MIMIC pneumonia patients data analysis and logistic regression

Presented by Yitong Feng, Hyojeong Kim

AI in Health

Instructor: Dr. Ying Ding

Summary

Pneumonia is an infection that inflames the air sacs in one or both lungs, causing cough with phlegm or pus, fever, chills, and difficulty breathing, which has been troubled human being for years already.
The objective of this analysis is to make data analysis on patients diagnosed with pneumonia, design a prediction algorithm using logistic regression and give the closest result. This dataset is originally from the MIMIC database. It contains data of male and female patients from age 19 to 89.

1.1 Data Overview and data clean

```
#import library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv("/content/patients with PNEUMONIA.csv")
data.head()
#data overview
data.describe()
data.info()
```

```
#data clean
#check nulls and outliers(patients
in their age over 120)
data =
data.drop(data[data['age']>120].ind
ex)
```

1.2 Extreme Value of pneumonia patients on Ethnicity

```
from pandas.core.frame import DataFrame

max_age = data['age'].groupby(data['ETHNICITY']).max()
min_age = data['age'].groupby(data['ETHNICITY']).min()
avg_age = data['age'].groupby(data['ETHNICITY']).mean()

ethnicityGroup = {'ETHNICITY': max_age.index.to_list(),
        'max': max_age.to_list(),
        'min': min_age.to_list(),
        'avg': avg_age.to_list()}
extremeValue = DataFrame(ethnicityGroup)
extremeValue
```

	ETHNICITY	max	min	avg
0	ASIAN	84	43	67.550000
1	ASIAN - CHINESE	86	69	78.000000
2	BLACK/AFRICAN	84	70	78.000000
3	BLACK/AFRICAN AMERICAN	88	23	61.247312
4	BLACK/CAPE VERDEAN	88	55	70.666667
5	BLACK/HAITIAN	79	76	77.500000
6	HISPANIC OR LATINO	87	28	59.642857
7	HISPANIC/LATINO - CENTRAL AMERICAN (OTHER)	78	78	78.000000
8	HISPANIC/LATINO - PUERTO RICAN	66	52	61.666667
9	MIDDLE EASTERN	79	79	79.000000
10	MULTI RACE ETHNICITY	65	65	65.000000
11	OTHER	82	34	63.000000
12	PATIENT DECLINED TO ANSWER	63	50	57.000000
13	PORTUGUESE	77	68	72.500000
14	UNABLE TO OBTAIN	87	66	77.666667
15	UNKNOWN/NOT SPECIFIED	87	19	68.769231
16	WHITE	89	19	66.359877
17	WHITE - RUSSIAN	87	70	80.000000

1.2 Extreme Value of pneumonia patients on LOS

```
# see the data overview of patients_icustays table
patients_icustays['LOS'] = patients_icustays['LOS'].round(0)
patients_icustays =
patients_icustays.drop(patients_icustays[patients_icustays['LOS']>10].index)
patients_icustays['LOS'].describe()
```

```
count    2402.000000
mean        2.943381
std         2.225547
min         0.000000
25%         1.000000
50%         2.000000
75%         4.000000
max        10.000000
Name: LOS, dtype: float64
```

1.3 Top 10 common diseases occur with Pneumonia

```
# see top common disease occur with pneumonia
# extract patients' subject_id who diagnosed with Pneumonia
temp_data =
patients_admissions[patients_admissions['SUBJECT_ID'].isin(data['SUBJECT_ID'].values.tolist()
)]
df1 = temp_data.groupby(['DIAGNOSIS'],as_index=False)['DIAGNOSIS'].agg({'cnt':'count'})
df1.sort_values('cnt', inplace=True, ascending=False)
df1[1:10]
```

