ClassifyMeister Midterm Evaluation

Group - 12

- Ajitesh Shree
- Jyotiraditya
- Sharvil Athaley
- Zoya Manjer

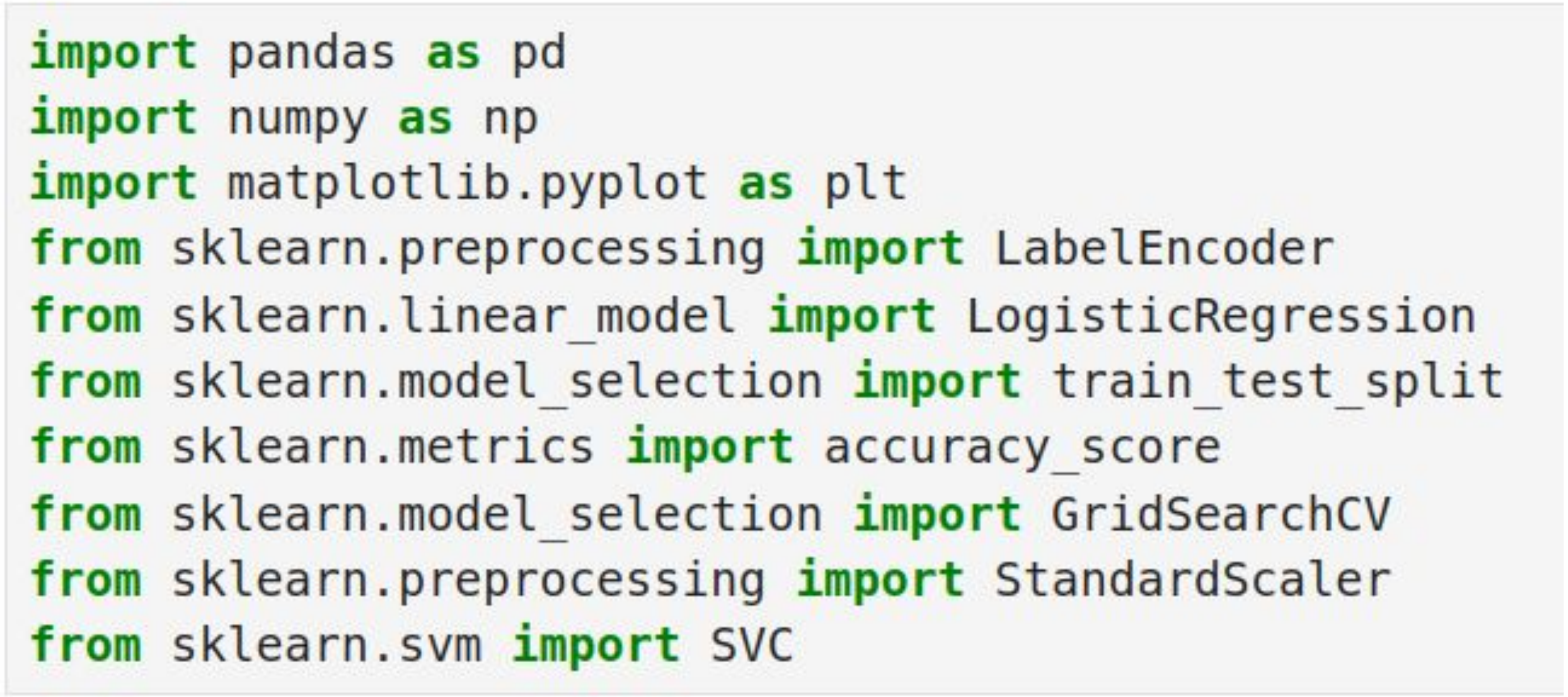
Objective

1. Implementing the methods we have learnt so far such as K-Nearest Neighbors , Logistic Regression and Support Vector Machines
2. Understanding how these methods work and why they are so accurate
3. Learning the various functions used in their implementation
4. Training models and extracting the maximum accuracy that we can from them

Dataset Description:

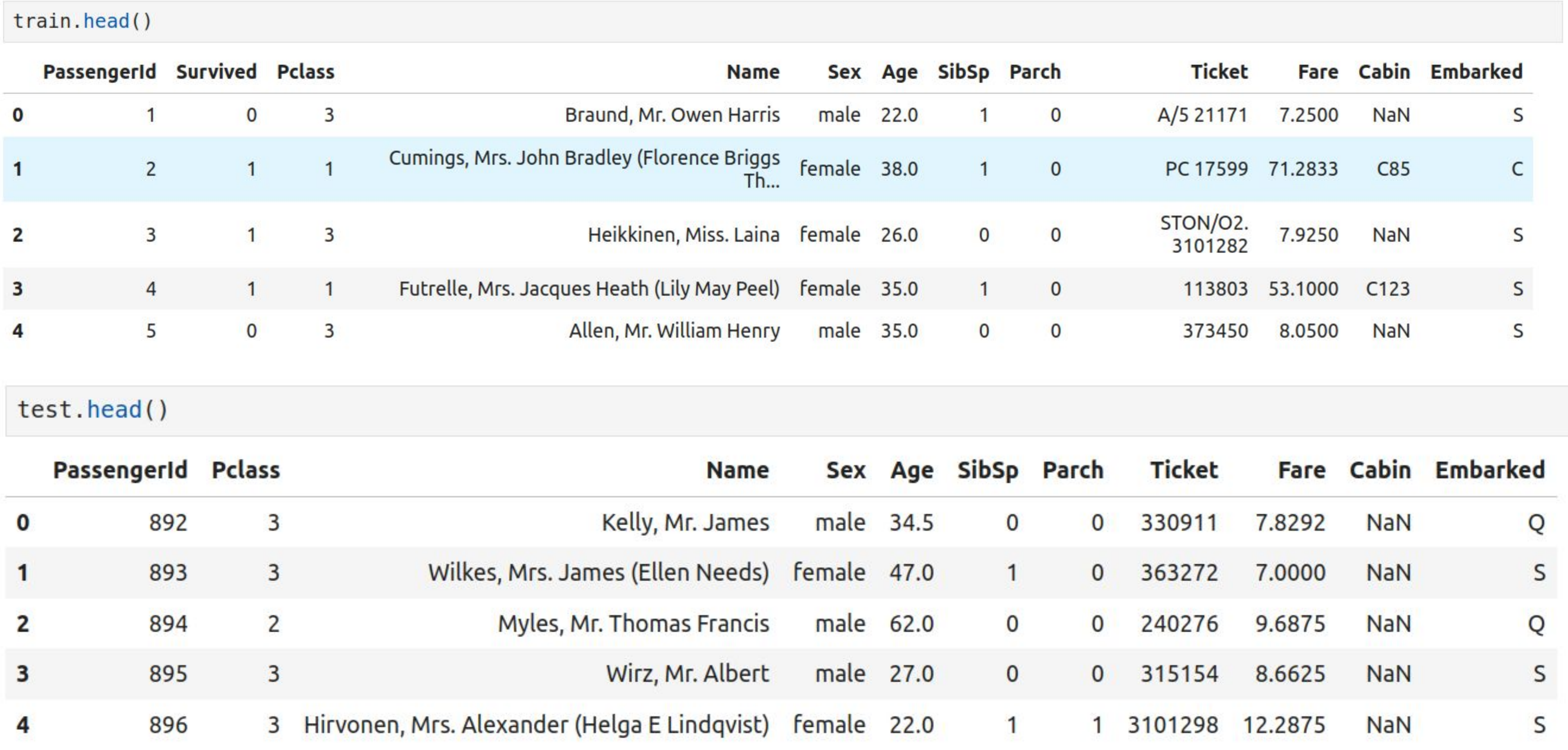
- Dataset used: Titanic Survivors
- The dataset includes various features that describe each passenger. These features are essential for understanding the factors that may have influenced a passenger's chance of survival. Here are the key features available in the Titanic survivor dataset:
  - Passenger ID: A unique identifier assigned to each passenger.
  - Survived: Denotes whether the passenger survived or not (0 = Did not survive, 1 = Survived).
  - Pclass: The passenger's ticket class (1 = First class, 2 = Second class, 3 = Third class).
  - Name: The passenger's name.
  - Sex: The gender of the passenger (Male or Female).
  - Age: The age of the passenger in years.
  - SibSp: The number of siblings/spouses aboard the Titanic.
  - Parch: The number of parents/children aboard the Titanic.
  - Ticket: The ticket number.
  - Fare: The fare paid by the passenger for the ticket.
  - Cabin: The cabin number where the passenger stayed. Some entries may have missing values.
  - Embarked: The port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton)

