ML_Term_Project.docx

by Eray Kütük

Submission date: 26-May-2024 08:50PM (UTC+0300)
Submission ID: 2388324376
File name: ML_Term_Project.docx (2.5M)
Word count: 5970
Character count: 34370

Enes Aydın & Eray Kütük

040200262 & 040200205

Machine Learning for Signal Processing

Topic: Titanic - Machine Learning from Disaster

CONTENTS

Solution of MANAV SEHGAL

Problem Definition: Given a dataset with information on passengers who survived or perished in the Titanic disaster, can our model predict whether passengers in a separate test dataset, lacking survival data, survived or not.

1. Libraries and Accessing Data

a. Libraries

```
# 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
```

The necessary libraries were imported.

b. Accessing Data

```
train_df = pd.read_csv('../input/train.csv')
test_df = pd.read_csv('../input/test.csv')
combine = [train_df, test_df]
```

There are two datasets containing similar passenger information such as name, age, gender, socio-economic class, etc. One is labeled as train.csv and the other as test.csv.Train.csv comprises information on a subset of the passengers (specifically 891 individuals) aboard the Titanic, crucially indicating whether they survived or not, referred to as the 'ground truth'. The test.csv dataset holds analogous information but doesn't provide the 'ground truth' for each passenger. The aim is to forecast these outcomes.

Utilizing the patterns identified in the train.csv data, anticipate the survival status of the remaining 418 passengers aboard (presented in test.csv).

2. Data Information

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

```
['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

First of all, let's examine the data in our train data. There are 12 different columns in the train dataset. Survived, Sex, Embarked and Class columns are categorical data. In addition, the Pclass column contains categorical ordinal data. The other columns that are Age, Fare, SibSp and Parch are numeric data. Age and Fare are continuous numeric data; SibSp and Parch are discontinuous numeric data.

For example, we can see the first 5 lines in the Picture2.1 and the last 5 lines in the Picture2.2.

```
train_df.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
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Picture2.1

```
train_df.tail()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

Picture2.2.

When we examine the 2 pictures, we see that the ticket and cabin columns are of mixed data type. It contains both numeric and alphanumeric characters. it also turns out that the PassengerId column is an index, and the Name column has a string expression. From a negative point of view, some data are empty or null. We'll have to look into these. We can look at the number of non-empty data when we run the code line that is train_df.info ().

```
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
```

Table2.1

3. Distribution of Feature Values

a. Distribution Numerical Features

In the solution I examined, Manav Sehgal drew the following conclusions for the numerical features in the data.

- The total of 891 samples represents 40% of the actual passenger count on the Titanic, which was 2,224.
- Survived is a categorical variable that takes on the values 0 or 1.
- Approximately 38% of the samples survived, which closely reflects the actual survival rate of 32%.
- The majority of passengers (over 75%) did not travel with either parents or children.
- Almost 30% of the passengers had siblings and/or a spouse on board.
- Fares varied greatly, with a small number of passengers (less than 1%) paying as much as $512.
- There were few elderly passengers (less than 1%) in the age range of 65 to 80.

On the other hand, we can obtain information by using the describe() function.

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Output of train_df.describe()

b. Distribution Categorical Features

Similarly, Manav Sehgal made some inferences for categorical data. These inferences:

- Each name in the dataset is unique, with a total count of 891.
- The "Sex" variable, with two potential values, is predominantly male, representing 65% of the dataset.
- There are multiple instances of duplicate cabin values across samples, indicating that several passengers shared a cabin.
- Embarked offers three potential values, with the majority of passengers utilizing the S port.
- The Ticket feature displays a high ratio (22%) of duplicate values, with a total of 681 unique entries.

In addition, we can obtain information by using the describe(include=['0']) function.

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Lesurer, Mr. Gustave J	male	1601	G6	S
freq	1	577	7	4	644

Output of train_df.describe(include=['0'])

4. Evaluating Features and Correlation

To validate certain observations and assumptions, we can promptly examine the correlations between features by pivoting them against each other. However, this analysis is feasible only for features without any missing values. Moreover, it's logical to conduct this analysis solely for features categorized as either categorical (e.g., Sex), ordinal (e.g., Pclass), or discrete (e.g., SibSp, Parch).

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

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

Output1

According to the output on the side, we can decipher that there is an important correlation between the "PClass" and the "Survived" columns. The survival average of the "Pclass" column in the data with 1 is 0.629630. On the other hand, where the value is 3, it is 0.242363. In other words, a low value of "Pclass" that is ordinal categorical feature increases the chances of survival.

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

	Sex	Survived
0	female	0.742038
1	male	0.188908

it is obvious that there is a strong correlation when we look at the correlation relationship between men and women. if we look at the table on the side, it seems that the survival rate of female is 3 times compared to male.

Output2

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

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

	SibSp	Survived
1	1	0.535885
2	2	0.464286
0	0	0.345395
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

SibSp and Parch features exhibit zero correlation for certain values. It might be optimal to derive a feature or a set of features from these individual attributes. For instance, by combining SibSp and Parch features, we could create a new attribute representing family size. This could offer a more meaningful insight into whether each passenger traveled with their family or not.

	Parch	Survived
3	3	0.600000
1	1	0.550847
2	2	0.500000
0	0	0.343658
5	5	0.200000
4	4	0.000000
6	6	0.000000

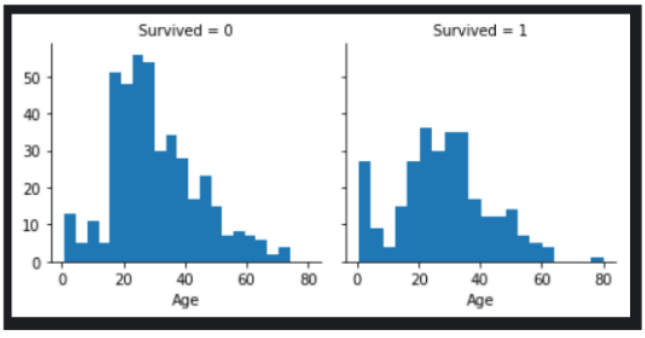
Output3

Output4

5. Visualizing Data

a. Correlation of Numerical Features

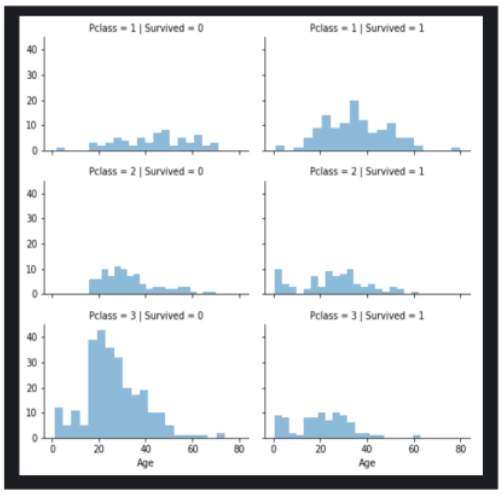
```
g = sns.FacetGrid(train_df, col='Survived')
g.map(plt.hist, 'Age', bins=20)
```



Graph5.a.1

In this part, creating a histogram can be useful for understanding the correlation between numerical features (such as age) and the survival goal. This helps us analyze the distribution of continuous variables like age. Based on our observations, we note that infants (up to 4 years old) had a high survival rate, the oldest passengers (at 80 years old) survived, and a significant number of passengers aged between 15-25 did not survive. This simple analysis confirms that we should consider the age feature in our model training, complete missing values, and create age groups.

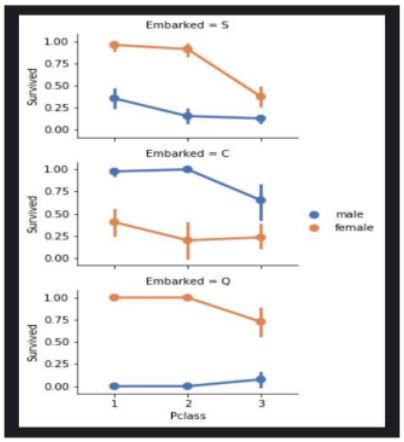
b. Correlation of Numerical and Ordinal Features



Graph5.b.1

To determine the relationships between numerical and ordinal features we can obtain correlations by combining multiple properties into a single graph. Although Pclass=3 had the most passengers, the majority of them did not survive. The majority of the baby passengers in Pclass=2 and Pclass=3 survived. Most passengers in Pclass =1 survived. These results confirm our classification assumption. Pclass differs in the age distribution of passengers. it is crucial that Consider Pclass for model training. Pclass is an important feature to improve the accuracy of the model. In particular, attention should be paid to the low survival rate of Pclass=3 and the high survival rate of Pclass=1. This information can help the model make more effective predictions.

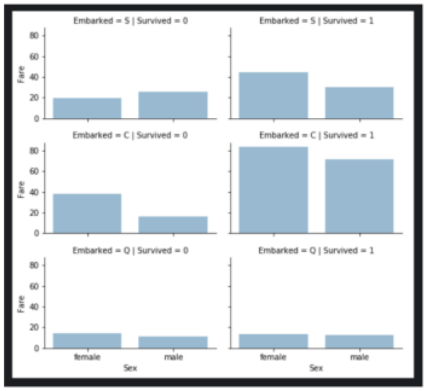
c. Correlation of Categorical Features



Graph5.c.1

Despite female passengers generally having a higher survival rate compared to males, there is an exception in Embarked=C where males have a higher survival rate, suggesting a potential correlation between Pclass and Embarked, and consequently between Pclass and Survived rather than a direct correlation between Embarked and Survived. Additionally, it is noteworthy that males in Pclass=3 have a better survival rate in ports C and Q compared to Pclass=2. Considering these observations, the model should include the Sex feature and the Embarked feature for training.

d. Correlation of Categorical and Numerical Features



Graph5.d.1

Understanding correlations between categorical non-numeric features and numerical features is crucial. For instance, we may consider correlating Embarked (categorical non-numeric), Sex (categorical non-numeric), and Fare (numerical continuous) features with Survived (categorical numeric). It has been observed that passengers who paid higher fares generally had a higher survival rate, validating the assumption to create fare ranges. Additionally, the port of embarkation seems to be associated with survival rates, supporting the correlation and completion of these features. In light of these observations, it is essential to consider binning the Fare feature for model training. This can enhance the model's predictive performance by capturing the potential impact of specific fare ranges on survival.

6. Feature engineering

Feature engineering involves transforming raw data into informative features to enhance machine learning model performance. It includes tasks like handling missing values, encoding categorical variables, creating new features, and selecting the most relevant ones for modeling. Effective feature engineering improves model accuracy and interpretability, ultimately contributing to the success of machine learning projects.

a. Dropping Features

Dropping features is an effective strategy to streamline data analysis by reducing the number of data points and speeding up processing. In accordance with our assumptions and decisions, we aim to drop the Cabin and Ticket features. It's important to note that when applicable, these operations should be applied consistently to both training and testing datasets to maintain coherence.

