Mini Project: Option 2 (Titanic)

Table of Contents

Part A: Exploration of Training and Test Datasets

There are 891 training samples and 418 test samples. The attributes in training and test data are arranged in columns. The *info()* function can provide an overview of a dataset where it outputs the total number of samples in a dataset, the number of attributes, and attributes' data type. Most importantly, it can show the number of null values in each attribute. According to Figure 1, there are null values in "Age", "Cabin", and "Embarked" attributes in the training data because the non-null counts do not tally with the number of entries. In the test data, there are null values in "Age", "Fare", and "Cabin" attributes. The summary of training and test datasets, and features is presented in Tables 1 and 2, respectively.

```
# Overview of training data
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
# Overview of test data
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

Figure 1: Feature dimension and number of samples in training data and test data.

Table 1: Summary of Training and Test Datasets

	Training Dataset	**Test Dataset**
Number of Samples	891	418
Number of Features	12	11
Shape	(891,12)	(418,11)

Table 2: Summary of Features

Attribute	**Description**	**Attribute Data Type**
PassengerId	Unique values in the range (1,1309) to identify every passenger.	Integer
Survived	Labels to predict in testing 1 = Survived 0 = Did not survive	Integer
Pclass	Ticket Class 1 = 1st Class 2 = 2nd Class 3 = 3rd Class	Integer
Name	Name of passenger	String
Sex	Male or Female	String
Age	Age in years	Float
SibSp	Number of siblings/spouse on board with passenger	Integer
Parch	Number of parents/children on board with passenger	Integer
Ticket	Ticket number	String
Fare	Ticket price	Float
Cabin	Cabin number	String
Embarked	Port of embarkation C = Cherbourg Q = Queenstown S = Southampton	String

The mean and variance of the values in each attribute can be computed using the *describe()* function, as shown in Figures 2 and 3. The mean and variance can only be computed if the attributes are integer or float data types. The summary of mean and variance of the values in each attribute is presented in Table 3.

```
train.describe()
```

	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

Figure 2: Mean and variance of the values in each attribute in training data.

```
test.describe()
```

	PassengerId	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200

Figure 3: Mean and variance of the values in each attribute in test data.

Table 3: Mean and Variance of Training and Test Datasets

Attributes	**Training Dataset**		**Test Dataset**	
	Mean	**Variance**	**Mean**	**Variance**
PassengerId	446	66231	1100.5	14595.167
Survived	0.384	0.237	-	-
Pclass	2.309	0.699	2.266	0.709
Age	29.699	211.019	30.273	201.107
SibSp	0.523	1.216	0.447	0.804
Parch	0.382	0.650	0.392	0.963
Fare	32.204	2469.437	35.627	3125.657

Based on Table 3, it can be inferred that “PassengerId”, “Age”, and “Fare” attributes have high variance. These features might not be good for training a classifier because of its high variance. More details on these features will be discussed in Part D.

Part B: Build Classifier

The training data is further split into 80% for training and 20% for validation. After feature engineering, the number of attributes used for training is 20. The summary of training, validation, and test datasets is showed in Table 4.

Table 4: Summary of Training, Validation, and Test Datasets

Training	**Validation**	**Test**
(712,20)	(179,20)	(418,20)

B.1 Support Vector Machine

Training Parameters:

- $Kernel =' linear'$

Support Vector Machine (SVM) is selected because it is also suitable for binary classification problem. The training strategy is to use the default training parameters except changing the SVM kernel to "linear". Linear kernel is chosen because the training dataset is linearly separable by a single line. Furthermore, training a SVM using a linear kernel is faster than other kernels.

B.2 Random Forest Classifier

Training Parameters:

- $Number\ of\ Trees = 100$
- $Max\ Depth = None$
- $Minimum\ number\ of\ samples\ required\ to\ split = 2$

Random Forest Classifier is chosen because it is suitable to work on a dataset with many attributes. Random Forest Classifier comprises of multiple decision trees. The classification result of Random Forest Classifier is decided by the combination of all decision trees through voting. This made Random Forest Classifier more accurate and stable than a single decision tree. Therefore, Random Forest Classifier is chosen for this classification task.

The training strategy is to use the default parameters of Random Forest Classifier and compare the training result with other classifiers.

B.3 Neural Network

Initial Structure of Model and Training Parameters:

- $32 - 64 - 128 - 64 - 32 - 1\ neural\ network$
- $Optimizer = Adam$
- $Learning\ Rate = 0.001$
- $Loss\ Function = Binary\ Crossentropy$
- $Batch\ Size = 8$
- $Training\ Epochs = 100$

The structure of the model is designed to double the number of neurons in the previous layer and halve the number of neurons when the layers get near the output layer. Training a neural network can involve large number of training parameters which results in high computational complexity. Therefore, this structure design can help to reduce the number of training parameters required.

The input to the first hidden layer in the neural network has shape $(N, 20)$, where N is the number of training samples and each training sample has 20 features. The output of the neural network is either a 1 or 0 which is produced by the sigmoid activation function.

Adam optimizer is selected because it is invariant to diagonal rescaling of gradients and is suitable for optimizing large number of parameters.

Learning rate is set as default value (0.001) for the initial training.

Binary crossentropy is selected as the loss function because its outputs are either 1 or 0 which makes it suitable for this binary classification task.

Batch size is a hyperparameter of gradient descent which controls the number of training samples used for estimating the gradient error. For a small training data, it is preferably set as 8.

Training Epochs is set as 100 because many iterations are required to train a neural network.

Codes:

```
# Fully Connected Neural Network
def model_nn():
    model = Sequential()
    model.add(InputLayer(input_shape=(X_train.shape[1],)))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    return model
```

```
model = model_nn()
model.summary()

# Compile the model
tf.random.set_seed(1234)
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 32)                672
_________________________________________________________________
dense_8 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_9 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_10 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_11 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 33
=================================================================
Total params: 21,473
Trainable params: 21,473
Non-trainable params: 0
_________________________________________________________________
```

Part C: Training Classifiers

Since the size of the training dataset and validation dataset are small, it is more accurate to get the average validation accuracy using K-Fold Cross Validation. Therefore, Neural Network, SVM, and Random Forest Classifier are trained using Stratified 10-Fold Cross Validation to get the average validation accuracies. Based on the classifiers' average validation accuracies, the best classifier will be used for predicting the test dataset.

C.1 Neural Network

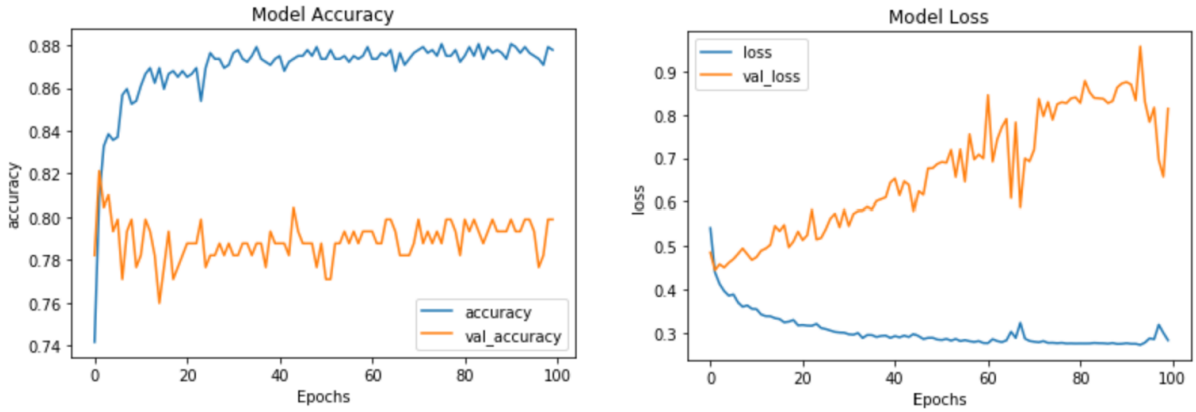

Figure 4: Initial Training Results of Neural Network.

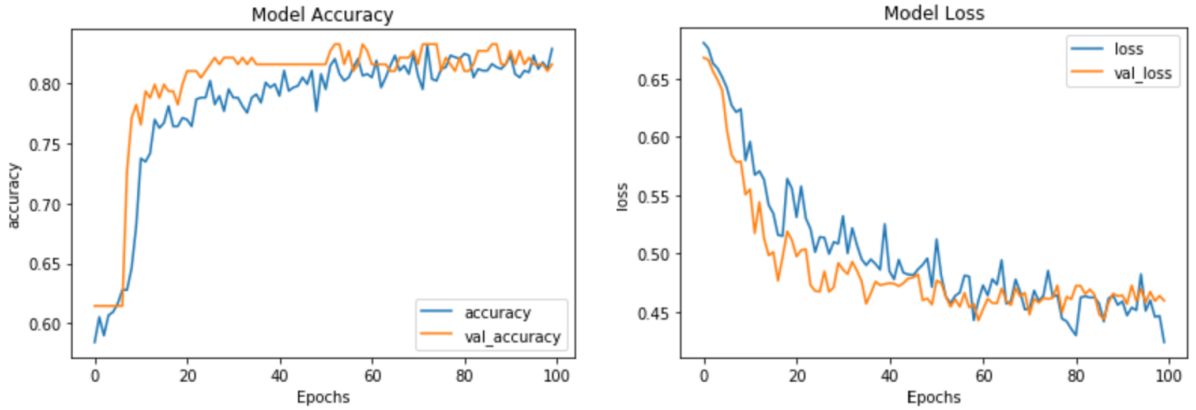

Figure 5: Best Training Results of Neural Network.

Based on the initial training results showed in Figure 4, it can be inferred that the validation loss starts to increase after the training started. This shows that the model did not converge. To overcome convergence problem, the training hyper-parameters need to be fine-tuned. Fine-tuning is done empirically until the best training hyper-parameters are found.

Firstly, the learning rate of Adam optimizer is gradually reduced to solve the convergence problem. After the learning rate was gradually reduced to 0.0005, the validation loss was still higher than training loss. Dropout layer was added after each fully connected layer with its dropout value set to between 0 and 1. Next, the number of hidden layers was increased from 5 to 9.

When there was still no improvement to the validation loss, the feature dimension was reduced from 20 to 15 by dropping “FareGroup_0”, “FareGroup_1”, “FareGroup_2”, “FareGroup_3”, and “FareGroup_4” attributes. The reason to drop these attributes is the number of features is too large for a small validation dataset.

When the model is trained on the reduced feature dataset, the training performance improved significantly as shown in Figure 5. The final neural network structure is showed in Figure 6. The reduced feature dataset is used for training other classifiers as well.

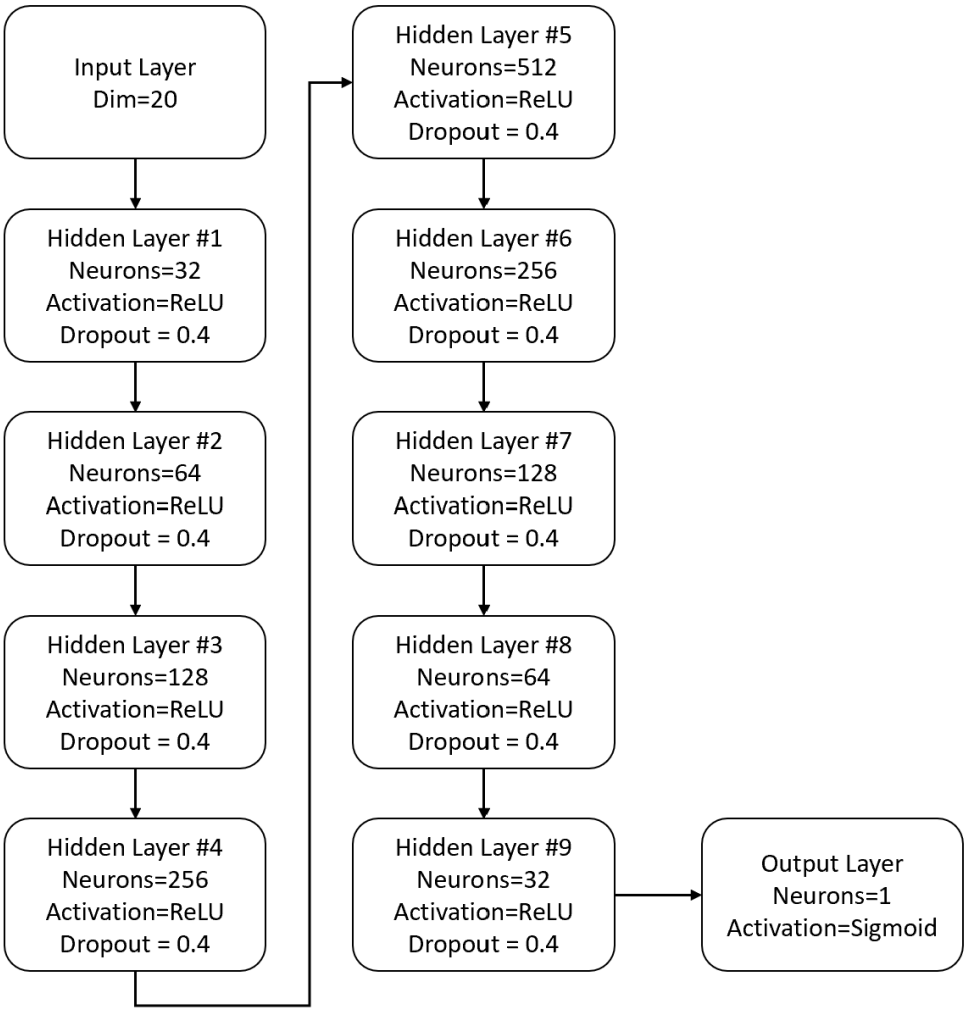

Figure 6: Final Neural Network Structure.

Stratified 10-Fold Cross Validation

Next, the neural network was trained using Stratified 10-Fold Cross Validation. It achieved an average validation accuracy of 81.0% on the validation dataset. The method to train neural network using cross validation is shown in Figures 7 and 8.