1.3 Top 10 common diseases occur with Pneumonia

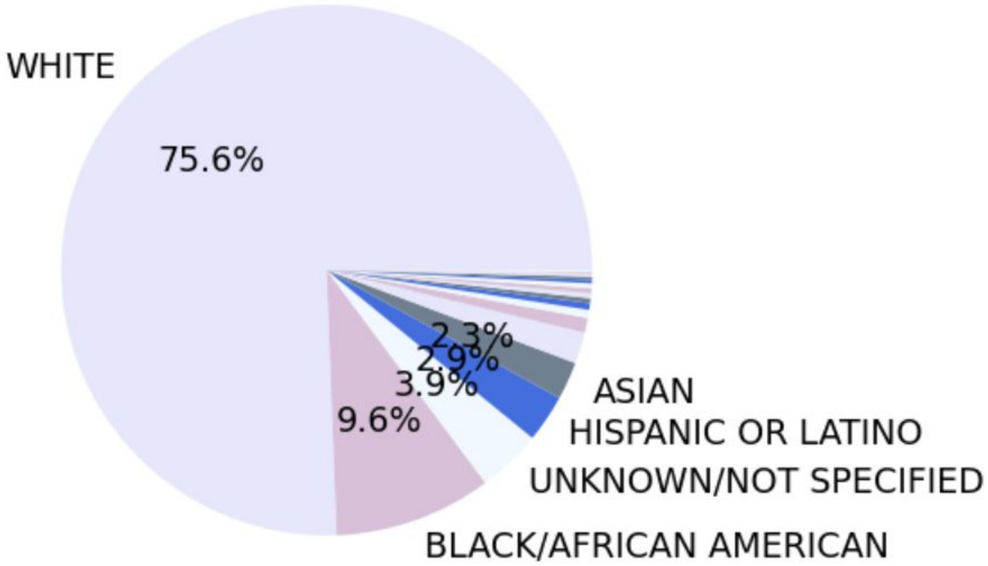
	DIAGNOSIS	cnt
469	SEPSIS	59
136	CONGESTIVE HEART FAILURE	47
203	FEVER	37
65	ASTHMA;COPD EXACERBATION	33
33	ALTERED MENTAL STATUS	30
483	SHORTNESS OF BREATH	28
266	HYPOTENSION	23
182	DYSPNEA	23
425	RESPIRATORY FAILURE	20

1.4 Visualization–Ethnicity distribution of patients

```
# demographic information of
patients--Ethnicity
# calculate number of patients by ethnicity
ethnicityGroup=data['ETHNICITY'].value_count
s()
colors=['lavender','thistle','aliceblue','ro
yalblue','slategrey']

top5Group = ethnicityGroup.index.to_list()
for i in range(5, len(top5Group)):
 top5Group[i] = ''
```

```
# Do not show the labels with small pieces of data
def my_autopct(pct):
return ('%.1f%%' % pct) if pct > 2 else ''
plt.pie(x=ethnicityGroup,
       colors=colors,
       autopct=my_autopct,
       labels=top5Group,
       pctdistance=0.6,
       labeldistance = 1.1,
       radius = 1.7,
       wedgeprops = {'linewidth': 2},
       textprops = {'fontsize':19 ,'color':'k'})
plt.show()
```

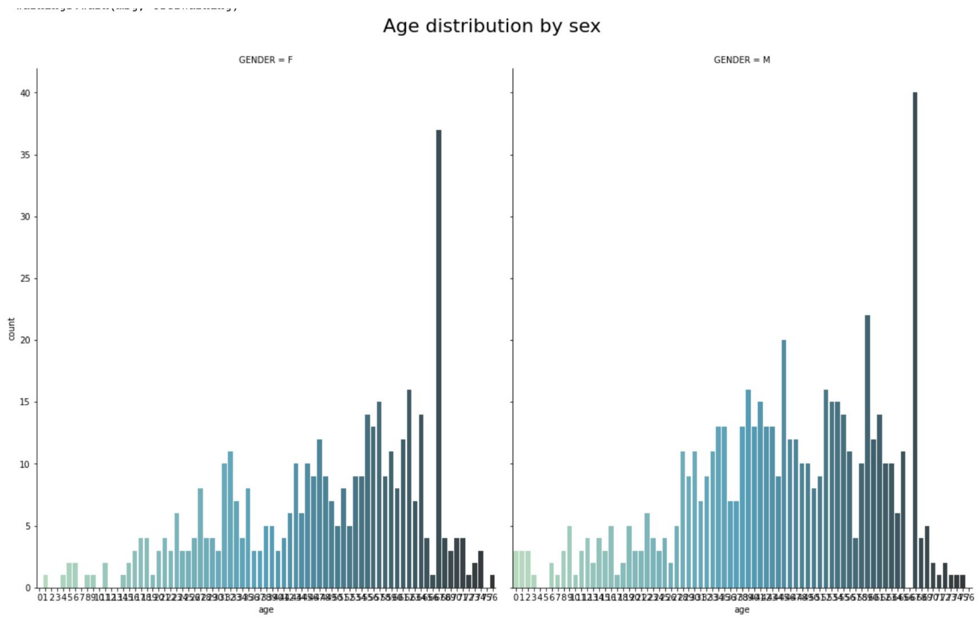
1.4 Visualization–Ethnicity distribution of patients



1.5 Visualization–Age distribution by gender

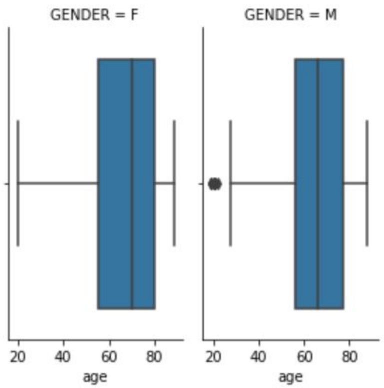
```
# age distribution by sex
agePeople=data['age'].value_counts()
g = sns.factorplot(x="age", col="GENDER",data=data, kind="count",size=10,
aspect=0.8,palette="GnBu_d");
g.set_xticklabels(np.arange(0,100));
# g.set_xticklabels(step=10);
g.fig.suptitle('Age distribution by sex',fontsize=22);
g.fig.subplots_adjust(top=.9)
```

