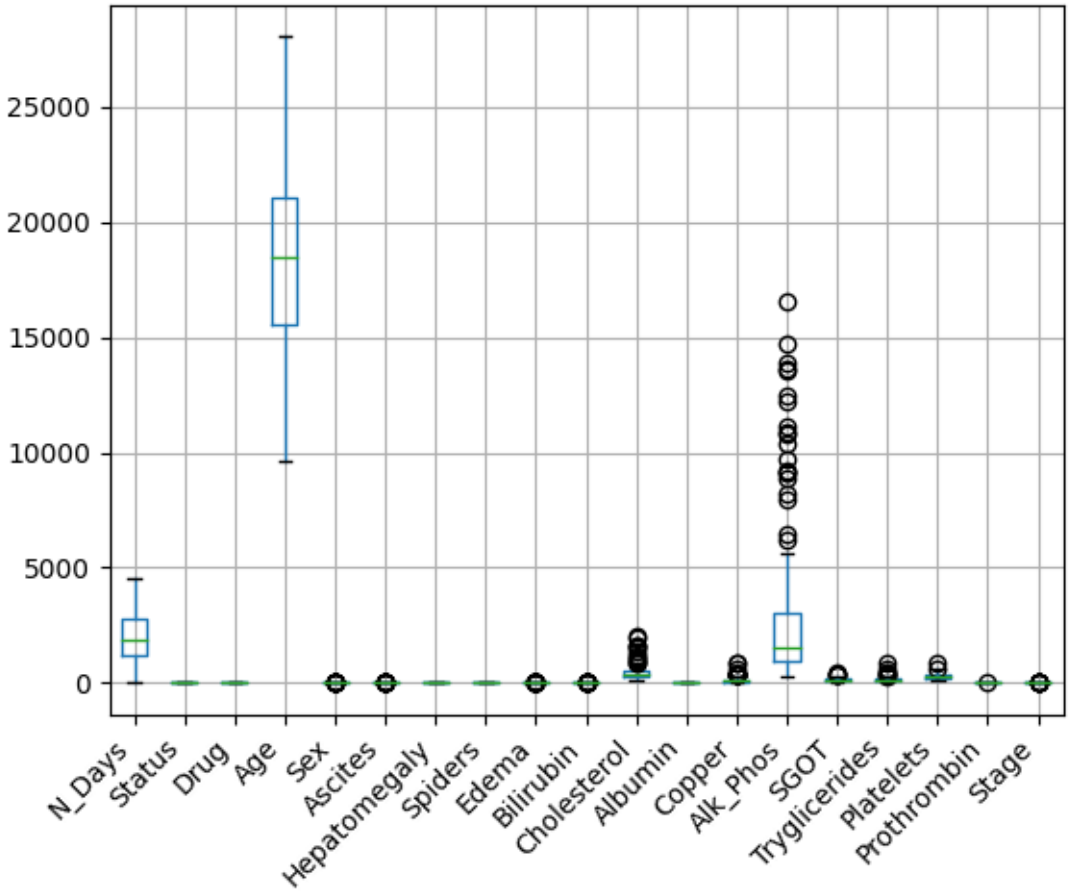
[ ]:
```python
# repeat for test data
imr = SimpleImputer(missing_values=np.nan, strategy='mean')
imr = imr.fit(test.values)
imputed_data = imr.transform(test.values)

test = pd.DataFrame(imputed_data, columns=test.columns)

# Printing to see if the NaN values are gone
print(f'Column Number of missing values ')
for c in train.columns:
    n_NaN = train[c].isnull().sum()
    print(f'{c:<32} {n_NaN}')

test.shape
```

```
Column Number of missing values
N_Days                          0
Status                          0
Drug                            0
Age                             0
Sex                             0
Ascites                         0
Hepatomegaly                    0
Spiders                         0
Edema                           0
Bilirubin                       0
Cholesterol                     0
Albumin                         0
Copper                          0
Alk_Phos                        0
SGOT                            0
Tryglicerides                   0
Platelets                       0
Prothrombin                     0
Stage                           0
```

