Data Analysis and Machine-Learning

Chapter 10.2.

ML Modelling Applications (2): Various Instruments for Effective EDA

Regular Expressions, Natural Language Data Processing, Pivot Tables, Missing Value Imputations, Dummy Variables

Pandas Crosstab, Groupby function, lambda application, Indexing



*In a world of constant change,*

*The fundamentals are most important than ever*

*- James C. Collins*

1. Introduction

So far we have covered the algorithmic concepts and mathematical calculations regarding various ML models, including Linear Models, Generalized Linear Models, Support Vector Machine, K-Nearest Neighbor, Decision-Tree Model, Random Forest Model, etc. We have also covered various data-science techniques for effective analytics and modelling, including feature selection methods (FS, BE, SS), Multicollinearity/Variance Inflation Factor processing, Variable Weight Evaluation Methodologies (e.g., eli5, Permutation Importance, Shap), scaling, stochastic modelling, regularizations (L1, L2), etc.

In actual applications, EDA (Exploratory Data Analysis), descriptive statistics of variables, visualizations, and preprocessing consist a big part in prior to the actual modelling process, as deeply learning about the data per se is essential to generate an effective model with best accuracy. Focusing on such aspects together with the purpose of demonstrating various modelling methodologies, this chapter will be focusing on various applications of ML analysis & models via data coding. In this subchapter in particular, we will be analyzing the famous titanic dataset, focusing particularly on demonstrating various useful instruments for efficient EDAs, including the usage of regular expressions for unstructured data processing, making pivot tables, imputations of missing values, generating dummy variables, and utilizing lambda function for quick data manipulations.

2. EDA (Exploratory Data Analysis) and Visualizations of Variables

#Import Essential Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

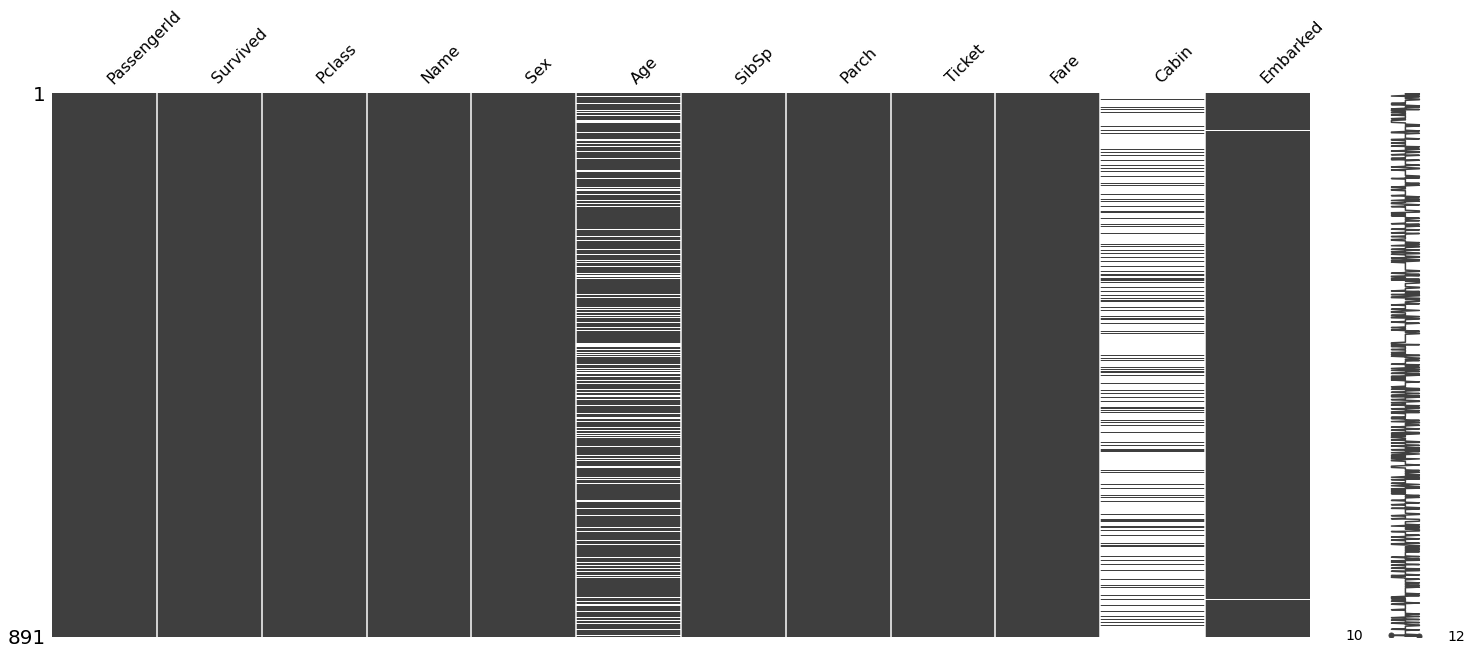
#Import Data

rawData = pd.read\_csv('Your File Path\\train.csv')

#Missing Values

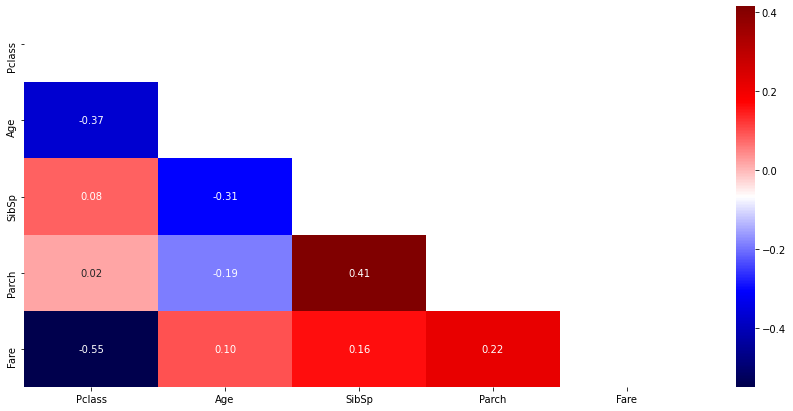
import missingno

missingno.matrix(rawData)



#Correlations

dt.visualCorr(rawData.drop(columns=['PassengerId', 'Survived']))



2.1. Distributions of the Variables (Categorical)

rawData['Survived'].value\_counts()