1.5 Visualization–Age distribution by gender



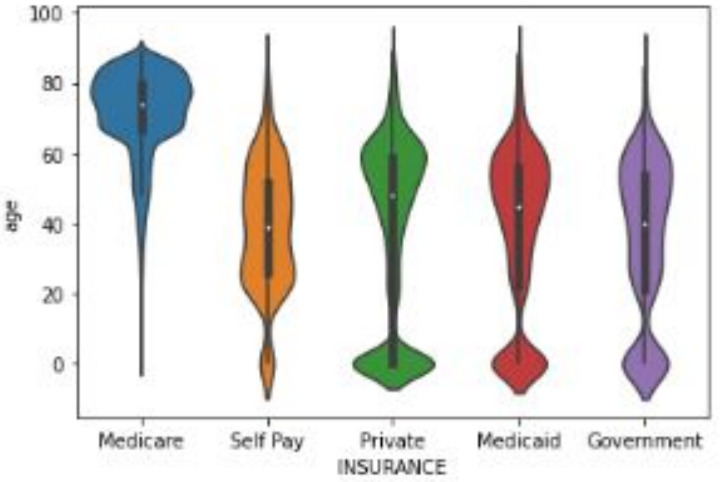
1.6 Visualization–Age distribution among gender

```
sns.factorplot(x='age',col='GENDER',data=data,kind='box',size=4,aspect=0.5)
```



1.7 Visualization–Insurance purchase by age

```
sns.violinplot(x=patients_admissions["INSURANCE"], y=patients_admissions["age"])
plt.show()
```

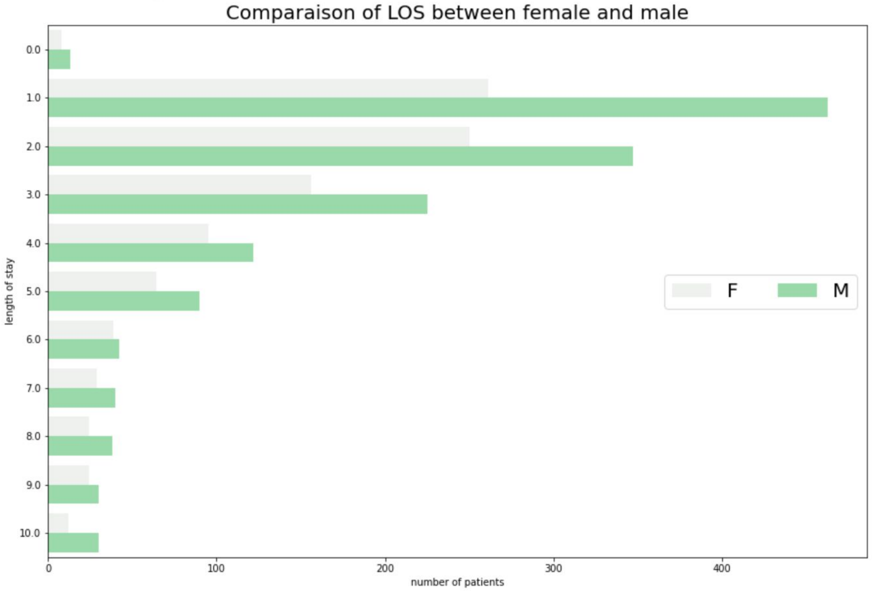


1.8 Visualization–Length of Stay staus by sex

```
#  LOS in ICU of patients diagnosed with pneumonia in different age groups among gender
f, ax = plt.subplots(sharex=True,figsize=(15, 10))
sns.set_color_codes("pastel")
g=sns.countplot(y='LOS',hue='GENDER', data=patients_icustays, ax=ax,color='g')
sns.set_color_codes("muted")

ax.legend(ncol=2, loc="center right", frameon=True,fontsize=20)
ax.set( ylabel="length of stay",xlabel="number of patients")
ax.set_title("Comparaison of LOS between female and male",fontsize=20)
```