```
# 10-fold Cross Validation

batch_size = 8
epochs = 100
kfold = 10

skf = StratifiedKFold(n_splits=kfold, shuffle=True, random_state=111)
skf_accuracy = []

for i, (train_index, val_index) in enumerate(skf.split(X_train, y_train)):
    print('[Fold %d/%d]' % (i + 1, kfold))
    X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]
    model = model_nn()
    tf.random.set_seed(1234)

    # Compile the model
    optimizer = Adam(learning_rate=0.0005)
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

    # Define callbacks list
    filepath = 'nn_crossval.hdf5'
    callbacks_list = [EarlyStopping(monitor='val_loss', verbose=1, patience=20),
                      ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_best_only=True)]
    history = model.fit(X_train_fold, y_train_fold, batch_size=batch_size, epochs=epochs,
                        validation_data=(X_val_fold, y_val_fold), verbose=1, callbacks=None)
    plot_graphs(history, 'accuracy', 'Model Accuracy')
    plot_graphs(history, 'loss', 'Model Loss')
    score = model.evaluate(X_val_fold, y_val_fold)
    val_acc = score[1]
    skf_accuracy.append(val_acc)
```

Figure 7: Method to train neural network using Stratified 10-Fold Cross Validation.

```
# Print the output
print('List of validation accuracy at each fold: {}'.format(skf_accuracy))
print('\nMaximum Validation Accuracy: {}%'.format(round(max(skf_accuracy)*100, 3)))
print('Minimum Validation Accuracy: {}%'.format(round(min(skf_accuracy)*100, 3)))
print('Overall Validation Accuracy: {}%'.format(round(mean(skf_accuracy)*100, 3)))
print('Standard Deviation is: {}'.format(round(stdev(skf_accuracy), 3)))
```

```
List of validation accuracy at each fold: [0.8333333134651184, 0.8202247023582458, 0.8089887499809265, 0.8089887499809265,
0.8202247023582458, 0.7415730357170105, 0.7977527976036072, 0.8539325594902039, 0.7865168452262878, 0.8202247023582458]

Maximum Validation Accuracy: 85.393%
Minimum Validation Accuracy: 74.157%
Overall Validation Accuracy: 80.918%
Standard Deviation is: 0.03
```

Figure 8: Method to achieve the average validation accuracy.

C.2 Support Vector Machine

SVM achieved an average validation accuracy of 80.5%. The method is shown in Figure 9.

```
# Support Vector Machine
svm = SVC(kernel='linear', random_state=7, verbose=True)
svm_model = svm.fit(X_train, y_train)
svm_score = cross_val_score(svm_model, X_val, y_val, cv=10)
print('\nList of validation accuracy at each fold: {}'.format(svm_score))
print('\nMaximum Validation Accuracy: {}%'.format(round(max(svm_score)*100, 3)))
print('Minimum Validation Accuracy: {}%'.format(round(min(svm_score)*100, 3)))
print('Overall Validation Accuracy: {}%'.format(round(mean(svm_score)*100, 3)))
#print("\nValidation Accuracy: {}".format(round(svm_model.score(X_val, y_val), 3)))
```

```
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
List of validation accuracy at each fold: [0.88888889 0.94444444 0.61111111 0.77777778 0.72222222 0.88888889
 0.88888889 0.83333333 0.66666667 0.82352941]

Maximum Validation Accuracy: 94.444%
Minimum Validation Accuracy: 61.111%
Overall Validation Accuracy: 80.458%
```

Figure 9: Method to train SVM using cross validation.

C.3 Random Forest Classifier

Random Forest Classifier achieved an average validation accuracy of 79.3%. The method is shown in Figure 10.

```
# Random Forest Classifier
rf = RandomForestClassifier(random_state=4, verbose=2)
rf_model = rf.fit(X_train, y_train)
rf_score = cross_val_score(rf_model, X_val, y_val, cv=10)
print('List of validation accuracy at each fold: {}'.format(rf_score))
print('\nMaximum Validation Accuracy: {}%'.format(round(max(rf_score)*100, 3)))
print('Minimum Validation Accuracy: {}%'.format(round(min(rf_score)*100, 3)))
print('Overall Validation Accuracy: {}%'.format(round(mean(rf_score)*100, 3)))
#print("\nValidation Accuracy: {}".format(round(rf_model.score(X_val, y_val), 3)))
```

```
building tree 92 of 100
building tree 93 of 100
building tree 94 of 100
building tree 95 of 100
building tree 96 of 100
building tree 97 of 100
building tree 98 of 100
building tree 99 of 100
building tree 100 of 100
List of validation accuracy at each fold: [0.83333333 0.94444444 0.66666667 0.83333333 0.72222222 0.94444444
 0.83333333 0.83333333 0.61111111 0.70588235]

Maximum Validation Accuracy: 94.444%
Minimum Validation Accuracy: 61.111%
Overall Validation Accuracy: 79.281%
```

Figure 10: Method to train Random Forest Classifier using cross validation.

The average validation accuracies of Random Forest Classifier, Support Vector Machine, and Neural Network are summarised in Table 5. Results showed that Neural Network has the best average validation accuracy. Therefore, Neural Network is selected for predicting the test dataset.

Table 5: Classification Accuracy of Different Classifiers

Classifier	**Validation Accuracy**
Neural Network	80.9%
Support Vector Machine	80.5%
Random Forest	79.3%

Part D: Data Pre-processing

D.1 Handling Missing Values

```
# Check for null values in training data
train.isnull().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
```

```
# Check for null values in test data
test.isnull().sum()
```

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

Figure 11: Number of missing values in each attribute in training data and test data.

Firstly, the missing values in "Age" attribute were handled by calculating the mean age and median age using the method showed in Figure 12.

```
print("The mean age in training data is: {}".format(round(X_train['Age'].mean(axis=0, skipna=True),2)))
print("The median age in training data is: {}".format(X_train['Age'].median()))

print("The mean age in test data is: {}".format(round(X_test['Age'].mean(axis=0, skipna=True),2)))
print("The median age in test data is: {}".format(X_test['Age'].median()))
```

```
The mean age in training data is: 29.7
The median age in training data is: 28.0
The mean age in test data is: 30.27
The median age in test data is: 27.0
```

Figure 12: Method to find mean age and median age in training data and test data.

Based on the calculated mean and median values, median age was selected to fill up the missing values in training data and test data as shown in Figure 13.

```
# Fill null values in Age column with median age
X_train['Age'].fillna(28.0, inplace=True)
X_test['Age'].fillna(27.0, inplace=True)
```

Figure 13: Method to fill up missing "Age" values.

After filling up the missing "Age" values with median age, the mean age and median age are re-computed as shown in Figure 14. Based on the results, there is no change in the median age but there is a minor change in the mean age.

```
# Check the updated mean and median age values
print("The updated mean age in training data is: {}".format(round(X_train['Age'].mean(axis=0, skipna=True),2)))
print("The updated median age in training data is: {}".format(X_train['Age'].median()))

print("The updated mean age in test data is: {}".format(round(X_test['Age'].mean(axis=0, skipna=True),2)))
print("The updated median age in test data is: {}".format(X_test['Age'].median()))
```

```
The updated mean age in training data is: 29.36
The updated median age in training data is: 28.0
The updated mean age in test data is: 29.6
The updated median age in test data is: 27.0
```

Figure 14: Updated mean age and median age in training data and test data.

Next, the missing values in "Embarked" attribute in training data was handled. There were 2 training samples with missing values in "Embarked" attribute as shown in Figure 15. Based on the training samples, it can be inferred that the embarkation point of these 2 samples are likely the same because both samples have the same ticket number.

```
# Display rows with Embarked column equals to null
print(train[train['Embarked'].isna()])
```

```
     PassengerId  Survived  Pclass                                      Name  \
61            62         1       1                       Icard, Miss. Amelie
829          830         1       1  Stone, Mrs. George Nelson (Martha Evelyn)

        Sex   Age  SibSp  Parch  Ticket  Fare Cabin Embarked
61   female  38.0      0      0  113572  80.0   B28      NaN
829  female  62.0      0      0  113572  80.0   B28      NaN
```

Figure 15: Training samples with missing values in "Embarked" attribute.

To find out the missing values in these 2 training samples, a simple search for $Ticket = 113572$ in training data and test data was conducted. Unfortunately, none other than the 2 training samples has $Ticket = 113572$.

Therefore, 3 sample dataframes were created from the training data by filtering all training samples with $Pclass = 1$ and $Embarked = Q, C, or\ S$ in each of the 3 sample dataframes. The filtering method is shown in Figures 16, 17 and 18. Both "Pclass" and "Embarked" attributes were selected as filters because these attributes contain relevant information to fill up the missing data in the 2 training samples.