```
print("Before", train_df.shape, test_df.shape, combine[0].shape, combine[1].shape)

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

"After", train_df.shape, test_df.shape, combine[0].shape, combine[1].shape
```

```
Before (891, 12) (418, 11) (891, 12) (418, 11)
('After', (891, 10), (418, 9), (891, 10), (418, 9))
```

Input6.a.1 and Output6.a.1

b. Creating New Feature Extracting from Existing

Before dropping the Name and PassengerId features, we aim to analyze whether the Name feature can be engineered to extract titles and investigate the correlation between titles and survival. Using regular expressions, we extract the Title feature by matching the first word that ends with a dot character within the Name feature. Observations from plotting Title, Age, and Survived indicate that most titles accurately bandAge groups, with certain titles showing variations in survival rates. For example, titles like Mme, Lady, and Sir mostly survived, while others like Don, Rev, and Jonkheer did not. Based on these observations, we decide to retain the new Title feature for model training.

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

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

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

Input6.b.1 and Output6.b.1

We have the option to substitute numerous titles with a more frequently occurring name or categorize them as "Rare".

```
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()
```

Input6.b.2

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

Output6.b.2

We have the possibility to transform the categorical titles into ordinal values.

```
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	0.0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C	0.0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S	0.0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	S	0.0
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	S	0.0

Input6.b.3 and Output6.b.3

Now, we can confidently remove the Name feature from both the training and testing datasets. Additionally, the PassengerId feature is unnecessary in the training dataset and can also be dropped.

```
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
```

```
((891, 9), (418, 9))
```

Input6.b.4 and Output6.b.4

c. Converting Categorical Feature

We can now proceed to convert string-type features into numerical values, a necessary step for most model algorithms. This conversion will also aid us in fulfilling the goal of completing features. To begin, we'll transform the Sex feature into a new feature named Gender, where female is represented as 1 and male as 0. This conversion simplifies the data representation and enables the algorithm to interpret the information more effectively, contributing to the overall analysis and modeling process.

```
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

Input6.c.1 and Output6.c.1

d. Completing Numerical Continuous Feature

Now, we must deal with missing or null values in the Age feature. There are several techniques available for estimating and completing numerical continuous features. One method involves generating random numbers within the range defined by the mean and standard deviation of the existing data. A more precise approach entails utilizing other correlated features. In our scenario, there exists a correlation among Age, Gender, and Class. We can approximate Age values by computing the median Age for each combination of Pclass and Gender. The third technique combines the first two methods, wherein random numbers are generated within the range defined by the mean and standard deviation, based on Pclass and Gender combinations, instead of using the median.

While the first and third approaches introduce random variation into our models and may yield different outcomes upon multiple executions, we choose the second method for its ability to provide more precise estimates by considering feature correlations. To begin, we'll initialize an empty array to store estimated Age values based on Class x Gender combinations. Next, we'll iterate over Sex (0 or 1) and Pclass (1, 2, 3) to compute estimated Age values for the six combinations. Subsequently, we'll establish Age bands and assess their correlations with Survived. Finally, we'll replace Age with ordinal values based on these bands, and subsequently, discard the AgeBand feature.