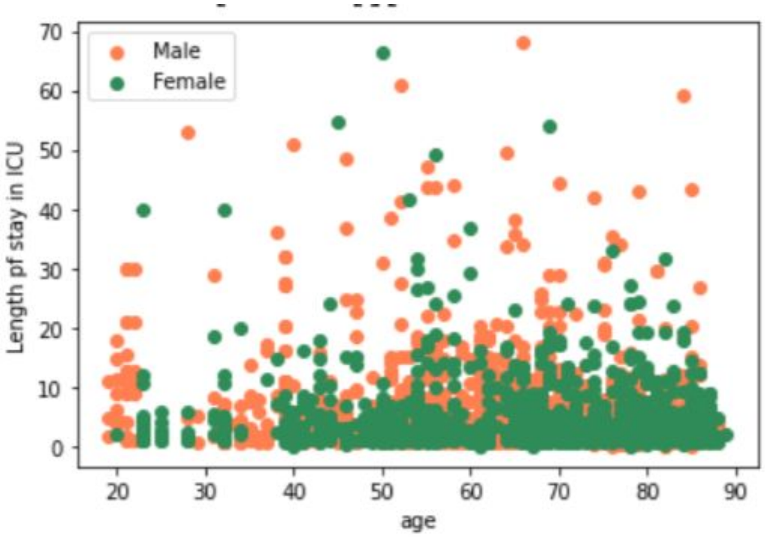
1.8 Visualization–Length of Stay staus by sex

```
Text(0.5, 1.0, 'Comparaison of LOS between female and male')
```

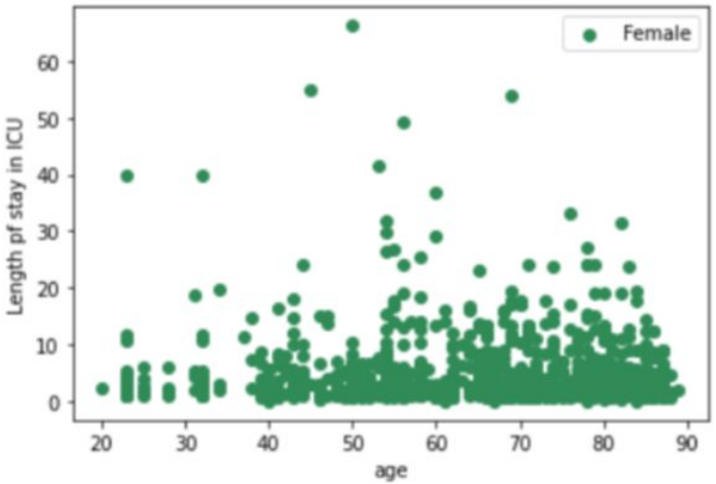


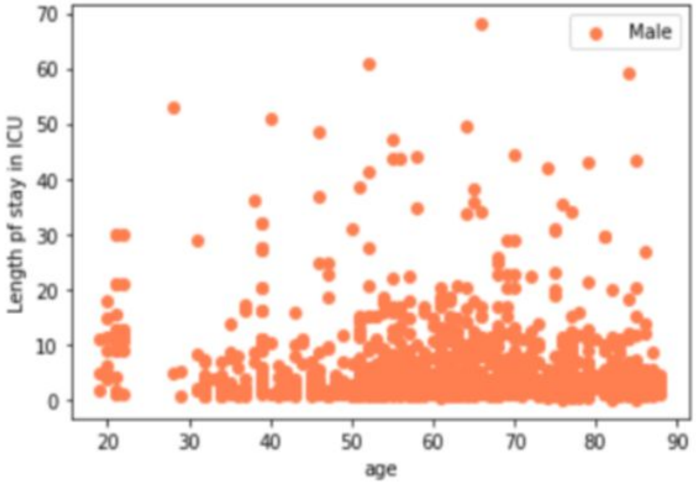
1.9 Visualization–Age distribibition by sex on LOS

```
#Indicate each color represents  what data
x_male=patients_icustays[ 'age'].loc[patients_icustays[ 'GENDER']=='M']
y_male=patients_icustays[ 'LOS'].loc[patients_icustays[ 'GENDER']=='M']
x_female=patients_icustays[ 'age'].loc[patients_icustays[ 'GENDER']=='F']
y_female=patients_icustays[ 'LOS'].loc[patients_icustays[ 'GENDER']=='F']

fig,ax=plt.subplots()
# set notes and form of graph
ax.scatter(x_male,y_male, c= 'coral',label='Male')
ax.scatter(x_female,y_female, c= 'seagreen',label='Female')
ax.set_xlabel('age')
ax.set_ylabel('Length pf stay in ICU' )

ax.legend()
plt.show
```



