



Titanic - Group A

Machine Learning from Disaster

Data science and Artificial Intelligence Society



Project Proposal

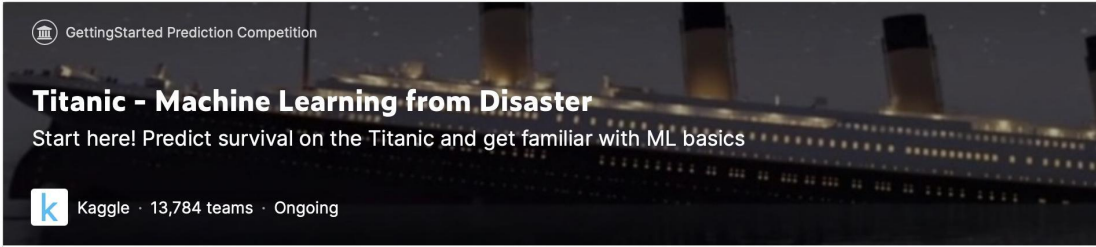
Topic: Titanic - Machine Learning from Disaster

Description: Build fundamentals in data science with titanic case from Kaggle & share insights learned

Expected Duration: 4 weeks

Team Member: Donggu Lee, Taekyung Lee, Hannah Lee, Sieun Park

Additional Deliverables



GettingStarted Prediction Competition

Titanic - Machine Learning from Disaster

Start here! Predict survival on the Titanic and get familiar with ML basics

Kaggle · 13,784 teams · Ongoing


Overview Data Code Discussion Leaderboard Rules

New Notebook ...

Filters

All Your Work Shared With You Bookmarks

Most Votes



Titanic Data Science Solutions
Updated 3Y ago
2034 comments · Titanic - Machine Learning from Disaster

8334

Gold

Referred to the Kaggle
Code with the Most Votes

Additional Deliverables



이번에는 캐글의 입문자를 위한 튜토리얼 문제라고 할 수 있는 Titanic: Machine Learning from Disaster의 예측 모델을 python으로 풀어보는 과정에 대해서 포스트를 할 것이다.

해당 포스팅에서 사용한 코드는 [여기](#)에서도 확인 할 수 있다.

데이터 분석 & 머신러닝

Kaggle(캐글) Titanic(타이타닉) 생존자 예측

컴공 K회 : 2021. 2. 4. 18:20



(주)피데스코리아 태블릿포스

25년 전문회사,태블릿포스,모바일포스,POS프로그램,관리비 용지무료,특가할인

Kaggle의 대표적인 문제 중 하나인 타이타닉 생존자 예측을 [Manav Sehgal](#)의 solution을 통해 정리해보았다.

github에 Kaggle 타이타닉 생존자 예측 관련
주피터 노트북과 데이터셋을 올려두었다.

▶ [github 링크](#) ◀

Titanic 생존자 예측

타이타닉 호 침몰 사건 당시의 사망자와 생존자를 구분하는 요인 분석을
통해, 승객들의 생존 여부를 예측

Utilized Tech Blog posts for appliances & feedback during weekly team meetings

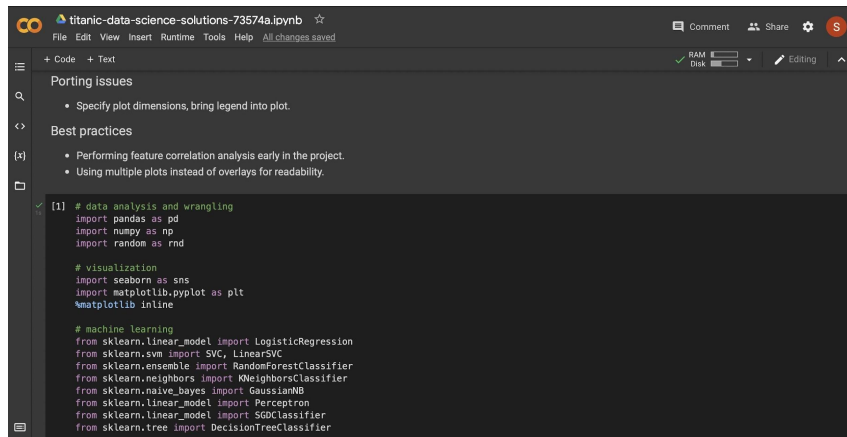


Additional Deliverables

Weekly Team Meeting (Every Tuesday)

52 lines of code – 13 lines every week

Shared useful documents via Google Drive



```
titanic-data-science-solutions-73574a.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
RAM 8 GB
Disk 100%
Editing

Porting issues
• Specify plot dimensions, bring legend into plot.

Best practices
• Performing feature correlation analysis early in the project.
• Using multiple plots instead of overlays for readability.

[1] # data analysis and wrangling
import pandas as pd
import numpy as np
import random as rnd

# visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

Describing Data

- Features

- Able to notice feature names