0 549

1 342

Name: Survived, dtype: int64

plt.figure(figsize=(15,2))

sns.countplot(data=rawData, y='Survived')



plt.figure(figsize=(12,7))

plt.subplot(1,2,1)

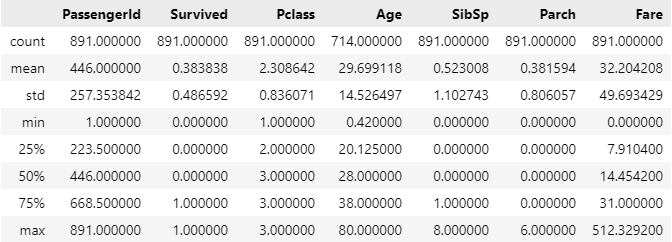
sns.countplot(data=rawData, x='Survived')

plt.subplot(1,2,2)

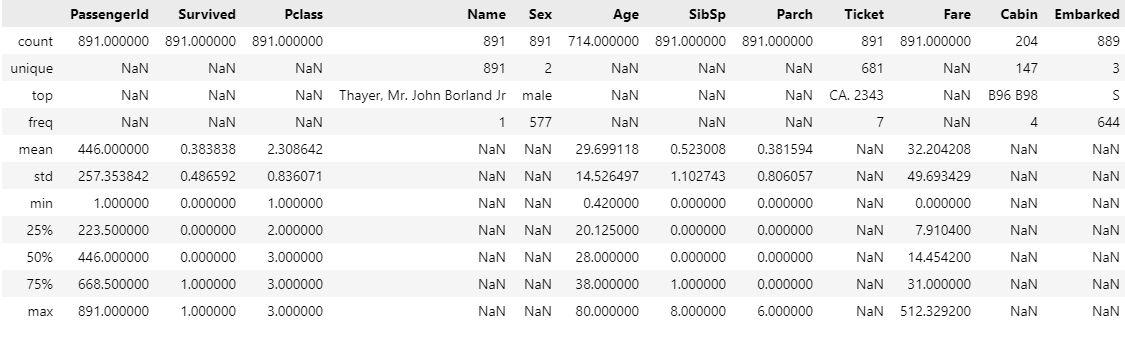
rawData['Survived'].value\_counts().plot.pie(autopct='%1.1f%%', explode=[0,0.03], wedgeprops={'width': 0.7, 'edgecolor': 'w', 'linewidth': 5})



rawData.describe()



rawData.describe(include='all')



rawData.columns

Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'], dtype='object')

rawData['Pclass'].value\_counts().sort\_values()

2 184

1 216

3 491

Name: Pclass, dtype: int64

rawData['Pclass'].value\_counts().sort\_index()

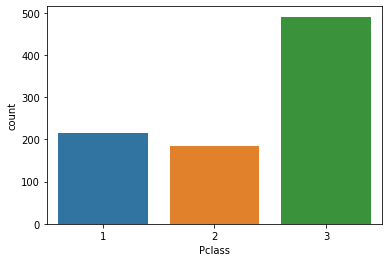
1 216

2 184

3 491

Name: Pclass, dtype: int64

sns.countplot(data=rawData, x='Pclass')



rawData['Sex'].unique()

array(['male', 'female'], dtype=object)

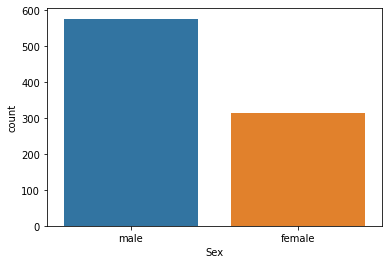
rawData['Sex'].value\_counts()

male 577

female 314

Name: Sex, dtype: int64

sns.countplot(data=rawData, x='Sex')



rawData['SibSp'].value\_counts()

0 608

1 209

2 28

4 18

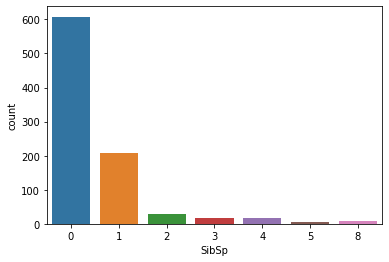
3 16

8 7

5 5

Name: SibSp, dtype: int64

sns.countplot(data=rawData, x='SibSp')



rawData['Parch'].value\_counts()

0 678

1 118

2 80

3 5

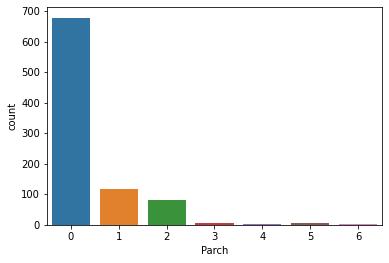
5 5

4 4

6 1

Name: Parch, dtype: int64

sns.countplot(data=rawData, x='Parch')



rawData['Embarked'].value\_counts()

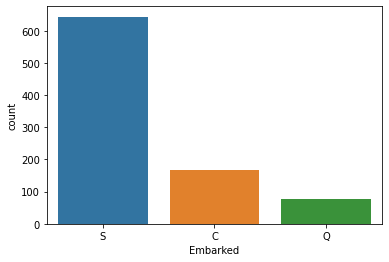
S 644

C 168

Q 77

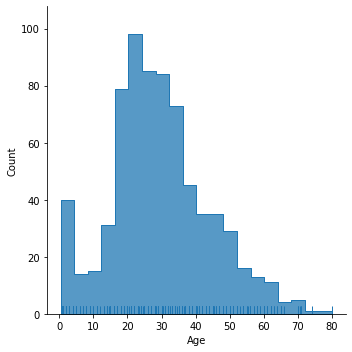
Name: Embarked, dtype: int64

sns.countplot(data=rawData, x='Embarked')

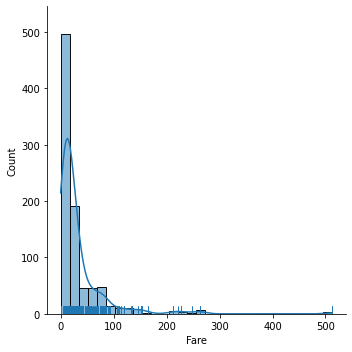


2.2. Distributions of the Variables (Continuous Data)

sns.displot(data=rawData, x='Age', element='step', rug=True)



sns.displot(data=rawData, x='Fare', bins=30, kde=True, rug=True)



