```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import plotly.express as px
from keras.utils import np_utils
from scipy.stats import multivariate_normal as mvn

#Import .py file of general algorithms
from google.colab import files
files.upload()
from general import accuracy, R2, OLS
from general import confusionMatrix
from general import SimpleLogisticRegression, ANN
from general import derivative, relu, linear, sigmoid, softmax, bin_cross_entropy, one_hot
```

Choose Files general.py

- **general.py**(n/a) - 12249 bytes, last modified: 6/11/2022 - 100% done

```
Saving general.py to general.py
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

Data Analysis

- Survival: 0=No, 1=Yes
- Sex: 0=Woman, 1=Man
-

Load the dataset

```
data = pd.read_csv('/content/drive/MyDrive/Enhance It/Training Projects/Hunger Games predicto
data.head()
```

	Sex	Age	SibSp	Parch	Fare	Title_Master	Title_Miss	Title_Mr	Title_Mrs	Titl
0	1	22.0	1	0	7.2500	0	0	1	0	
1	0	38.0	1	0	71.2833	0	0	0	1	
2	0	26.0	0	0	7.9250	0	1	0	0	

```
data['Age'] = data['Age'].astype(int)
data.dtypes
```

```
Sex              int64
Age              int64
SibSp            int64
Parch            int64
Fare           float64
                 ...
FamilySize       int64
Singleton        int64
SmallFamily      int64
LargeFamily      int64
Survived         int64
Length: 69, dtype: object
```

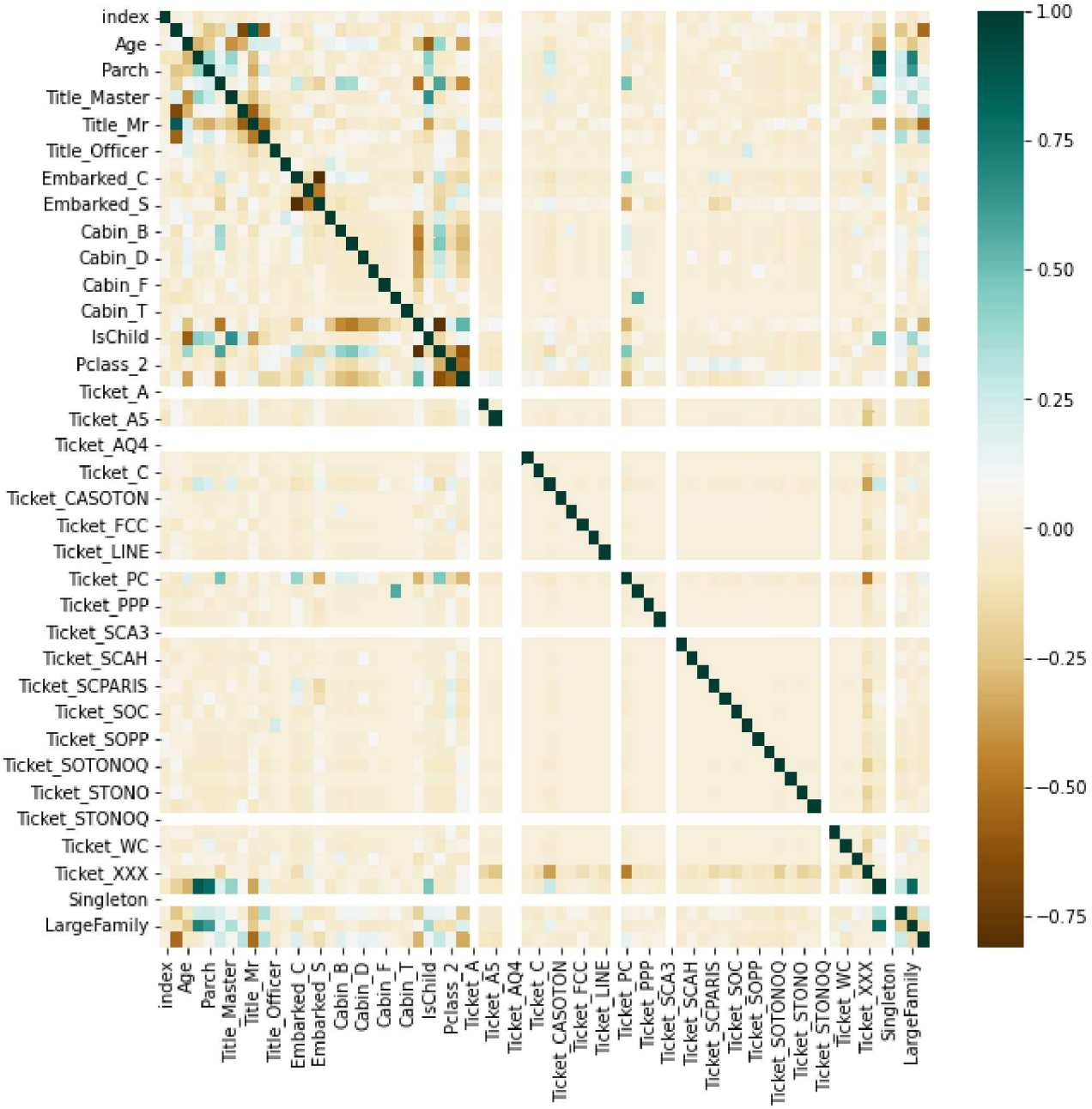
```
#Check the size of the data
print(f"Size of dataset: {data.shape}")