```
print(train_df.columns.values)
```

- Data type

- Which data is categorical (Survived, Sex, Embarked, Pclass) or numerical (Age, Fare)
- which data is float, int, or object

```
print(train_df.tail())  
train_df.info()
```

- Finding errors

- Are there any empty values

- Assumption

- based on the data, we can make assumption that which features are useful for solving the problem.

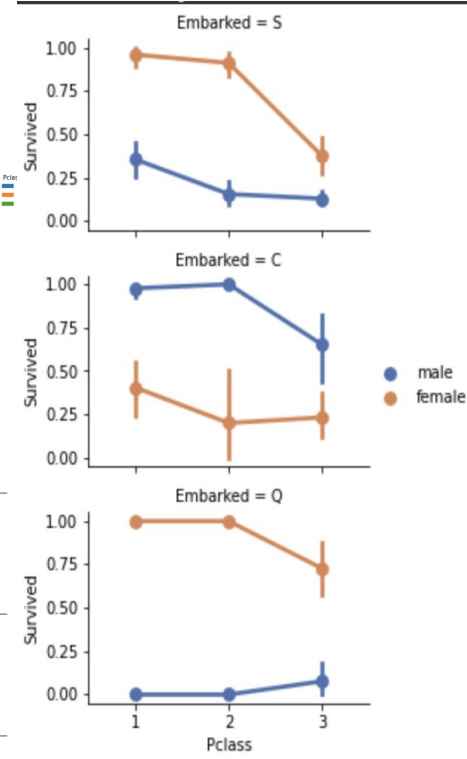
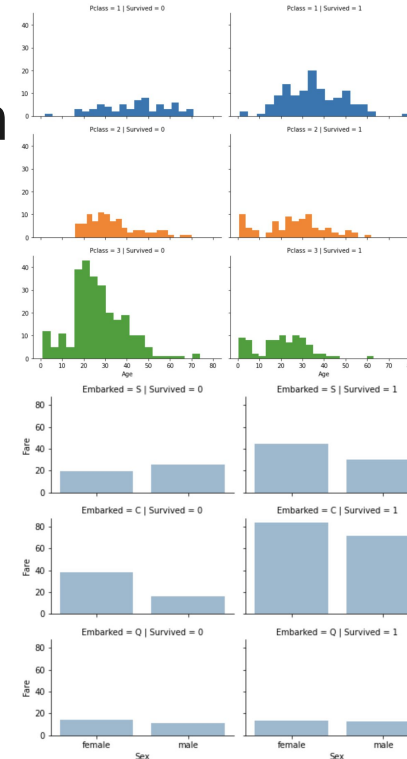
```
print(train_df.describe())  
train_df.describe(include=['O'])
```

```
[ 'PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'  
  'Ticket' 'Fare' 'Cabin' 'Embarked']  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 418 entries, 0 to 417  
Data columns (total 11 columns):  
#   Column             Non-Null Count  Dtype    
---  ---               
0   PassengerId         418 non-null    int64    
1   Pclass              418 non-null    int64    
2   Name                418 non-null    object    
3   Sex                 418 non-null    object    
4   Age                 332 non-null    float64   
5   SibSp               418 non-null    int64    
6   Parch               418 non-null    int64    
7   Ticket              418 non-null    object    
8   Fare                417 non-null    float64   
9   Cabin               91 non-null     object    
10  Embarked            418 non-null    object    
dtypes: float64(2), int64(4), object(5)  
memory usage: 36.0+ KB  
  
   PassengerId  Survived  Pclass  ...    SibSp  Parch    Fare  
count  891.000000   891.000000   891.000000  ...    891.000000   891.000000   891.000000  
mean    446.000000     0.383838     2.308642  ...     0.523008     0.381594    32.204208  
std    257.353842     0.486592     0.836071  ...     1.102743     0.806057    49.693429  
min      1.000000     0.000000     1.000000  ...     0.000000     0.000000     0.000000  
25%    223.500000     0.000000     2.000000  ...     0.000000     0.000000     7.910400  
50%    446.000000     0.000000     3.000000  ...     0.000000     0.000000    14.454200  
75%    668.500000     1.000000     3.000000  ...     1.000000     0.000000    31.000000  
max    891.000000     1.000000     3.000000  ...     8.000000     6.000000   512.329200
```

Line 14 - 16: Data Correlation

```
#grid = sns.FacetGrid(train_df, col = 'Pclass', hue = 'Survived')
grid = sns.FacetGrid(train_df, col = 'Survived', row = 'Pclass', hue = 'Pclass', height = 3, aspect = 1.6)
grid.map(plt.hist, 'Age', alpha = 1, bins = 20)
grid.add_legend();
```

- Visualizing the relationship between given data
- Check the correlation between features when predicting survivals.
- Decide which features are not needed.





Line 17: Data Drop