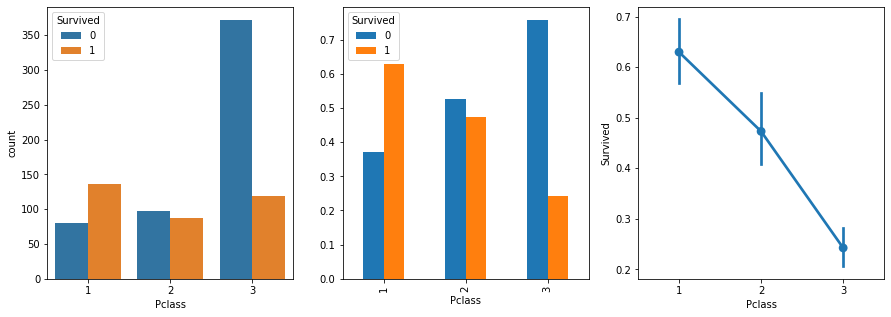
2.3. Effects on the Target Variable (Relationship with Y)

fig, ax = plt.subplots(1,3, figsize=(15,5))

sns.countplot(data=rawData, x='Pclass', hue='Survived', ax=ax[0])

pd.crosstab(rawData['Pclass'], rawData['Survived'], normalize='index').plot.bar(ax=ax[1])

sns.pointplot(data=rawData, x='Pclass', y='Survived', ax=ax[2])

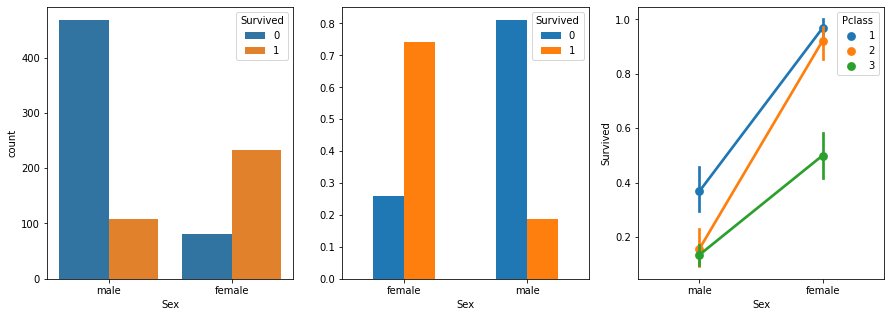


fig, ax = plt.subplots(1,3, figsize=(15,5))

sns.countplot(data=rawData, x='Sex', hue='Survived', ax=ax[0])

pd.crosstab(rawData['Sex'], rawData['Survived'], normalize='index').plot.bar(ax=ax[1], rot=0)

sns.pointplot(data=rawData, x='Sex', y='Survived', hue='Pclass')



fig, ax = plt.subplots(2,2, figsize=(15,15))

sns.countplot(data=rawData, x='SibSp', hue='Survived', ax=ax[0,0])

ax[0,0].set\_title('SibSp (Counts)')

pd.crosstab(rawData['SibSp'], rawData['Survived'], normalize='index').plot.bar(ax = ax[0,1])

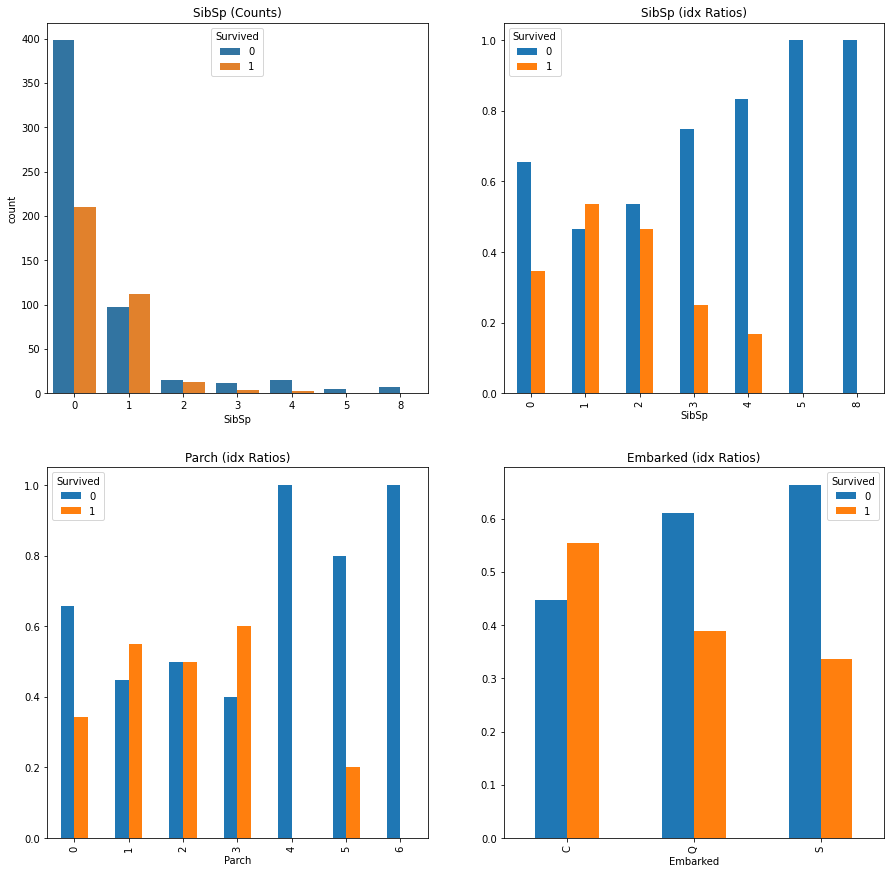
ax[0,1].set\_title('SibSp (idx Ratios)')

pd.crosstab(rawData['Parch'], rawData['Survived'], normalize='index').plot.bar(ax = ax[1,0])

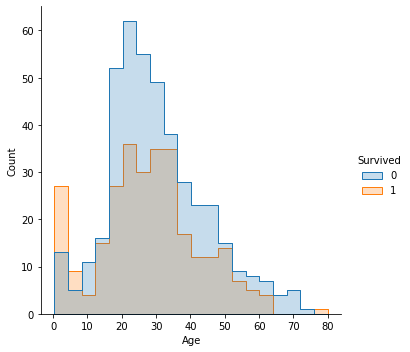
ax[1,0].set\_title('Parch (idx Ratios)')

pd.crosstab(rawData['Embarked'], rawData['Survived'], normalize='index').plot.bar(ax = ax[1,1])

