

Report For Titanic Machine Learning And Performance Measurement

First of all we imported necessary libraries (Pandas, Numpy, Matplotlib, Seaborn) for data extraction, editing and visualization. Then reading my train csv file, I checked for null values.

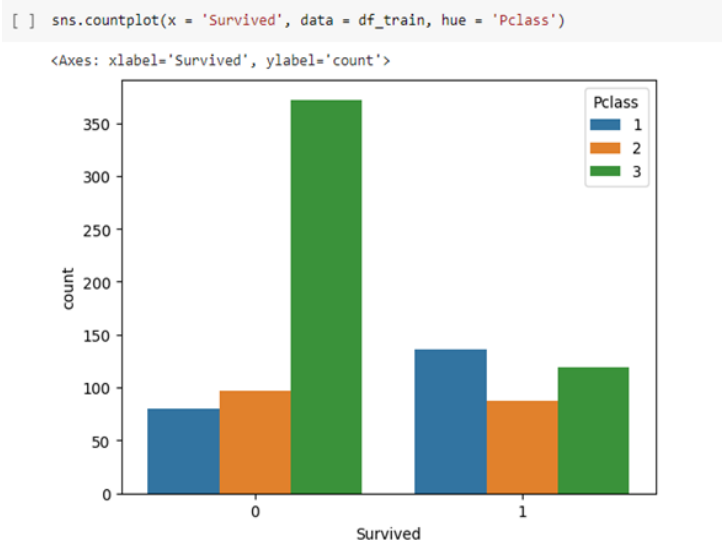
```
[ ] df_train.info()

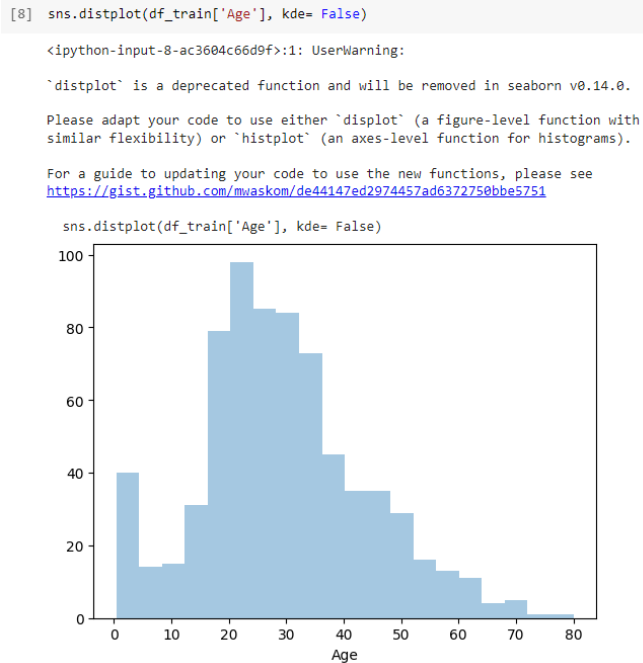
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column             Non-Null Count  Dtype
---  -
0   PassengerId         891 non-null    int64
1   Survived            891 non-null    int64
2   Pclass              891 non-null    int64
3   Name                891 non-null    object
4   Sex                 891 non-null    object
5   Age                 714 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  Cabin               204 non-null    object
11  Embarked            889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

I got the output you see. I made comments with some of the non-null columns on this output. I made the interpretations on graphics with seaborn and matplotlib library.

One bar will be for passengers who did not survive, and the other will be for those who did. The number of passengers in each group is indicated by the height of the corresponding bar. The bars will also be further distinguished by color, with each hue standing for a distinct passenger class. (Pclass).

Understanding how the survival rate differed by passenger class can be done using this plot. For instance, if the plot indicates that more first-class passengers escaped the disaster than those traveling in second or third class, it could be inferred that having been in a higher class may have given one an advantage.





Visualization of the age distribution in the df_train dataset is provided by the output code. The spread of age values will be represented by the x-axis, and the frequency or count of people who fit each age value will be shown on the y-axis. The height of the bar in each bin on the histogram indicates how many people in the dataset have ages that lie within that range, while the width of the bar in each bin represents a range of age values.

Overall, this depiction will contribute to the dissemination of insights into the distribution of ages in the dataset, including metrics such as the median and skewness, which can be used to guide data analysis and modeling choices.