```
print("Before", train_df.shape, test_df.shape)
```

```
# remove column(axis=1)
```

```
train_df = train_df.drop(['Ticket', 'Cabin'], axis='columns')
```

```
test_df = test_df.drop(['Ticket', 'Cabin'], axis='columns')
```

```
combine = [train_df, test_df]
```

```
print("After", train_df.shape, test_df.shape)
```

Before (891, 12) (418, 11)

After (891, 10) (418, 9)

- Check how the datasets are organized (891 Line with 12 columns).
- Dropping the data that is not useful for predicting survivals.
- Check how the datasets had changed.

Line 18: New Variable

Sex	female	male
Title		
Capt	0	1
Col	0	2
Countess	1	0
Don	0	1
Dr	1	6
Jonkheer	0	1
Lady	1	0
Major	0	2
Master	0	40
Miss	182	0
Mlle	2	0
Mme	1	0
Mr	0	517
Mrs	125	0
Ms	1	0
Rev	0	6
Sir	0	1

```
for dataset in combine:
    dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=False)

pd.crosstab(train_df['Title'], train_df['Sex'])
```

- Extracted `title` from `Name` and renamed it `Title`
- Extracted entities those start with blank space and end with "." and saved it under the new variable `Title`
- Utilized crosstab function for analysis on distributions among `Title` and `Sex`
- "Miss", "Mr", and "Mrs" were the most common titles among many other titles.

Line 19: Simplification

```
for dataset in combine:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col',\
        'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')

    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

train_df[['Title', 'Survived']].groupby(['Title'], as_index=False).mean()
```

	Title	Survived
0	Master	0.575000
1	Miss	0.702703
2	Mr	0.156673
3	Mrs	0.793651
4	Rare	0.347826

- For convenience sake, variables with low frequencies have been integrated as “Rare” ('Lady', 'Countess','Capt', 'Col', 'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona' → 'Rare')
- Simplified titles those indicate female ('Mlle', 'Ms' → Miss / 'Mme' → 'Mrs')
- Title: Master, Miss, Mr, Mrs, Rare
- Extracted survival rate by sex groups
- Survival rate in descending order: Mrs > Miss > Master > Rare > Mr
- Simple analysis that female had higher chances in survival

Line 20: Character variable → Numeric variable

```
title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}
for dataset in combine:
    dataset['Title'] = dataset['Title'].map(title_mapping)
    dataset['Title'] = dataset['Title'].fillna(0)

train_df.head()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	Title	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C	3
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S	2
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	S	3
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	S	1

- Title → numeric
 - If NaN, Title = 0

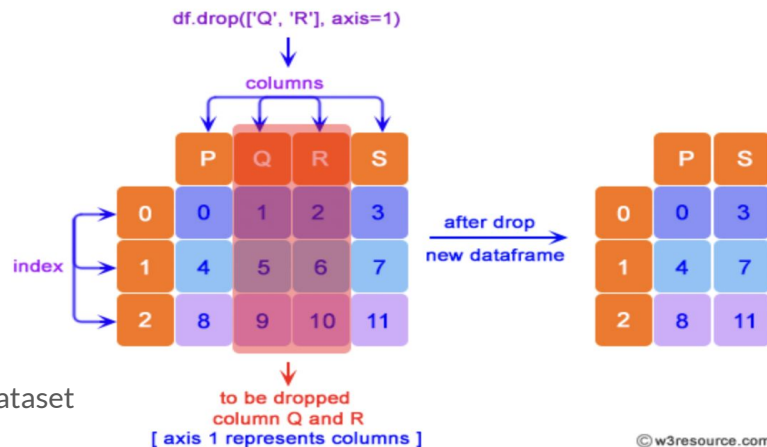
Line 21: Data Drop (Name, PassengerID 제거)

```
train_df = train_df.drop(['Name', 'PassengerId'], axis=1)
test_df = test_df.drop(['Name'], axis=1)
combine = [train_df, test_df]
train_df.shape, test_df.shape
```

Before (891, 12) (418, 11)

After (891, 10) (418, 9)

- **Drop function:** Dropped entities under `Name` & `PassengerID` from each dataset
- **Combine Dataset:** combine train and test datasets
- **Shape function:** Returns the shape of the array (ex. 891 Lines 10 Columns)





Line 22: Converting a Categorical Feature

```
for dataset in combine:
    dataset['Sex'] = dataset['Sex'].map( {'female': 1, 'male': 0} ).astype(int)

train_df.head()
```