```
for dataset in combine:
    for i in range(0, 2):
        for j in range(0, 3):
            guess_df = dataset[(dataset['Sex'] == i) & \
                               (dataset['Pclass'] == j+1)]['Age'].dropna()
            age_guess = guess_df.median()
            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()
```

Input6.d.1

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22	1	0	7.2500	S	1
1	1	1	1	38	1	0	71.2833	C	3
2	1	3	1	26	0	0	7.9250	S	2
3	1	1	1	35	1	0	53.1000	S	3
4	0	3	0	35	0	0	8.0500	S	1

Output6.d.1

	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

Output6.d.2

```
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)
```

Input6.d.2

```
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()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	AgeBand
0	0	3	0	1	1	0	7.2500	S	1	(16.0, 32.0]
1	1	1	1	2	1	0	71.2833	C	3	(32.0, 48.0]
2	1	3	1	1	0	0	7.9250	S	2	(16.0, 32.0]
3	1	1	1	2	1	0	53.1000	S	3	(32.0, 48.0]
4	0	3	0	2	0	0	8.0500	S	1	(32.0, 48.0]

Input6.d.3 and Output6.d.3

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

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

Input6.d.4 and Output6.d.4

e. Create New Feature Combining Existing Features

We have the option to generate a new feature called Family Size by combining the Parch and SibSp features, allowing us to discard Parch and SibSp from our datasets. Additionally, we can introduce another feature named IsAlone. By doing so, we can drop Parch, SibSp, and FamilySize features and utilize IsAlone instead. Furthermore, we can create a synthetic feature by combining Pclass and Age. This approach aims to simplify the dataset while retaining relevant information about family size and passenger status, ultimately enhancing the modeling process.

```
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)
```

Input6.e.1

	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

Output6.e.1

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

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

Input6.e.2

	IsAlone	Survived
0	0	0.505650
1	1	0.303538

Output6.e.2

```
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()
```

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone
0	0	3	0	1	7.2500	S	1	0
1	1	1	1	2	71.2833	C	3	0
2	1	3	1	1	7.9250	S	2	1
3	1	1	1	2	53.1000	S	3	0
4	0	3	0	2	8.0500	S	1	1

Input6.e.3 and Output6.e.3

```
for dataset in combine:
    dataset['Age*Class'] = dataset.Age * dataset.Pclass

train_df.loc[:, ['Age*Class', 'Age', 'Pclass']].head(5)
```

	Age*Class	Age	Pclass
0	3	1	3
1	2	2	1
2	3	1	3
3	2	2	1
4	6	2	3

Input6.e.4 and Output6.e.4

f. Completing Categorical Feature

The Embarked feature contains values denoting the ports of embarkation, namely S (Southampton), Q (Queenstown), and C (Cherbourg). In our training dataset, there are two missing values in this feature. To address this, we will impute these missing values with the most frequently occurring value, which is the simplest method of handling missing categorical data. This ensures that the dataset remains complete and suitable for analysis without introducing bias or compromising the integrity of the information.

```
freq_port = train_df.Embarked.dropna().mode()[0]
print(freq_port)  # 'S'
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)
```

Input6.f.1

S

	Embarked	Survived
0	C	0.553571
1	Q	0.389610
2	S	0.339009

Output6.f.1

g. Converting Categorical Feature to Numeric

We will convert the Embarked feature into a new numeric Port feature.

```
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

Input6.g.1 and Output6.g.1

h. Quick Completing and Converting Numeric Feature

Now, we can fill in the sole missing value in the Fare feature of the test dataset by utilizing the mode of the feature, representing the most frequently occurring value. This task can be executed in a single line of code. Importantly, given that we're solely replacing a single value, there's no necessity to generate an intermediary new feature or engage in additional correlation analysis to estimate the missing feature. This completion objective guarantees that the model algorithm functions solely on non-null values, aligning with the specified requirement. Additionally, we might contemplate rounding the fare to two decimal places to ensure accurate currency representation.

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

	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

Input6.h.1 and Output6.h.1

	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

Output6.h.2

```
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)
```

Input6.h.2

```
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(7)
```

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	0	3	0	1	0	0	1	0	3
1	1	1	1	2	3	1	3	0	2
2	1	3	1	1	1	0	2	1	3
3	1	1	1	2	3	0	3	0	2
4	0	3	0	2	1	0	1	1	6
5	0	3	0	1	1	2	1	1	3
6	0	1	0	3	3	0	1	1	3

Input6.h.3 and Output6.h.3

And the first 7 rows of test dataset.

```
test_df.head(7)
```

	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

Input6.h.4 and Output6.h.4

7. Model, Predict and Solve

Having finished data preprocessing, we're now ready to train a model for prediction. The choice of algorithm depends on the problem type and solution requirements. Our problem involves both classification and regression tasks, aiming to identify the relationship between the output variable (Survived or not) and other input features (such as Gender, Age, and Port). We need to evaluate a few models to find the best fit for this specific case. Additionally, since we're training our model with a given dataset, we're dealing with supervised learning. Given these criteria, we can focus our choice of models on a select few. These may include:

- Logistic Regression
- Support Vector Machines
- k-Nearest Neighbors
- Gaussian Naive Bayes
- Perceptron
- Linear SVC
- Stochastic Gradient Descent
- Decision Tree
- Random Forest

```
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))
```

This code snippet separates the training data's features (X_train) and target variable (Y_train) by excluding the "Survived" column from the training dataframe. Similarly, it selects the features (X_test) for the test data by removing the "PassengerId" column from the test dataframe. The "shape" function is utilized to display the dimensions (number of rows and columns) of each dataframe. This code segment is essential for preparing the datasets for training machine learning models, ensuring the correct segregation of input features (independent variables) and the target variable (dependent variable).

a. Logistic Regression

Early integration of Logistic Regression into the workflow is essential, as it examines the correlation between the categorical dependent variable (feature) and one or more independent variables (features) by estimating probabilities through a logistic function, particularly the cumulative logistic distribution. This approach enables us to predict the probability of a binary outcome based on provided input features. The confidence score generated by the model, derived from our training dataset, signifies the reliability or certainty of the model's predictions. A higher confidence score implies greater confidence in the model's predictions, whereas a lower score indicates less certainty. This metric is vital for assessing the model's performance and its suitability for the given task.

```
# 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
# output 80.36
```

We can utilize Logistic Regression to validate our assumptions and decisions regarding feature creation and completion objectives. This involves calculating the coefficients of the features in the decision function. Positive coefficients signify a rise in the log-odds of the response (resulting in an increased probability), whereas negative coefficients denote a decline in the log-odds of the response (leading to a decreased probability). For instance, the highest positive coefficient corresponds to the Sex feature, indicating that as the Sex value rises (from male: 0 to female: 1), the likelihood of Survived=1 increases

significantly. Conversely, as Pclass increases, the probability of Survived=1 decreases the most. Additionally, the artificial feature Age*Class demonstrates the second highest negative correlation with Survived, making it a valuable feature for modeling. Similarly, the Title feature exhibits the second highest positive correlation.

This examination helps us grasp the influence of each feature on the probability of survival, directing our decisions regarding feature engineering and offering insights into the associations between features and the target variable.

