Using Logistic Regression

Trying to predict a classification- survival or deceased. Implementing Logistic Regression in Python for classification.

Used a "**semi-cleaned**" version of the Titanic data set, if you use the data set hosted directly on Kaggle, you may need to do some additional cleaning not shown in this lecture notebook.

```
Start coding or generate with AI.
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Data

reading the titanic_train.csv file into a pandas data frame.

```
train = pd.read_csv('titanic_train.csv')
```

```
train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S

Exploratory Data Analysis

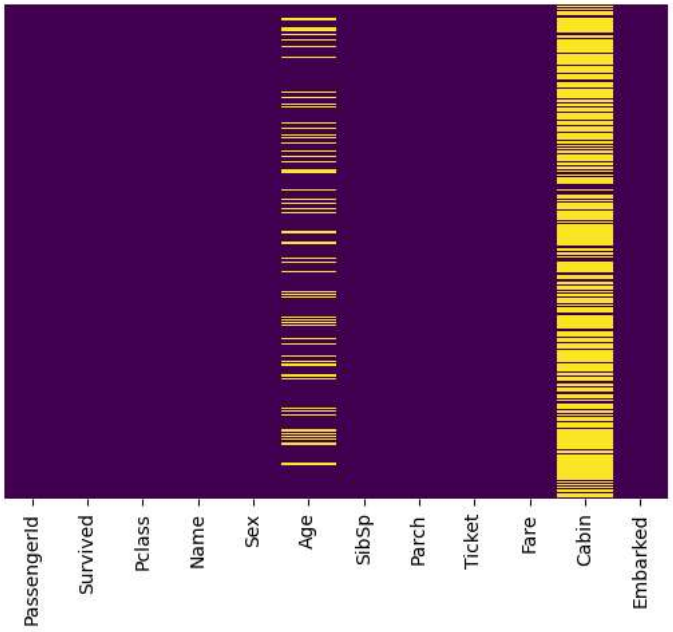
```
train.isnull()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...
886	False	False	False	False	False	False	False	False	False	False	True	False
887	False	False	False	False	False	False	False	False	False	False	False	False
888	False	False	False	False	False	True	False	False	False	False	True	False
889	False	False	False	False	False	False	False	False	False	False	False	False
890	False	False	False	False	False	False	False	False	False	False	True	False

891 rows × 12 columns

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

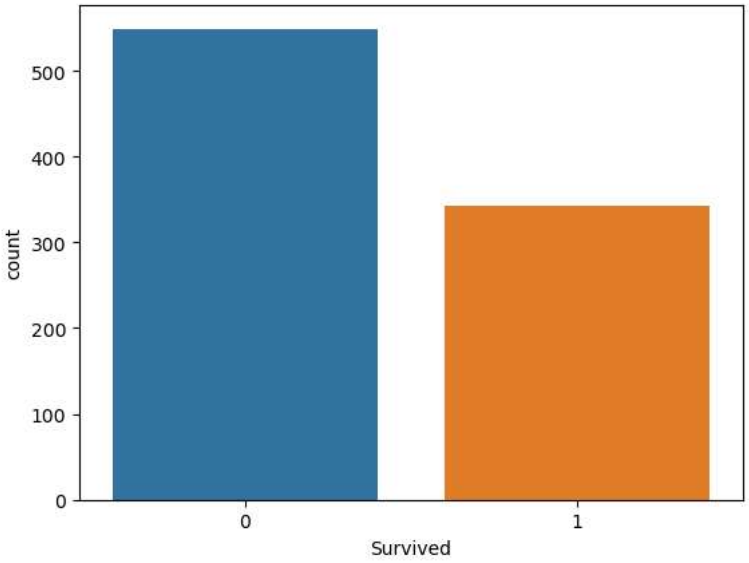
```
<Axes: >
```



We'll probably drop this later, or change it to another feature like "Cabin Known: 1 or 0"

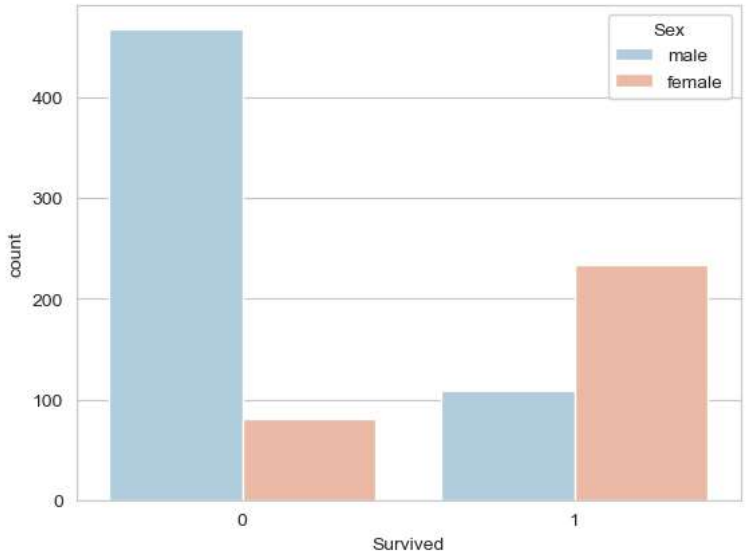
```
sns.countplot(x='Survived',data=train)
```