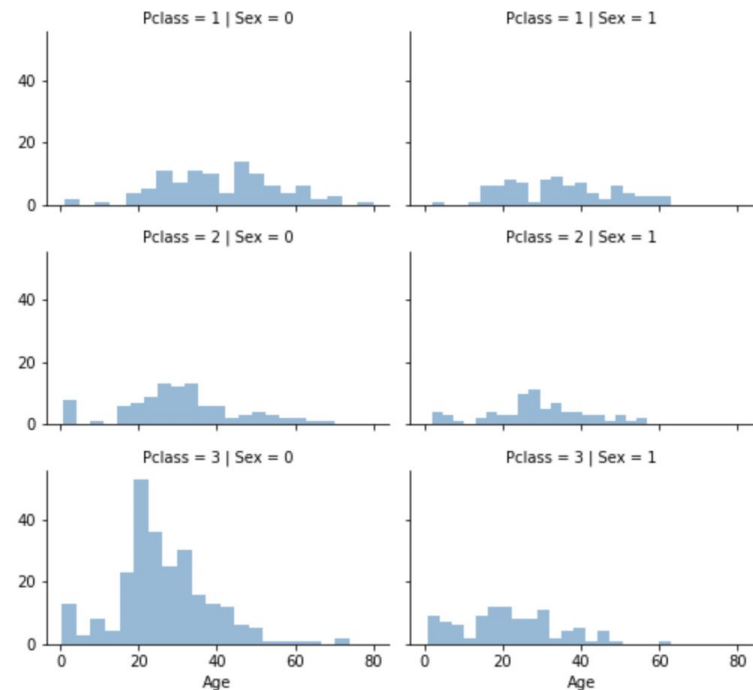
	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22.0	1	0	7.2500	S	1
1	1	1	1	38.0	1	0	71.2833	C	3
2	1	3	1	26.0	0	0	7.9250	S	2
3	1	1	1	35.0	1	0	53.1000	S	3
4	0	3	0	35.0	0	0	8.0500	S	1

- **For loop:** Converted Sex variable within Combined data set (train + test) into data type of int and visualized it via chart
- **head function:** return the first n rows to verify if the function processed correctly

Line 23: Pclass and Sex Visualization

```
grid = sns.FacetGrid(train_df, row='Pclass', col='Sex', height=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend()
```

- **sns.Facetgrid()** function: utilized in demand of various plot views
 - Aspect * Height : Gives the width of each facet in inches
- **grid.map()** function: Histogram conversion
- **grid.add_legend()** function: Explain Pclass & Sex variables





Line 24 - 25: Using For Loop Iteration (Sequence)

```
guess_ages = np.zeros((2,3))  
guess_ages
```

Prepare an empty **array** that contains age values based on Pclass and Gender

```
age_guess = guess_df.median()  
  
# Convert random age float to nearest .5 age  
guess_ages[i,j] = int( age_guess/0.5 + 0.5 ) * 0.5  
  
for i in range(0, 2):  
    for j in range(0, 3):  
        dataset.loc[ (dataset.Age.isnull()) & (dataset.Sex == i) & (dataset.Pclass == j+1),\  
                    'Age'] = guess_ages[i,j]  
  
        dataset['Age'] = dataset['Age'].astype(int)  
  
train_df.head()
```

Use **for** loop for iterating over a sequence of Sex (0 or 1) and Pclass (1, 2, 3)

Calculate guessed values of **age** for the six combinations



Line 26: Age Bands

```
train_df['AgeBand'] = pd.cut(train_df['Age'], 5)
train_df[['AgeBand', 'Survived']].groupby(['AgeBand'], as_index=False).mean().sort_
_values(by='AgeBand', ascending=True)
```

	AgeBand	Survived
0	(-0.08, 16.0]	0.550000
1	(16.0, 32.0]	0.337374
2	(32.0, 48.0]	0.412037
3	(48.0, 64.0]	0.434783
4	(64.0, 80.0]	0.090909

Age band is the **range** of ages that determines the possible amount for each participant

Using age bands, we determine **correlation** with survived.

Next week, we'll replace age with ordinal based on these sets.