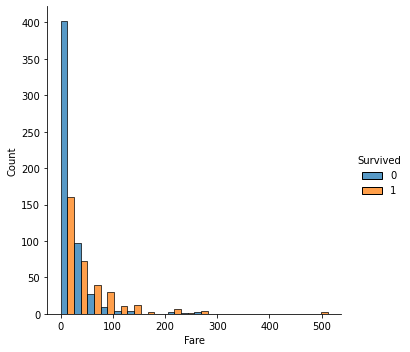
ax[1,1].set\_title('Embarked (idx Ratios)')



sns.displot(data=rawData, x='Age', hue='Survived', multiple='layer', element='step')

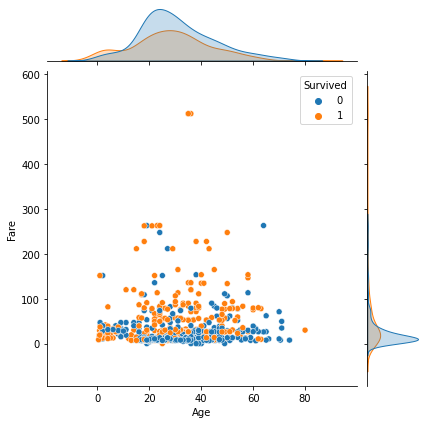


sns.displot(data=rawData, x='Fare', hue='Survived', bins=20, multiple='dodge')



#Jointplot

sns.jointplot(data=rawData, x='Age', y='Fare', hue='Survived')



rawData['Cabin'].isna().sum()

687

display(rawData[rawData['Pclass']==1]['Cabin'].isna().sum())

display(rawData[rawData['Pclass']==2]['Cabin'].isna().sum())

rawData[rawData['Pclass']==3]['Cabin'].isna().sum()

40

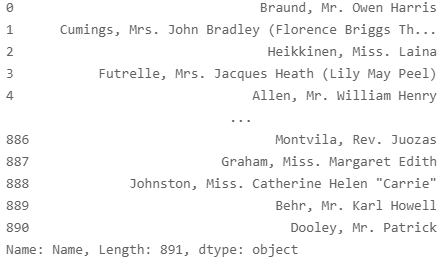
168

479

2.4. Processing Natural Language Data using Regular Expressions

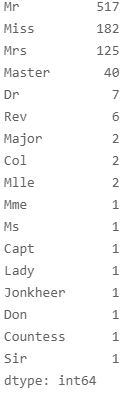
#Name Analysis

rawData.Name



#Regular Expressions (To extract middle names only)

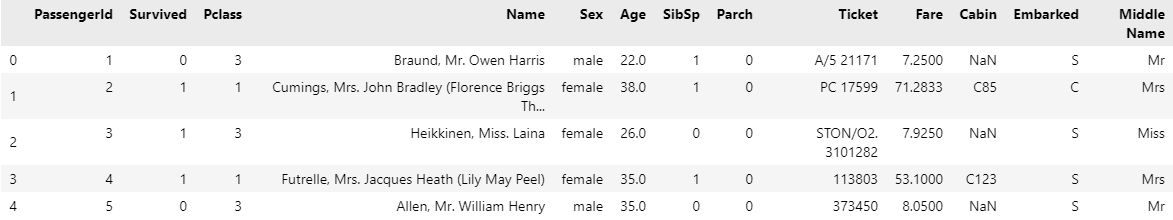
rawData.Name.str.extract(r'([A-Za-z]+)\.').value\_counts()



#New Column: Middle Name

rawData['Middle Name'] = rawData.Name.str.extract(r'([A-Za-z]+)\.')

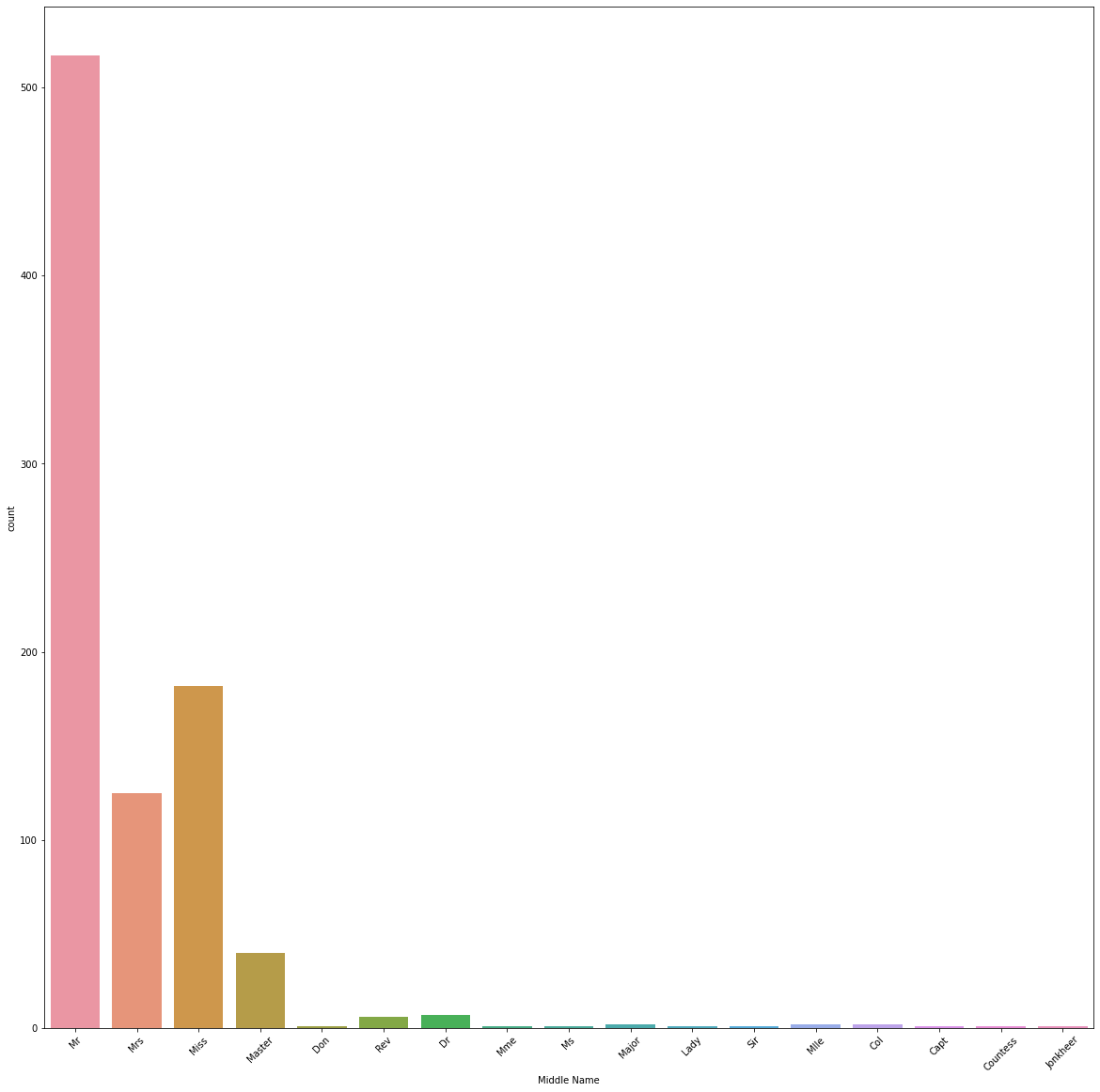
rawData.head()