```
# Find training samples having Pclass=1 and Embarked=Queenstown
Q_Class1 = train.loc[(train['Pclass']==1) & (train['Embarked']=='Q')]
Q_Class1
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
245	246	0	1	Minahan, Dr. William Edward	male	44.0	2	0	19928	90.0	C78	Q
412	413	1	1	Minahan, Miss. Daisy E	female	33.0	1	0	19928	90.0	C78	Q

Figure 16: Filtering of "Pclass" and "Embarked" attributes to create dataframe "Q_Class1".

```
# Find training samples having Pclass=1 and Embarked=Cherbourg
C_Class1 = train.loc[(train['Pclass']==1) & (train['Embarked']=='C')]
C_Class1
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38.0	1	0	PC 17599	71.2833	C85	C
30	31	0	1	Uruchurtu, Don. Manuel E	male	40.0	0	0	PC 17601	27.7208	NaN	C
31	32	1	1	Spencer, Mrs. William Augustus (Marie Eugenie)	female	28.0	1	0	PC 17569	146.5208	B78	C
34	35	0	1	Meyer, Mr. Edgar Joseph	male	28.0	1	0	PC 17604	82.1708	NaN	C
52	53	1	1	Harper, Mrs. Henry Sleeper (Myna Haxtun)	female	49.0	1	0	PC 17572	76.7292	D33	C
...	...	...	...	...	...	...	...	...	...	...	...	...
839	840	1	1	Marechal, Mr. Pierre	male	28.0	0	0	11774	29.7000	C47	C
842	843	1	1	Serepeca, Miss. Augusta	female	30.0	0	0	113798	31.0000	NaN	C
849	850	1	1	Goldenberg, Mrs. Samuel L (Edwiga Grabowska)	female	28.0	1	0	17453	89.1042	C92	C
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11767	83.1583	C50	C
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C

85 rows × 12 columns

Figure 17: Filtering of "Pclass" and "Embarked" attributes to create dataframe "C_Class1".

```
# Find training samples having Pclass=1 and Embarked=Southampton
S_Class1 = train.loc[(train['Pclass']==1) & (train['Embarked']=='S')]
S_Class1
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
23	24	1	1	Sloper, Mr. William Thompson	male	28.0	0	0	113788	35.5000	A6	S
27	28	0	1	Fortune, Mr. Charles Alexander	male	19.0	3	2	19950	263.0000	C23 C25 C27	S
...	...	...	...	...	...	...	...	...	...	...	...	...
862	863	1	1	Swift, Mrs. Frederick Joel (Margaret Welles Barron)	female	48.0	0	0	17466	25.9292	D17	S
867	868	0	1	Roebling, Mr. Washington Augustus II	male	31.0	0	0	PC 17590	50.4958	A24	S
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monypeny)	female	47.0	1	1	11751	52.5542	D35	S
872	873	0	1	Carlsson, Mr. Frans Olof	male	33.0	0	0	695	5.0000	B51 B53 B55	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S

127 rows × 12 columns

Figure 18: Filtering of "Pclass" and "Embarked" attributes to create dataframe "S_Class1".

Next, the mean and median ticket fare of each sample dataframe were computed using the method showed in Figure 19. Based on the results, the median ticket fare for $Pclass = 1$ at "Cherbourg" embarkation point is the nearest value to the ticket fare of the 2 training samples with missing value in "Embarked" attribute as shown in Figure 20. Therefore, the "Embarked" column of the 2 training samples will be filled as "C" using the method showed in Figure 21.

```
print("The mean ticket fare for PClass 1 at Queenstown is: {}".format(round(Q_Class1['Fare'].mean(axis=0, skipna=True),2)))
print("The median ticket fare for PClass 1 at Queenstown: {}".format(Q_Class1['Fare'].median()))

print("The mean ticket fare for PClass 1 at Cherbourg is: {}".format(round(C_Class1['Fare'].mean(axis=0, skipna=True),2)))
print("The median ticket fare for PClass 1 at Cherbourg: {}".format(C_Class1['Fare'].median()))

print("The mean ticket fare for PClass 1 at Southampton is: {}".format(round(S_Class1['Fare'].mean(axis=0, skipna=True),2)))
print("The median ticket fare for PClass 1 at Southampton: {}".format(S_Class1['Fare'].median()))
```

```
The mean ticket fare for PClass 1 at Queenstown is: 90.0
The median ticket fare for PClass 1 at Queenstown: 90.0
The mean ticket fare for PClass 1 at Cherbourg is: 104.72
The median ticket fare for PClass 1 at Cherbourg: 78.2667
The mean ticket fare for PClass 1 at Southampton is: 70.36
The median ticket fare for PClass 1 at Southampton: 52.0
```

Figure 19: Mean and median ticket fare at 3 embarkation points.

```
# Display rows with Embarked column equals to null
print(X_train[X_train['Embarked'].isna()])
```

```
     Pclass     Sex   Age  SibSp  Parch  Fare Embarked
61        1  female  38.0      0      0  80.0      NaN
829       1  female  62.0      0      0  80.0      NaN
```

Figure 20: Training samples with missing values in "Embarked" attribute.

```
# Fill null values in Embarked column with "Cherbourg"
train['Embarked'].fillna('C', inplace=True)
```

Figure 21: Method to replace null values in "Embarked" column.

Next, the missing value in "Fare" attribute in the test data was handled. The test sample with missing data is showed in Figure 22. The method used to fill up the missing ticket fare showed in Figures 23 and 24 is similar to the one used in filling the missing embarkation point of previous training samples.

As for "Cabin" attribute, there are almost 80% of missing values. Due to the large number, this makes cleaning difficult. In addition, "Cabin" attribute contains high uniqueness of data which is not a good attribute for training the classifier. Therefore, this attribute will be excluded.

```
# Display rows with Fare column equals to null in test data
print(test[test['Fare'].isna()])
```

```
     PassengerId  Pclass                Name   Sex   Age  SibSp  Parch Ticket  \
152         1044       3  Storey, Mr. Thomas  male  60.5      0      0   3701

     Fare Cabin Embarked
152   NaN   NaN        S
```

Figure 22: Test sample with missing “Fare” value.

```
# Find test samples having Pclass=3 and Embarked=Southampton
S_Class3 = test.loc[(test['Pclass']==3) & (test['Embarked']=='S')]
S_Class3
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
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
5	897	3	Svensson, Mr. Johan Cervin	male	14.0	0	0	7538	9.2250	NaN	S
9	901	3	Davies, Mr. John Samuel	male	21.0	2	0	A/4 48871	24.1500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...
409	1301	3	Peacock, Miss. Treasteall	female	3.0	1	1	SOTON/O.Q. 3101315	13.7750	NaN	S
412	1304	3	Henriksson, Miss. Jenny Lovisa	female	28.0	0	0	347086	7.7750	NaN	S
413	1305	3	Spector, Mr. Woolf	male	27.0	0	0	A.5. 3236	8.0500	NaN	S
415	1307	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN	S
416	1308	3	Ware, Mr. Frederick	male	27.0	0	0	359309	8.0500	NaN	S

142 rows × 11 columns

Figure 23: Filtering of “Pclass” and “Embarked” attributes to create dataframe “S_Class3”.

```
print("The mean ticket fare for PClass 3 at Southampton is: {}".format(round(S_Class3['Fare'].mean(axis=0, skipna=True),2)))
print("The median ticket fare for PClass 3 at Southampton: {}".format(S_Class3['Fare'].median()))
```

```
The mean ticket fare for PClass 3 at Southampton is: 13.91
The median ticket fare for PClass 3 at Southampton: 8.05
```

```
# Fill null values in Fare column with median ticket fare at "Cherbourg"
test['Fare'].fillna(8.05, inplace=True)
test.iloc[152]
```

```
PassengerId                 1044
Pclass                         3
Name           Storey, Mr. Thomas
Sex                          male
Age                          60.5
SibSp                           0
Parch                           0
Ticket                       3701
Fare                         8.05
Cabin                         NaN
Embarked                        S
Name: 152, dtype: object
```

Figure 24: Methods to calculate mean and median ticket fares and fill up missing data using median ticket fare.

D.2 Attributes Selection

The more important attributes were selected for training the classifier. Firstly, “PassengerId”, “Name”, and “Ticket” attributes contain many unique entries because they served as an identity. This characteristic is not useful to train a classifier due to its high variance. Secondly, there are nearly 80% of missing data in the “Cabin” attribute in both training and test data. This becomes a difficult task to fill up the missing values for “Cabin” attribute. In addition, “Cabin” attribute also served as an identity which is not a useful feature for this classification task. Therefore, “PassengerId”, “Name”, “Ticket”, and “Cabin” attributes were not selected for training the classifier. Finally, the “Survived” column in the training data was extracted and assigned as training labels. The method to select the important attributes is shown in Figure 25.

```
# Only important attributes are selected
X_train = train[['Pclass','Sex','Age','SibSp','Parch','Fare','Embarked']]
X_test = test[['Pclass','Sex','Age','SibSp','Parch','Fare','Embarked']]
y_train = train[['Survived']]
```

Figure 25: Columns selected for training data, test data, and training labels.

D.3 Feature Engineering

D.3.1 SibSp and Parch Attribute

Both the "SibSp" and "Parch" attributes can be combined into 1 attribute called "FamilySize" by summing the values of both columns at each row. Furthermore, we can categorise whether a passenger is onboard with any family members by using the following conditions:

$$if\ FamilySize \geq 1, passenger\ is\ with\ family\ member(s)$$

$$if\ FamilySize < 1, passenger\ is\ not\ with\ family\ members$$

The method to perform the above task is showed in Figure 26.

```
# Combine 'SibSp' and 'Parch' into one attribute called 'FamilySize'
X_train['FamilySize'] = X_train['SibSp'] + X_train['Parch']
X_test['FamilySize'] = X_test['SibSp'] + X_test['Parch']

# If FamilySize>=1 means passenger is onboard with family members, otherwise no.
X_train['Family'] = np.where(X_train['FamilySize']>=1, 'yes', 'no')
X_test['Family'] = np.where(X_test['FamilySize']>=1, 'yes', 'no')
```

Figure 26: Method to determine whether a passenger is onboard with family members.

```
X_train.head()
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	FamilySize	Family
0	3	male	22.0	1	0	7.2500	S	1	yes
1	1	female	38.0	1	0	71.2833	C	1	yes
2	3	female	26.0	0	0	7.9250	S	0	no
3	1	female	35.0	1	0	53.1000	S	1	yes
4	3	male	35.0	0	0	8.0500	S	0	no

```
X_test.head()
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	FamilySize	Family
0	3	male	34.5	0	0	7.8292	Q	0	no
1	3	female	47.0	1	0	7.0000	S	1	yes
2	2	male	62.0	0	0	9.6875	Q	0	no
3	3	male	27.0	0	0	8.6625	S	0	no
4	3	female	22.0	1	1	12.2875	S	2	yes

Figure 27: Updated training and test data after feature engineering.

Next, “SibSp”, “Parch”, and “FamilySize” columns are dropped from the dataframe because they are not needed anymore. The method to do this is showed in Figure 28.