```
<Axes: xlabel='Survived', ylabel='count'>
```



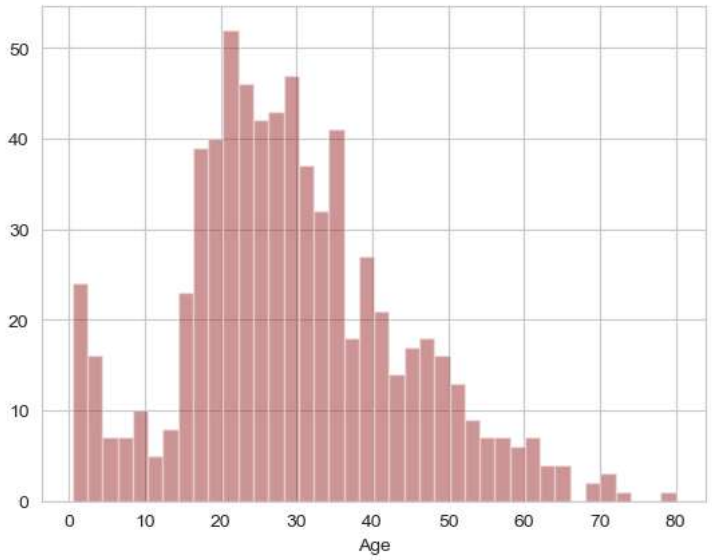

```
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

```
<Axes: xlabel='Survived', ylabel='count'>
```



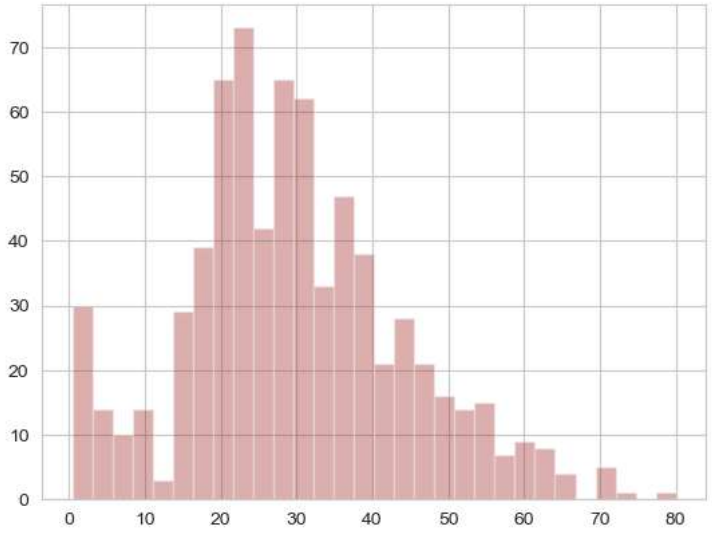

```
sns.distplot(train['Age'].dropna(),kde=False,color='darkred',bins=40)
```

```
C:\Users\dille\AppData\Local\Temp\ipykernel_7428\2002818437.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(train['Age'].dropna(),kde=False,color='darkred',bins=40)
<Axes: xlabel='Age'>
```




```
train['Age'].hist(bins=30,color='darkred',alpha=0.3)
```

```
<Axes: >
```