plt.figure(figsize=(20,20))

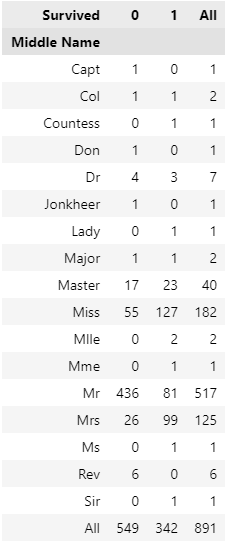
sns.countplot(data=rawData, x='Middle Name')

plt.xticks(rotation=45)



#Crosstab Information

pd.crosstab(rawData['Middle Name'], rawData['Survived'], margins=True)

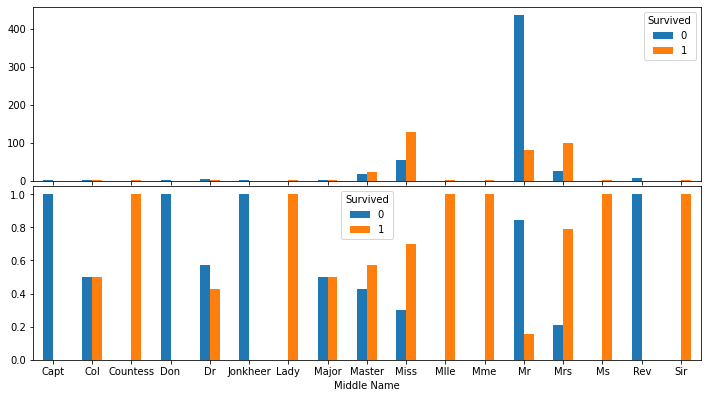


fig, ax = plt.subplots(2,1, figsize=(10,30))

fig.tight\_layout()

pd.crosstab(rawData['Middle Name'], rawData['Survived']).plot.bar(figsize=(10,5), rot=0, ax=ax[0])

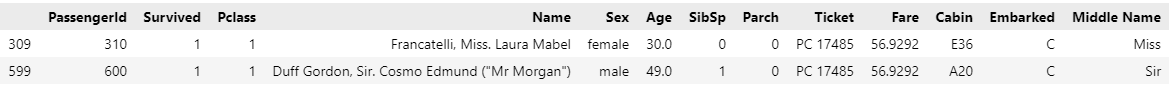
pd.crosstab(rawData['Middle Name'], rawData['Survived'], normalize='index').plot.bar(figsize=(10,5), rot=0, ax=ax[1])



for \_, df in rawData.groupby(['Ticket']):

    if len(df) >= 2 and df['Middle Name'].str.contains('Sir').any():

        display(df)



rawData['Cabin'].unique()

array([nan, 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6', 'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33', 'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101', 'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4', 'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35', 'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19', 'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54', 'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40', 'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44', 'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14', 'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38', 'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68', 'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48', 'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63', 'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30', 'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36', 'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42', 'C148'], dtype=object)

#lambda application on dataframe to extract 'decks' information from the 'Cabin' Column

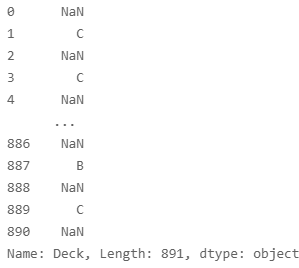
def extract\_decks(x):

    if not x is np.nan: return x.split()[-1][0:1]

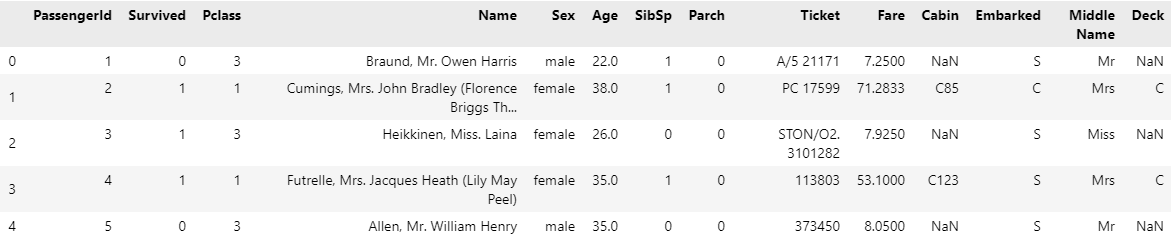
    else: return np.nan

rawData['Deck'] = rawData['Cabin'].apply(lambda x: extract\_decks(x))

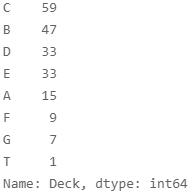
rawData['Deck']



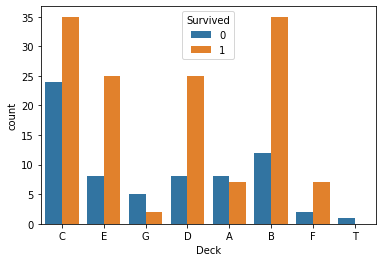
rawData.head()



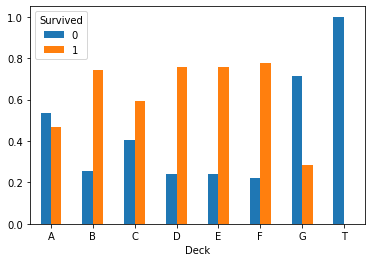
rawData['Deck'].value\_counts()