```
[53] print(df_train[df_train['Pclass'] == 1]['Age'].mean())
      print(df_train[df_train['Pclass'] == 2]['Age'].mean())
      print(df_train[df_train['Pclass'] == 3]['Age'].mean())

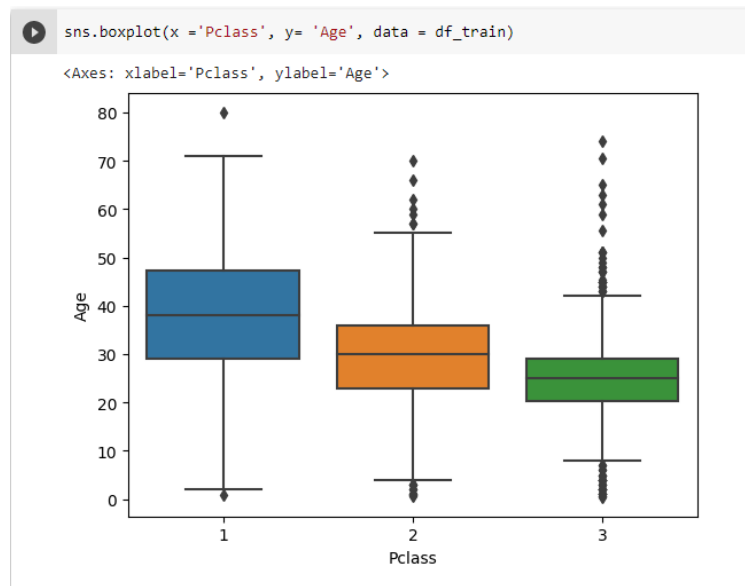
38.233440860215055
29.87763005780347
25.14061971830986
```

```
[54] def fill_the_na_values(cols):
      age= cols[0]
      pclass = cols[1]

      if pd.isna(age):
          if pclass == 1:
              return round(df_train[df_train['Pclass'] == 1]['Age'].mean())
          if pclass == 2:
              return round(df_train[df_train['Pclass'] == 2]['Age'].mean())
          if pclass == 3:
              return round(df_train[df_train['Pclass'] == 3]['Age'].mean())
          else:
              return age

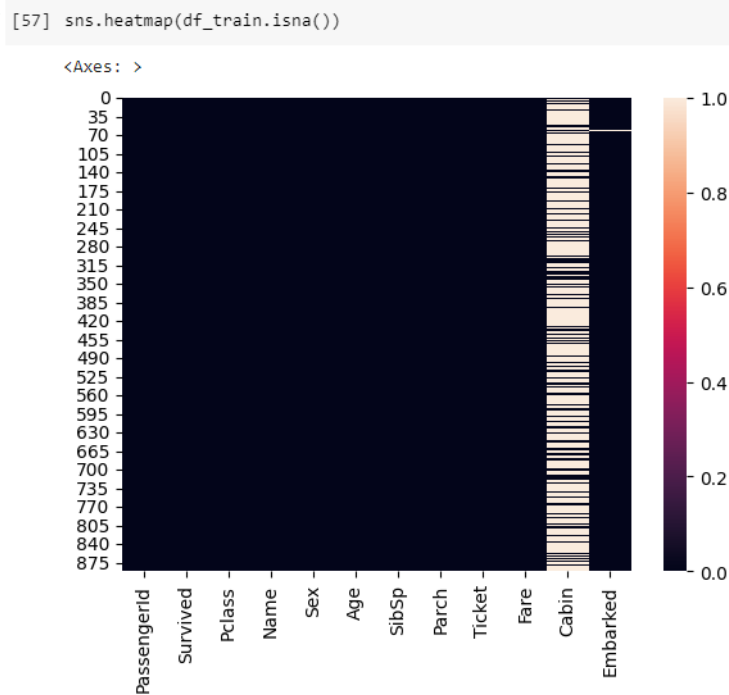
      df_train['Age'] = df_train[['Age', 'Pclass']].apply(fill_the_na_values, axis=1)
```

I filled in the null values in the age column according to the average age of the 1,2,3rd classes in the pclass column. Normally this may not be true, but at the moment I have limited data so I have to act as a guess.



I can see that from this graph. 1st class passengers are all older, so older people are more wealthy. I can make a comment like this.

After that I can use heatmap for null values. Apart from the age column, I also have empty values in our cabin and embarked columns.



After all graphs and interpret I can drop the cabin , PassengerId, Name ,Ticket and other null values on Embarked. Because I only want values that can be numeric and that I can comment on.

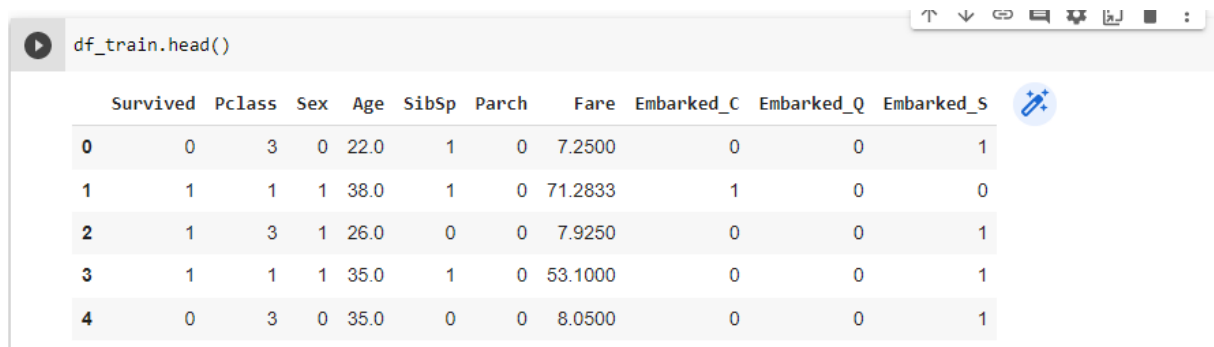
But I still have string values in my df. These columns are the sex and embarked columns. I apply operations to these columns as 0 and 1.