```
# Drop 'SibSp', 'Parch', and 'FamilySize' columns
X_train = X_train.drop(['SibSp','Parch','FamilySize'], axis=1)
X_test = X_test.drop(['SibSp','Parch','FamilySize'], axis=1)
```

Figure 28: Method to drop columns in dataframe.

D.3.2 Age Attribute

In Part A, it was observed that the values in “Age” and “Fare” attributes have high variance. To reduce the variance, age and fare values were categorised into different groups.

Firstly, age values were converted to integer datatype. Next, the range of age values in training dataset were categorised into 5 groups. Different age range was tried out to obtain low variance. The final age range is summarised in Table 6. Similar approach was carried out on the test dataset and the age range is summarised in Table 7.

The high number of counts for age range (27, 35) in training dataset is due to 202 passengers that have $age = 28$, which majority of the samples were filled up to replace the missing values. On the other hand, the high number of counts for age range (18, 27) in test dataset is due to 98 passengers that have $age = 27$, which majority of the samples were filled up to replace the missing values.

Table 6: Different range of ages in training dataset categorised into 5 distinct groups

Age Range	**Age Group**	**Count**
$age \leq 17$	0	113
$17 < age \leq 27$	1	224
$27 < age \leq 35$	2	337
$35 < age \leq 45$	3	116
$age > 45$	4	101

Table 7: Different range of ages in test dataset categorised into 5 distinct groups

Age Range	Age Group	Count
$age \leq 17$	0	41
$17 < age \leq 27$	1	212
$27 < age \leq 35$	2	60
$35 < age \leq 45$	3	53
$age > 45$	4	52

After the transformation of "Age" attributes, the mean and variance were computed and compared with the mean and variance before transformation. The result in Table 8 showed that the variance is significantly reduced.

Table 8: Comparison of mean and variance of "Age" attribute after transformation.

	Before Transformation		**After Transformation**	
	Mean	**Variance**	**Mean**	**Variance**
"Age" Attribute in Training Dataset	29.362	169.513	1.852	1.322
"Age" attribute in Test Dataset	29.599	161.386	1.672	1.420

The methods to categorise the range of ages in both training and test datasets into 5 age groups are shown in Figure 29.

```
# Categorise range of ages in training dataset into 5 age groups
for row in X_train:
    X_train['AgeGroup'] = X_train['Age'].astype(int)
    X_train.loc[X_train['AgeGroup']<=17, 'AgeGroup']=0
    X_train.loc[(X_train['AgeGroup']>17) & (X_train['AgeGroup']<=27), 'AgeGroup']=1
    X_train.loc[(X_train['AgeGroup']>27) & (X_train['AgeGroup']<=35), 'AgeGroup']=2
    X_train.loc[(X_train['AgeGroup']>35) & (X_train['AgeGroup']<=45), 'AgeGroup']=3
    X_train.loc[X_train['AgeGroup']>45, 'AgeGroup']=4

X_train['AgeGroup'].value_counts()
```

```
2    337
1    224
3    116
0    113
4    101
Name: AgeGroup, dtype: int64
```

```
# Categorise range of ages in test dataset into 5 age groups
for row in X_test:
    X_test['AgeGroup'] = X_test['Age'].astype(int)
    X_test.loc[X_test['AgeGroup']<=17, 'AgeGroup']=0
    X_test.loc[(X_test['AgeGroup']>17) & (X_test['AgeGroup']<=27), 'AgeGroup']=1
    X_test.loc[(X_test['AgeGroup']>27) & (X_test['AgeGroup']<=35), 'AgeGroup']=2
    X_test.loc[(X_test['AgeGroup']>35) & (X_test['AgeGroup']<=45), 'AgeGroup']=3
    X_test.loc[X_test['AgeGroup']>45, 'AgeGroup']=4

X_test['AgeGroup'].value_counts()
```

```
1    212
2     60
3     53
4     52
0     41
Name: AgeGroup, dtype: int64
```

Figure 29: Methods to categorise the age range into 5 age groups in both datasets.

Next, “Age” attribute is dropped from the dataframe because it is not needed anymore. The method to do this is showed in Figure 30.

```
# Drop 'Age' column
X_train = X_train.drop(['Age'], axis=1)
X_test = X_test.drop(['Age'], axis=1)
```

Figure 30. Method to drop “Age” attribute.

D.3.3 Fare Attribute

Different range of fare values were worked out to obtain low variance. Firstly, the range of fare values in training dataset were categorised into 5 groups. Different fare range was tried out to obtain low variance. The final fare range is summarised in Table 9. Similar approach was carried out on the test dataset and the fare range is summarised in Table 10.

Table 9: Different range of fares in training dataset categorised into 5 distinct groups

Fare Range	Fare Group	Count
$fare \leq 8$	0	241
$8 < fare \leq 20$	1	274
$20 < fare \leq 50$	2	216
$50 < fare \leq 80$	3	86
$fare > 80$	4	74

Table 10: Different range of fares in test dataset categorised into 5 distinct groups

Fare Range	Fare Group	Count
$fare \leq 8$	0	119
$8 < fare \leq 20$	1	119
$20 < fare \leq 50$	2	100
$50 < fare \leq 80$	3	39
$fare > 80$	4	41

After the transformation of "Fare" attributes, the mean and variance were computed and compared with the mean and variance before transformation. The result in Table 11 showed that the variance is significantly reduced. Next, "Fare" attribute is dropped from the dataframe because it is not needed anymore.

Table 11: Comparison of mean and variance of "Fare" attribute after transformation.

	Before Transformation		**After Transformation**	
	Mean	**Variance**	**Mean**	**Variance**
"Fare" Attribute in Training Dataset	32.204	2469.437	1.414	1.477
"Fare" attribute in Test Dataset	35.561	3119.981	1.435	1.594

The methods to categorise the range of fares in both training and test datasets into 5 fare groups are shown in Figure 31.

```
# Categorise range of fares in training dataset into 5 fare groups
for row in X_train:
    X_train['FareGroup'] = X_train['Fare']
    X_train.loc[X_train['FareGroup']<=8, 'FareGroup']=0
    X_train.loc[(X_train['FareGroup']>8) & (X_train['FareGroup']<=20), 'FareGroup']=1
    X_train.loc[(X_train['FareGroup']>20) & (X_train['FareGroup']<=50), 'FareGroup']=2
    X_train.loc[(X_train['FareGroup']>50) & (X_train['FareGroup']<=80), 'FareGroup']=3
    X_train.loc[X_train['FareGroup']>80, 'FareGroup']=4

X_train['FareGroup'] = X_train['FareGroup'].astype(int)
X_train['FareGroup'].value_counts()
```

```
1    274
0    241
2    216
3     86
4     74
Name: FareGroup, dtype: int64
```

```
# Categorise range of fares in test dataset into 5 fare groups
for row in X_test:
    X_test['FareGroup'] = X_test['Fare']
    X_test.loc[X_test['FareGroup']<=8, 'FareGroup']=0
    X_test.loc[(X_test['FareGroup']>8) & (X_test['FareGroup']<=20), 'FareGroup']=1
    X_test.loc[(X_test['FareGroup']>20) & (X_test['FareGroup']<=50), 'FareGroup']=2
    X_test.loc[(X_test['FareGroup']>50) & (X_test['FareGroup']<=80), 'FareGroup']=3
    X_test.loc[X_test['FareGroup']>80, 'FareGroup']=4

X_test['FareGroup'] = X_test['FareGroup'].astype(int)
X_test['FareGroup'].value_counts()
```

```
1    119
0    119
2    100
4     41
3     39
Name: FareGroup, dtype: int64
```

Figure 31: Methods to categorise the fare range into 5 fare groups in both datasets.

D.4 One-Hot Encoding

Finally, one-hot encoding is performed on categorical attributes as showed in Figure 32.

```
# One-hot encoding for categorical variables
X_train = pd.get_dummies(X_train, prefix=['Pclass','Sex','Embarked','Family', 'AgeGroup', 'FareGroup'],
                         columns=['Pclass','Sex','Embarked','Family', 'AgeGroup', 'FareGroup'])
X_test = pd.get_dummies(X_test, prefix=['Pclass','Sex','Embarked','Family', 'AgeGroup', 'FareGroup'],
                         columns=['Pclass','Sex','Embarked','Family', 'AgeGroup', 'FareGroup'])

X_train.head()
```

	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S	Family_no	Family_yes	AgeGroup_0	AgeGroup_1	Age
0	0	0	1	0	1	0	0	1	0	1	0	1	
1	1	0	0	1	0	1	0	0	0	1	0	0	
2	0	0	1	1	0	0	0	1	1	0	0	1	
3	1	0	0	1	0	0	0	1	0	1	0	0	
4	0	0	1	0	1	0	0	1	1	0	0	0	

```
X_test.head()
```

	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S	Family_no	Family_yes	AgeGroup_0	AgeGroup_1	Age
0	0	0	1	0	1	0	1	0	1	0	0	0	
1	0	0	1	1	0	0	0	1	0	1	0	0	
2	0	1	0	0	1	0	1	0	1	0	0	0	
3	0	0	1	0	1	0	0	1	1	0	0	1	
4	0	0	1	1	0	0	0	1	0	1	0	1	

Figure 32: One-hot encoding for categorical attributes.

The final dimensions, $(number\ of\ samples, number\ of\ attributes)$, of training and test datasets which will be used for training and prediction are shown in Table 12.

Table 12: Dimensions of training and test datasets used for training and prediction.

	Training Dataset	**Test Dataset**
Shape	$(891, 20)$	$(418, 20)$

Part F: Analysis on the Predicted Test Dataset

The Neural Network was trained using the following parameters:

- Optimizer = Adam
- Learning Rate = 0.0005
- Loss Function = Binary Crossentropy
- Batch Size = 8
- Epochs = 100
- Early Stopping: patience = 20, monitored by validation loss

There is an addition of "Early Stopping" criterion to prevent overfitting the model. For this training, the patience is set to 20 and it is monitored by validation loss. This means that the training will stop when the validation loss does not improve after 20 epochs.

The training result in Figures 33 and 34 showed that the training stops at Epoch 52. This model achieved a validation accuracy of 81.6% on the validation dataset as shown in Figure 35. The model is saved as "nn_crossval.hdf5" and is used for predicting the test dataset.