#Check and delete for duplicates
print(f"Number of duplicate rows: {data[data.duplicated()].shape}\n")
data.drop_duplicates(subset=None, keep="first", inplace=True)
print(f"Final size of dataset: {data.shape}")

#Check for null values
print(data.isnull().sum())

data = data.reset_index()
```

```
Size of dataset: (891, 69)
Number of duplicate rows: (96, 69)

Final size of dataset: (795, 69)
Sex            0
Age            0
SibSp          0
Parch          0
Fare           0
              ..
FamilySize     0
Singleton      0
SmallFamily    0
LargeFamily    0
Survived       0
Length: 69, dtype: int64
```

Make EDA

```
plt.figure(figsize=(10,10))
heat1 = data.corr()
sns.heatmap(heat1,cmap="BrBG",annot=False)
```

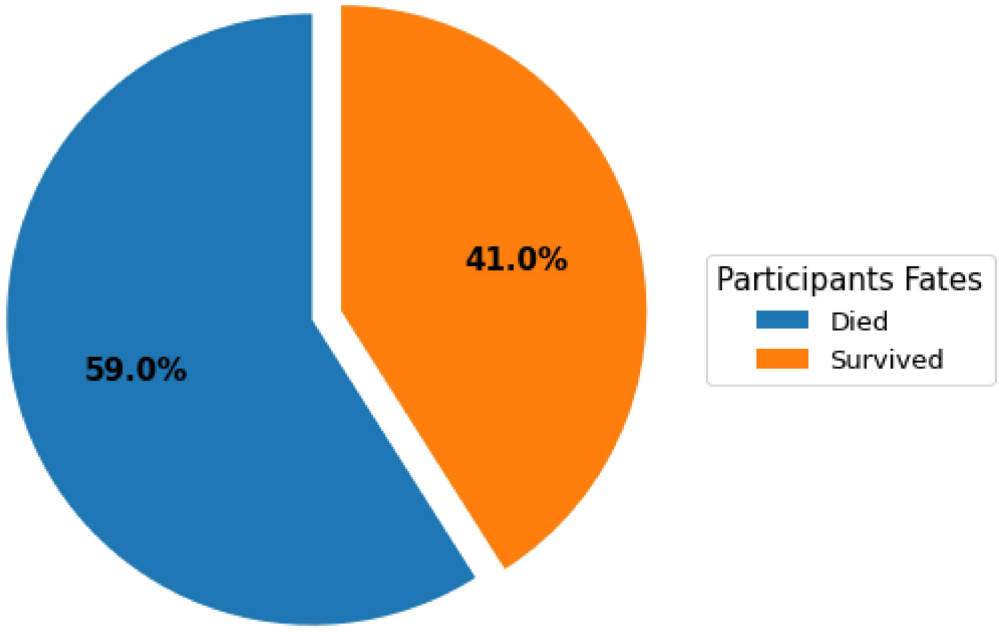
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff4fbb92e10>
```




```
sur_count = [len(data['Survived'][data['Survived']==0]), len(data['Survived'][data['Survived'
```