```
[66] df_train['Sex'] = df_train['Sex'].replace({'male': 0, 'female': 1})
```

```
[68] Embarked = pd.get_dummies(df_train.Embarked, prefix= 'Embarked')  
Embarked
```

```
[69] df_train = df_train.join(Embarked)  
df_train.drop(['Embarked'],axis=1,inplace = True)
```

I'm dropping my old embarked column from my df while adding my embarked column to which I applied dummies to my df.



	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked_C	Embarked_Q	Embarked_S
0	0	3	0	22.0	1	0	7.2500	0	0	1
1	1	1	1	38.0	1	0	71.2833	1	0	0
2	1	3	1	26.0	0	0	7.9250	0	0	1
3	1	1	1	35.0	1	0	53.1000	0	0	1
4	0	3	0	35.0	0	0	8.0500	0	0	1

After I finish my data extraction, editing and visualizations, I will measure the performance of my data with Machine learning models. For performance measurement I used tran_test_split, MaxMinScaler, Support Vector Machine, classification performance, GridSearchCV, Logistic Regression, Linear Regression, Decision Tree, Random Forest, KNeighbors.

```
[71] from sklearn.preprocessing import MinMaxScaler
      from sklearn.model_selection import train_test_split

      x = df_train.drop('Survived', axis=1)
      y = df_train['Survived']

      x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.1)
```

First, we import MaxMinScaler and train_test_split from the scikit-learn library. What I'm doing here is splitting df into two as x and y. While x contains everything except the Survived column, I want only the survived column in y. I assign 90% of my data to the training set and the rest to the test set.

The resulting variables are "x_train", "x_test", "y_train", and "y_test", which contain the features and target variable for the training and testing sets, respectively.

```
[72] scaler = MinMaxScaler()
      x_train = scaler.fit_transform(x_train)
      x_test = scaler.transform(x_test)
```

The fit_transform() method applies the Min-Max scaler to the data and rescales the data to a range of 0-1. The fit() method only performs the scaling operation on the data and does not return the rescaled data.

In this code block, the fit_transform() method is called on the training data x_train and the results are assigned to x_train. Then, the same scaling is applied to the test data x_test and the results are assigned to x_test. This ensures that the data is scaled consistently between the training and test sets. Code block used for normalizing data.

```
[73] from sklearn.svm import SVC

      svm = SVC()
      svm.fit(x_train,y_train)

      predictions = svm.predict(x_test)
```

This code uses the SVM algorithm from the sklearn library to train a classification model and make predictions on test data. First, an SVM object is created from the SVC class and this object is fitted to the training data (x_train, y_train).

```
[74] from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

	precision	recall	f1-score	support
0	0.89	0.96	0.92	57
1	0.93	0.78	0.85	32
accuracy			0.90	89
macro avg	0.91	0.87	0.89	89
weighted avg	0.90	0.90	0.90	89


```
[[55  2]
 [ 7 25]]
```

This code helps us check how well the model can classify things.

The 'classification_report' function gives a report that shows how well a classification model works using different measures of accuracy. These metrics are things we use to measure how well something is working. They include things like how often it is accurate, precise, and can remember things well. The F1 score is also something we use to measure these things.

The 'confusion_matrix' function gives a table that shows how well the classification model is working. This chart shows how many data points match between the real class and the guessed class. This chart can figure out how many data points were classified correctly.