```
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)
```

	Feature	Correlation
1	Sex	2.201527
5	Title	0.398234
2	Age	0.287163
4	Embarked	0.261762
6	IsAlone	0.129140
3	Fare	-0.085150
7	Age*Class	-0.311200
0	Pclass	-0.749007

b. Support Vector Machines

We will now utilize Support Vector Machines (SVM) for modeling, which are supervised learning algorithms commonly employed for classification and regression tasks. SVMs employ learning algorithms to analyze data and make predictions. Given a set of labeled training samples, where each belongs to one of two categories, SVM constructs a model to classify new test samples into these categories. SVM serves as a non-probabilistic binary linear classifier, meaning it assigns data points to categories without offering probability estimates. It is noteworthy that SVM produces a confidence score, typically higher than that of Logistic Regression models. This elevated confidence score reflects greater confidence in the model's predictions, rendering SVMs particularly beneficial for achieving high precision and reliability in classification tasks.

```
# 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

# output 83.84
```

c. k-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm is a widely used method in machine learning used for classification and regression assignments. Its principle is grounded on the notion that akin data points typically share akin labels or values. Throughout the training process, the KNN algorithm preserves the complete training dataset for reference purposes. The value of k, typically a small positive integer, determines the number of neighbors to consider. For instance, when k is set to 1, the data point is assigned to the class of its single nearest neighbor. KNN is intuitive and effective for datasets with clear separation between classes, but careful selection of k is crucial to balance bias and variance.

It's important to note that while k-NN generates a confidence score, it tends to perform better than Logistic Regression models but not as well as Support Vector Machines in terms of confidence. This indicates that k-NN may offer intermediate reliability in its predictions compared to the other two models. This algorithm is particularly effective in scenarios where the decision boundaries are complex or nonlinear.

```
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

# output 84.74
```

d. Gaussian Naive Bayes

Naive Bayes classifiers are fundamental probabilistic models used in machine learning, applying Bayes' theorem under the assumption of strong feature independence. These classifiers are valued for their scalability, as they require a linear number of parameters relative to the features in a given problem. However, while Naive Bayes classifiers offer simplicity and scalability, their confidence scores are typically lower compared to models like Logistic Regression, Support Vector Machines, and k-Nearest Neighbors. This suggests that Naive Bayes classifiers may offer less certainty in their predictions. Nonetheless, they remain effective in certain contexts, especially when dealing with large datasets and high-dimensional feature spaces.

```
# 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
```

e. Perceptron

The perceptron is a supervised learning technique utilized for training binary classifiers, determining whether an input (represented as a numeric vector) belongs to a specific class. Acting as a linear classifier, it predicts based on a linear predictor function that combines a set of weights with the feature vector. Notably, the perceptron algorithm supports online learning, processing training set elements individually, allowing for ongoing model adjustment based on new data. Fundamentally, the perceptron iteratively updates its weights to minimize classification errors and enhance its ability to accurately classify input data into desired categories. This characteristic renders it valuable for tasks involving binary classification and real-time processing, like spam detection or sentiment analysis.

```
# 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

# 78.0
```

f. Linear SVC

Linear SVC is a classification algorithm used to find the optimal linear decision boundary that separates data points into different classes. Unlike other SVM variants, LinearSVC operates directly in the original feature space and is typically suitable for large-scale and high-dimensional datasets. Its main goal is to maximize the margin between classes, aiming to achieve better generalization performance and noise robustness.

```
# 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

#output 79.12
```

g. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an optimization technique used in the training of machine learning models. Unlike computing gradients over the entire dataset, SGD updates model parameters using small batches of data, enhancing efficiency, particularly for large datasets. It iteratively modifies parameters to minimize a loss function, usually employing a predetermined learning rate. SGD is extensively utilized for its rapidity and scalability, rendering it applicable across diverse machine learning tasks.

```
# 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   #72.28
```

h. Decision Tree

This model utilizes a decision tree as its predictive tool, where features are linked to outcomes regarding the target value. Classification trees are utilized when predicting a finite set of values, with branches indicating feature combinations leading to class labels, while regression trees are employed for continuous value prediction. This model boasts the highest confidence score among those assessed. Decision trees are favored for their straightforwardness in grasping datasets and offering strong interpretability.

```
# 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

#output 86.76
```

i. Random Forest

The Random Forests algorithm is widely recognized as one of the most preferred machine learning techniques, belonging to the ensemble learning methods category, which combines multiple models to enhance predictive accuracy. During training, Random Forests build numerous decision trees, typically around 100 trees, and aggregate their predictions to make the final decision. For classification tasks, the output is determined by the most common class among the individual trees, while for regression tasks, it relies on the average prediction of all trees. With the highest confidence score among the models assessed, Random Forests exhibit a notable level of confidence in their predictions. As a result, we have opted to utilize the output (Y_pred) of this model for our competition submission. Renowned for their resilience, adaptability, and ability to handle high-dimensional data, Random Forests remain a preferred choice for various classification and regression tasks in machine learning.

```
# 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

#output 86.76
```

8. Model Evaluation

Now, we can assess and rank all the models to determine the best one for our problem. Although both the Decision Tree and Random Forest models achieve the same score, we opt to utilize the **Random Forest** model. The reason behind this decision is that Random Forests address the tendency of decision trees to overfit the training set. By constructing multiple decision trees and aggregating their predictions, Random Forests offer improved generalization performance and robustness against overfitting. Therefore, despite scoring equally with the Decision Tree model, we choose Random Forests for their ability to mitigate the overfitting issue commonly associated with decision trees.

```
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)
```

	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
6	Stochastic Gradient Decent	78.90
7	Linear SVC	78.90
5	Perceptron	78.00
4	Naive Bayes	72.28

SOLUTION OF NADIN TAMER

Introduction

Since the “Sinking of the Titanic” is one of the biggest disasters in history, it is an event that has attracted the attention of many people due to its tragic circumstances, significant loss of life, and the various human stories that emerged from it. That's why we are going to investigate and predict survival rates by the given dataset about the passengers, which includes various demographic and personal information such as age, sex, and class. By creating parameters from the given datasets, we can better understand the factors that influenced survival. Subsequently, we will train the machine with these parameters using several machine learning algorithms, which are depicted in the figure above. This process involves data cleaning, feature selection, model training, and evaluation. Before delving into the machine learning aspect, there are 5 essential preparatory steps that must be completed first, including data exploration, visualization, and handling missing values.

- Gaussian Naive Bayes
- Logistic Regression
- Support Vector Machines
- Perceptron
- Decision Tree Classifier
- Random Forest Classifier
- KNN or k-Nearest Neighbors
- Stochastic Gradient Descent
- Gradient Boosting Classifier

Figure 2

Figure 1

1. Importing Necessary Libraries

Firstly, it is imperative to import the required libraries such as pandas, mathplotlib, seaborn and numpy , in order to execute the algorithms discussed in the introduction segment like visualization of given dataset.