Line 27: Continuous Variable → Categorical Variable

```
for dataset in combine:
    dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3
    dataset.loc[ dataset['Age'] > 64, 'Age']
train_df.head()
```

Conversion of Continuous Variable `Age` into Categorical Variable within corresponding AgeBand

	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Title	AgeBand
0	0	3	0	1	1	0	A/5 21171	7.2500	NaN	S	1	(16.0, 32.0]
1	1	1	1	2	1	0	PC 17599	71.2833	C85	C	3	(32.0, 48.0]
2	1	3	1	1	0	0	STON/O2. 3101282	7.9250	NaN	S	2	(16.0, 32.0]
3	1	1	1	2	1	0	113803	53.1000	C123	S	3	(32.0, 48.0]
4	0	3	0	2	0	0	373450	8.0500	NaN	S	1	(32.0, 48.0]



Line 28: Drop AgeBand Variable

```
train_df = train_df.drop(['AgeBand'], axis=1)
combine = [train_df, test_df]
train_df.head()
```

- Drop AgeBand variable

	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Title
0	0	3	0	1	1	0	A/5 21171	7.2500	NaN	S	1
1	1	1	1	2	1	0	PC 17599	71.2833	C85	C	3
2	1	3	1	1	0	0	STON/O2. 3101282	7.9250	NaN	S	2
3	1	1	1	2	1	0	113803	53.1000	C123	S	3
4	0	3	0	2	0	0	373450	8.0500	NaN	S	1

Line 29: Integrate two numeric variables

```
for dataset in combine:
    dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1

train_df[['FamilySize', 'Survived']].groupby(['FamilySize'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

FamilySize	Survived	
3	4	0.724138
2	3	0.578431
1	2	0.552795
6	7	0.333333
0	1	0.303538
4	5	0.200000
5	6	0.136364
7	8	0.000000
8	11	0.000000

- Aggregated sum of SibSp (Head count of Sibling & Spouse), Parch (Head count of Parent & Child) and 1 (individuals) into variable `FamilySize`
- Family of 4 had relatively higher rate of survival
- Single individual had 30% of survival rate



Line 30- 31: Combining Existing Features (IsAlone)

```
for dataset in combine:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1

train_df[['IsAlone', 'Survived']].groupby(['IsAlone'], as_index=False).mean()
```

```
train_df = train_df.drop(['Parch', 'SibSp', 'FamilySize'], axis=1)
test_df = test_df.drop(['Parch', 'SibSp', 'FamilySize'], axis=1)
combine = [train_df, test_df]

train_df.head()
```

- After creating FamilySize, we create another feature (**IsAlone**)
- Drop Parch, SibSp, and FamilySize features in favor of **IsAlone**.

Line 33-34: Data Replacement

```
freq_port = train_df.Embarked.dropna().mode()[0]
freq_port

for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].fillna(freq_port)

train_df[['Embarked', 'Survived']].groupby(['Embarked'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

'S'

- By using dropna function, drop a null value in Embarked column.
- After dropping all the null values, check which value is the most common value in Embarked columns.

	Embarked	Survived
0	C	0.553571
1	Q	0.389610
2	S	0.339009

- By using fillna function, fill a null value with the most frequent value in Embarked columns
- After filling the dataset, check the survival rate of people on board at each port.

Line 35: Data Replacement

```
for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)

train_df.head()
```

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	0	3	0	1	7.2500	0	1	0	3
1	1	1	1	2	71.2833	1	3	0	2
2	1	3	1	1	7.9250	0	2	1	3
3	1	1	1	2	53.1000	0	3	0	2
4	0	3	0	2	8.0500	0	1	1	6

- Replacing a object data into int data in Embarked columns by using .map () function.
- Check the data had changed properly by using .head () function.

Line 36: Fare Feature

```
test_df['Fare'].fillna(test_df['Fare'].dropna().median(), inplace=True)
test_df.head()
```

- Substituted variable `Fare` by its median
- fillna() method: replaces the NULL values with a specified value
- dropna() method: removes the rows that contains NULL values
- inplace parameter: overwrite the existing data values

	PassengerId	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	892	3	0	2	7.8292	2	1	1	6
1	893	3	1	2	7.0000	0	3	0	6
2	894	2	0	3	9.6875	2	1	1	6
3	895	3	0	1	8.6625	0	1	1	3
4	896	3	1	1	12.2875	0	3	0	3

Line 37: Cutting the Data by Range

```
train_df['FareBand'] = pd.qcut(train_df['Fare'], 4)
train_df[['FareBand', 'Survived']].groupby(['FareBand'], as_index=False).mean().sort_values(by='FareBand', ascending=True)
```

	FareBand	Survived
0	(-0.001, 7.91]	0.197309
1	(7.91, 14.454]	0.303571
2	(14.454, 31.0]	0.454955
3	(31.0, 512.329]	0.581081

- Utilized pd.qcut() to Data Cut 'Fare' Column into Quartile
- Represented 'Survived' (Survival Rate) by its respective Range
- Wider the FareBand's range, higher the Survival rate

Line 38-39

```
for dataset in combine:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare'] = 2
    dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3
    dataset['Fare'] = dataset['Fare'].astype(int)

train_df = train_df.drop(['FareBand'], axis=1)
combine = [train_df, test_df]

train_df.head(10)

#And the test dataset**
test_df.head(10)
```