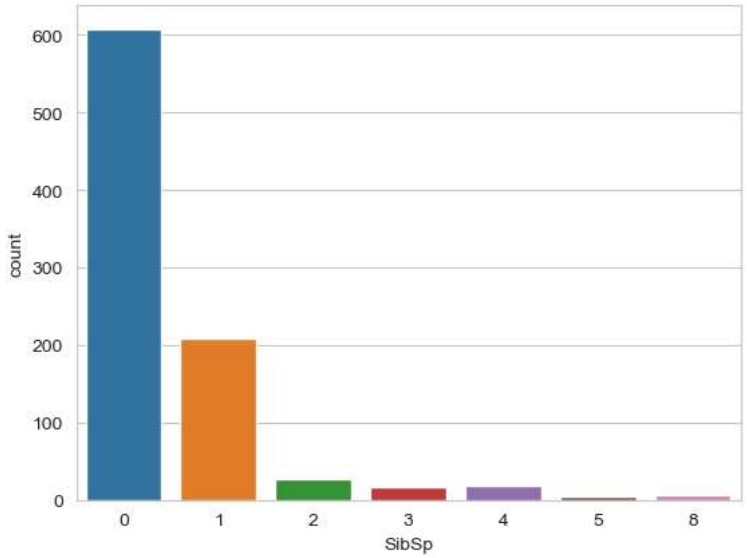



```
train['SibSp'].unique()
```

```
array([1, 0, 3, 4, 2, 5, 8], dtype=int64)
```

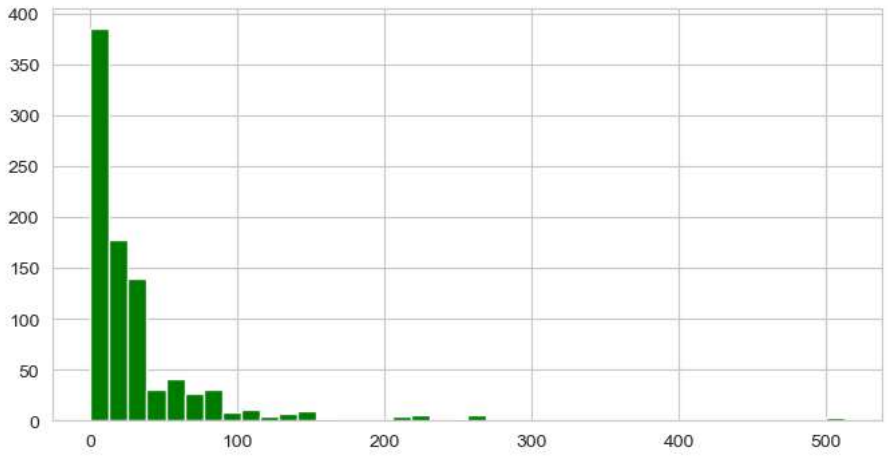
```
sns.countplot(x='SibSp',data=train)
```

```
<Axes: xlabel='SibSp', ylabel='count'>
```




```
train['Fare'].hist(color='green',bins=40,figsize=(8,4))
```

```
<Axes: >
```