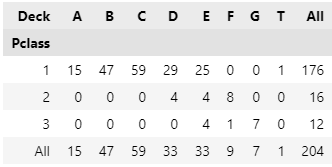
sns.countplot(data=rawData, x='Deck', hue='Survived')



pd.crosstab(rawData['Deck'], rawData['Survived'], normalize='index').plot.bar(rot=0)



pd.crosstab(rawData['Deck'], rawData['Pclass'], margins=True).T



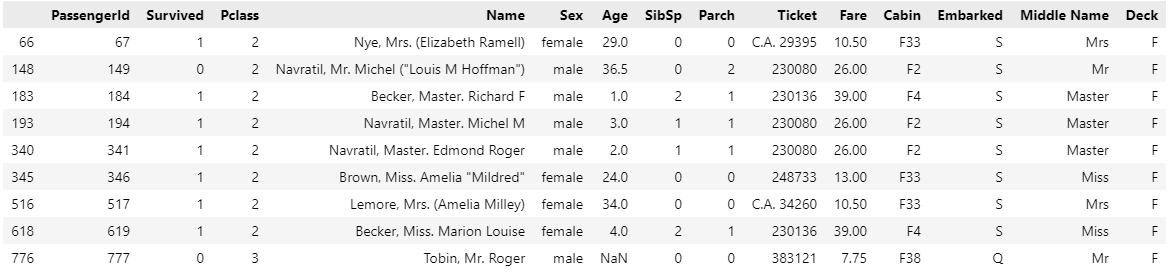
rawData[rawData['Deck']=='T']



rawData[rawData['Deck']=='A']

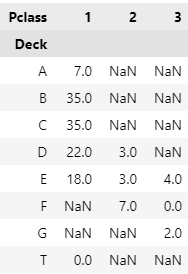


rawData[rawData['Deck']=='F']



#Using pivot table enables to summarize the combined variable information efficiently, as follows.

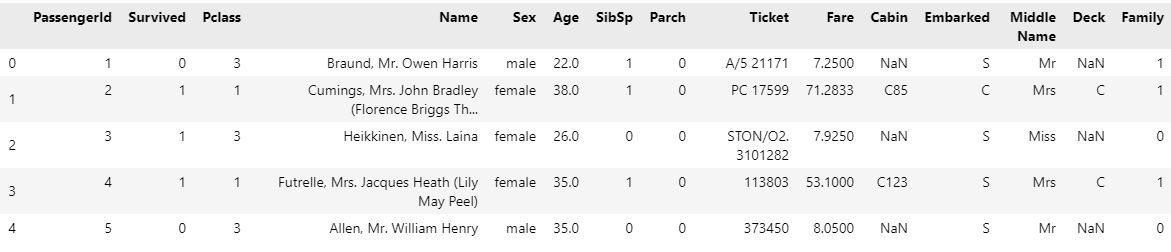
pd.pivot\_table(rawData, index='Deck', columns='Pclass', values='Survived', aggfunc='sum')



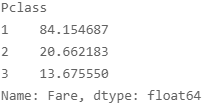
#Genearting Family Column

rawData['Family'] = rawData['SibSp'] + rawData['Parch']

rawData.head()



rawData.groupby('Pclass')['Fare'].mean()



#Fellow Passenger Information

for \_, group in rawData.groupby('Ticket'):

    if len(group) > 1:

        if len( group[ (group['Family'] == 0) & (group['Fare'] > 70) ] ):

            idx = group[ (group['Family'] == 0) & (group['Fare'] > 70) ].index

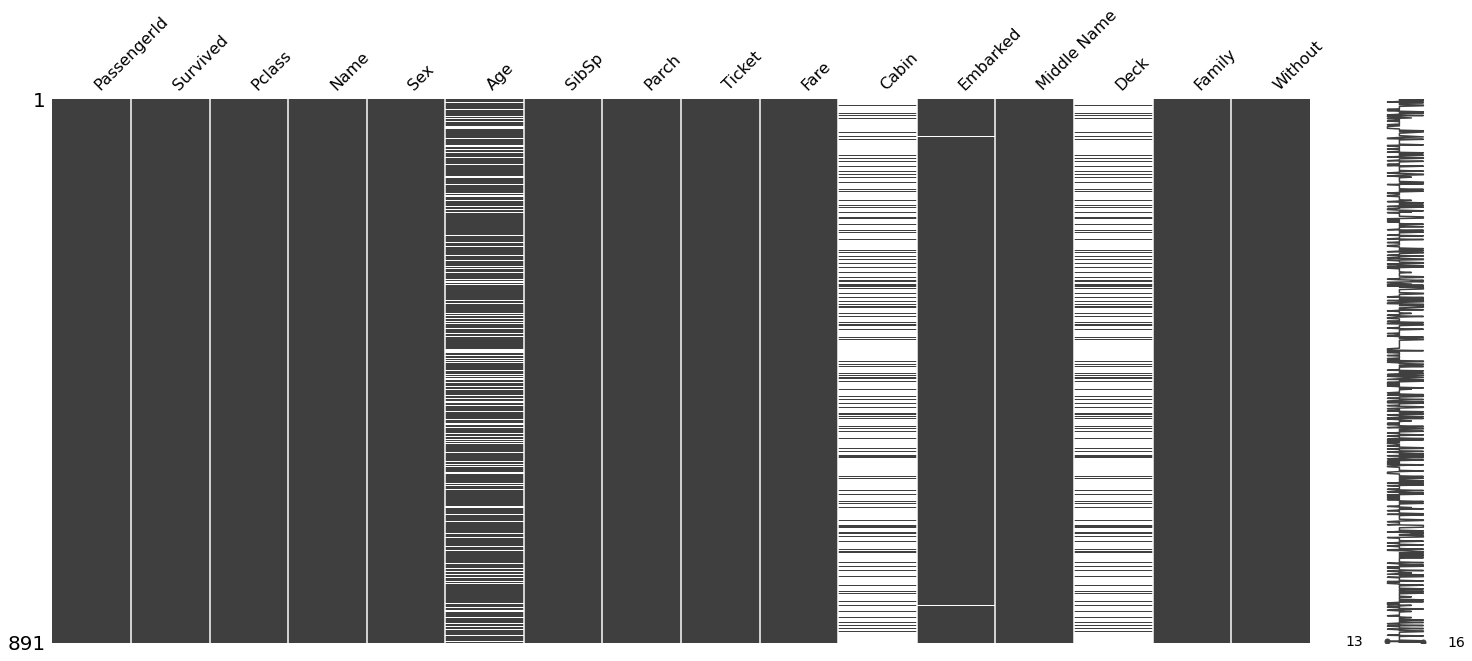
            rawData.loc[idx, 'Without'] = 1

idx = rawData[ rawData['Without'] != 1]['Without'].index

rawData.loc[idx, 'Without'] = 0

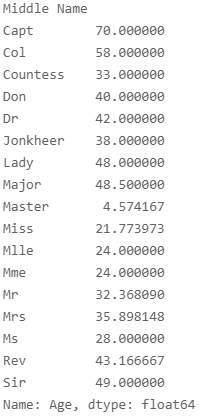
3. Preprocessing

missingno.matrix(rawData)