1.9 Visualization–Age distribition by sex on LOS





2. Logistic regression

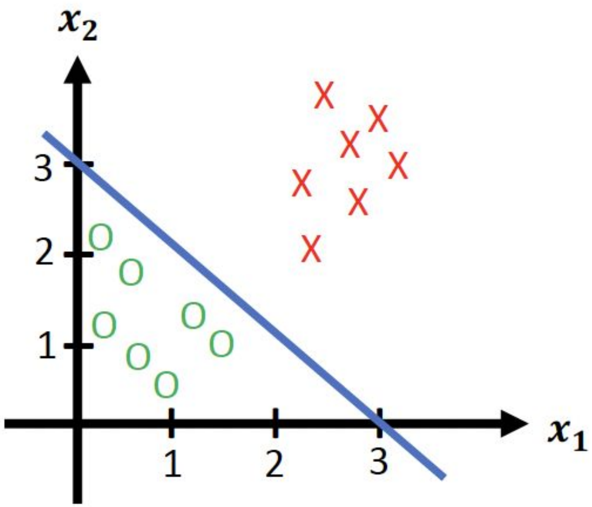
Logistic regression is a process of modeling the probability of a discrete outcome given an input variable. The most common logistic regression models a binary outcome; something that can take two values such as true/false, yes/no, and so on.

Classification problem

- Email: Spam/ Not Spam
- Document: Relevant/Not relevant
- Transaction: Fraudlent(Yes/No)

Binary Classification Problem

$y \in \{0, 1\}$
0 : Negative Class
1 : Positive Class



To make predictions on $y = f x$,
threshold classifier output condition:
If G $\boldsymbol{MTx} \geq 0$
$y = 1$ *predict Spam class*

2.1 Import libraries and loading the dataset

The libraries we will use for this project:

1. **pandas:** The first library that we need to import is pandas, which is a portmanteau of "panel data" and is the most popular Python library for working with tabular data.
2. **numpy:** we'll need to import NumPy, which is a popular library for numerical computing. Numpy is known for its NumPy array data structure as well as its useful methods reshape, arange, and append.
3. **%matplotlib:** we need to import matplotlib, which is Python's most popular library for data visualization.
4. **seaborn:** we will want to import seaborn, which is another Python data visualization library that makes it easier to create beautiful visualizations using matplotlib.

+ Code + Text

RAM Disk Editing

```
[ ] from sklearn import datasets

    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    %matplotlib inline
    sns.set_style("white")
```

```
[ ] from google.colab import files
    uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
[ ] train = pd.read_csv('./patients with PNEUMONIA.csv') # Training set is already available
    train.head()
```

	ROW_ID	SUBJECT_ID	GENDER	DOB	DOD	DOD_HOSP	DOD_SSN	EXPIRE_FLAG	ROW_ID.1	SUBJECT_ID.1	...	INSURANCE	LANGUAGE	RELIGION	MARITAL_STATUS	ETHNICITY	EDREGTIME	EDOUTTIME	DIAGNOSIS	H
0	637	674	F	2113-12-14 00:00:00	2195-02-17 00:00:00	2195-02-17 00:00:00	2195-02-17 00:00:00	1	830	674	...	Medicare	ENGL	JEWISH	MARRIED	WHITE	6/19/2193 18:06	6/19/2193 20:51	PNEUMONIA	
1	664	705	M	2087-10-23 00:00:00	2156-08-26 00:00:00	2156-08-26 00:00:00	2156-08-26 00:00:00	1	869	705	...	Medicare	NaN	UNOBTAINABLE	NaN	ASIAN	8/22/2156 21:03	8/23/2156 4:32	PNEUMONIA	
2	665	707	F	2026-04-05 00:00:00	2116-01-04 00:00:00	NaN	2116-01-04 00:00:00	1	870	707	...	Medicare	NaN	UNOBTAINABLE	SINGLE	WHITE	2/28/2115 16:38	2/28/2115 22:19	PNEUMONIA	
3	669	711	M	2100-03-06 00:00:00	2185-05-26 00:00:00	NaN	2185-05-26 00:00:00	1	878	711	...	Medicare	ENGL	JEWISH	MARRIED	WHITE	3/22/2185 6:59	3/22/2185 11:14	PNEUMONIA	
4	683	726	F	2046-09-03 00:00:00	2123-07-16 00:00:00	2123-07-16 00:00:00	2123-07-16 00:00:00	1	895	726	...	Medicare	NaN	NOT SPECIFIED	MARRIED	WHITE	7/1/2123 14:54	7/1/2123 20:15	PNEUMONIA	

5 rows × 27 columns

2.2 Understand the dataset

The info() function is used to print a concise summary of a DataFrame. This method prints information about a DataFrame including the index dtype and column dtypes, non-null values and memory usage.

As we can see in the output, the summary includes list of all columns with their data types and the number of non-null values in each column. we also have the value of rangeindex provided for the index axis.