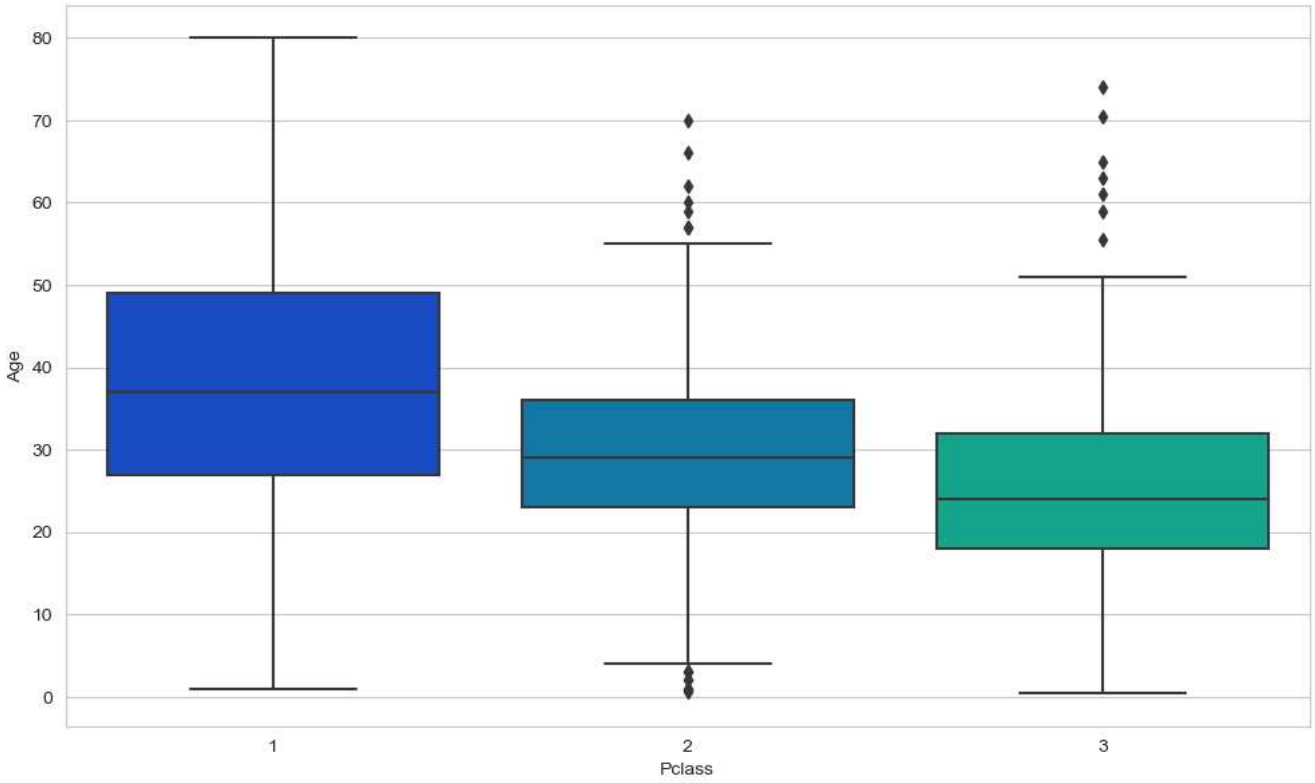


Data Cleaning

We want to fill in missing age data instead of just dropping the missing age data rows. One way to do this is by filling in the mean age of all the passengers (imputation). However we can be smarter about this and check the average age by passenger class. For example:

```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

```
<Axes: xlabel='Pclass', ylabel='Age'>
```



We can see the wealthier passengers in the higher classes tend to be older, which makes sense. We'll use these average age values to impute based on Pclass for Age.

```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age
```

Now apply that function!

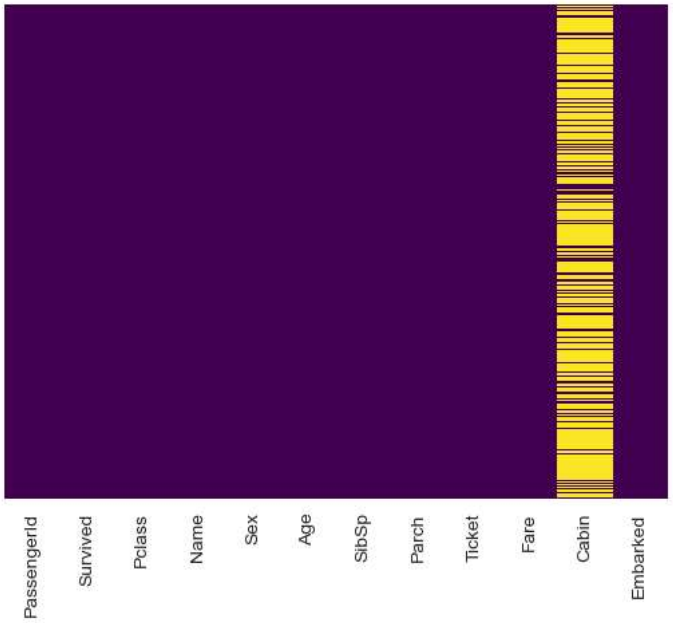
```
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

```
C:\Users\dille\AppData\Local\Temp\ipykernel_7428\822839471.py:2: FutureWarning: Series.__getitem__ treating keys as positions is dep
  Age = cols[0]
C:\Users\dille\AppData\Local\Temp\ipykernel_7428\822839471.py:3: FutureWarning: Series.__getitem__ treating keys as positions is dep
  Pclass = cols[1]
```

Now let's check that heat map again!