#Imputation of Missing Values

rawData.groupby('Middle Name')['Age'].mean()



lists = rawData['Middle Name'].unique()

for i in lists:

    rawData['Mean Age'] = rawData.loc[rawData['Middle Name']==i, 'Age'].mean()

rawData['Mean Age'].unique()

array([38.])

len(rawData['Mean Age'])

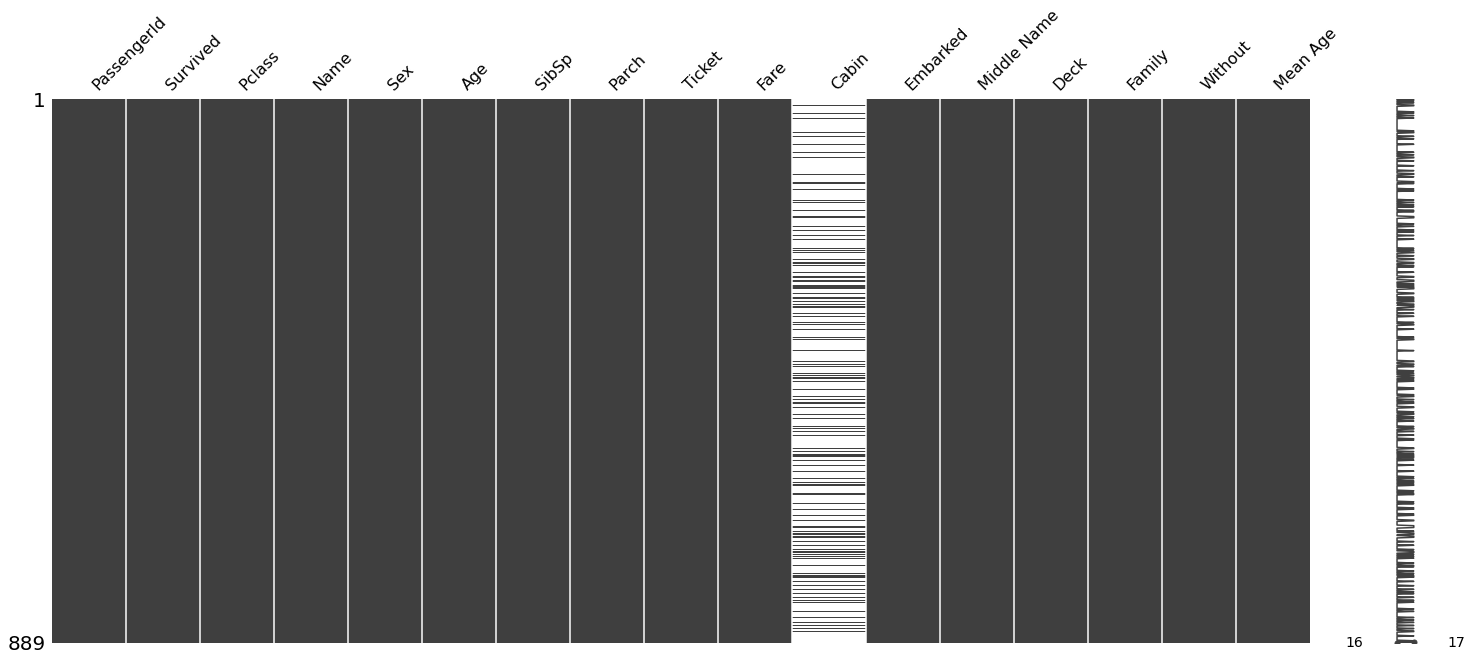
891

rawData['Age'].fillna(rawData['Mean Age'], inplace=True)

rawData['Deck'].fillna('None', inplace=True)

rawData.dropna(subset=['Embarked'], inplace=True)

missingno.matrix(rawData)



rawData.columns

Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'Middle Name', 'Deck', 'Family', 'Without', 'Mean Age'], dtype='object')

rawData.drop(columns = ['PassengerId', 'Name', 'SibSp', 'Parch', 'Ticket', 'Cabin', 'Middle Name', 'Mean Age'], inplace=True)

#Spotting and Processing Outliers

rawData.Age.unique()

array([22. , 38. , 26. , 35. , 54. , 2. , 27. , 14. , 4. , 58. , 20. , 39. , 55. , 31. , 34. , 15. , 28. , 8. , 19. , 40. , 66. , 42. , 21. , 18. , 3. , 7. , 49. , 29. , 65. , 28.5 , 5. , 11. , 45. , 17. , 32. , 16. , 25. , 0.83, 30. , 33. , 23. , 24. , 46. , 59. , 71. , 37. , 47. , 14.5 , 70.5 , 32.5 , 12. , 9. , 36.5 , 51. , 55.5 , 40.5 , 44. , 1. , 61. , 56. , 50. , 36. , 45.5 , 20.5 , 62. , 41. , 52. , 63. , 23.5 , 0.92, 43. , 60. , 10. , 64. , 13. , 48. , 0.75, 53. , 57. , 80. , 70. , 24.5 , 6. , 0.67, 30.5 , 0.42, 34.5 , 74. ])

rawData.Age.max()