Step2: Basic operations to understand the dataset

```
t=train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 27 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   ROW_ID                1000 non-null   int64
 1   SUBJECT_ID            1000 non-null   int64
 2   GENDER                1000 non-null   object
 3   DOB                   1000 non-null   object
 4   DOD                   665 non-null    object
 5   DOD_HOSP              464 non-null    object
 6   DOD_SSN               565 non-null    object
 7   EXPIRE_FLAG           1000 non-null   int64
 8   ROW_ID.1              1000 non-null   int64
 9   SUBJECT_ID.1          1000 non-null   int64
 10  HADM_ID               1000 non-null   int64
 11  ADMITTIME             1000 non-null   object
 12  DISCHTIME             1000 non-null   object
 13  DEATHTIME             165 non-null    object
 14  ADMISSION_TYPE        1000 non-null   object
 15  ADMISSION_LOCATION    1000 non-null   object
 16  DISCHARGE_LOCATION    1000 non-null   object
 17  INSURANCE             1000 non-null   object
 18  LANGUAGE              696 non-null    object
 19  RELIGION              993 non-null    object
 20  MARITAL_STATUS        972 non-null    object
 21  ETHNICITY             1000 non-null   object
 22  EDREGTIME             944 non-null    object
 23  EDOUTTIME             944 non-null    object
 24  DIAGNOSIS             1000 non-null   object
 25  HOSPITAL_EXPIRE_FLAG  1000 non-null   int64
 26  HAS_CHARTEVENTS_DATA  1000 non-null   int64
dtypes: int64(8), object(19)
memory usage: 211.1+ KB
```

2.2 Understand the dataset

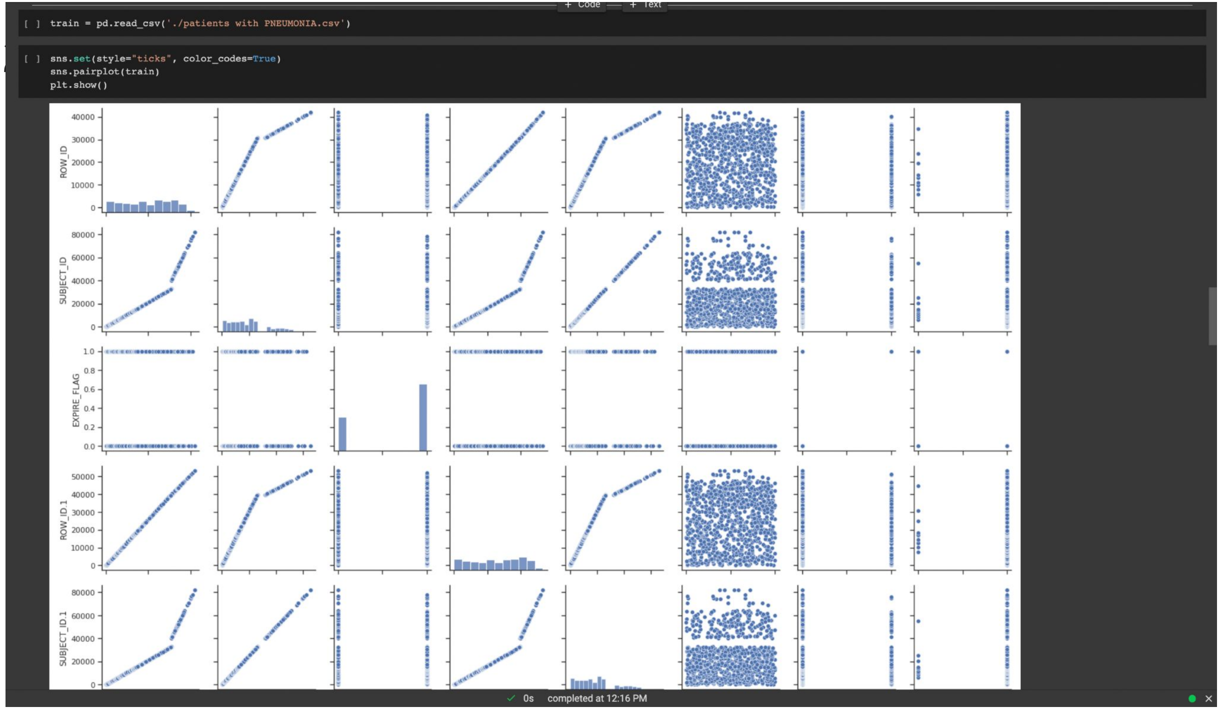
The drop() function is used to drop specified labels from rows or columns. Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names. When using a multi-index, labels on different levels can be removed by specifying the level.

```
[ ] # data overview
    train.describe()
```