```
[ ]  #data analysis libraries
     import numpy as np
     import pandas as pd

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

     #ignore warnings
     import warnings
     warnings.filterwarnings('ignore')
```

Figure 3

2. Read in and Explore the Data

Once the datasets are retrieved from ".csv" files, which include both training and testing data, the subsequent step entails generating a thorough statistical summary exclusively for the training dataset (train). This summary serves the purpose of providing a detailed overview of the dataset's structure and content, thereby enabling a deeper understanding of its characteristics and facilitating subsequent analysis and interpretation.

```
[ ]  #import train and test CSV files
     train = pd.read_csv("../input/train.csv")
     test = pd.read_csv("../input/test.csv")

     #take a look at the training data
     train.describe(include="all")
```

Figure 4

3. Data Analysis

For a thorough analysis of the dataset, we utilize a simple one-liner code to explore the different features present in its columns. This straightforward method allows us to quickly grasp the characteristics of the dataset, aiding in subsequent analysis and interpretation.

```
[30] #see a sample of the dataset to get an idea of the variables
     train.sample(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
67	68	0	3	Crease, Mr. Ernest James	male	19.0	0	0	S.P. 3464	8.1583	NaN	S
188	189	0	3	Bourke, Mr. John	male	40.0	1	1	364849	15.5000	NaN	Q
263	264	0	1	Harrison, Mr. William	male	40.0	0	0	112059	0.0000	B94	S
490	491	0	3	Hagland, Mr. Konrad Mathias Reiersen	male	NaN	1	0	65304	19.9667	NaN	S
741	742	0	1	Cavendish, Mr. Tyrell William	male	36.0	1	0	19877	78.8500	C46	S

Tablo 1

From the figure above, we can observe the number of features in the given dataset. If we need to identify these value's data types, we can categorize them by:

- Survived: int
- Pclass: int
- Name: string
- Sex: string
- Age: float
- SibSp: int
- Parch: int
- Ticket: string
- Fare: float
- Cabin: string
- Embarked: string

The lines of code below, play a crucial role in enhancing our comprehension of the dataset by meticulously listing the names of its features and providing a comprehensive summary of each feature through the utilization of the describe() function with the parameter include="all". This summary encompasses various statistical metrics, such as mean, standard deviation, minimum and maximum values, quartiles, and more, offering valuable insights into the distribution and characteristics of the dataset. By leveraging this summary, we gain a deeper understanding of the dataset's underlying structure and content, thereby laying a solid foundation for subsequent data analysis and decision-making processes and also in Data Cleaning process, we can detect the NaN values to fill them with new values by an algorithm which is the topic of next content.

```
[ ]  #see a summary of the training dataset
     train.describe(include = "all")
```

Figure 5

Here are a few observations based on our dataset:

- ❖ Our training set comprises a total of 891 passengers.
- ❖ The Age feature exhibits a notable absence, with approximately 19.8% of its values being missing. Given the presumed significance of age in relation to survival, addressing these gaps is imperative.
- ❖ Conversely, the Cabin feature displays a much higher absence rate, with approximately 77.1% of its values missing. Due to the substantial extent of missing data, it's impractical to attempt to fill these gaps, suggesting that we should consider dropping this feature from our dataset.
- ❖ The Embarked feature, on the other hand, has only 0.22% of its values missing, which is relatively insignificant and shouldn't pose any major issues.

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	204	889
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147	3
top	NaN	NaN	NaN	Braund, Mr. Owen Harris	male	NaN	NaN	NaN	347082	NaN	B96 B98	S
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4	644
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN	32.204208	NaN	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN	49.693429	NaN	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN	NaN

Tablo 2

```
#check for any other unusable values
print(pd.isnull(train).sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Figure 6

Here are some observations based on the data before getting more accurate results in visualization section:

- ❖ Gender: Survival rates are higher among females.
- ❖ Family Status (SibSp/Parch): Individuals traveling alone have a greater chance of survival.
- ❖ Age: Survival rates tend to be higher among youn[6]t children.
- ❖ Socioeconomic Status (Pclass): Passengers from higher socioeconomic classes are more likely to survive.

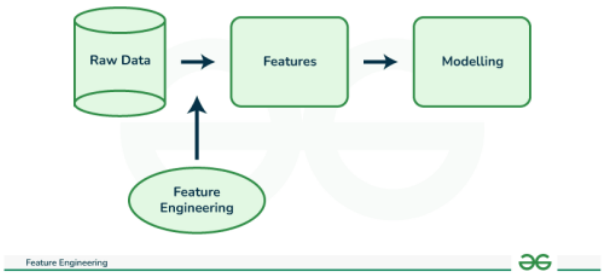


Figure 7

4. Data Visualization

Visualizing the data allows us to validate the predictions made during the data analysis phase, enabling us to discern whether they hold true or prove false. Through visual representations, such as plots and graphs, we gain deeper insights into the relationships and patterns within the dataset, thereby enhancing our ability to make more accurate predictions. By leveraging visualization techniques, we can effectively identify trends, anomalies, and correlations, empowering us to refine our predictive models and derive more robust conclusions from the data analysis process.

a. Sex Feature

This feature facilitates the visualization and analysis of survival rates based on the passengers' sex variable aboard the Titanic. Through simple visualizations like bar charts or pie charts, we can easily observe and assess the differences in survival rates between male and female passengers.

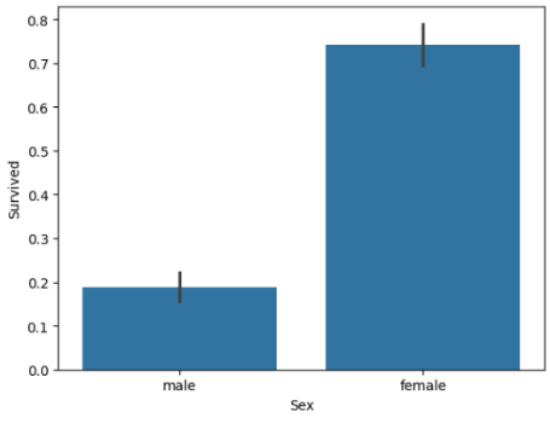
```
#draw a bar plot of survival by sex
sns.barplot(x="Sex", y="Survived", data=train)

#print percentages of females vs. males that survive
print("Percentage of females who survived:", train["Survived"][train["Sex"] == 'female'].value_counts(normalize = True)[1]*100)

print("Percentage of males who survived:", train["Survived"][train["Sex"] == 'male'].value_counts(normalize = True)[1]*100)
```

```
Percentage of females who survived: 74.20382165605095
Percentage of males who survived: 18.890814558058924
```

Based on our observations, it appears that female passengers exhibit a notably higher likelihood of survival compared to their male counterparts. This disparity in survival rates can be attributed to the implementation of the well-known "women and children first" protocol during the evacuation procedure of the Titanic. This protocol prioritized the boarding of lifeboats by women and children, potentially explaining the higher survival rates among female passengers.



b. Pclass Feature

This feature serves as a crucial tool for visualizing and conducting an in-depth analysis of survival rates based on the socioeconomic class of passengers aboard the Titanic. Through the utilization of various visualization techniques such as histograms, box plots, or violin plots, we can effectively illustrate and scrutinize the distribution and patterns of survival outcomes across different passenger classes.