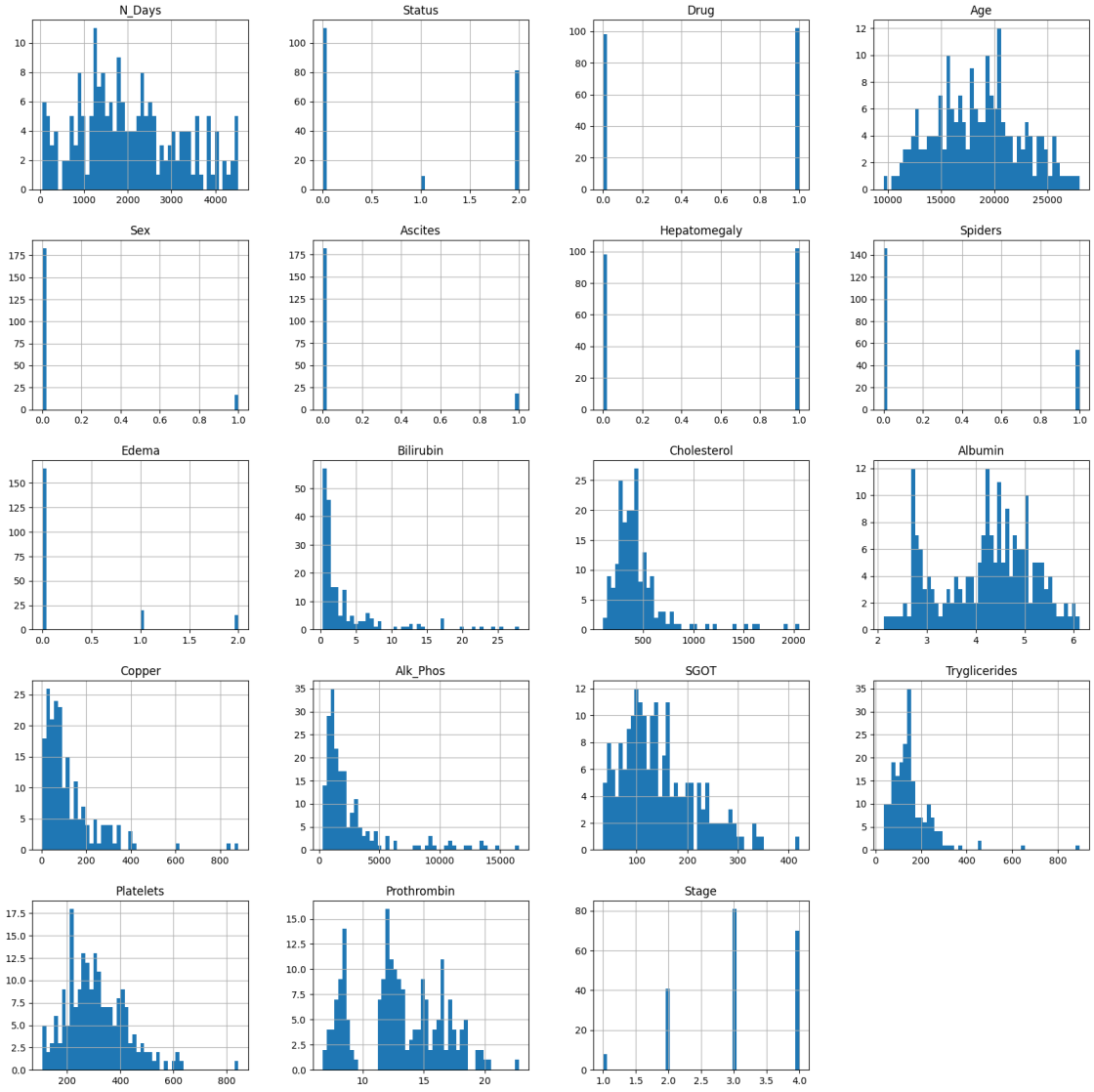
[ ]: (109, 18)