80.0

idx = rawData[ rawData['Age'] == 80 ].index

rawData.drop(index = idx, inplace=True)

rawData.Age.unique()

array([22. , 38. , 26. , 35. , 54. , 2. , 27. , 14. , 4. , 58. , 20. , 39. , 55. , 31. , 34. , 15. , 28. , 8. , 19. , 40. , 66. , 42. , 21. , 18. , 3. , 7. , 49. , 29. , 65. , 28.5 , 5. , 11. , 45. , 17. , 32. , 16. , 25. , 0.83, 30. , 33. , 23. , 24. , 46. , 59. , 71. , 37. , 47. , 14.5 , 70.5 , 32.5 , 12. , 9. , 36.5 , 51. , 55.5 , 40.5 , 44. , 1. , 61. , 56. , 50. , 36. , 45.5 , 20.5 , 62. , 41. , 52. , 63. , 23.5 , 0.92, 43. , 60. , 10. , 64. , 13. , 48. , 0.75, 53. , 57. , 70. , 24.5 , 6. , 0.67, 30.5 , 0.42, 34.5 , 74. ])

rawData.Age.max()

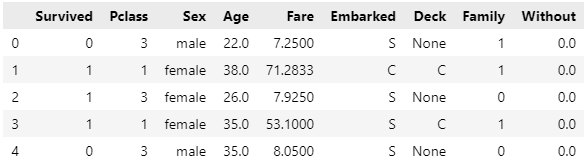
74.0

rawData[rawData['Age']==74]



#Generating Dummy Variables for Categorical Data

rawData.head()



rawData = pd.get\_dummies(data=rawData, columns=['Sex', 'Embarked', 'Deck'])

rawData.columns

Index(['Survived', 'Pclass', 'Age', 'Fare', 'Family', 'Without', 'Sex\_female', 'Sex\_male', 'Embarked\_C', 'Embarked\_Q', 'Embarked\_S', 'Deck\_A', 'Deck\_B', 'Deck\_C', 'Deck\_D', 'Deck\_E', 'Deck\_F', 'Deck\_G', 'Deck\_None', 'Deck\_T'], dtype='object')

rawData['Deck\_T'].value\_counts()

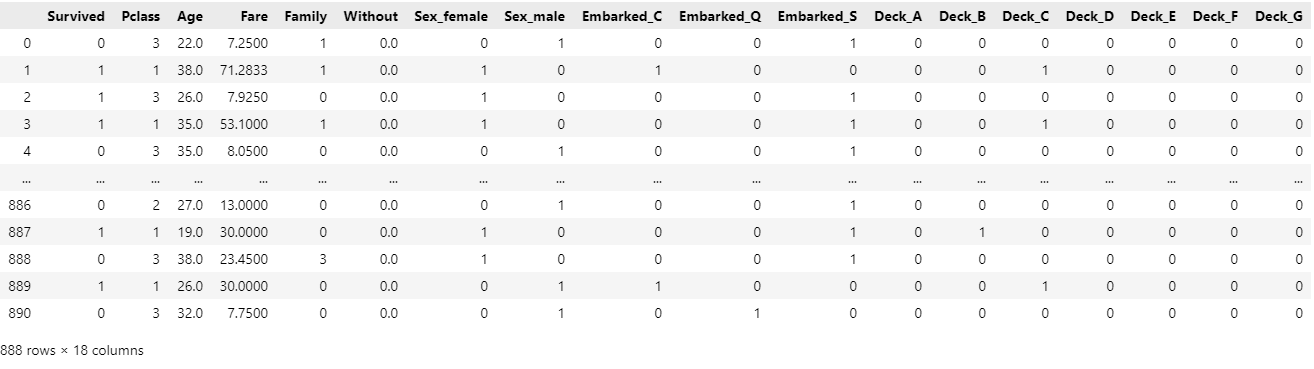
0 887

1 1

Name: Deck\_T, dtype: int64

rawData.drop(columns=['Deck\_None', 'Deck\_T'], inplace=True)

rawData



4. Modelling

4.1. Logistic Regression

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

from sklearn.model\_selection import train\_test\_split, GridSearchCV, StratifiedKFold, cross\_val\_score

x = rawData.drop(columns=['Survived'])

y = rawData['Survived']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, shuffle=True, random\_state=123)

folds = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=123)

baseModel = LogisticRegression(max\_iter=100000000)

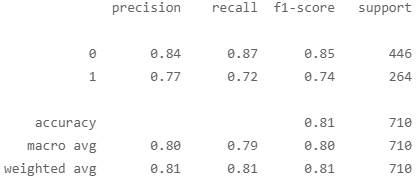
baseModel.fit(x\_train, y\_train)

yhat\_train = baseModel.predict(x\_train)

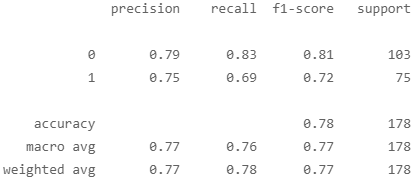
yhat\_test = baseModel.predict(x\_test)

#Evaluations

print(classification\_report(y\_train, yhat\_train))



print(classification\_report(y\_test, yhat\_test))



scores\_train = cross\_val\_score(baseModel, x\_train, y\_train, scoring='roc\_auc', cv=folds)

scores\_test = cross\_val\_score(baseModel, x\_test, y\_test, scoring='roc\_auc', cv=folds)

display(scores\_train.mean())

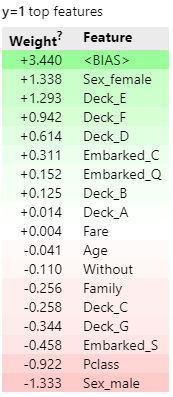
display(scores\_test.mean())

0.8603561253561255

0.8377435064935066

import eli5

eli5.show\_weights(baseModel, feature\_names = x\_train.columns.tolist())



4.2. Support Vector Machine

from sklearn.svm import SVC

svmModel = SVC()

params = {

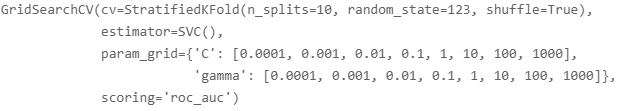
    'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],

    'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

}

search = GridSearchCV(svmModel, param\_grid=params, cv=folds, scoring='roc\_auc')

search.fit(x\_train, y\_train)



display(search.best\_params\_)

display(search.best\_estimator\_)

display(search.best\_score\_)

{'C': 100, 'gamma': 0.0001}

SVC(C=100, gamma=0.0001)

0.8505257705257707

#Evaluations (Classification Report)

yhat\_train = search.predict(x\_train)

yhat\_test = search.predict(x\_test)

print(classification\_report(y\_test, yhat\_test))