```
#draw a bar plot of survival by Pclass
sns.barplot(x="Pclass", y="Survived", data=train)

#print percentage of people by Pclass that survived
print("Percentage of Pclass = 1 who survived:", train["Survived"][train["Pclass"] == 1].value_counts(normalize = True)[1]*100)

print("Percentage of Pclass = 2 who survived:", train["Survived"][train["Pclass"] == 2].value_counts(normalize = True)[1]*100)

print("Percentage of Pclass = 3 who survived:", train["Survived"][train["Pclass"] == 3].value_counts(normalize = True)[1]*100)
```

```
Percentage of Pclass = 1 who survived: 62.96296296296296
Percentage of Pclass = 2 who survived: 47.28260869565217
Percentage of Pclass = 3 who survived: 24.236252545824847
```

Upon analysis, it becomes apparent that passengers belonging to higher socioeconomic classes exhibit a significantly higher likelihood of survival compared to those from lower classes. This discrepancy in survival rates can be attributed to the preferential assistance and access to resources afforded to individuals of higher social standing during the evacuation process.

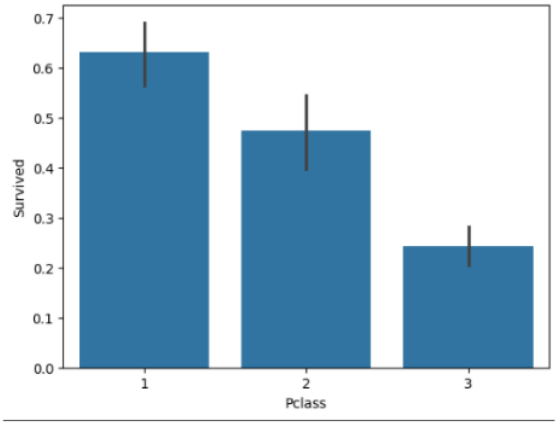


Figure 8

c. SibSp Feature

This feature proves invaluable in visualizing and conducting a comprehensive analysis of survival rates based on the number of siblings (SibSp) accompanying passengers aboard the Titanic. By employing various visualization techniques such as bar charts, histograms, or scatter plots, we can effectively illustrate and scrutinize the distribution and patterns of survival outcomes across different sibling counts.

```
#draw a bar plot for SibSp vs. survival
sns.barplot(x="SibSp", y="Survived", data=train)

#I won't be printing individual percent values for all of these.
print("Percentage of SibSp = 0 who survived:", train["Survived"][train["SibSp"] == 0].value_counts(normalize = True)[1]*100)

print("Percentage of SibSp = 1 who survived:", train["Survived"][train["SibSp"] == 1].value_counts(normalize = True)[1]*100)

print("Percentage of SibSp = 2 who survived:", train["Survived"][train["SibSp"] == 2].value_counts(normalize = True)[1]*100)
```

```
Percentage of SibSp = 0 who survived: 34.53947368421053
Percentage of SibSp = 1 who survived: 53.588516746411486
Percentage of SibSp = 2 who survived: 46.42857142857143
```

So it seems that passengers traveling with family members, such as siblings or spouses, could assist each other during the chaos of the evacuation. This mutual support would help them navigate to lifeboats more effectively but more siblings may had struggles to stay together in chaotic enviroment.

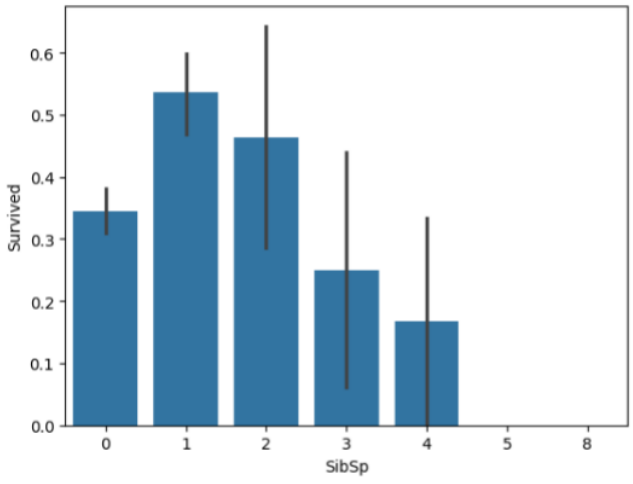


Figure 9

d. Age Feature

This feature serves as a pivotal tool for visualizing and conducting a comprehensive analysis of survival rates based on age groups among passengers aboard the Titanic. Through the utilization of various visualization techniques such as histograms, kernel density plots, or age distribution plots segmented by survival status, we can effectively illustrate and scrutinize the distribution and patterns of survival outcomes across different age brackets

```
#sort the ages into logical categories
train["Age"] = train["Age"].fillna(-0.5)
test["Age"] = test["Age"].fillna(-0.5)
bins = [-1, 0, 5, 12, 18, 24, 35, 60, np.inf]
labels = ['Unknown', 'Baby', 'Child', 'Teenager', 'Student', 'Young Adult', 'Adult', 'Senior']
train['AgeGroup'] = pd.cut(train["Age"], bins, labels = labels)
test['AgeGroup'] = pd.cut(test["Age"], bins, labels = labels)

#draw a bar plot of Age vs. survival
sns.barplot(x="AgeGroup", y="Survived", data=train)
plt.show()
```

Indeed, upon closer examination, it becomes evident that infant passengers were given priority during the evacuation process, likely due to their vulnerability and need for immediate assistance. This prioritization of babies aligns with the longstanding maritime tradition of ensuring the safety of the youngest passengers first. Conversely, elderly passengers faced greater challenges and lower survival rates due to their physical limitations, which may have hindered their mobility and ability to navigate the chaotic evacuation procedures effectively.

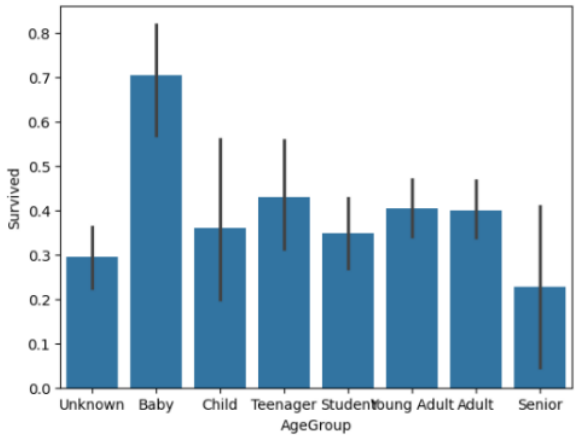


Figure 10

5. Data Cleaning

During this stage of the process, unnecessary data segments will be identified and removed, while missing values within the provided dataset will be effectively handled and completed. This meticulous data cleaning and preprocessing phase are crucial for ensuring the integrity and accuracy of subsequent analyses and predictive modeling.

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked
count	418.000000	418.000000	418	418	332.000000	418.000000	418.000000	417.000000	418
unique	NaN	NaN	418	2	NaN	NaN	NaN	NaN	3
top	NaN	NaN	Kelly, Mr. James	male	NaN	NaN	NaN	NaN	S
freq	NaN	NaN	1	266	NaN	NaN	NaN	NaN	270
mean	1100.500000	2.265550	NaN	NaN	30.272590	0.447368	0.392344	35.627188	NaN
std	120.810458	0.841838	NaN	NaN	14.181209	0.896760	0.981429	55.907576	NaN
min	892.000000	1.000000	NaN	NaN	0.170000	0.000000	0.000000	0.000000	NaN
25%	996.250000	1.000000	NaN	NaN	21.000000	0.000000	0.000000	7.895800	NaN
50%	1100.500000	3.000000	NaN	NaN	27.000000	0.000000	0.000000	14.454200	NaN
75%	1204.750000	3.000000	NaN	NaN	39.000000	1.000000	0.000000	31.500000	NaN
max	1309.000000	3.000000	NaN	NaN	76.000000	8.000000	9.000000	512.329200	NaN

Tablo 3

Considering their limited relevance to our analysis, we have decided to eliminate the Cabin and Ticket features from our dataset. These features contain either too much missing data or do not offer significant insights that could contribute meaningfully to our analysis.