1.0.5 Data exploration after cleaning

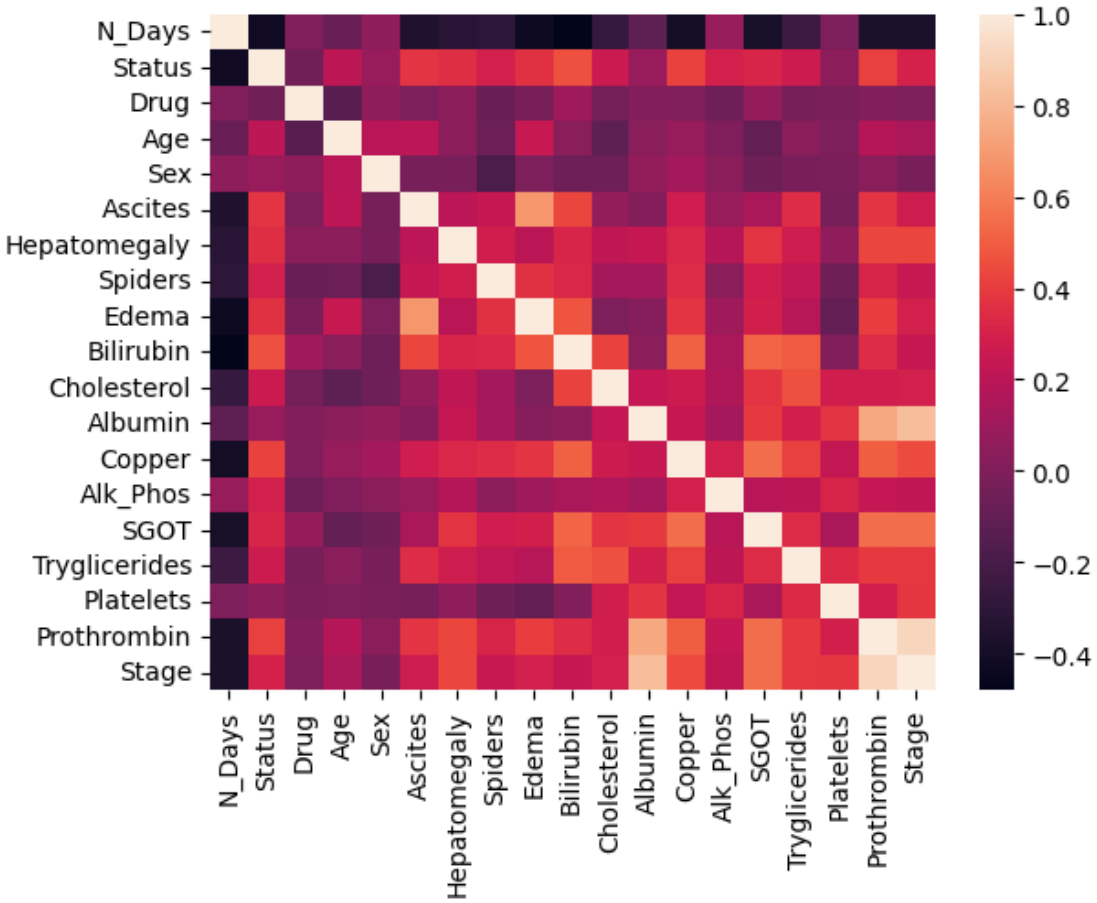
[ ]:
```
train.boxplot()
plt.xticks(rotation=45, ha='right')
plt.show()
```



[ ]:
```
train.hist(figsize=(20,20), bins=50)
plt.show()
```



[ ]:
```
sns.heatmap(train.corr())
plt.show()
```



1.0.6 Data preprocessing

Train test split

[ ]:
```
# splitting into X and y
y = train.Stage
X = train.iloc[:,0:18]

# Split data into training and test data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=6, stratify=y)
```

Scaling

[ ]:
```
# Scaling is performed later
```

1.0.7 Modelling

Data pipeline with kernel

[ ]:
```
pipe_svc = make_pipeline(StandardScaler(), PCA(), SVC(random_state=1))

# Define ranges of parameter values:
param_range  = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
param_range2 = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0]

param_grid   = [{'svc__C': param_range},
                {'svc__C': param_range, 'svc__gamma': param_range2}]

gs_svc = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='f1_macro',
                  cv=10,
                  n_jobs=-1)

gs_svc_test = gs_svc.fit(X_train, y_train)
clf_svc_test = gs_svc_test.best_estimator_
clf_svc_test.fit(X_train, y_train)
print(clf_svc_test.score(X_test, y_test))
```

```
/Users/joridholmen/Library/Python/3.9/lib/python/site-
packages/sklearn/model_selection/_split.py:700: UserWarning: The least populated
class in y has only 5 members, which is less than n_splits=10.
  warnings.warn(
```

```
0.9166666666666666
```

[ ]:
```
gs_svc_final = gs_svc.fit(X, y)
clf_svc_final = gs_svc.best_estimator_
clf_svc_final.fit(X, y)
y_pred = clf_svc_final.predict(test)

# write the results to a csv file
with open('kaggle_submission_svc.csv', 'w') as f:
    w = csv.writer(f)

    w.writerow(['index','Stage'])

    for r in range(0, 109):
        w.writerow([r, int(y_pred[r])])
```

```
/Users/joridholmen/Library/Python/3.9/lib/python/site-
packages/sklearn/model_selection/_split.py:700: UserWarning: The least populated
class in y has only 8 members, which is less than n_splits=10.
  warnings.warn(
```

The accuracy score on the training data is 0.9167, but when uploading to kaggle the score is 0.6927.