Libraries Used throughout:



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

Snippets:



```
train.head()
```

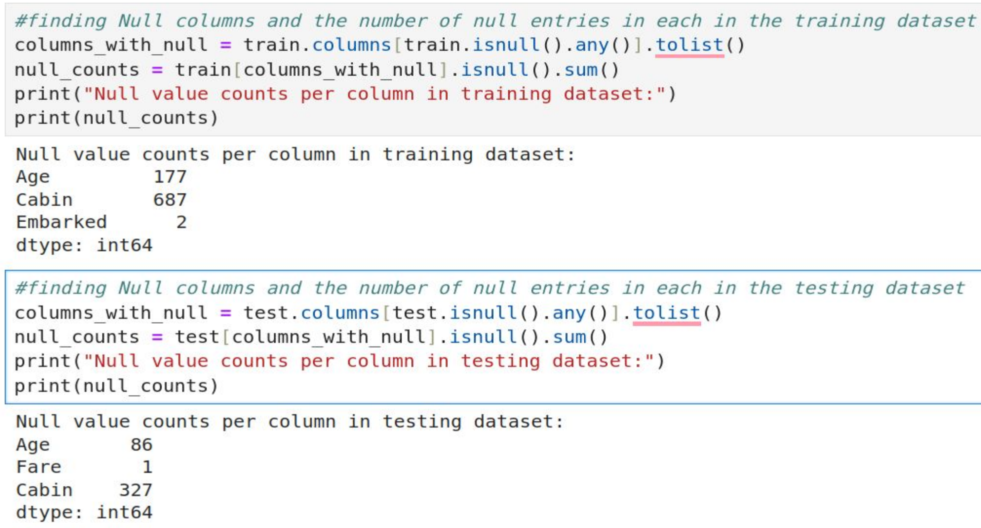
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
test.head()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

Filling Null Values:

- Null Values in Training dataset: { ‘Age’ : 177 ; ‘Cabin’ : 687 ; ‘Embarked’ : 2 }
- Null Values in Testing dataset: { ‘Age’ : 86 ; Fare : 1 ; Cabin : 327 }
- We fill the Null value in Age and Fare features by taking the average mean as the filling value.
- Also, for filling the null values in Embarked and Cabin features, we take the value which occurred the most and the second-most and fill them in a randomly, i.e, without knowing which null value got filled by the most occurring or the second-most occurring value.



```
#finding Null columns and the number of null entries in each in the training dataset
columns_with_null = train.columns[train.isnull().any()].tolist()
null_counts = train[columns_with_null].isnull().sum()
print("Null value counts per column in training dataset:")
print(null_counts)
```

```
Null value counts per column in training dataset:
Age         177
Cabin       687
Embarked      2
dtype: int64
```

```
#finding Null columns and the number of null entries in each in the testing dataset
columns_with_null = test.columns[test.isnull().any()].tolist()
null_counts = test[columns_with_null].isnull().sum()
print("Null value counts per column in testing dataset:")
print(null_counts)
```

```
Null value counts per column in testing dataset:
Age       86
Fare       1
Cabin    327
dtype: int64
```

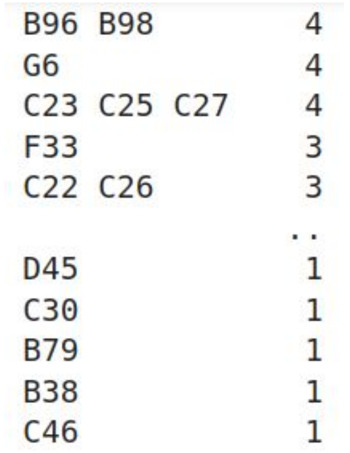
Code Snippets filling Null values:

Filling Null values of Age and Fare:

1. train['Age'].fillna(train['Age'].mean(), inplace=True)
2. test['Age'].fillna(test['Age'].mean(), inplace=True)
3. test['Fare'].fillna(test['Fare'].mean(), inplace=True)