	PassengerId	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	892	3	0	2	0	2	1	1	6
1	893	3	1	2	0	0	3	0	6
2	894	2	0	3	1	2	1	1	6
3	895	3	0	1	1	0	1	1	3
4	896	3	1	1	1	0	3	0	3
5	897	3	0	0	1	0	1	1	0
6	898	3	1	1	0	2	2	1	3
7	899	2	0	1	2	0	1	0	2
8	900	3	1	1	0	1	3	1	3
9	901	3	0	1	2	0	1	0	3



Line 43

```
# Support Vector Machines  
  
svc = SVC()  
svc.fit(X_train, Y_train)  
Y_pred = svc.predict(X_test)  
acc_svc = round(svc.score(X_train, Y_train) * 100, 2)  
acc_svc
```

83.84

Support Vector Machines (SVM)

- supervised learning models used for classification and regression
- compute confidence scores for each method



Line 44

```
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
acc_knn
```

84.74

K-Nearest Neighbor (KNN)

- non-parametric method used for classification and regression



Line 45

```
# Gaussian Naive Bayes

gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)
Y_pred = gaussian.predict(X_test)
acc_gaussian = round(gaussian.score(X_train, Y_train) * 100, 2)
acc_gaussian
```

72.28

Gaussian Naive Bayes (Gaussian NB)

- non-probabilistic binary linear classifier
- lowest confidence score



Line 40: Get the Dataset

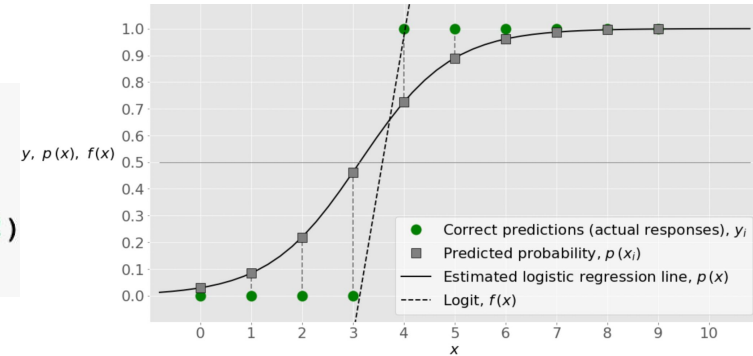
```
X_train = train_df.drop("Survived", axis=1)
Y_train = train_df["Survived"]
X_test  = test_df.drop("PassengerId", axis=1).copy()
X_train.shape, Y_train.shape, X_test.shape
```

((891, 8), (891,), (418, 8))

- Prepare necessary data for the model
- .drop() function → delete variable ("Survived")
- Y_train = Independent Variable ("Survived")
- X_test = Data to be predicted

Line 41: Logistic Regression

```
logreg = LogisticRegression()  
logreg.fit(X_train, Y_train)  
Y_pred = logreg.predict(X_test)  
acc_log = round(logreg.score(X_train, Y_train) * 100, 2)  
acc_log
```



- Logistic Regression: A model that analyzes probability by predicting the probability that data will fall into a category
- `logreg.fit()` function & `logreg.predict()` function: Fit the data in to the logistic regression and find the equation/probability
- Chances of categorization under Survived = 80.36%



Line 42: Logistic Regression

```
coeff_df = pd.DataFrame(train_df.columns.delete(0))
coeff_df.columns = ['Feature']
coeff_df["Correlation"] = pd.Series(logreg.coef_[0])
coeff_df.sort_values(by='Correlation', ascending=False)
```

- Calculate correlation in between the features
- Variable `sex` = Highest positive correlation
 - the most influential variable

	Feature	Correlation
1	Sex	2.201619
5	Title	0.397888
2	Age	0.287011
4	Embarked	0.261473
6	IsAlone	0.126553
3	Fare	-0.086655
7	Age*Class	-0.311069
0	Pclass	-0.750700

Line 43: Support Vector Machines

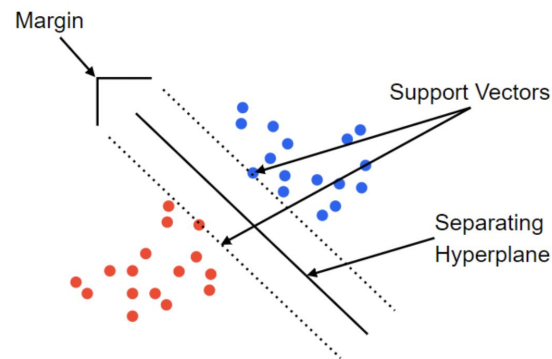
```
# Support Vector Machines

svc = SVC()
svc.fit(X_train, Y_train)
Y_pred = svc.predict(X_test)
acc_svc = round(svc.score(X_train, Y_train) * 100, 2)
acc_svc
```