	ROW_ID	SUBJECT_ID	EXPIRE_FLAG	ROW_ID.1	SUBJECT_ID.1	HADM_ID	HOSPITAL_EXPIRE_FLAG	HAS_CHARTEVENTS_DATA
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	19774.888000	24687.885000	0.665000	25508.516000	24687.885000	149999.458000	0.165000	0.989000
std	11272.786471	18158.488381	0.472227	14490.284071	18158.488381	28884.903681	0.371366	0.104355
min	61.000000	68.000000	0.000000	69.000000	68.000000	100030.000000	0.000000	0.000000
25%	10099.750000	10672.750000	0.000000	13050.250000	10672.750000	125618.500000	0.000000	1.000000
50%	20092.000000	21289.000000	1.000000	25987.500000	21289.000000	150052.000000	0.000000	1.000000
75%	29512.750000	31415.750000	1.000000	38203.750000	31415.750000	174600.500000	0.000000	1.000000
max	41904.000000	82211.000000	1.000000	53375.000000	82211.000000	199951.000000	1.000000	1.000000

```
[ ] train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 23 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   ROW_ID              1000 non-null   int64
 1   SUBJECT_ID          1000 non-null   int64
 2   GENDER              1000 non-null   object
 3   DOD_HOSP            464 non-null    object
 4   DOD_SSN             565 non-null    object
 5   EXPIRE_FLAG         1000 non-null   int64
 6   ROW_ID.1            1000 non-null   int64
 7   SUBJECT_ID.1        1000 non-null   int64
 8   HADM_ID             1000 non-null   int64
 9   ADMITTIME           1000 non-null   object
 10  DISCHTIME           1000 non-null   object
 11  DEATHTIME           165 non-null    object
 12  ADMISSION_TYPE      1000 non-null   object
 13  ADMISSION_LOCATION  1000 non-null   object
```

add analysis: If the probability is 0.000, the __regression model is statistically valid



Another useful way that you can learn about this data set is by generating a pairplot. You can use the seaborn method pairplot for this, and pass in the entire DataFrame as a parameter.

2.3 Build a linear regression

Creating Training Data and Test Data

Next, it's time to split train_data into training data and test data. As before, we will use built-in functionality from scikit-learn to do this. First, we need to divide our data into x values (the data we will be using to make predictions) and y values (the data we are attempting to predict).

Step3: Build a Linear Regression

scikit-learn makes it very easy to divide our data set into training data and test data. To do this, we'll need to import the function train_test_split from the model_selection module of scikit-learn.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

2.4 Splitting data

Creating Training Data and Test Data

We can use the train_test_split function combined with list unpacking to generate our training data and test data:

Step4: Logistic Regression/ Splitting Data

The first thing we need to do is split our data into an x-array (which contains the data that we will use to make predictions) and a y-array (which contains the data that we are trying to predict.

First, we should decide which columns to include. You can generate a list of the DataFrame's columns using train.columns, which outputs:

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3)
```

```
def convert_to_binary(x):
    if x == "PNEUMONIA":
        return 1
    else : return 0
```

```
x = train['SUBJECT_ID'].astype(float)
y = train['DIAGNOSIS'].apply(lambda x: convert_to_binary(x))
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=1)
```

```
y
```

```
0      1
1      1
2      1
3      1
4      1
      ..
995    1
996    1
997    1
998    1
999    1
Name: DIAGNOSIS, Length: 1000, dtype: int64
```

```
def convert_to_binary(x):
float
lambda
```

2.5 Model development

To train our model, we will first need to import the appropriate model from scikit-learn. To train the model, we need to call the fit method on the LogisticRegression object we just created and pass in our x_training_data and y_training_data variable.

Step5: Model Development & Training

Building and Training the Model

```
[ ] from sklearn.linear_model import LinearRegression
```

Next, we need to create an instance of the Linear Regression Python object. We will assign this to a variable called model

```
[ ] x = np.array([2.0 , 2.4, 1.5, 3.5, 3.5, 3.5, 3.5, 3.7, 3.7])
    y = np.array([196, 221, 136, 255, 244, 230, 232, 255, 267])

    lr = LinearRegression()
    lr.fit(x.reshape(-1, 1), y)

    LinearRegression()
```

```
[ ] print(lr.predict([[2.4]]))

    [198.36539227]
```

```
[ ] print(model.fit)

    <bound method LinearRegression.fit of LinearRegression()>
```

2.6 Prediction

Scikit-learn makes it very easy to make predictions from a machine learning model. You simply need to call the predict method on the model variable that we created earlier. The predictions variable holds the predicted values of the features stored in x_test. Since we used the train_test_split method to store the real values in y_test.