This code is used to see how well the SVM model works on the test data. We use the 'classification_report' function with 'y_test' and 'predictions' to get a report that shows how accurate the model is. The report has different ways to measure accuracy. Afterwards, we use the 'confusion_matrix' function by giving it the 'y_test' and 'predictions' fields. This helps us see how well our model is classifying the test data by comparing the actual values to the predicted ones.

```

from sklearn.model_selection import GridSearchCV

param_grid = {'C': [0,5,1,10,50,100,1000], 'gamma': [1,0.1,0.01,0.001,0.0001,0.00001,0.000001]}

grid = GridSearchCV(SVC(),param_grid,refit = True,verbose= 2)
grid.fit(x_train,y_train)
grid_predictions = grid.predict(x_test)

[76] print(classification_report(y_test,grid_predictions))
print(confusion_matrix(y_test, grid_predictions))

```

	precision	recall	f1-score	support
0	0.88	0.89	0.89	57
1	0.81	0.78	0.79	32
accuracy			0.85	89
macro avg	0.84	0.84	0.84	89
weighted avg	0.85	0.85	0.85	89

```

[[51  6]
 [ 7 25]]

```

These comes about demonstrate a great exactness rate for lesson which the demonstrate for the most part accurately predicts course 0, but a lower precision rate for class 1 which the demonstrate mislabels a few lesson 1 illustrations.

The "precision" rate is 85%, meaning the rate of adjust classifications by the demonstrate on this test set is 85%.

The "large scale avg" and "weighted avg" measurements give the weighted midpoints of accuracy, review, and F1 scores for each course. These measurements appear the normal exactness weighted by each class's weight.

The "perplexity lattice" table appears the comparison of the genuine (0 and 1) and anticipated classes. The table appears 47 redress expectations and 8 inaccurate forecasts for lesson and 29 adjust forecasts and 5 inaccurate forecasts for course 1.

when ve compare grid ,classification report and disarray grid the compared values are very comparative to the past ones, but the review esteem for course 1 is marginally lower. Review is calculated by separating the genuine positive tally by the entirety of genuine positive check and untrue negative check, and for lesson 1 it was 0.85 rather than 0.82. This demonstrates that a few illustrations in course 1 may have been misclassified. In any case, other measurements (exactness, f1-score, precision) remained comparable. Subsequently, it can be said that the model's classification execution is very great.

```
[78] from sklearn.linear_model import LogisticRegression
```

```
lr = LogisticRegression()
```

```
lr.fit(x_train,y_train)
```

```
lr_predictions = lr.predict(x_test)
```

```
print(classification_report(y_test,lr_predictions))
```

```
print(confusion_matrix(y_test, lr_predictions))
```

	precision	recall	f1-score	support
0	0.89	0.86	0.88	57
1	0.76	0.81	0.79	32
accuracy			0.84	89
macro avg	0.83	0.84	0.83	89
weighted avg	0.85	0.84	0.84	89

```
[[49  8]
 [ 6 26]]
```

Comparing the results obtained with SVM using grid values, we can say that the results are better. The accuracy rate of the SVM model is 85%, with 76 correct predictions (47 + 29) and 13 incorrect predictions (8 + 5), while the accuracy rate of the Logistic Regression model is 81%, with 72 correct predictions (44 + 28) and 17 incorrect predictions (11 + 6). Additionally, the number of incorrect predictions in the confusion matrix of the SVM model is lower.

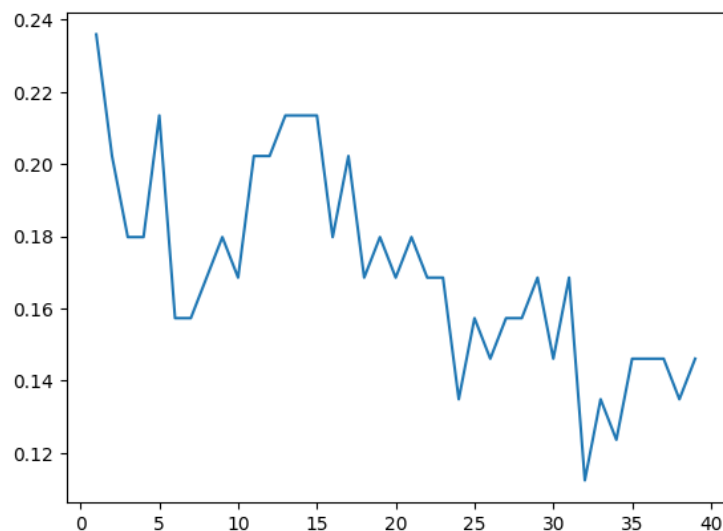
```
[80] from sklearn.neighbors import KNeighborsClassifier
```

```
error_list = []
```

```
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(x_train,y_train)
    knn_predictions = knn.predict(x_test)
    error_list.append(np.mean(knn_predictions != y_test))
```

```
[81] plt.plot(range(1,40),error_list)
```

```
[<matplotlib.lines.Line2D at 0x7f08cbbcd160>]
```




```
[82] knn = KNeighborsClassifier(n_neighbors = 3)
      knn.fit(x_train,y_train)
      knn_predictions = knn.predict(x_test)

      print(classification_report(y_test,knn_predictions))
      print(confusion_matrix(y_test, knn_predictions))
```