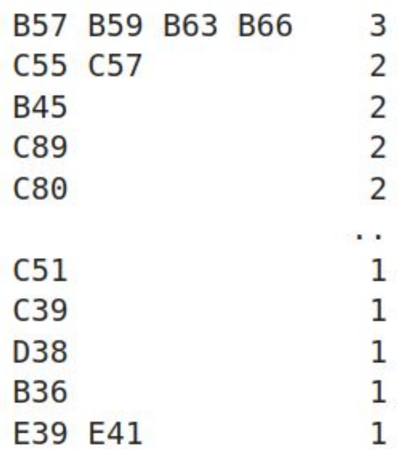
Code to get the most occurring and 2nd most occurring value in Embarked and Cabin features:

train['Cabin'].value_counts()



```
B96 B98        4
G6             4
C23 C25 C27    4
F33            3
C22 C26        3
              ..
D45            1
C30            1
B79            1
B38            1
C46            1
```

test['Cabin'].value_counts()



```
B57 B59 B63 B66    3
C55 C57            2
B45                2
C89                2
C80                2
                  ..
C51                1
C39                1
D38                1
B36                1
E39 E41            1
```

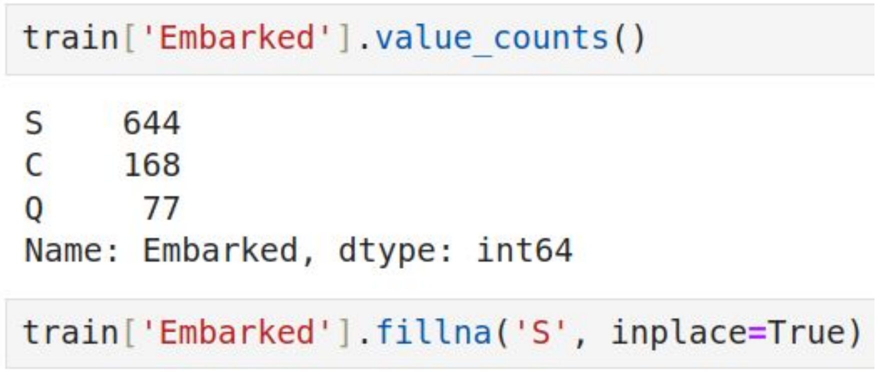
Code Snippets filling Null values:

Creating function to fill value in Embarked and Cabin and using them:

```
def fillNullValInCabin(df, value1, value2):
    toFill = np.array([])
    value_counts = df['Cabin'].value_counts()

    for value, count in value_counts.items():
        if count == value1 or count == value2:
            toFill = np.append(toFill, value)

    random_index = np.random.randint(0, len(toFill))
    df['Cabin'].fillna(toFill[random_index], inplace=True)
fillNullValInCabin(train, 4, 3)
fillNullValInCabin(test, 2, 3)
```



```
train['Embarked'].value_counts()
```

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

```
train['Embarked'].fillna('S', inplace=True)
```

Encoding and Deleting Irrelevant features:

- Deleted the features ‘Name’ , ‘Ticket’ and ‘PassengerID’ from both datasets as all of them had almost unique values throughout entries, and thus were of no use in finding similarity and pattern in fitting the model parameters.

Code Snippets:

```
train = train.drop('Name', axis =1)
test = test.drop('Name', axis =1)
```

- Before going further, we OneHotEncoded features ‘Embarked’ and ‘Sex’ and Label Encoded feature ‘Cabin’ to convert the categorical value to binary and numerical values respectively.

Encoding and Deleting Irrelevant features:

Encoding Code:

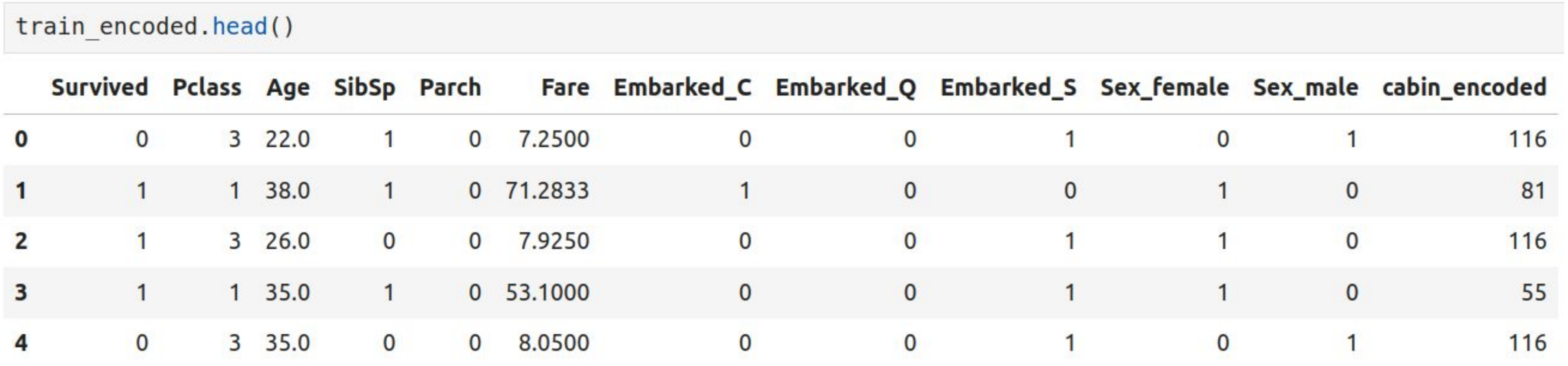
```
train_encoded = pd.concat([train, pd.get_dummies(train['Embarked'], prefix= 'Embarked'), pd.get_dummies(train['Sex'], prefix= 'Sex')], axis=1)
train_encoded.drop(['Embarked','Sex'], axis =1, inplace =True)
test_encoded = pd.concat([test, pd.get_dummies(test['Embarked'], prefix= 'Embarked'), pd.get_dummies(test['Sex'], prefix= 'Sex')], axis=1)
test_encoded.drop(['Embarked','Sex'], axis =1, inplace =True)
encoder = LabelEncoder()
train_encoded['cabin_encoded'] =encoder.fit_transform(train_encoded['Cabin'])
train_encoded.drop('Cabin', axis=1, inplace=True)
test_encoded['cabin_encoded'] =encoder.fit_transform(test_encoded['Cabin'])
test_encoded.drop('Cabin', axis=1, inplace=True)
```

Deleting Code:

```
train_encoded.drop('PassengerId',axis=1,inplace = True)
test_passengerID = test_encoded['PassengerId']
test_encoded.drop('PassengerId',axis=1,inplace = True)
train_encoded.drop('Ticket',axis=1,inplace = True)
test_encoded.drop('Ticket',axis=1,inplace = True)
```

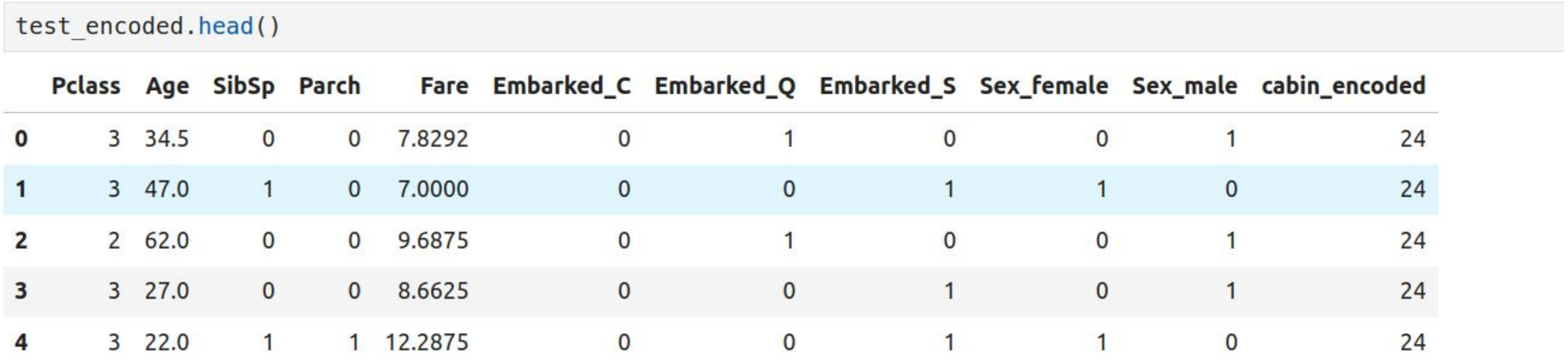
Encoding and Deleting Irrelevant features:

Snippet showing transformed Datasets:



```
train_encoded.head()
```

	Survived	Pclass	Age	SibSp	Parch	Fare	Embarked_C	Embarked_Q	Embarked_S	Sex_female	Sex_male	cabin_encoded
0	0	3	22.0	1	0	7.2500	0	0	1	0	1	116
1	1	1	38.0	1	0	71.2833	1	0	0	1	0	81
2	1	3	26.0	0	0	7.9250	0	0	1	1	0	116
3	1	1	35.0	1	0	53.1000	0	0	1	1	0	55
4	0	3	35.0	0	0	8.0500	0	0	1	0	1	116



```
test_encoded.head()
```

	Pclass	Age	SibSp	Parch	Fare	Embarked_C	Embarked_Q	Embarked_S	Sex_female	Sex_male	cabin_encoded
0	3	34.5	0	0	7.8292	0	1	0	0	1	24
1	3	47.0	1	0	7.0000	0	0	1	1	0	24
2	2	62.0	0	0	9.6875	0	1	0	0	1	24
3	3	27.0	0	0	8.6625	0	0	1	0	1	24
4	3	22.0	1	1	12.2875	0	0	1	1	0	24

Scaling Dataset:

- Dataset ‘train_encoded’ was separated into X and y representing the features to be used in fitting and the label to be predicted respectively. Code:
  - X = train_encoded.drop('Survived', axis=1)
  - y = train_encoded['Survived']
- X and y were further split into training and testing dataset with 9:1 proportion of the whole. Code:
  - X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
- Both the X_train and X_test dataset were scaled using StandardScaler library to get their means 0, and thus help in getting better weights and bias values. Note: Used training dataset for fitting model and transforming both training and testing data using that to maintain uniformity. Code:
  - scaler = StandardScaler()
  - scaler.fit(X_train)
  - X_train_scaled = scaler.transform(X_train)
  - X_test_scaled = scaler.transform(X_test)

Methodology and Techniques used

We have use the following techniques to evaluate the dataset-

1. K-Nearest Neighbors
2. Logistic Regression
3. Support Vector Machines

   We have explained each of them below in detail.

Logistic Regression

- Logistic Regression is a supervised learning algorithm used for binary classification problems, where the target variable has two possible outcomes (e.g., yes/no, true/false).
- It is based on the concept of the logistic function (sigmoid function), which maps any real-valued number to a value between 0 and 1. This function is used to estimate the probability of the binary outcome.
- Logistic Regression assumes a linear relationship between the input features and the log-odds (also known as logit) of the target variable. The log-odds are then transformed into probabilities using the logistic function.
- The algorithm estimates the parameters of the logistic function using maximum likelihood estimation. It aims to find the best-fitting line (or hyperplane in higher dimensions) that separates the two classes, based on the training data.

Process used for Logistic Regression:

- The scaled datasets were fit in the model. Code:
  - logreg = LogisticRegression()
  - logreg.fit(X_train_scaled,y_train)
- Predictions were made using this fit and the accuracy was calculated by comparing the predicted values on ‘X_test_scaled’ with the given values ‘y_test’.Code:
  - y_pred = logreg.predict(X_test_scaled)
  - accuracy_score(y_pred, y_test)
  - Accuracy got: 0.8333333333333334
- Now, the given test set in the problem was scaled using the same fit. Code:
  - test_encoded_scaled = scaler.transform(test_encoded)
- Then, the scaled test dataset was used to get predictions. Code:
  - y_logistic_predicted = logreg.predict(test_encoded_scaled)
- Lastly, these predictions were compared from the given test outputs in the problem. Code:
  - tested_output_value = np.array(tested_output['Survived'])
  - accuracy_score(y_logistic_predicted, tested_output_value)
- Finally, the output of our model gave an accuracy of **<u>0.9354066985645934</u>** on the test dataset.

Support Vector Machine

- It is a Supervised Learning Algorithm that works by the dividing the data points using a hyperplane in N-dimensional space. The dimension of the hyperplane depends on the number of features in the input.
- The hyperplane is taken such that the support vectors, i.e. the points of the classes under consideration nearest to a hyperplane are as far apart as possible.
- What is an SVM kernel- It is a function that takes low dimensional input and transforms it into higher dimensions. It is useful if the categories under consideration have non separable points. We can use the kernel function to convert it to a higher dimension where they are separable and hence we can take a hyperplane.

