Applied Machine Learning
Assignment 1

Name	YAP LI XEN
Student ID	P7414389
Lecturer	MR. PAUL

PART A: CLASSIFICATION

(1) How is your prediction task defined? And what is the meaning of the output variable?

- The prediction task is to predict categorical data “Survived”
- The output variables “Survived” consists of 2 values (1=“Survived”, 0=”Not Survived”)
- This is 2-class variables

(2) How do you represent your data as features?

- Target: Survived
- Features: All columns except Survived

A	B	C	D	E	F	G	H	I	J	K	L
PassengerId	**Survived**	**Pclass**	**Name**	**Sex**	**Age**	**SibSp**	**Parch**	**Ticket**	**Fare**	**Cabin**	**Embarked**
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cumings, Mrs. John Bradley (I	female	38	1	0	PC 17599	71.2833	C85	C

(3) Did you process the features in any way?

- Feature Selection

To remove features that doesn't bring impact or less important to the classification

```
df = df.drop(['PassengerId', 'Name', 'Ticket'], 1)
df.shape
```

- Column: Fare

To group Fare into multiple folds

```
fare_bins=[0,10,20,40,60,80,100,200,600]
fare_labels=[1,2,3,4,5,6,7,8]
df['Fare_Group'] = pd.cut(df['Fare'], bins=fare_bins, labels=fare_labels, right=False)
df = df.drop('Fare', 1)
print(df.head())
df.shape
```

```
   Survived  Pclass     Sex   Age Cabin Embarked hasFamilyAboard Fare_Group
0         0       3    male  22.0   NaN        S              No          1
1         1       1  female  38.0   C85        C              No          5
2         1       3  female  26.0   NaN        S              No          1
3         1       1  female  35.0  C123        S              No          4
4         0       3    male  35.0   NaN        S              No          1
```

- Column: Cabin

There are many missing data in this column. Transform into "Yes" (with records) and "No" (without record)

```
df['Cabin'].fillna('No', inplace=True)
df['Cabin'].replace(regex=r'^((?!No).)*$',value='Yes',inplace=True)
print(df.head())
df.shape
```

```
   Survived  Pclass     Sex   Age Cabin Embarked hasFamilyAboard Fare_Group
0         0       3    male  22.0    No        S              No          1
1         1       1  female  38.0   Yes        C              No          5
2         1       3  female  26.0    No        S              No          1
3         1       1  female  35.0   Yes        S              No          4
4         0       3    male  35.0    No        S              No          1
```

- ### Column: Age
  - To replace missing value (NaN) with mean value of Age
  - To normalise Age into 3 categories
    - Children (0-12)
    - Adult (13-59)
    - Elderly (60 and above)

```python
df['Age'].fillna(df['Age'].mean(), inplace=True)
age_bins=[0,13,60,120]
age_labels=['Children','Adult','Elderly']
df['Age_Group'] = pd.cut(df['Age'], bins=age_bins, labels=age_labels, right=False)
df = df.drop('Age', 1)
print(df.head())
df.shape
```

```
   Survived  Pclass     Sex Cabin Embarked hasFamilyAboard Fare_Group  \
0         0       3    male    No        S              No          1
1         1       1  female   Yes        C              No          5
2         1       3  female    No        S              No          1
3         1       1  female   Yes        S              No          4
4         0       3    male    No        S              No          1

  Age_Group
0     Adult
1     Adult
2     Adult
3     Adult
4     Adult
```

- ### Column: Embarked

To replace missing value (NaN) with mode value of Embarked.

```python
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
print(df.head())
df.shape
```

```
   Survived  Pclass     Sex Cabin Embarked hasFamilyAboard Fare_Group  \
0         0       3    male    No        S              No          1
1         1       1  female   Yes        C              No          5
2         1       3  female    No        S              No          1
3         1       1  female   Yes        S              No          4
4         0       3    male    No        S              No          1
```

- ### Encoding

Encode all categorical columns

```python
df = pd.get_dummies(df)
print(df.head())
df.shape
```

```
   Survived  Pclass  Sex_female  Sex_male  Cabin_No  Cabin_Yes  Embarked_C  \
0         0       3           0         1         1          0           0
1         1       1           1         0         0          1           1
2         1       3           1         0         1          0           0
3         1       1           1         0         0          1           0
4         0       3           0         1         1          0           0

   Embarked_Q  Embarked_S  hasFamilyAboard_No  ...  Fare_Group_2  \
0           0           1                   1  ...             0
1           0           0                   1  ...             0
2           0           1                   1  ...             0
3           0           1                   1  ...             0
4           0           1                   1  ...             0

   Fare_Group_3  Fare_Group_4  Fare_Group_5  Fare_Group_6  Fare_Group_7  \
0             0             0             0             0             0
1             0             0             1             0             0
2             0             0             0             0             0
3             0             1             0             0             0
4             0             0             0             0             0

   Fare_Group_8  Age_Group_Children  Age_Group_Adult  Age_Group_Elderly
0             0                   0                1                  0
1             0                   0                1                  0
2             0                   0                1                  0
3             0                   0                1                  0
4             0                   0                1                  0
```

(4) Did you bring in any additional sources of data?