83.84

Support Vector Machines (SVM)

- supervised learning models used for classification and regression
- design decision boundary
- compute confidence scores for each method and evaluate its performance
- multiply the score by 100 and round up to two decimal places



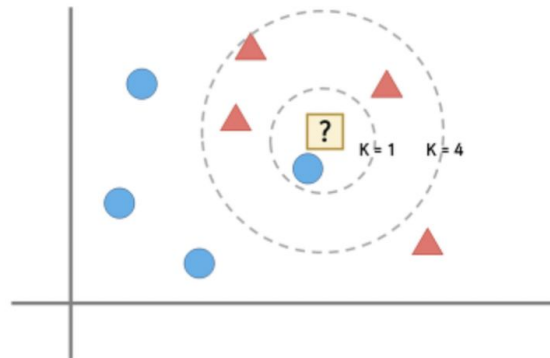
Line 44: K-Nearest Neighbor



```
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
acc_knn
```

84.74

K-Nearest Neighbor (KNN)

- non-parametric method used for classification and regression
- classify samples by majority vote of its neighbors
- assign samples to the class most common among its k nearest neighbors



K=1: ? = 
K=4: ? = 

Line 45: Gaussian Naive Bayes

```
# Gaussian Naive Bayes

gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)
Y_pred = gaussian.predict(X_test)
acc_gaussian = round(gaussian.score(X_train, Y_train) * 100, 2)
acc_gaussian
```

72.28

Gaussian Naive Bayes (Gaussian NB)

- non-probabilistic binary linear classifier
- family of simple probabilistic classifiers based on applying Bayes' theorem
- strong naive independence assumptions between the features
- lowest confidence score among six classification methods

$$P(y | x_1, \dots, x_n) = \frac{P(x_1 | y) P(x_2 | y) \dots P(x_n | y) P(y)}{P(x_1)P(x_2) \dots P(x_n)}$$

$$P(y | x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^n P(x_i | y)}{P(x_1)P(x_2) \dots P(x_n)}$$

$$P(y | x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i | y)$$

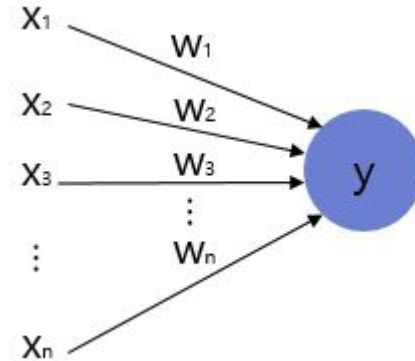
$$y = \operatorname{argmax}_y P(y) \prod_{i=1}^n P(x_i | y)$$

Line 46: Perceptron

```
# Perceptron

perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
Y_pred = perceptron.predict(X_test)
acc_perceptron = round(perceptron.score(X_train, Y_train) * 100, 2)
acc_perceptron
```

78.34



Perceptron

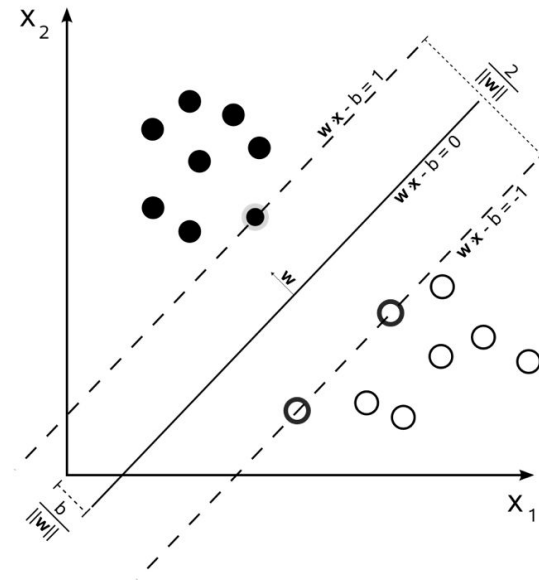
- It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

Line 47: Linear SVC

```
# Linear SVC
```

```
linear_svc = LinearSVC()  
linear_svc.fit(X_train, Y_train)  
Y_pred = linear_svc.predict(X_test)  
acc_linear_svc = round(linear_svc.score(X_train, Y_train) * 100, 2)  
acc_linear_svc
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:1208: ConvergenceWarning,  
79.12
```