Advantages of SVM

- Effective in high dimensional cases and complex problems.
- Memory efficient as it uses a subset of training points (support vectors) in the decision function.
- Different kernel functions can be specified for different input and hence we can have custom kernels.
- It is relatively stable i.e. a small change to the data may not affect the hyperplane.

Process used for SVM:

- All the steps done for Logistic regression model were repeated for SVM, starting from splitting the training dataset till the accuracy of the prediction on testing data. Code:
  - x_train, x_test, yo_train, yo_test = train_test_split(X, y, test_size=0.1, random_state=54)
  - scaler.fit(x_train)
  - x_train_scaled = scaler.transform(x_train)
  - x_test_scaled = scaler.transform(x_test)
  - model2 = SVC()
  - model2.fit(x_train_scaled, yo_train)
  - yo_pred = model2.predict(x_test_scaled)
  - accuracy_score(yo_pred,yo_test)
  - Accuracy : 0.8222222222222222
- Predicting on the given test data:
  - Test_encoded_scaled = scaler.transform(test_encoded)
  - y_SVM_predicted = model2.predict(Test_encoded_scaled)
  - accuracy_score(y_SVM_predicted, tested_output_value)
- Finally, the accuracy on the prediction on given tested data came out to be **0.9569377990430622**

K-Nearest Neighbors (KNN)

- KNN is an algorithm based on Supervised Learning.
- It works by taking K Neighbors of the give test case and putting it into the category closest to that of the test case. How ‘close’ it is determined by taking its distance with its neighbors and seeing which category most neighbors fall into.
- This is called a ‘lazy learner’ as it classifies and trains a model when it gets the test data and not beforehand.
- K is generally taken as odd to prevent ties from occurring in case of binary classification. The choice of K is tricky as keeping it too small might make model inaccurate but keeping it too large may be computationally expensive
- It is generally taken around the square root of number of data points. Here we had data on around 891 passengers for training and hence took K as 29.

Advantages of KNN

- It is easy to understand and implement.
- Since there is no explicit training step we can keep adding new data to the dataset, the prediction is adjusted without training a new model.
- There is only one parameter- the value K, which makes tuning the parameter easier. We can also use different distance metrics such as Euclidean, Manhattan, Minnowski etc.

Process used for KNN

- We took the dataset as input. We then replaced blank values in columns such as ‘Age’ and ‘Fare’ with the average of those columns.
- Attributes such as Name, Passenger Id were not considered as they were unique and hence would not be useful.
- Non numerical attributes such as ‘Embarked’, ‘Sex’ and ‘Cabin’ were converted to numerical ones as we need numerical values to take distance.
- We then scaled down the data to lie within a certain range

Code snippets for KNN

```
blank_not_accepted = [ 'Age','Fare']
for column in blank_not_accepted:
   mean = int(dataset[column].mean(skipna= True))
   dataset[column]=dataset[column].replace(np.NaN, mean)
   mean1 = int(test[column].mean(skipna= True))
   test[column]=test[column].replace(np.NaN, mean1)
```

```
dataset=dataset.drop( 'Ticket',axis=1)
test=test.drop('Ticket',axis=1)

dataset=dataset.drop( 'PassengerId',axis=1)
test=test.drop('PassengerId',axis=1)

dataset.Sex[dataset.Sex ==  'male'] =0
dataset.Sex[dataset.Sex ==  'female'] = 1
```

Code Snippets and Observation

```
classifier=KNeighborsClassifier(n_neighbors=29,p=2,metric='euclidean')
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)
print(accuracy_score(y_test,y_pred))
```

KNN model

Accuracy obtained was 0.8222222222222222

Code Snippets and observation

```
test_scaled=scaler.transform(test)
y_pred_knn=classifier.predict(test_scaled)
tested_output_value=np.array(test_op[ 'Survived' ])
y_pred_knn.shape == tested_output_value.shape

print(accuracy_score(y_pred_knn,tested_output_value))
```

Accuracy obtained against test data was `0.9449760765550239`

Conclusion:

1. Accuracy for Logistic Regression: 93.54066985645934 %
2. Accuracy for SVM: 95.69377990430622 %
3. Accuracy for KNN: 94.49760765550239 %