```
[ ] x= x.reshape(-1, 1)
    y= y.reshape(-1, 1)
```

```
[ ] x_test=x.reshape(-1,1)
```

```
[ ] y_test=y.reshape(-1,1)
```

```
[ ] y_pred = lr.predict(x_test)
```

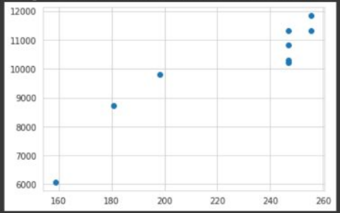
```
[ ] predictions=lr.predict(y_test)
```

An easy way to do this is plot the two arrays using a scatterplot. It's easy to build matplotlib scatterplots using the plt.scatter method. Here's the code for this:

+ Code + Text

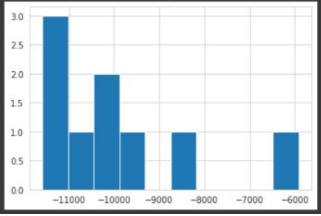
```
[ ] plt.scatter(y_pred, predictions)
```

```
<matplotlib.collections.PathCollection at 0x7fa84af37cd0>
```




```
[ ] plt.hist(y_pred - predictions)
```

```
(array([3., 1., 2., 1., 0., 1., 0., 0., 0., 1.]),
 array([-11581.11051374, -11014.59056153, -10448.07060932,  -9881.55065711,
         -9315.0307049 ,  -8748.51075269,  -8181.99080048,  -7615.47084827,
         -7048.95089606,  -6482.43094385,  -5915.91099164]),
 <a list of 10 Patch objects>)
```



2.7 Model evaulation

Measuring the Performance of a Logistic Regression Machine Learning Model

scikit-learn has an excellent built-in module called classification_report that makes it easy to measure the performance of a classification machine learning model.

```
[ ] from sklearn.metrics import classification_report

[ ] y_true = [0, 1, 2, 2, 2]
    y_pred = [0, 0, 2, 2, 1]
    target_names = ['class 0', 'class 1', 'class 2']

[ ] print(classification_report(y_true, y_pred, target_names=target_names))

                  precision    recall  f1-score   support

         class 0       0.50      1.00      0.67         1
         class 1       0.00      0.00      0.00         1
         class 2       1.00      0.67      0.80         3

        accuracy                           0.60         5
       macro avg       0.50      0.56      0.49         5
    weighted avg       0.70      0.60      0.61         5
```

2.7 Model evaulation

Confusion matrix

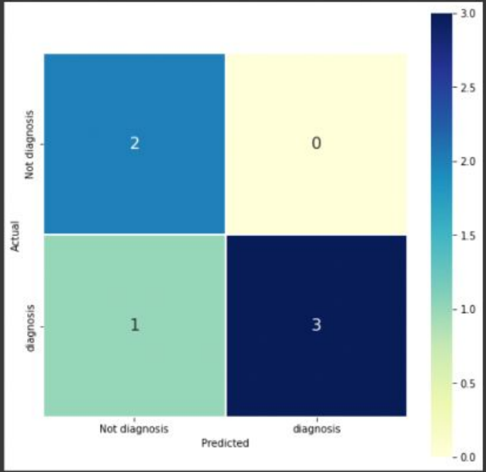
Compute confusion matrix to evaluate the accuracy of a classification. By definition a confusion matrix is such that is equal to the number of observations known to be in group and predicted to be in group .

```
# basic
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, y_pred)

df_cm = pd.DataFrame(cm)
df_cm.rename(columns={0:'Not diagnosis', 1:'diagnosis'}, index={0:'Not diagnosis', 1:'diagnosis'}, inplace=True)
df_cm
```

	Not diagnosis	diagnosis
Not diagnosis	2	0
diagnosis	1	3

```python
fig, ax = plt.subplots(figsize=(8,8))
sns.heatmap(df_cm, fmt='.0f',
            cmap='YlGnBu', linewidth=1,
            square=True, annot=True, annot_kws={'fontsize':16}, ax=ax)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')

plt.show()
```



2.7 Model evaulation

Logistic function flot

Compute confusion matrix to evaluate the accuracy of a classification. By definition a confusion matrix is such that is equal to the number of observations known to be in group and predicted to be in group .

3.Logistic Function Plot

```
coef = predicted.coef_
coef
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-66c2d13729b5> in <module>()
----> 1 coef = predicted.coef_
      2 coef

NameError: name 'predicted' is not defined
```

SEARCH STACK OVERFLOW

```
[ ] coef = coef.flatten()
    coef
```

```
[ ] intercept = model.intercept_
    intercept
```

Conclusion

Conclusion

Yes it can be used to predict whether a person had pneumonia or not. The accuracy and precision are above **70%.** But, it's not a good model. Especially if we look at the recall value **60%**, it's only about **61%** f1 - score. It means, this model only can catch__of patients who had pneumonia..

Root of Problems

I'm not doing any data preprocessing method, such as scaling the data and balancing the target data.I use a very simple logistic regression model

Future Works

Do some data preprocessing methods and use a more complex model