Titanic Classification

Building a predictive model to determine the likelihood of survival for passengers on the Titanic using data science techniques in Python.

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
!kaggle competitions download -c titanic
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.jso
Downloading titanic.zip to /content
  0% 0.00/34.1k [00:00<?, ?B/s]
100% 34.1k/34.1k [00:00<00:00, 55.5MB/s]
```

```
import zipfile
zip_ref = zipfile.ZipFile('/content/titanic.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()
```

Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
titanic_data = pd.read_csv('/content/train.csv')
```

```
titanic_data.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

visualize the correlation matrix

```
sns.heatmap(titanic_data.corr(), cmap="Spectral")
plt.show
```

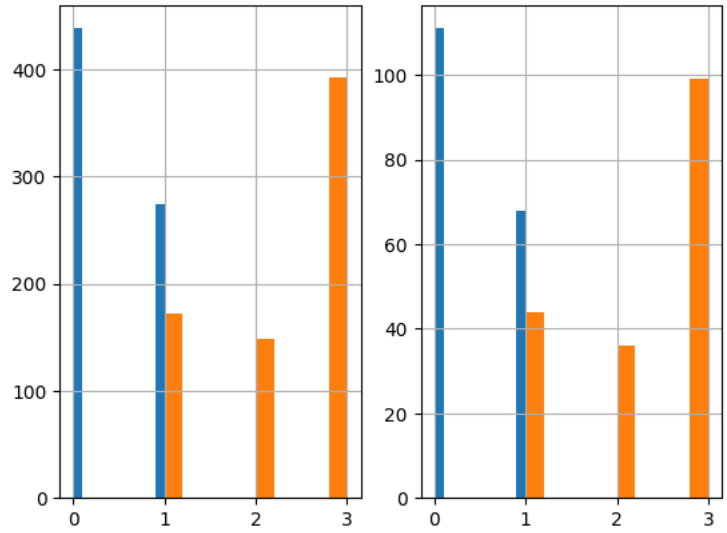
```
<ipython-input-17-e66693fdec0c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  sns.heatmap(titanic_data.corr(), cmap="Spectral")
<function matplotlib.pyplot.show(close=None, block=None)>
```

split the titanic_data DataFrame into stratified train and test sets.


```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2)
for train_indices, test_indices in split.split(titanic_data, titanic_data[["Survived","Pclass","Sex"]]):
  strat_train_set = titanic_data.loc[train_indices]
  strat_test_set = titanic_data.loc[test_indices]
```

creating two subplots and visualize the distribution of two features in both the training and testing sets

```
plt.subplot(1,2,1)
strat_train_set['Survived'].hist()
strat_train_set['Pclass'].hist()

plt.subplot(1,2,2)
strat_test_set['Survived'].hist()
strat_test_set['Pclass'].hist()

plt.show()
#Blue Bars are Survival
#Orange Bars are Pclass
```




```
strat_train_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 712 entries, 69 to 285
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  712 non-null    int64
 1   Survived     712 non-null    int64
 2   Pclass       712 non-null    int64
 3   Name         712 non-null    object
 4   Sex          712 non-null    object
 5   Age          573 non-null    float64
 6   SibSp        712 non-null    int64
 7   Parch        712 non-null    int64
 8   Ticket       712 non-null    object
 9   Fare         712 non-null    float64
 10  Cabin        157 non-null    object
 11  Embarked     710 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 72.3+ KB
```

Provides a convenient way to fill missing values in the "Age" column of a DataFrame

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.impute import SimpleImputer

class AgeImputer(BaseEstimator, TransformerMixin):

  def fit(self,X, y=None):
    return self

  def transform(self, X):
    imputer = SimpleImputer(strategy = "mean")
    X['Age'] = imputer.fit_transform(X[['Age']])
    return X
```

FeatureEncoder class performs one-hot encoding for both "Embarked" and "Sex" features in the X DataFrame. It utilizes the OneHotEncoder class from scikit-learn and creates new columns in the DataFrame to store the encoded data.

```
from sklearn.preprocessing import OneHotEncoder

class FeatureEncoder(BaseEstimator, TransformerMixin):

  def fit(self, X, y=None):
    return self

  def transform(self,X):
    encoder = OneHotEncoder()
    matrix = encoder.fit_transform(X[['Embarked']]).toarray()

    column_names = ["C", "S", "Q", "N"]

    for i in range(len(matrix.T)):
      X[column_names[i]] = matrix.T[i]

    matrix = encoder.fit_transform(X[['Sex']]).toarray()

    column_names = ["Female", "Male"]

    for i in range(len(matrix.T)):
      X[column_names[i]] = matrix.T[i]

    return X
```

FeatureDropper class removes several features from the X DataFrame based on a pre-defined list

```
class FeatureDropper(BaseEstimator, TransformerMixin):

  def fit(self, X, y=None):
    return self

  def transform(self, X):
    return X.drop(["Embarked","Name","Ticket","Cabin","Sex","N"], axis=1, errors = 'ignore')
```

Defining a pipeline that combines the three custom transformer classes

```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([("ageimputer", AgeImputer()),
                     ("featureencoder", FeatureEncoder()),
                      ("featuredropper", FeatureDropper())])


strat_train_set = pipeline.fit_transform(strat_train_set)


strat_train_set
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	C	S	Q	Female	Male
69	70	0	3	26.000000	2	0	8.6625	0.0	0.0	1.0	0.0	1.0
660	661	1	1	50.000000	2	0	133.6500	0.0	0.0	1.0	0.0	1.0
280	281	0	3	65.000000	0	0	7.7500	0.0	1.0	0.0	0.0	1.0
204	205	1	3	18.000000	0	0	8.0500	0.0	0.0	1.0	0.0	1.0