```
batch_size = 8
epochs = 100

# Early Stopping and Save Best NN model for prediction
filepath = 'nn_crossval.hdf5'
callbacks_list = [EarlyStopping(monitor='val_loss', verbose=1, patience=20),
                  ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_best_only=True)]
history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
                    validation_data=(X_val, y_val), verbose=1, callbacks=callbacks_list)
```

```
y: 0.8268

Epoch 00049: val_loss did not improve from 0.44481
Epoch 50/100
712/712 [==============================] - 1s 1ms/step - loss: 0.4289 - accuracy: 0.8272 - val_loss: 0.4490 - val_accurac
y: 0.8156

Epoch 00050: val_loss did not improve from 0.44481
Epoch 51/100
712/712 [==============================] - 1s 1ms/step - loss: 0.4588 - accuracy: 0.8118 - val_loss: 0.4629 - val_accurac
y: 0.8156

Epoch 00051: val_loss did not improve from 0.44481
Epoch 52/100
712/712 [==============================] - 1s 848us/step - loss: 0.4567 - accuracy: 0.8244 - val_loss: 0.4590 - val_accur
acy: 0.8156

Epoch 00052: val_loss did not improve from 0.44481
Epoch 00052: early stopping
```

Figure 33: Training stops at Epoch 52.

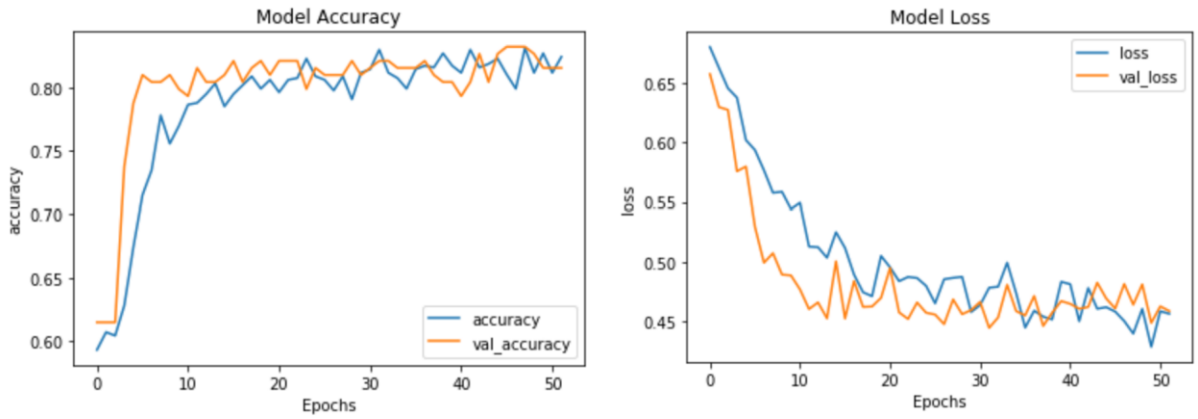

Figure 34: Training curves of model accuracy and model loss.

```
# Load saved NN model. Comment out the below code to load saved model.
#model = load_model('nn_crossval.hdf5')
val_loss, val_acc = model.evaluate(X_val, y_val, verbose=1)

print("Validation Accuracy: {}".format(round(val_acc,3)))
print("Validation Loss: {}".format(round(val_loss,3)))
```

```
179/179 [==============================] - 0s 73us/step
Validation Accuracy: 0.816
Validation Loss: 0.459
```

Figure 35: Neural Network's validation accuracy on validation dataset.

The predicted labels were saved in "submission_nn.csv" file and appended to the test dataset for analysis. The method to do this is shown in Figure 36.

```
y_pred = model.predict_classes(X_test)
submission['Survived'] = y_pred
submission.to_csv('submission_nn.csv')
```

```
submission.head()
```

PassengerId	Survived
892	0
893	0
894	0
895	0
896	1

```
# Join predicted labels to test dataset
submission = pd.read_csv('submission_nn.csv')
X_test = pd.read_csv('X_test_cleaned1.csv')
X_test = X_test.drop(['FareGroup_0', 'FareGroup_1', 'FareGroup_2', 'FareGroup_3', 'FareGroup_4'], axis=1)
X_test['Survived_Pred'] = submission['Survived']
X_test.head()
```

_male	Embarked_C	Embarked_Q	Embarked_S	Family_no	Family_yes	AgeGroup_0	AgeGroup_1	AgeGroup_2	AgeGroup_3	AgeGroup_4	Survived_Pred
1	0	1	0	1	0	0	0	1	0	0	0
0	0	0	1	0	1	0	0	0	0	1	0
1	0	1	0	1	0	0	0	0	0	1	0
1	0	0	1	1	0	0	1	0	0	0	0
0	0	0	1	0	1	0	1	0	0	0	1

Figure 36: Method to analyse predicted test dataset.

Answers to Part F questions

1) How many of these 418 passengers are classified as survivors from the Titanic tragedy?

   Ans: Out of 418 passengers, only 139 survived from the Titanic tragedy.

2) Among these 139 survivors,

   (i) 132 are female survivors.

   (ii) 16 survivors are below 18 years old.

   (iii) 63 survivors have no family members onboard.

   (iv) Passengers from ticket class 3 have the least chance of surviving the tragedy.

   (v) Passengers embarked at Southampton have the least chance of surviving the tragedy.

The codes to obtain the above analysis are shown in Figures 37, 38, and 39.

```
# How many passengers survived?
# 0 = Did not survive
# 1 = Survived
X_test['Survived_Pred'].value_counts()
```

```
0    279
1    139
Name: Survived_Pred, dtype: int64
```

```
# How many female survivors?
female_survivor = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Sex_female']==1)]
print("Number of female survivors: {}".format(female_survivor.shape[0]))
```

```
Number of female survivors: 132
```

```
# How many survivors are aged 18 and below
below_18 = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['AgeGroup_0']==1)]
print("Number of survivors below 18 years old: {}".format(below_18.shape[0]))
```

```
Number of survivors below 18 years old: 16
```

```
# How many survivors are without family members onboard?
no_family = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Family_no']==1)]
print("Number of survivors that have no family members onboard: {}".format(no_family.shape[0]))
```

```
Number of survivors that have no family members onboard: 63
```

Figure 37: Codes to obtain 1), 2(i), 2(ii), and 2(iii).

```
# Number of passengers in each ticket class
class1 = X_test.loc[(X_test['Pclass_1']==1)]
class2 = X_test.loc[(X_test['Pclass_2']==1)]
class3 = X_test.loc[(X_test['Pclass_3']==1)]
print("Number of passengers in Class 1: {}".format(class1.shape[0]))
print("Number of passengers in Class 2: {}".format(class2.shape[0]))
print("Number of passengers in Class 3: {}".format(class3.shape[0]))
```

```
Number of passengers in Class 1: 107
Number of passengers in Class 2: 93
Number of passengers in Class 3: 218
```

```
# Passengers from which ticket class have the least chance to survive?
bad_class1 = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Pclass_1']==1)]
bad_class2 = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Pclass_2']==1)]
bad_class3 = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Pclass_3']==1)]
print("Percentage of survivors in Class 1: {}%".format(round((bad_class1.shape[0]/class1.shape[0])*100,2)))
print("Percentage of survivors in Class 2: {}%".format(round((bad_class2.shape[0]/class2.shape[0])*100,2)))
print("Percentage of survivors in Class 3: {}%".format(round((bad_class3.shape[0]/class3.shape[0])*100,2)))
```

```
Percentage of survivors in Class 1: 49.53%
Percentage of survivors in Class 2: 34.41%
Percentage of survivors in Class 3: 24.77%
```

Figure 38: Codes to obtain 2(iv).

```
emC = X_test.loc[(X_test['Embarked_C']==1)]
emQ = X_test.loc[(X_test['Embarked_Q']==1)]
emS = X_test.loc[(X_test['Embarked_S']==1)]
print("Number of passengers embarked at Cherbourg: {}".format(emC.shape[0]))
print("Number of passengers embarked at Queenstown: {}".format(emQ.shape[0]))
print("Number of passengers embarked at Southampton: {}".format(emS.shape[0]))
```

```
Number of passengers embarked at Cherbourg: 102
Number of passengers embarked at Queenstown: 46
Number of passengers embarked at Southampton: 270
```

```
# Passengers from which port of embarkation have the least chance to survive?
bad_portc = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Embarked_C']==1)]
bad_portq = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Embarked_Q']==1)]
bad_ports = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Embarked_S']==1)]
print("Percentage of survivors embarked at Cherbourg: {}%".format(round((bad_portc.shape[0]/emC.shape[0])*100,2)))
print("Percentage of survivors embarked at Queenstown: {}%".format(round((bad_portq.shape[0]/emQ.shape[0])*100,2)))
print("Percentage of survivors embarked at Southampton: {}%".format(round((bad_ports.shape[0]/emS.shape[0])*100,2)))
```

```
Percentage of survivors embarked at Cherbourg: 41.18%
Percentage of survivors embarked at Queenstown: 50.0%
Percentage of survivors embarked at Southampton: 27.41%
```

Figures 39: Codes to obtain 2(v).

Part G: Build Convolutional Neural Network

Convolutional Neural Network (CNN) is selected to compare the results obtained from Neural Network (NN), SVM, and Random Forest Classifier.

Since CNN's input layer only accept 3-dimensional data, the training and test samples need to be reshaped. The method to reshape the datasets is shown in Figure 40.

```
# Reshape training and test dataset
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1],1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1],1)

print(X_train.shape)
print(X_test.shape)
```

```
(891, 15, 1)
(418, 15, 1)
```

Figure 40: Method to reshape training and test datasets.

Next, Conv1D layer is selected to convolve the data samples because each data sample is represented as a 1-dimensional array. Each Conv1D layer contains different number of $(1x3)$ kernels to convolve the data samples to create feature maps. Since the number of training features is only 15, a small kernel length of 3 is used so that it does not heavily reduce the feature dimension. The activation function used in each Conv1D layer is Rectified Linear Unit (ReLU). Lastly, the activation function used in the output layer is Sigmoid because it is suitable for binary classification.

The CNN structure and training parameters were fine-tuned based on the training accuracy, training loss, validation accuracy, and validation loss. This is to prevent underfitting, overfitting of the model by observing the training curves.

The best performing CNN structure contains 5 Conv1D layers, each having 32, 64, 128, 64, and 32 kernels, respectively. Dropout layer is added after each Conv1D layer to reduce overfitting. The feature maps are flattened before passing to the output layer. The final CNN structure that will be used to compare with NN, SVM, and Random Forest Classifier is shown in Figure 41.

The training parameters of the CNN is listed as follows:

- 10-fold cross validation
- Batch Size = 8
- Training Epochs = 100
- Optimizer = Adam
- Learning Rate = 0.0005
- Loss Function = Binary Crossentropy

```
# Convolutional Neural Network
def model_cnn():
    model = Sequential()
    model.add(InputLayer(input_shape=(X_train.shape[1],1)))
    model.add(Conv1D(32, 3, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Conv1D(64, 3, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Conv1D(128, 3, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Conv1D(64, 3, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Conv1D(32, 3, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))

    return model
```

Figure 41: Final CNN structure.

CNN is trained using Stratified 10-Fold Cross Validation as shown in Figures 42 and 43. The average validation accuracy achieved by CNN is 81.5%.

```
# 10-fold Cross Validation

batch_size = 8
epochs = 100
kfold = 10

skf = StratifiedKFold(n_splits=kfold, shuffle=True, random_state=111)
skf_accuracy = []

for i, (train_index, val_index) in enumerate(skf.split(X_train, y_train)):
    print('[Fold %d/%d]' % (i + 1, kfold))
    X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]
    model = model_cnn()
    tf.random.set_seed(4321)

    # Compile the model
    optimizer = Adam(learning_rate=0.0005)
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

    # Define callbacks list
    filepath = 'cnn_crossval.hdf5'
    callbacks_list = [EarlyStopping(monitor='val_loss', verbose=1, patience=20),
                      ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_best_only=True)]
    history = model.fit(X_train_fold, y_train_fold, batch_size=batch_size, epochs=epochs,
                        validation_data=(X_val_fold, y_val_fold), verbose=1, callbacks=None)
    plot_graphs(history, 'accuracy', 'Model Accuracy')
    plot_graphs(history, 'loss', 'Model Loss')
    score = model.evaluate(X_val_fold, y_val_fold)
    val_acc = score[1]
    skf_accuracy.append(val_acc)
```

Figure 42: Method to train CNN using Stratified 10-Fold Cross Validation.