```
#we'll start off by dropping the Cabin featu
train = train.drop(['Cabin'], axis = 1)
test = test.drop(['Cabin'], axis = 1)
```

```
#we can also drop the Ticket feature since it's un
train = train.drop(['Ticket'], axis = 1)
test = test.drop(['Ticket'], axis = 1)
```

Before proceeding with any further analysis, it's imperative to address the issue of missing values within the Embarked feature. To begin, a few lines of code are implemented to provide an overview of the number of missing values present in the current dataset.

```
#now we need to fill in the missing values in the Embarked feature
print("Number of people embarking in Southampton (S):")
southampton = train[train["Embarked"] == "S"].shape[0]
print(southampton)

print("Number of people embarking in Cherbourg (C):")
cherbourg = train[train["Embarked"] == "C"].shape[0]
print(cherbourg)

print("Number of people embarking in Queenstown (Q):")
queenstown = train[train["Embarked"] == "Q"].shape[0]
print(queenstown)
```

```
Number of people embarking in Southampton (S):
644
Number of people embarking in Cherbourg (C):
168
Number of people embarking in Queenstown (Q):
77
```

Upon examination, it becomes evident that the most prevalent value (mode) within the Embarked feature is "Southampton" by a considerable margin. Given this observation, a practical approach for addressing the missing values entails employing the method of imputation, wherein the missing values are replaced with the mode value of the dataset, which in this case is "Southampton".

```
[ ]  #replacing the missing values in the Embarked feature with S
     train = train.fillna({"Embarked": "S"})
```

In addition to the missing values in the Embarked feature, we also encounter numerous missing entries within the Age feature. To address this issue, we implement a strategic approach by leveraging the titles extracted from passengers' names and mapping them into appropriate age categories. This innovative technique enables us to infer the missing ages more accurately based on the demographic characteristics implied by passengers' titles

```
#replace various titles with more common names
for dataset in combine:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Capt', 'Col',
    'Don', 'Dr', 'Major', 'Rev', 'Jonkheer', 'Dona'], 'Rare')

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

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

To group titles into the same age category, we utilize the code above. This code helps us identify and consolidate titles that correspond to similar age groups, thereby simplifying the categorization process and improving the accuracy of our age imputation technique.

```
[17] #map each of the title groups to a numerical valueThere
     title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Royal": 5, "Rare": 6}
     for dataset in combine:
         dataset['Title'] = dataset['Title'].map(title_mapping)
         dataset['Title'] = dataset['Title'].fillna(0)

     train.head()
```

Once we categorize the titles into specific age groups, the next step involves handling the unknown values of age within both the train and test datasets. To accomplish this, we systematically scan through the age values of both datasets. When encountering unknown values, we utilize the categorized title names to impute the missing ages effectively. By leveraging the information derived from the title name categorization, we ensure that the imputed ages are aligned with the corresponding age groups inferred from passengers' titles.

```
# fill missing age with mode age group for each title
mr_age = train[train["Title"] == 1]["AgeGroup"].mode() #Young Adult
miss_age = train[train["Title"] == 2]["AgeGroup"].mode() #Student
mrs_age = train[train["Title"] == 3]["AgeGroup"].mode() #Adult
master_age = train[train["Title"] == 4]["AgeGroup"].mode() #Baby
royal_age = train[train["Title"] == 5]["AgeGroup"].mode() #Adult
rare_age = train[train["Title"] == 6]["AgeGroup"].mode() #Adult

age_title_mapping = {1: "Young Adult", 2: "Student", 3: "Adult", 4: "Baby", 5: "Adult", 6: "Adult"}

#I tried to get this code to work with using .map(), but couldn't.
#I've put down a less elegant, temporary solution for now.
#train = train.fillna({"Age": train["Title"].map(age_title_mapping)})
#test = test.fillna({"Age": test["Title"].map(age_title_mapping)})

for x in range(len(train["AgeGroup"])):
    if train["AgeGroup"][x] == "Unknown":
        train["AgeGroup"][x] = age_title_mapping[train["Title"][x]]

for x in range(len(test["AgeGroup"])):
    if test["AgeGroup"][x] == "Unknown":
        test["AgeGroup"][x] = age_title_mapping[test["Title"][x]]
```

Considering their limited relevance to our analysis, we also have decided to eliminate the name feature from our dataset. These feature contain either too much missing data or do not offer significant insights that could contribute meaningfully to our analysis.

```
#drop the name feature since it contains
train = train.drop(['Name'], axis = 1)
test = test.drop(['Name'], axis = 1)
```

At this stage, we need to categorize the fare values into logical groups, which will help simplify and enhance our analysis. Additionally, we must address the single missing fare value present in the test dataset. To accomplish this, we will first identify and fill the missing fare value by calculating the mean fare for the corresponding passenger class from the train dataset. Following this, we will divide the fare values into several logical groups, using quartiles, to create a new categorized feature.

```
#fill in missing Fare value in test set based on mean fare for that Pclass
for x in range(len(test["Fare"])):
    if pd.isnull(test["Fare"][x]):
        pclass = test["Pclass"][x] #Pclass = 3
        test["Fare"][x] = round(train[train["Pclass"] == pclass]["Fare"].mean(), 4)

#map Fare values into groups of numerical values
train['FareBand'] = pd.qcut(train['Fare'], 4, labels = [1, 2, 3, 4])
test['FareBand'] = pd.qcut(test['Fare'], 4, labels = [1, 2, 3, 4])

#drop Fare values
train = train.drop(['Fare'], axis = 1)
test = test.drop(['Fare'], axis = 1)
```

This code first fills in the missing Fare values in the test dataset by using the mean fare for the corresponding passenger class (Pclass) from the train dataset. Then, it creates a new feature called FareBand in both the train and test datasets by dividing the Fare values into four equal groups (quartiles) using pd.qcut, and labels each group with numerical values from 1 to 4. Finally, it drops the original Fare column from both datasets, as the fare information is now represented by the FareBand feature. This process helps in handling fare information more effectively during the analysis and modeling stages.