- Column: hasFamilyAboard

To normalise the number of sibling/spouse/parents/children aboard the Titanic into single column **hasFamilyAboard** with the value "Yes" or "No"

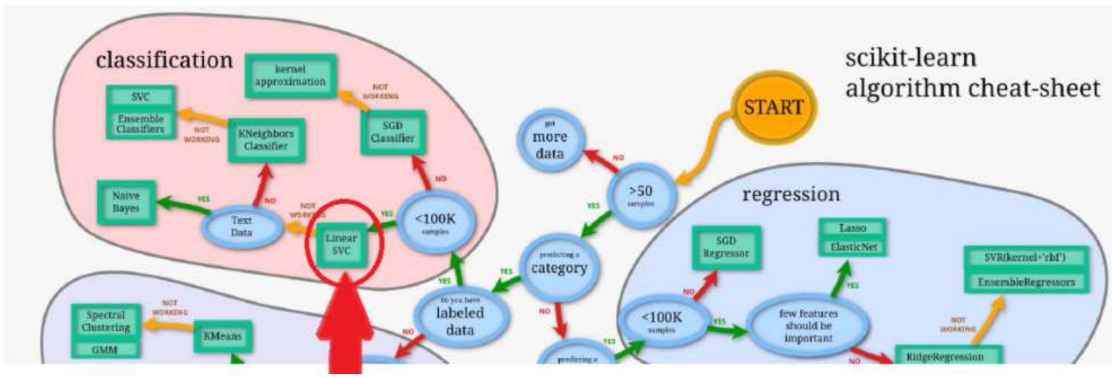
```
df['hasFamilyAboard'] = np.where((df['SibSp'] > 0) & (df['Parch'] > 0), 'Yes', 'No')
df = df.drop(['SibSp', 'Parch'], 1)
print(df.head())
df.shape
```

```
   Survived  Pclass     Sex   Age     Fare Cabin Embarked hasFamilyAboard
0         0       3    male  22.0   7.2500   NaN        S              No
1         1       1  female  38.0  71.2833   C85        C              No
2         1       3  female  26.0   7.9250   NaN        S              No
3         1       1  female  35.0  53.1000  C123        S              No
4         0       3    male  35.0   8.0500   NaN        S              No
```

(5) How did you select which learning algorithms to use?

- Referring to scikit-learn algorithm cheat-sheet recommendation

Linear SVC is selected as training model following the cheat-sheet in our use case



(6) Did you try to tune the hyperparameters of the learning algorithm, and in that case how?

- Hyperparameter Tuning: GridSearchCV
- To include a range of values in chosen parameter and assign to param grid
- To set cv to 5 (split the train-validation data into 5 folds)

To use GridSearchCV which includes Cross Validation to identify best paramter and best score.

```
from sklearn.model_selection import GridSearchCV

svm = LinearSVC()
param_grid = {'C': [0.01,0.1,1.0,10.0,100.0]}
grid_search = GridSearchCV(svm, param_grid=param_grid, cv=5, verbose=3, return_train_score=True)
grid_search.fit(X_train, y_train);
```

- Then it will generate the best param and best score

```
print("Best Param: {}".format(grid_search.best_params_))
print("Best Score: {:.2f}%".format(grid_search.best_score_*100))
```

```
Best Param: {'C': 0.01}
Best Score: 78.93%
```

(7) How do you evaluate the quality of your system?

- Score the trained model using both train data and test data, then compare the result
- We can see how well the trained model can predict the test data
- The comparison can also help us to determine if this is under-fitting, appropriate-fitting or over-fitting

```
training_data_score = model.score(X_train, y_train)
print("Training Data Score: {:.2f}%".format(training_data_score*100))

test_data_score = model.score(X_test, y_test)
print("Test Data Score: {:.2f}%".format(test_data_score*100))
```

```
Training Data Score: 80.76%
Test Data Score: 79.89%
```

(8) How well does your system compare to a stupid baseline?

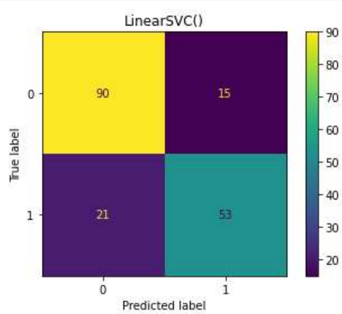
-

(9) Can you say anything about the errors that the system makes? For a classification task, you may consider a confusion matrix.

- Using confusion matrix, it will show us the total number of
  - True Positive
  - False Positive
  - True Negative
  - False Negative

```
from sklearn.metrics import plot_confusion_matrix

plot_confusion_matrix(model, X_test, y_test)
plt.title(model)
plt.show()
```



- Using classification report
- With the true positive (TP) and and false positive (FP), we will be able to calculate the precision
- With the true positive (TP) and and false negative (FN), we will be able to calculate the recall

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.81      0.86      0.83       105
           1       0.78      0.72      0.75        74

    accuracy                           0.80       179
   macro avg       0.80      0.79      0.79       179
weighted avg       0.80      0.80      0.80       179
```

(10) Is it possible to say something about which features the model considers important? (Whether this is possible depends on the type of classifier you are using)

- Once all data has transformed into numerical, we are able to determine which features are correlation to the target.

```
corr = df.corr()
corr.sort_values(["Survived"], ascending = False, inplace = True)
print(corr["Survived"])
```

```
Survived              1.000000
Sex_female            0.543351
Cabin_Yes             0.316912
Embarked_C            0.168240
Fare_Group_6          0.162583
Fare_Group_7          0.150716
Age_Group_Children    0.116691
Fare_Group_4          0.099358
Fare_Group_8          0.098513
Fare_Group_5          0.055730
Fare_Group_3          0.051066
hasFamilyAboard_Yes   0.047257
Fare_Group_2          0.042006
Embarked_Q            0.003650
Age_Group_Elderly    -0.040857
hasFamilyAboard_No   -0.047257
Age_Group_Adult      -0.078779
Embarked_S           -0.149683
Fare_Group_1         -0.295081
Cabin_No             -0.316912
Pclass               -0.338481
Sex_male             -0.543351
Name: Survived, dtype: float64
```