```
# Print the output
print('List of validation accuracy at each fold: {}'.format(skf_accuracy))
print('\nMaximum Validation Accuracy: {}%'.format(round(max(skf_accuracy)*100, 3)))
print('Minimum Validation Accuracy: {}%'.format(round(min(skf_accuracy)*100, 3)))
print('Overall Validation Accuracy: {}%'.format(round(mean(skf_accuracy)*100, 3)))
print('Standard Deviation is: {}'.format(round(stdev(skf_accuracy), 3)))
```

```
List of validation accuracy at each fold: [0.7888888716697693, 0.8202247023582458, 0.7752808928489685, 0.7977527976036072,
0.8314606547355652, 0.7752808928489685, 0.8202247023582458, 0.8651685118675232, 0.8202247023582458, 0.8539325594902039]

Maximum Validation Accuracy: 86.517%
Minimum Validation Accuracy: 77.528%
Overall Validation Accuracy: 81.484%
Standard Deviation is: 0.031
```

Figure 43: Method to obtain CNN's average validation accuracy.

Table 12: Average Validation Accuracies of Different Classifiers.

Classifier	**Validation Accuracy**
Neural Network	80.9%
Support Vector Machine	80.5%
Random Forest	79.3%
Convolutional Neural Network	81.5%

Based on Table 12, CNN outperformed all the other classifiers. Since CNN is a type of neural network, it is likely to have a better performance than machine learning models such as SVM and Random Forest Classifier.

CNN performs slightly better than NN because both of their network structures are different. CNN has 5 convolutional layers while NN has 9 fully-connected layers. This shows that having more hidden layers does not guarantee a better training performance.

The dropout parameter values of both networks are also different. This was done through detailed fine-tuning of training hyper-parameters to improve training curves. Therefore, CNN has the best performance among other classifiers used in this classification problem.

Part H: Image Classification for Cifar-10 Dataset

Cifar-10 dataset contains 60,000 32x32 colour images in 10 classes, namely airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is separated into labelled training and test datasets. Distribution of Cifar-10 dataset is shown in Table 13.

Table 13: Distribution of Cifar-10 dataset.

	Training Dataset	**Test Dataset**
Shape	(50000, 32, 32, 3)	(10000, 32, 32, 3)
Number of Samples	50000	10000
Image Size	32 x 32	32 x 32
Image Channels	3	3
Number of Class Labels	10	10
Shape of Class Labels	(50000, 1)	(10000, 1)
Number of Samples per Class Labels	5000	1000

Based on Table 13, it can be inferred that this is a multi-class image classification. Therefore, a CNN is selected to work on this classification task since CNN is suitable for image data. The CNN built in Part G was extended to this image classification problem except with the following changes in the structure:

1. Input Shape is changed to (32, 32, 3) because each colour image sample has dimensions 32x32 and 3 colour channels, RGB.
2. Conv1D is changed to Conv2D because image dimensions are 2D.
3. Kernel size is changed to 3x3 because image dimensions are 2D.
4. Padding is added to retain the image border.
5. MaxPooling layer is added to reduce computational parameters.
6. Number of output neurons is changed to 10 because there are 10 classes to predict.
7. Activation function at output layer is changed to “Softmax” because “Sigmoid” can only work on 2 classes.
8. The initial CNN structure is shown in Figure 44.

```python
# Convolutional Neural Network
def model_cnn():
    model = Sequential()
    model.add(InputLayer(input_shape=(32,32,3)))
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    model.add(Flatten())
    model.add(Dense(10, activation='softmax'))

    return model
```

Figure 44: Initial CNN structure to train Cifar-10 dataset.

Training Parameters:

- Optimizer = Adam
- Learning Rate = 0.001
- Loss Function = Categorical Crossentropy (for multi-class)
- Batch Size = 128
- Training Epochs = 50

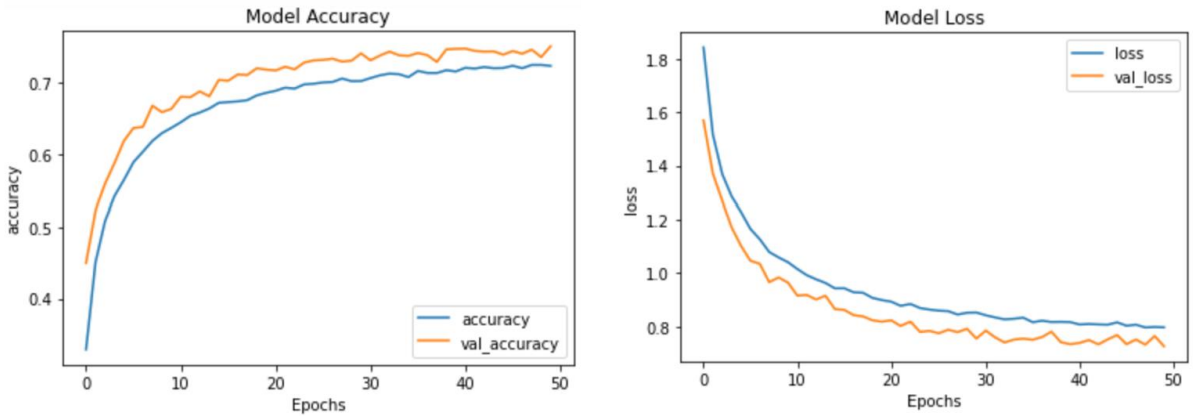

Figure 45: Initial training results of CNN.

```
# Prediction on test dataset
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=1)

print("Test Accuracy: {}%".format(round((test_acc*100),3)))
print("Test Loss: {}".format(round(test_loss,3)))
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.7284 - accuracy: 0.7497
Test Accuracy: 74.97%
Test Loss: 0.728
```

Figure 46: Initial prediction accuracy on Cifar-10 test dataset.

Based on the training results showed in Figure 45, it can be inferred that CNN model has good generalization because the validation accuracy is higher than the training accuracy. The error gap between the validation loss and training loss is small. The model is used to predict the Cifar-10 test dataset and achieved a test accuracy of 74.97%. These results showed that CNN performance still can be improved.

```
# Convolutional Neural Network
def model_cnn():
    model = Sequential()
    model.add(InputLayer(input_shape=(32,32,3)))
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    model.add(Flatten())
    model.add(Dense(10, activation='softmax'))

    return model
```

Figure 47: Improved CNN structure.

To improve the performance of CNN, the number of layers is increased by duplicating every convolutional layer as shown in Figure 47. The improved CNN structure is trained and predicted on the Cifar-10 test dataset, achieving an improved test accuracy of 76.13% as shown in Figure 48.

```
# Prediction on test dataset
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=1)

print("Test Accuracy: {}%".format(round((test_acc*100),3)))
print("Test Loss: {}".format(round(test_loss,3)))
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.9316 - accuracy: 0.7613
Test Accuracy: 76.13%
Test Loss: 0.932
```

Figure 48. Improved prediction accuracy on Cifar-10 test dataset.

Based on the above test results, it can be concluded that the classification accuracy of Cifar-10 dataset can be improved by increasing the depth of CNN. However, by doing so, it also increases the number of trainable parameters. At the same time, high computational power is required to train the model.

Despite the slight improvement in the test results, the test accuracy of Cifar-10 dataset is still relatively low compared to the validation accuracy of Titanic dataset. The reason is due to the different size of the datasets used for training a model. A large dataset like the Cifar-10, requires very deep neural network and high computational resources to achieve high training accuracy. However, due to limited computational resources, a very deep CNN cannot be built to train the Cifar-10 dataset.

Part I: Source Codes for Titanic Survivors Classification

```
import pandas as pd
import numpy as np
import tensorflow as tf
import keras
from keras.layers import Dense, InputLayer, Dropout, Conv1D, Flatten
from keras.models import Sequential, load_model
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.metrics import accuracy_score
from statistics import mean, stdev

pd.set_option('display.max_colwidth', 140)

train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

train.head()

test.head()

## 1.1 Exploratory Data Analysis

# Overview of training data
train.info()

train.describe()

# Overview of test data
test.info()
```

```
test.describe()

print(train.shape)
print(test.shape)

# Check for null values in training data
train.isnull().sum()

# Check for null values in test data
test.isnull().sum()

## 1.2 Data Pre-processing

### 1.2.1 Handling Missing Values

print("The mean age in training data is: {}".format(round(train['Age'].mean(axis=0, skipna=True),2)))
print("The median age in training data is: {}".format(train['Age'].median()))

print("The mean age in test data is: {}".format(round(test['Age'].mean(axis=0, skipna=True),2)))
print("The median age in test data is: {}".format(test['Age'].median()))

# Fill null values in Age column with median age
train['Age'].fillna(28.0, inplace=True)
test['Age'].fillna(27.0, inplace=True)

train.isnull().sum()

test.isnull().sum()

# Check the updated mean and median age values
print("The updated mean age in training data is: {}".format(round(train['Age'].mean(axis=0, skipna=True),2)))
print("The updated median age in training data is: {}".format(train['Age'].median()))

print("The updated mean age in test data is: {}".format(round(test['Age'].mean(axis=0, skipna=True),2)))
print("The updated median age in test data is: {}".format(test['Age'].median()))
```

```
# Display rows with Embarked column equals to null
print(train[train['Embarked'].isna()])


# Find training samples having Pclass=1 and Embarked=Queenstown
Q_Class1 = train.loc[(train['Pclass']==1) & (train['Embarked']=='Q')]
Q_Class1


# Find training samples having Pclass=1 and Embarked=Cherbourg
C_Class1 = train.loc[(train['Pclass']==1) & (train['Embarked']=='C')]
C_Class1


# Find training samples having Pclass=1 and Embarked=Southampton
S_Class1 = train.loc[(train['Pclass']==1) & (train['Embarked']=='S')]
S_Class1


print("The mean ticket fare for PClass 1 at Queenstown is: {}".format(round(Q_Class1['Fare'].mean(axis=0, skipna=True),2)))
print("The median ticket fare for PClass 1 at Queenstown: {}".format(Q_Class1['Fare'].median()))


print("The mean ticket fare for PClass 1 at Cherbourg is: {}".format(round(C_Class1['Fare'].mean(axis=0, skipna=True),2)))
print("The median ticket fare for PClass 1 at Cherbourg: {}".format(C_Class1['Fare'].median()))


print("The mean ticket fare for PClass 1 at Southampton is: {}".format(round(S_Class1['Fare'].mean(axis=0, skipna=True),2)))
print("The median ticket fare for PClass 1 at Southampton: {}".format(S_Class1['Fare'].median()))


# Fill null values in Embarked column with "Cherbourg"
train['Embarked'].fillna('C', inplace=True)


train.isnull().sum()


# Display rows with Fare column equals to null in test data
print(test[test['Fare'].isna()])


# Find test samples having Pclass=3 and Embarked=Southampton
S_Class3 = test.loc[(test['Pclass']==3) & (test['Embarked']=='S')]
```