```
strat_train_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 712 entries, 69 to 285
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  712 non-null    int64  
 1   Survived     712 non-null    int64  
 2   Pclass       712 non-null    int64  
 3   Age          712 non-null    float64
 4   SibSp        712 non-null    int64  
 5   Parch        712 non-null    int64  
 6   Fare         712 non-null    float64
 7   C            712 non-null    float64
 8   S            712 non-null    float64
 9   Q            712 non-null    float64
 10  Female       712 non-null    float64
 11  Male         712 non-null    float64
dtypes: float64(7), int64(5)
memory usage: 72.3 KB
```

Prepares the data for training a machine learning model by:

1) Dropping the target variable from the features.

2) Separating the target variable into a dedicated variable.

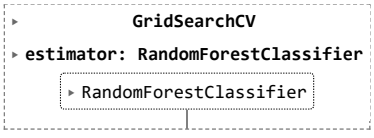
3) Applying standard scaling to normalize the features.

4) Converting the target variable to a NumPy array.

```
from sklearn.preprocessing import StandardScaler

X = strat_train_set.drop(['Survived'], axis = 1)
y = strat_train_set['Survived']

scaler = StandardScaler()
X_data = scaler.fit_transform(X)
y_data = y.to_numpy()
```

Optimizing the hyperparameters of a machine learning model using GridSearchCV and By exploring different combinations of parameters and evaluating their performance on held-out data, we find the configuration that leads to the most accurate and generalizable model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

clf = RandomForestClassifier()

param_grid = [
    {"n_estimators":[10,100,200,500], "max_depth": [None,5,10], "min_samples_split": [2,3,4]}
]

grid_search = GridSearchCV(clf, param_grid, cv=3, scoring = 'accuracy', return_train_score=True)
grid_search.fit(X_data, y_data)
```



```
final_clf = grid_search.best_estimator_
```

```
final_clf
```

```
RandomForestClassifier(max_depth=5, n_estimators=200)
```

```
strat_test_set = pipeline.fit_transform(strat_test_set)
```

```
X_test = strat_test_set.drop(['Survived'], axis=1)
y_test = strat_test_set['Survived']

scaler = StandardScaler()
X_data_test = scaler.fit_transform(X_test)
y_data_test = y_test.to_numpy()
```

```
final_clf.score(X_data_test, y_data_test)
```

```
0.8268156424581006
```

```
final_data = pipeline.fit_transform(titanic_data)
```

```
final_data
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	C	S	Q	Female	Male
0	1	0	3	22.000000	1	0	7.2500	0.0	0.0	1.0	0.0	1.0
1	2	1	1	38.000000	1	0	71.2833	1.0	0.0	0.0	1.0	0.0
2	3	1	3	26.000000	0	0	7.9250	0.0	0.0	1.0	1.0	0.0
3	4	1	1	35.000000	1	0	53.1000	0.0	0.0	1.0	1.0	0.0
4	5	0	3	35.000000	0	0	8.0500	0.0	0.0	1.0	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	27.000000	0	0	13.0000	0.0	0.0	1.0	0.0	1.0
887	888	1	1	19.000000	0	0	30.0000	0.0	0.0	1.0	1.0	0.0
888	889	0	3	29.699118	1	2	23.4500	0.0	0.0	1.0	1.0	0.0
889	890	1	1	26.000000	0	0	30.0000	1.0	0.0	0.0	0.0	1.0
890	891	0	3	32.000000	0	0	7.7500	0.0	1.0	0.0	0.0	1.0

891 rows × 12 columns

```
X_final = final_data.drop(['Survived'], axis=1)
y_final = final_data['Survived']

scaler = StandardScaler()
X_data_final = scaler.fit_transform(X_final)
y_data_final = y_final.to_numpy()
```

```
prod_clf = RandomForestClassifier()

param_grid = [
    {"n_estimators":[10,100,200,500], "max_depth": [None,5,10], "min_samples_split": [2,3,4]}
]

grid_search = GridSearchCV(prod_clf, param_grid, cv=3, scoring = 'accuracy', return_train_score=True)
grid_search.fit(X_data_final, y_data_final)
```

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

```
prod_final_clf = grid_search.best_estimator_
```

```
prod_final_clf
```

```
RandomForestClassifier(max_depth=5, min_samples_split=3)
```

```
titanic_test_data = pd.read_csv("/content/test.csv")
```

```
final_test_data = pipeline.fit_transform(titanic_test_data)
```

```
X_final_test = final_test_data
X_final_test = X_final_test.fillna(method="ffill")

scaler = StandardScaler()
X_data_final_test = scaler.fit_transform(X_final_test)
```

```
predictions = prod_final_clf.predict(X_data_final_test)
```

```
final_df = pd.DataFrame(titanic_test_data['PassengerId'])
final_df['Survived'] = predictions
final_df.to_csv("/content/predictions.csv", index = False)
```

```
final_df
```

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	1
...	...	...
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

418 rows × 2 columns

```
predictions
```

```
array([0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0])
```