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
<Axes: >
```



Great! Let's go ahead and drop the Cabin column and the row in Embarked that is NaN.

```
train.drop('Cabin',axis=1,inplace=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[25], line 1
----> 1 train.drop('Cabin',axis=1,inplace=True)

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:5344, in DataFrame.drop(self, labels, axis, index, columns, level, inplace,
errors)
   5196 def drop(
   5197     self,
   5198     labels: IndexLabel | None = None,
   (...)
   5205     errors: IgnoreRaise = "raise",
   5206 ) -> DataFrame | None:
   5207     """
   5208     Drop specified labels from rows or columns.
   5209
   (...)
   5342             weight  1.0     0.8
   5343     """
-> 5344     return super().drop(
   5345         labels=labels,
   5346         axis=axis,
   5347         index=index,
   5348         columns=columns,
   5349         level=level,
   5350         inplace=inplace,
   5351         errors=errors,
   5352     )

File ~\anaconda3\Lib\site-packages\pandas\core\generic.py:4711, in NDFrame.drop(self, labels, axis, index, columns, level, inplace,
errors)
   4709 for axis, labels in axes.items():
   4710     if labels is not None:
-> 4711         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4713 if inplace:
   4714     self._update_inplace(obj)

File ~\anaconda3\Lib\site-packages\pandas\core\generic.py:4753, in NDFrame._drop_axis(self, labels, axis, level, errors,
only_slice)
   4751         new_axis = axis.drop(labels, level=level, errors=errors)
   4752     else:
-> 4753         new_axis = axis.drop(labels, errors=errors)
   4754     indexer = axis.get_indexer(new_axis)
   4756 # Case for non-unique axis
   4757 else:

File ~\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:7000, in Index.drop(self, labels, errors)
   6998 if mask.any():
   6999     if errors != "ignore":
-> 7000         raise KeyError(f"{labels[mask].tolist()} not found in axis")
   7001     indexer = indexer[~mask]
   7002 return self.delete(indexer)
```

```
train.head()
```

```
train.dropna(inplace=True)
```

Converting Categorical Features

We'll need to convert categorical features to dummy variables using pandas! Otherwise our machine learning algorithm won't be able to directly take in those features as inputs.

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          891 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
```

```
pd.get_dummies(train['Embarked'],drop_first=True).head()
```

	Q	S
0	False	True
1	False	False
2	False	True
3	False	True
4	False	True

```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

```
train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
```

```
train.head()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
0	1	0	3	22.0	1	0	7.2500
1	2	1	1	38.0	1	0	71.2833
2	3	1	3	26.0	0	0	7.9250
3	4	1	1	35.0	1	0	53.1000
4	5	0	3	35.0	0	0	8.0500

```
train = pd.concat([train,sex,embark],axis=1)
```

```
train.head()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	0	3	22.0	1	0	7.2500	True	False	True
1	2	1	1	38.0	1	0	71.2833	False	False	False
2	3	1	3	26.0	0	0	7.9250	False	False	True
3	4	1	1	35.0	1	0	53.1000	False	False	True
4	5	0	3	35.0	0	0	8.0500	True	False	True

Building a Logistic Regression model

```
## Train Test Split
train.drop('Survived',axis=1).head()
```

	PassengerId	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	3	22.0	1	0	7.2500	True	False	True
1	2	1	38.0	1	0	71.2833	False	False	False
2	3	3	26.0	0	0	7.9250	False	False	True
3	4	1	35.0	1	0	53.1000	False	False	True
4	5	3	35.0	0	0	8.0500	True	False	True

```
train['Survived'].head()
```

```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(train.drop('Survived',axis=1),
                                                    train['Survived'], test_size=0.30,
                                                    random_state=101)
```

Training and Predicting

```
from sklearn.linear_model import LogisticRegression


logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
C:\Users\dille\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (stat
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

LogisticRegression

LogisticRegression()

```
predictions = logmodel.predict(X_test)


from sklearn.metrics import confusion_matrix


accuracy=confusion_matrix(y_test,predictions)


accuracy
```

```
array([[133,  21],
       [ 42,  72]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score


accuracy=accuracy_score(y_test,predictions)
accuracy
```

```
0.7191011235955056
```

```
predictions
```

```
array([0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
```

```
0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0
```