```
S_Class3


print("The mean ticket fare for PClass 3 at Southampton is: {}".format(round(S_Class3['Fare'].mean(axis=0, skipna=True),2)))

print("The median ticket fare for PClass 3 at Southampton: {}".format(S_Class3['Fare'].median()))


# Fill null values in Fare column with median ticket fare at "Cherbourg"

test['Fare'].fillna(8.05, inplace=True)

test.iloc[152]


test.isnull().sum()


### 1.2.2 Attributes Selection


# Only important attributes are selected

X_train = train[['Pclass','Sex','Age','SibSp','Parch','Fare','Embarked']]

X_test = test[['Pclass','Sex','Age','SibSp','Parch','Fare','Embarked']]

y_train = train[['Survived']]


X_train.isnull().sum()


X_test.isnull().sum()


X_train.head()


X_test.head()


y_train.head()


print(X_train.shape)

print(X_test.shape)

print(y_train.shape)


### 1.2.3 Feature Engineering


#### 1.2.3.1 SibSp and Parch Attributes
```

```
# Combine 'SibSp' and 'Parch' into one attribute called 'FamilySize'
X_train['FamilySize'] = X_train['SibSp'] + X_train['Parch']
X_test['FamilySize'] = X_test['SibSp'] + X_test['Parch']

# If FamilySize>=1 means passenger is onboard with family members, otherwise no.
X_train['Family'] = np.where(X_train['FamilySize']>=1, 'yes', 'no')
X_test['Family'] = np.where(X_test['FamilySize']>=1, 'yes', 'no')

X_train.head()

X_test.head()

# Drop 'SibSp', 'Parch', and 'FamilySize' columns
X_train = X_train.drop(['SibSp','Parch','FamilySize'], axis=1)
X_test = X_test.drop(['SibSp','Parch','FamilySize'], axis=1)

#### 1.2.3.2 Age Attribute

X_train['Age'].value_counts()

X_test['Age'].value_counts()

X_train['Age'].describe()

X_test['Age'].describe()

# Categorise range of ages in training dataset into 5 age groups
for row in X_train:
    X_train['AgeGroup'] = X_train['Age'].astype(int)
    X_train.loc[X_train['AgeGroup']<=17, 'AgeGroup']=0
    X_train.loc[(X_train['AgeGroup']>17) & (X_train['AgeGroup']<=27), 'AgeGroup']=1
    X_train.loc[(X_train['AgeGroup']>27) & (X_train['AgeGroup']<=35), 'AgeGroup']=2
    X_train.loc[(X_train['AgeGroup']>35) & (X_train['AgeGroup']<=45), 'AgeGroup']=3
    X_train.loc[X_train['AgeGroup']>45, 'AgeGroup']=4
```

```
X_train['AgeGroup'].value_counts()

# Categorise range of ages in test dataset into 5 age groups
for row in X_test:
    X_test['AgeGroup'] = X_test['Age'].astype(int)
    X_test.loc[X_test['AgeGroup']<=17, 'AgeGroup']=0
    X_test.loc[(X_test['AgeGroup']>17) & (X_test['AgeGroup']<=27), 'AgeGroup']=1
    X_test.loc[(X_test['AgeGroup']>27) & (X_test['AgeGroup']<=35), 'AgeGroup']=2
    X_test.loc[(X_test['AgeGroup']>35) & (X_test['AgeGroup']<=45), 'AgeGroup']=3
    X_test.loc[X_test['AgeGroup']>45, 'AgeGroup']=4

X_test['AgeGroup'].value_counts()

X_train['AgeGroup'].describe()

X_test['AgeGroup'].describe()

# Drop 'Age' column
X_train = X_train.drop(['Age'], axis=1)
X_test = X_test.drop(['Age'], axis=1)

X_train.head()

#### 1.2.3.3 Fare Attribute

X_train['Fare'].describe()

X_test['Fare'].describe()

# Categorise range of fares in training dataset into 5 fare groups
for row in X_train:
    X_train['FareGroup'] = X_train['Fare']
    X_train.loc[X_train['FareGroup']<=8, 'FareGroup']=0
    X_train.loc[(X_train['FareGroup']>8) & (X_train['FareGroup']<=20), 'FareGroup']=1
    X_train.loc[(X_train['FareGroup']>20) & (X_train['FareGroup']<=50), 'FareGroup']=2
    X_train.loc[(X_train['FareGroup']>50) & (X_train['FareGroup']<=80), 'FareGroup']=3
```

```
    X_train.loc[X_train['FareGroup']>80, 'FareGroup']=4

X_train['FareGroup'] = X_train['FareGroup'].astype(int)
X_train['FareGroup'].value_counts()

# Categorise range of fares in test dataset into 5 fare groups
for row in X_test:
    X_test['FareGroup'] = X_test['Fare']
    X_test.loc[X_test['FareGroup']<=8, 'FareGroup']=0
    X_test.loc[(X_test['FareGroup']>8) & (X_test['FareGroup']<=20), 'FareGroup']=1
    X_test.loc[(X_test['FareGroup']>20) & (X_test['FareGroup']<=50), 'FareGroup']=2
    X_test.loc[(X_test['FareGroup']>50) & (X_test['FareGroup']<=80), 'FareGroup']=3
    X_test.loc[X_test['FareGroup']>80, 'FareGroup']=4

X_test['FareGroup'] = X_test['FareGroup'].astype(int)
X_test['FareGroup'].value_counts()

X_train['FareGroup'].describe()

X_test['FareGroup'].describe()

# Drop 'Fare' column
X_train = X_train.drop(['Fare'], axis=1)
X_test = X_test.drop(['Fare'], axis=1)

X_train.head()

### 1.2.4 One-Hot Encoding for Categorical Variables

# One-hot encoding for categorical variables
X_train = pd.get_dummies(X_train, prefix=['Pclass','Sex','Embarked','Family', 'AgeGroup', 'FareGroup'],
                         columns=['Pclass','Sex','Embarked','Family', 'AgeGroup', 'FareGroup'])
X_test = pd.get_dummies(X_test, prefix=['Pclass','Sex','Embarked','Family', 'AgeGroup', 'FareGroup'],
                         columns=['Pclass','Sex','Embarked','Family', 'AgeGroup', 'FareGroup'])

X_train.head()
```

```
X_test.head()

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)

## 1.3 Save Cleaned Training Data and Test Data

X_train.to_csv('X_train_cleaned1.csv', index=False, header=True)
X_test.to_csv('X_test_cleaned1.csv', index=False, header=True)
y_train.to_csv('y_train_cleaned1.csv', index=False, header=True)

## 1.4 Import Cleaned Training Data and Test Data

X_train = pd.read_csv("X_train_cleaned1.csv")
X_test = pd.read_csv("X_test_cleaned1.csv")
y_train = pd.read_csv("y_train_cleaned1.csv")

X_train.head()

# Reduce feature dimension by dropping FareGroup Columns
X_train = X_train.drop(['FareGroup_0', 'FareGroup_1', 'FareGroup_2', 'FareGroup_3', 'FareGroup_4'], axis=1)
X_test = X_test.drop(['FareGroup_0', 'FareGroup_1', 'FareGroup_2', 'FareGroup_3', 'FareGroup_4'], axis=1)

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=222)

print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
print(y_train.shape)
print(y_val.shape)

# Convert dataframe to numpy array
X_train = X_train.to_numpy()
X_val = X_val.to_numpy()
```

```
X_test = X_test.to_numpy()

print(X_train.shape)
print(X_val.shape)
print(X_test.shape)

# Convert labels to numpy arrays
y_train = y_train.values
y_val = y_val.values
print(type(y_train))
print(type(y_val))

## 1.5 Create Models for Stratified 10-Fold Cross Validation Training

### 1.5.1 Random Forest Classifier

# Random Forest Classifier
rf = RandomForestClassifier(random_state=4, verbose=2)
rf_model = rf.fit(X_train, y_train)
rf_score = cross_val_score(rf_model, X_val, y_val, cv=10)
print('List of validation accuracy at each fold: {}'.format(rf_score))
print('\nMaximum Validation Accuracy: {}%'.format(round(max(rf_score)*100, 3)))
print('Minimum Validation Accuracy: {}%'.format(round(min(rf_score)*100, 3)))
print('Overall Validation Accuracy: {}%'.format(round(mean(rf_score)*100, 3)))
#print("\nValidation Accuracy: {}".format(round(rf_model.score(X_val, y_val), 3)))

### 1.5.2 Support Vector Machine

# Support Vector Machine
svm = SVC(kernel='linear', random_state=7, verbose=True)
svm_model = svm.fit(X_train, y_train)
svm_score = cross_val_score(svm_model, X_val, y_val, cv=10)
print('\nList of validation accuracy at each fold: {}'.format(svm_score))
print('\nMaximum Validation Accuracy: {}%'.format(round(max(svm_score)*100, 3)))
print('Minimum Validation Accuracy: {}%'.format(round(min(svm_score)*100, 3)))
print('Overall Validation Accuracy: {}%'.format(round(mean(svm_score)*100, 3)))
```

```
#print("\nValidation Accuracy: {}".format(round(svm_model.score(X_val, y_val), 3)))


### 1.5.3 Neural Network


# Fully Connected Neural Network
def model_nn():
    model = Sequential()
    model.add(InputLayer(input_shape=(X_train.shape[1],)))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid'))

    return model


import matplotlib.pyplot as plt
%matplotlib inline

def plot_graphs(history, metrics, title):
    plt.plot(history.history[metrics])
    plt.plot(history.history['val_'+metrics])
```

```
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel(metrics)
    plt.legend([metrics, 'val_'+metrics], loc='best')
    plt.show()

#### 1.5.3.1 Stratified 10-Fold Cross Validation

X_train = pd.read_csv("X_train_cleaned1.csv")
X_test = pd.read_csv("X_test_cleaned1.csv")
y_train = pd.read_csv("y_train_cleaned1.csv")

X_train = X_train.drop(['FareGroup_0', 'FareGroup_1', 'FareGroup_2', 'FareGroup_3', 'FareGroup_4'], axis=1)
X_test = X_test.drop(['FareGroup_0', 'FareGroup_1', 'FareGroup_2', 'FareGroup_3', 'FareGroup_4'], axis=1)

X_train = X_train.to_numpy()
X_test = X_test.to_numpy()
y_train = y_train.values

print(type(X_train))
print(type(X_test))
print(type(y_train))

print(X_train.shape)
print(X_test.shape)

# 10-fold Cross Validation

batch_size = 8
epochs = 100
kfold = 10

skf = StratifiedKFold(n_splits=kfold, shuffle=True, random_state=111)
skf_accuracy = []

for i, (train_index, val_index) in enumerate(skf.split(X_train, y_train)):
```

```
    print('[Fold %d/%d]' % (i + 1, kfold))
    X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]
    model = model_nn()
    tf.random.set_seed(1234)

    # Compile the model
    optimizer = Adam(learning_rate=0.0005)
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

    # Define callbacks list
    filepath = 'nn_crossval.hdf5'
    callbacks_list = [EarlyStopping(monitor='val_loss', verbose=1, patience=20),
                      ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_best_only=True)]
    history = model.fit(X_train_fold, y_train_fold, batch_size=batch_size, epochs=epochs,
                        validation_data=(X_val_fold, y_val_fold), verbose=1, callbacks=None)
    plot_graphs(history, 'accuracy', 'Model Accuracy')
    plot_graphs(history, 'loss', 'Model Loss')
    score = model.evaluate(X_val_fold, y_val_fold)
    val_acc = score[1]
    skf_accuracy.append(val_acc)