	PassengerId	Survived	Pclass	Sex	SibSp	Parch	Embarked	AgeGroup	CabinBool	Title	FareBand
0	1	0	3	0	1	0	1	4.0	0	1	1
1	2	1	1	1	1	0	2	6.0	1	3	4
2	3	1	3	1	0	0	1	5.0	0	2	2
3	4	1	1	1	1	0	1	5.0	1	3	4
4	5	0	3	0	0	0	1	5.0	0	1	2

Tablo 4

6. Choosing the Best Model

Among the various machine learning training models available, it is crucial to identify the most accurate model to develop a superior machine learning mechanism. To achieve this, we must compare the output of the trained data with the test data. This process involves training multiple models, evaluating their performance on the test dataset, and selecting the model that delivers the highest accuracy and best generalizes to unseen data. To achieve that objective we will use 22% of training data.

```
from sklearn.model_selection import train_test_split

predictors = train.drop(['Survived', 'PassengerId'], axis=1)
target = train["Survived"]
x_train, x_val, y_train, y_val = train_test_split(predictors, target, test_size = 0.22, random_state = 0)
```

Models to be tested are:

- Gaussian Naive Bayes
- Logistic Regression
- Support Vector Machines
- Perceptron
- Decision Tree Classifier
- Random Forest Classifier
- KNN or k-Nearest Neighbors
- Stochastic Gradient Descent
- Gradient Boosting Classifier

```
# Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

gaussian = GaussianNB()
gaussian.fit(x_train, y_train)
y_pred = gaussian.predict(x_val)
acc_gaussian = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_gaussian)
```

```
68
```

Model 1: Gaussian Naive Bayes

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_val)
acc_logreg = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_logreg)
```

```
79.7
```

Model 2: Logistic Regression

```
# Support Vector Machines
from sklearn.svm import SVC

svc = SVC()
svc.fit(x_train, y_train)
y_pred = svc.predict(x_val)
acc_svc = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_svc)
```

```
82.74
```

Model 3: Support Vector Machines

```
# Linear SVC
from sklearn.svm import LinearSVC

linear_svc = LinearSVC()
linear_svc.fit(x_train, y_train)
y_pred = linear_svc.predict(x_val)
acc_linear_svc = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_linear_svc)
```

```
78.68
```

Model 4: Linear SVC

```
# Perceptron
from sklearn.linear_model import Perceptron

perceptron = Perceptron()
perceptron.fit(x_train, y_train)
y_pred = perceptron.predict(x_val)
acc_perceptron = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_perceptron)
```

```
78.68
```

Model 5: Perception

```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier

decisiontree = DecisionTreeClassifier()
decisiontree.fit(x_train, y_train)
y_pred = decisiontree.predict(x_val)
acc_decisiontree = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_decisiontree)
```

```
80.71
```

Model 6: Decision Tree

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier

randomforest = RandomForestClassifier()
randomforest.fit(x_train, y_train)
y_pred = randomforest.predict(x_val)
acc_randomforest = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_randomforest)
```

```
84.26
```

Model 7: Random Forest

```
# KNN or k-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
y_pred = knn.predict(x_val)
acc_knn = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_knn)
```

```
82.23
```

Model 8: KNN or k-Nearest Neighbors

```
# Stochastic Gradient Descent
from sklearn.linear_model import SGDClassifier

sgd = SGDClassifier()
sgd.fit(x_train, y_train)
y_pred = sgd.predict(x_val)
acc_sgd = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_sgd)
```

```
78.68
```

Model 9: Stochastic Gradient Descent

```
# Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier

gbk = GradientBoostingClassifier()
gbk.fit(x_train, y_train)
y_pred = gbk.predict(x_val)
acc_gbk = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_gbk)
```

```
84.26
```

Model 10: Gradient Boosting Classifier

After evaluating various machine learning models, the ones that yield the best results are the Random Forest and Gradient Boosting classifiers. These models stand out due to their superior performance in terms of accuracy and reliability when compared to other models tested.

	Model	Score
3	Random Forest	84.26
9	Gradient Boosting Classifier	84.26
0	Support Vector Machines	82.74
1	KNN	82.23
7	Decision Tree	80.71
2	Logistic Regression	79.70
4	Naive Bayes	78.68
5	Perceptron	78.68
6	Linear SVC	78.68
8	Stochastic Gradient Descent	78.68

References

Kaggle. "Titanic - Machine Learning from Disaster." 2012. https://www.kaggle.com/competitions/titanic.

Kaggle. "Titanic Data Science Solutions." by Manav Sehgal. https://www.kaggle.com/code/startupsci/titanic-data-science-solutions.

Kaggle. "Titanic Survival Predictions (Beginner)." by Nadin Tamer. https://www.kaggle.com/code/nadintamer/titanic-survival-predictions-beginner.

GeeksforGeeks. "What is Feature Engineering?" December 21, 2024. https://www.geeksforgeeks.org/what-is-feature-engineering/.

Malzemebilimi.net. "Titanic Batışı (Darbe Deneyi)." https://malzemebilimi.net/titanic-batisi-darbe-deneyi.html.

ML_Term_Project.docx

ORIGINALITY REPORT

16%	16%	1%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.kaggle.com Internet Source	7%
2	unsupervisedlearning.co.uk Internet Source	2%
3	www.csdn.net Internet Source	1%
4	github.com Internet Source	1%
5	computer-science-student.tistory.com Internet Source	1%
6	freesoft.dev Internet Source	1%
7	Cesar Garcia, Alexis Ivan Andrade Valle, Angel Alberto Silva Conde, Nestor Ulloa et al. "Predicting the impact of adding metakaolin on the splitting strength of concrete using ensemble ML classification and symbolic regression techniques –a comparative study", Frontiers in Built Environment, 2024 Publication	<1%

8	serpdotai.gitbook.io Internet Source	<1 %
9	huggingface.co Internet Source	<1 %
10	hynnjnn.tistory.com Internet Source	<1 %
11	medium.com Internet Source	<1 %
12	onlinesciencepublishing.com Internet Source	<1 %
13	fastercapital.com Internet Source	<1 %
14	Singh, Shekhar. "Facial Expression Recognition Using Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs) for Data Augmentation and Image Generation", University of Nevada, Las Vegas, 2024 Publication	<1 %
15	klnce.edu Internet Source	<1 %
16	www.frontiersin.org Internet Source	<1 %
17	www.soffner.com.br Internet Source	<1 %

18	www.techscience.com Internet Source	<1%
19	www.ijcaonline.org Internet Source	<1%
20	radar.brookes.ac.uk Internet Source	<1%
21	acme.byu.edu Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		