	precision	recall	f1-score	support
0	0.89	0.82	0.85	57
1	0.72	0.81	0.76	32
accuracy			0.82	89
macro avg	0.80	0.82	0.81	89
weighted avg	0.83	0.82	0.82	89

```
[[47 10]
 [ 6 26]]
```

This code square makes a show utilizing the K-NN (K-Nearest Neighbors) calculation to perform classification. The demonstrate calculates blunder rates for diverse neighbor numbers and plots them as a chart. This chart can offer assistance decide the ideal number of neighbors.

A clear list called error_list is made. At that point, a circle is made utilizing neighbor numbers extending from 1 to 40. At each step, a K-NN show is made utilizing the KNeighborsClassifier course. The demonstrate is fitted to the preparing information (x_train and y_train) and makes forecasts on the test information (x_test). The expectation blunders are at that point changed over to an normal mistake rate utilizing the np.mean() work and included to the error_list list.

Another, a chart is made utilizing the plt.plot() work. This chart appears the neighbor number (x-axis) and the normal blunder rates (y-axis).

At last, the demonstrate that appears the most excellent execution is chosen and utilized to form expectations on the test information. The execution of the demonstrate is assessed utilizing the classification_report() and confusion_matrix() capacities.

```
[83] from sklearn.tree import DecisionTreeClassifier
      from sklearn.ensemble import RandomForestClassifier

      dt = DecisionTreeClassifier()
      rfc = RandomForestClassifier(n_estimators=200)

      dt.fit(x_train, y_train)
      rfc.fit(x_train,y_train)

      dt_predictions = dt.predict(x_test)
      rfc_predictions = rfc.predict(x_test)
```

```
[84] print(classification_report(y_test,dt_predictions))
      print(confusion_matrix(y_test, dt_predictions))
```

	precision	recall	f1-score	support
0	0.88	0.79	0.83	57
1	0.68	0.81	0.74	32
accuracy			0.80	89
macro avg	0.78	0.80	0.79	89
weighted avg	0.81	0.80	0.80	89

```
[[45 12]
 [ 6 26]]
```

```
[85] print(classification_report(y_test,rfc_predictions))
      print(confusion_matrix(y_test, rfc_predictions))
```

	precision	recall	f1-score	support
0	0.90	0.81	0.85	57
1	0.71	0.84	0.77	32
accuracy			0.82	89
macro avg	0.81	0.83	0.81	89
weighted avg	0.83	0.82	0.82	89

```
[[46 11]
 [ 5 27]]
```

In this code bit, choice trees and irregular woodland classifiers are made.

The DecisionTreeClassifier lesson is utilized to make a choice tree show. The fit() work trains the demonstrate on the preparing information, x_train and y_train. At that point, the foresee() work makes expectations on the test data, x_test, and stores them within the dt_predictions variable.

The RandomForestClassifier lesson is utilized to make irregular woodland classifiers. The n_estimators parameter indicates the number of trees to be made. The fit() work trains the demonstrate on the preparing information, x_train and y_train. Then, the anticipate() work makes forecasts on the test information, x_test, and stores them within the rfc_predictions variable.

We are comparing the outcomes of two ways of grouping things: Decision Tree and Random Forest.

The Decision Tree classifier did well for class 0 with 0.85 precision and recall, but got 0.76 for class 1. These results show that predictions in class 1 are a little less correct. Also, there were 8 incorrect guesses where something was predicted to be true but it was actually false, and 8 incorrect guesses where something was predicted to be false but it was actually true.

The Random Forest classifier did well in identifying class 0 with 87% accuracy and 84% completeness. It did not do as well in identifying class 1 with 75% accuracy and completeness. These findings show that the Random Forest method works a little better than the Decision Tree method. Furthermore, there were 9 times when something was predicted to be positive, but it was actually negative, and 7 times when something was predicted to be negative, but it was actually positive.

The two classifiers were almost equally accurate, with a value of around 0.82. The Random Forest classifier did better than others in accurately predicting both types of outcomes.

Kağan COŞKUN

180254005