Data pipeline with regularization

[ ]:
```
pipe_l2 = make_pipeline(
    StandardScaler(),
    LogisticRegression(solver='liblinear')
)

param_grid = {
    'logisticregression__C': [0.01, 0.1, 1, 10, 100],
    'logisticregression__fit_intercept': [True, False],
    'logisticregression__penalty': ['l1', 'l2']
}

gs_lr = GridSearchCV(estimator=pipe_l2,
                     param_grid=param_grid,
                     scoring='f1_macro',
                     cv=10,
                     n_jobs=-1)

gs_lr_test = gs_lr.fit(X_train, y_train)
clf_lr_test = gs_lr_test.best_estimator_
clf_lr_test.fit(X_train, y_train)
print(clf_lr_test.score(X_test, y_test))
```

```
/Users/joridholmen/Library/Python/3.9/lib/python/site-
packages/sklearn/model_selection/_split.py:700: UserWarning: The least populated
class in y has only 5 members, which is less than n_splits=10.
  warnings.warn(
```

```
0.9
```

[ ]:
```
gs_lr_final = gs_lr.fit(X, y)
clf_lr_final = gs_lr.best_estimator_
clf_lr_final.fit(X, y)
y_pred = clf_lr_final.predict(test)

# write the results to a csv file
with open('kaggle_submission_lr.csv', 'w') as f:
    w = csv.writer(f)

    w.writerow(['index','Stage'])

    for r in range(0, 109):
        w.writerow([r, int(y_pred[r])])
```

```
/Users/joridholmen/Library/Python/3.9/lib/python/site-
packages/sklearn/model_selection/_split.py:700: UserWarning: The least populated
class in y has only 8 members, which is less than n_splits=10.
  warnings.warn(
```

The accuracy score on the training data is 0.9, but when uploading to kaggle the score is 0.6589.

Other models used for Kaggle submission

[ ]:
```
rf = make_pipeline(RandomForestClassifier())
param_grid = {
    'randomforestclassifier__n_estimators': [50, 100, 200],
    'randomforestclassifier__max_depth': [None, 5, 10],
    'randomforestclassifier__min_samples_split': [2, 5, 10],
    'randomforestclassifier__min_samples_leaf': [1, 2, 4]
}

gs_rf = GridSearchCV(estimator=rf,
                     param_grid=param_grid,
                     scoring='f1_macro',
                     cv=10,
                     n_jobs=-1)

gs_rf_test = gs_rf.fit(X_train, y_train)
clf_rf_test = gs_rf_test.best_estimator_
clf_rf_test.fit(X_train, y_train)
print(clf_rf_test.score(X_test, y_test))
```

```
/Users/joridholmen/Library/Python/3.9/lib/python/site-
packages/sklearn/model_selection/_split.py:700: UserWarning: The least populated
class in y has only 5 members, which is less than n_splits=10.
  warnings.warn(
```

```
0.9166666666666666
```

[ ]:
```
gs_rf_final = gs_rf.fit(X, y)
clf_rf_final = gs_rf.best_estimator_
clf_rf_final.fit(X, y)
y_pred = clf_rf_final.predict(test)

# write the results to a csv file
with open('kaggle_submission_rf.csv', 'w') as f:
    w = csv.writer(f)

    w.writerow(['index','Stage'])

    for r in range(0, 109):
        w.writerow([r, int(y_pred[r])])
```

```
/Users/joridholmen/Library/Python/3.9/lib/python/site-
packages/sklearn/model_selection/_split.py:700: UserWarning: The least populated
class in y has only 8 members, which is less than n_splits=10.
  warnings.warn(
```

The accuracy score here is about the same as kaggle, and therefore I will be submitting this to kaggle.

1.0.8 Kaggle submission

[ ]:
```
kaggle = RandomForestClassifier()

kaggle.fit(X, y)
y_pred = kaggle.predict(test)

# write the results to a csv file
with open('kaggle_submission.csv', 'w') as f:
    w = csv.writer(f)

    w.writerow(['index','Stage'])

    for r in range(0, 109):
        w.writerow([r, int(y_pred[r])])
```