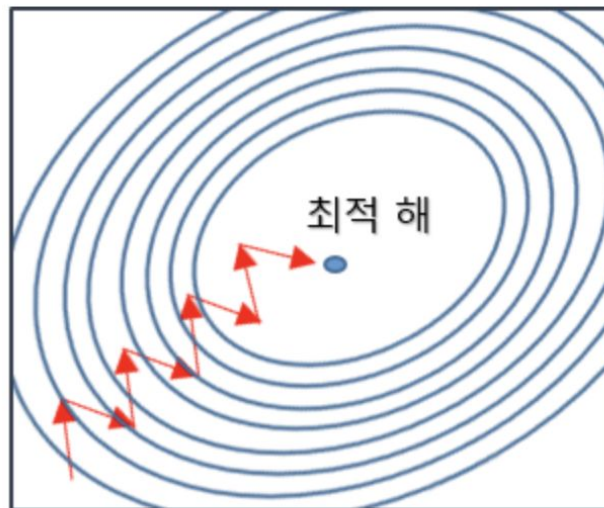


Line 48: Stochastic Gradient Descent

```
# Stochastic Gradient Descent
```

```
sgd = SGDClassifier()  
sgd.fit(X_train, Y_train)  
Y_pred = sgd.predict(X_test)  
acc_sgd = round(sgd.score(X_train, Y_train) * 100, 2)  
acc_sgd
```

77.33



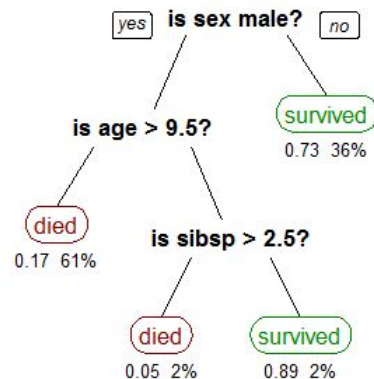
확률적 경사 하강법

Line 49: Decision Tree

Decision Tree

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)
Y_pred = decision_tree.predict(X_test)
acc_decision_tree = round(decision_tree.score(X_train, Y_train) * 100, 2)
acc_decision_tree
```

- Model confidence score is the **highest** among models evaluated
- Classification target variable can take a **finite** set of values
- Splits a set of data into smaller and smaller groups (called nodes)
- If subset of the data is split, predictions become more **accurate** if each of the resulting subgroups are similar than before

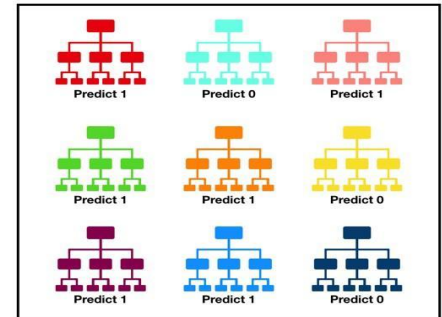
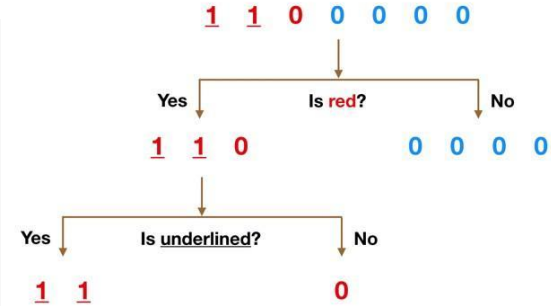


Line 50: Random Forest

```
# Random Forest
```

```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)
Y_pred = random_forest.predict(X_test)
random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
acc_random_forest
```

- Ensembled as a learning method for **classification regression** that operate by constructing multiple decision trees
- Also produced the model 's highest confidence score with **86.76**
- Random Forest is determined based on randomly selected subset of the data and randomly selected fields





Line 51: Model Evaluation

```
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree'],
    'Score': [acc_svc, acc_knn, acc_log,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_linear_svc, acc_decision_tree])
models.sort_values(by='Score', ascending=False)
```

- Both Decision Tree and Random Forest scores the **same**
- Choose Random Forest as they correct for decision trees' habit of **overfitting** to their training set

	Model	Score
3	Random Forest	86.76
8	Decision Tree	86.76
1	KNN	84.74
0	Support Vector Machines	83.84
2	Logistic Regression	80.36
7	Linear SVC	79.12
6	Stochastic Gradient Decent	78.56
5	Perceptron	78.00
4	Naive Bayes	72.28



Conclusion

- Comparing the accuracy of the different models, Random Forest and Random Tree is the best amongst classification models with 86.76 percent accuracy
- Random Forest model highlights the importance of predictors (Sex, Pclass, Fare, Age)
- After analyzing all models, we can conclude that predictors Sex, Pclass, and Age played the biggest role for Titanic survivors.



Lessons learned..

- Problem solving approach by using Machine Learning
- Application of various Machine Learning & Data Visualization methodologies
- Hands-on experience of many Python libraries (numpy, pandas, seaborn, matplotlib, scikit learn)
- Collaborative teamwork ! :)



Thank you!