# Print the output
print('List of validation accuracy at each fold: {}'.format(skf_accuracy))
print('\nMaximum Validation Accuracy: {}%'.format(round(max(skf_accuracy)*100, 3)))
print('Minimum Validation Accuracy: {}%'.format(round(min(skf_accuracy)*100, 3)))
print('Overall Validation Accuracy: {}%'.format(round(mean(skf_accuracy)*100, 3)))
print('Standard Deviation is: {}'.format(round(stdev(skf_accuracy), 3)))
```

1.5.4 Convolutional Neural Network

1.5.4.1 Stratified 10-Fold Cross Validation

```
X_train = pd.read_csv("X_train_cleaned1.csv")
X_test = pd.read_csv("X_test_cleaned1.csv")
```

```
y_train = pd.read_csv("y_train_cleaned1.csv")

X_train = X_train.drop(['FareGroup_0', 'FareGroup_1', 'FareGroup_2', 'FareGroup_3', 'FareGroup_4'], axis=1)
X_test = X_test.drop(['FareGroup_0', 'FareGroup_1', 'FareGroup_2', 'FareGroup_3', 'FareGroup_4'], axis=1)

X_train = X_train.to_numpy()
X_test = X_test.to_numpy()
y_train = y_train.values

print(type(X_train))
print(type(X_test))
print(type(y_train))
print(X_train.shape)
print(X_test.shape)

# Reshape training and test dataset
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1],1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1],1)

print(X_train.shape)
print(X_test.shape)

# Convolutional Neural Network
def model_cnn():
    model = Sequential()
    model.add(InputLayer(input_shape=(X_train.shape[1],1)))
    model.add(Conv1D(32, 3, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Conv1D(64, 3, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Conv1D(128, 3, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Conv1D(64, 3, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Conv1D(32, 3, activation='relu'))
    model.add(Dropout(0.3))
```

```
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))

    return model

import matplotlib.pyplot as plt
%matplotlib inline

def plot_graphs(history, metrics, title):
    plt.plot(history.history[metrics])
    plt.plot(history.history['val_'+metrics])
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel(metrics)
    plt.legend([metrics, 'val_'+metrics], loc='best')
    plt.show()

# 10-fold Cross Validation

batch_size = 8
epochs = 100
kfold = 10

skf = StratifiedKFold(n_splits=kfold, shuffle=True, random_state=111)
skf_accuracy = []

for i, (train_index, val_index) in enumerate(skf.split(X_train, y_train)):
    print('[Fold %d/%d]' % (i + 1, kfold))
    X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]
    model = model_cnn()
    tf.random.set_seed(4321)

    # Compile the model
    optimizer = Adam(learning_rate=0.0005)
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
```

```
    # Define callbacks list
    filepath = 'cnn_crossval.hdf5'
    callbacks_list = [EarlyStopping(monitor='val_loss', verbose=1, patience=20),
                      ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_best_only=True)]
    history = model.fit(X_train_fold, y_train_fold, batch_size=batch_size, epochs=epochs,
                        validation_data=(X_val_fold, y_val_fold), verbose=1, callbacks=None)
    plot_graphs(history, 'accuracy', 'Model Accuracy')
    plot_graphs(history, 'loss', 'Model Loss')
    score = model.evaluate(X_val_fold, y_val_fold)
    val_acc = score[1]
    skf_accuracy.append(val_acc)

# Print the output
print('List of validation accuracy at each fold: {}'.format(skf_accuracy))
print('\nMaximum Validation Accuracy: {}%'.format(round(max(skf_accuracy)*100, 3)))
print('Minimum Validation Accuracy: {}%'.format(round(min(skf_accuracy)*100, 3)))
print('Overall Validation Accuracy: {}%'.format(round(mean(skf_accuracy)*100, 3)))
print('Standard Deviation is: {}'.format(round(stdev(skf_accuracy), 3)))

## 1.6 Prediction on Test Dataset using NN

model = model_nn()
model.summary()

# Compile the model
tf.random.set_seed(1234)
optimizer = Adam(learning_rate=0.0005)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

batch_size = 8
epochs = 100

# Early Stopping and Save Best NN model for prediction
filepath = 'nn_crossval.hdf5'
callbacks_list = [EarlyStopping(monitor='val_loss', verbose=1, patience=20),
```

```
            ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_best_only=True)]
history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
              validation_data=(X_val, y_val), verbose=1, callbacks=callbacks_list)

plot_graphs(history, 'accuracy', 'Model Accuracy')
plot_graphs(history, 'loss', 'Model Loss')

# Load saved NN model. Comment out the below code to load saved model.
#model = load_model('nn_crossval.hdf5')
val_loss, val_acc = model.evaluate(X_val, y_val, verbose=1)

print("Validation Accuracy: {}".format(round(val_acc,3)))
print("Validation Loss: {}".format(round(val_loss,3)))

submission = pd.read_csv('submission.csv', index_col='PassengerId')
submission.head()

y_pred = model.predict_classes(X_test)
print(y_pred.shape)
submission['Survived'] = y_pred
submission.to_csv('submission_nn.csv')

submission.head()

# Join predicted labels to test dataset
submission = pd.read_csv('submission_nn.csv')
X_test = pd.read_csv('X_test_cleaned1.csv')
X_test = X_test.drop(['FareGroup_0', 'FareGroup_1', 'FareGroup_2', 'FareGroup_3', 'FareGroup_4'], axis=1)
X_test['Survived_Pred'] = submission['Survived']
X_test.head()

# How many passengers survived?
# 0 = Did not survive
# 1 = Survived
X_test['Survived_Pred'].value_counts()
```

```
# How many female survivors?
female_survivor = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Sex_female']==1)]
print("Number of female survivors: {}".format(female_survivor.shape[0]))

# How many survivors are aged 18 and below
below_18 = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['AgeGroup_0']==1)]
print("Number of survivors below 18 years old: {}".format(below_18.shape[0]))

# How many survivors are without family members onboard?
no_family = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Family_no']==1)]
print("Number of survivors that have no family members onboard: {}".format(no_family.shape[0]))

# Number of passengers in each ticket class
class1 = X_test.loc[(X_test['Pclass_1']==1)]
class2 = X_test.loc[(X_test['Pclass_2']==1)]
class3 = X_test.loc[(X_test['Pclass_3']==1)]
print("Number of passengers in Class 1: {}".format(class1.shape[0]))
print("Number of passengers in Class 2: {}".format(class2.shape[0]))
print("Number of passengers in Class 3: {}".format(class3.shape[0]))

# Passengers from which ticket class have the least chance to survive?
bad_class1 = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Pclass_1']==1)]
bad_class2 = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Pclass_2']==1)]
bad_class3 = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Pclass_3']==1)]
print("Percentage of survivors in Class 1: {}%".format(round((bad_class1.shape[0]/class1.shape[0])*100,2)))
print("Percentage of survivors in Class 2: {}%".format(round((bad_class2.shape[0]/class2.shape[0])*100,2)))
print("Percentage of survivors in Class 3: {}%".format(round((bad_class3.shape[0]/class3.shape[0])*100,2)))

# Number of passengers embarked at each port
emC = X_test.loc[(X_test['Embarked_C']==1)]
emQ = X_test.loc[(X_test['Embarked_Q']==1)]
emS = X_test.loc[(X_test['Embarked_S']==1)]
print("Number of passengers embarked at Cherbourg: {}".format(emC.shape[0]))
print("Number of passengers embarked at Queenstown: {}".format(emQ.shape[0]))
print("Number of passengers embarked at Southampton: {}".format(emS.shape[0]))
```

```
# Passengers from which port of embarkation have the least chance to survive?

bad_portc = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Embarked_C']==1)]

bad_portq = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Embarked_Q']==1)]

bad_ports = X_test.loc[(X_test['Survived_Pred']==1) & (X_test['Embarked_S']==1)]

print("Percentage of survivors embarked at Cherbourg: {}%".format(round((bad_portc.shape[0]/emC.shape[0])*100,2)))

print("Percentage of survivors embarked at Queenstown: {}%".format(round((bad_portq.shape[0]/emQ.shape[0])*100,2)))

print("Percentage of survivors embarked at Southampton: {}%".format(round((bad_ports.shape[0]/emS.shape[0])*100,2)))
```

Part J: Source Codes for Cifar-10 Image Classification

```
# 2. Cifar-10 Image Classification

import numpy as np

import tensorflow as tf

import keras

from keras.datasets import cifar10

from keras.layers import Dense, InputLayer, Dropout, Conv2D, MaxPooling2D, Flatten,
BatchNormalization

from keras.models import Sequential, load_model

from keras.optimizers import Adam, SGD

from keras.regularizers import l2

from sklearn.model_selection import train_test_split, StratifiedKFold

from sklearn.metrics import accuracy_score

from statistics import mean, stdev

import matplotlib.pyplot as plt

%matplotlib inline


# Load Cifar-10 Dataset

(X_train, y_train), (X_test, y_test) = cifar10.load_data()


print(X_train.shape)

print(X_test.shape)

print(y_train.shape)

print(y_test.shape)


print(type(X_train))

print(type(X_test))

print(type(y_train))

print(type(y_test))


trainLabel, train_count = np.unique(y_train, return_counts=True)

testLabel, test_count = np.unique(y_test, return_counts=True)

print("Training Labels: {}".format(trainLabel))

print("Counts of each training class: {}".format(train_count))
```

```
print("\n")
print("Test Labels: {}".format(testLabel))
print("Counts of each test class: {}".format(test_count))

# Print a sample of training image
print(X_train[0].shape)
print(X_train[0])

# Normalize the pixel values to values between 0 and 1
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

X_train = X_train/255
X_test = X_test/255

print(X_train[0].shape)
print(X_train[0])

y_train

# Convert array of integers into array of binary values. Number of columns = Number of class.
num_class = 10
y_train = keras.utils.to_categorical(y_train, num_class)
y_test = keras.utils.to_categorical(y_test, num_class)

print(y_train.shape)
print(y_train[0])

# Further split training data into training data and validation data.
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=333)

print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
print(y_train.shape)
print(y_val.shape)
print(y_test.shape)

# Convolutional Neural Network
def model_cnn():
    model = Sequential()
    model.add(InputLayer(input_shape=(32,32,3)))
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    model.add(Flatten())
    model.add(Dense(10, activation='softmax'))

    return model

def plot_graphs(history, metrics, title):
    plt.plot(history.history[metrics])
    plt.plot(history.history['val_'+metrics])
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel(metrics)
```

```
    plt.legend([metrics, 'val_'+metrics], loc='best')
    plt.show()

model = model_cnn()
model.summary()

# Compile the model
tf.random.set_seed(2468)
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

batch_size = 128
epochs = 50

history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
                    validation_data=(X_val, y_val), verbose=1)

plot_graphs(history, 'accuracy', 'Model Accuracy')
plot_graphs(history, 'loss', 'Model Loss')

# Prediction on test dataset
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=1)

print("Test Accuracy: {}%".format(round((test_acc*100),3)))
print("Test Loss: {}".format(round(test_loss,3)))
```