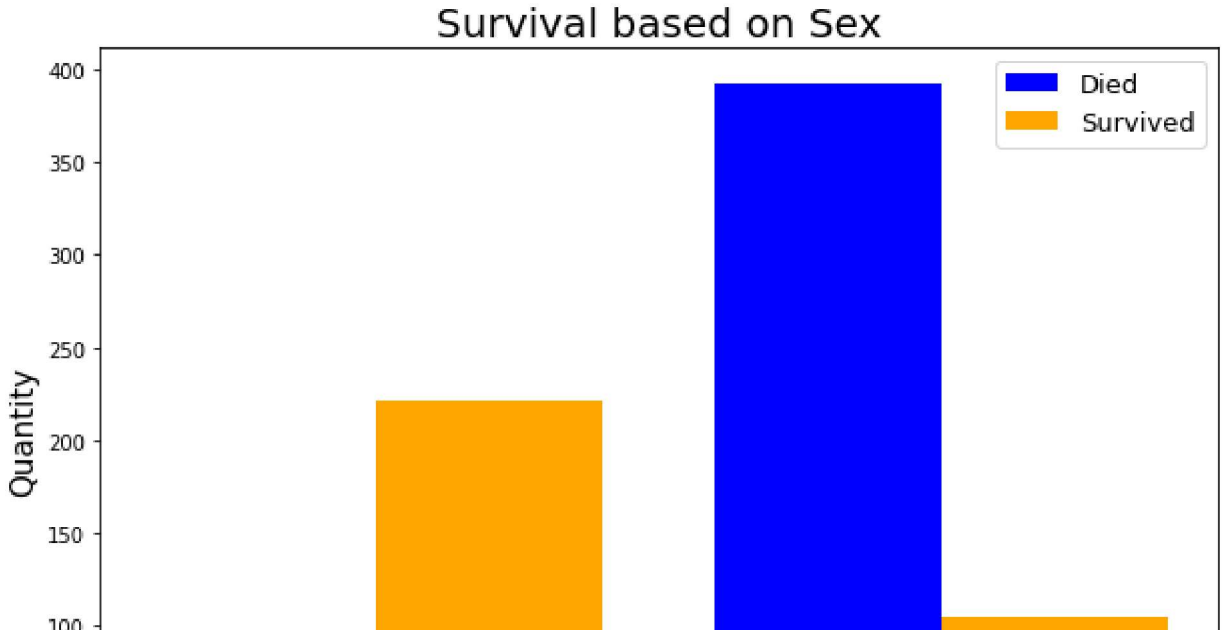
```
plt.figure(figsize=(10,7))
wedges, texts, autotexts = plt.pie(sur_count, explode=[0,0.1], autopct='%1.1f%%', startangle=
plt.legend(wedges, ['Died','Survived'],
          title="Participants Fates",
```

```
            title_fontsize=15,
            loc="center left",
            bbox_to_anchor=(1, 0.5),
            fontsize=13)
    plt.setp(autotexts, size=15, weight="bold")
    plt.show()
```



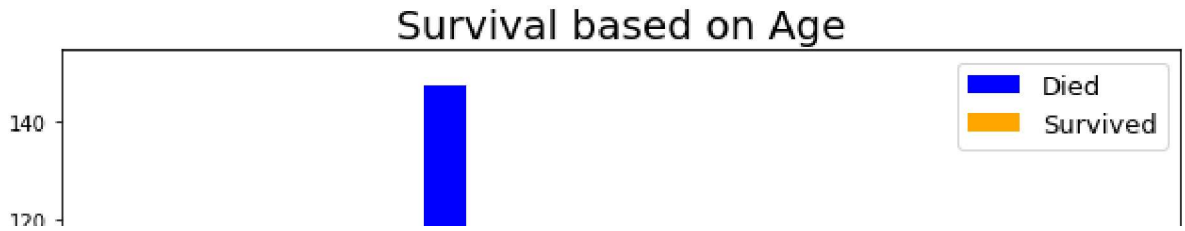

```
men_data = [data['Sex'][(data['Survived']==0) & (data['Sex']==1)], data['Sex'][(data['Survive
wom_data = [data['Sex'][(data['Survived']==0) & (data['Sex']==0)], data['Sex'][(data['Survive

plt.figure()
fig, ax = plt.subplots(figsize=(10,7))
ax.hist(men_data, 1, color=['blue','orange'], label=['Died','Survived'])
ax.hist(wom_data, 1, color=['blue','orange'])
ax.legend(fontsize=13)
ax.set_xticks(np.arange(2))
ax.set_xticklabels(['Women','Men'], size=10)
plt.ylabel('Quantity', size=15)
plt.xlabel('Sex', size=15)
plt.title('Survival based on Sex', size=20)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/fromnumeric.py:3208: VisibleDeprecatic
  return asarray(a).size
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprec
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
Text(0.5, 1.0, 'Survival based on Sex')
<Figure size 432x288 with 0 Axes>
```




```
age_counts = [data['Age'][data['Survived']==0], data['Age'][data['Survived']==1]]

plt.figure(figsize=(10,7))
plt.hist(age_counts, color=['blue','orange'], label=['Died','Survived'])
plt.legend(fontsize=13)
plt.ylabel('Quantity', size=15)
plt.xlabel('Age', size=15)
plt.title('Survival based on Age', size=20)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/fromnumeric.py:3208: VisibleDeprecatic
  return asarray(a).size
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprec
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
Text(0.5, 1.0, 'Survival based on Age')
```



Classify each person on their family size

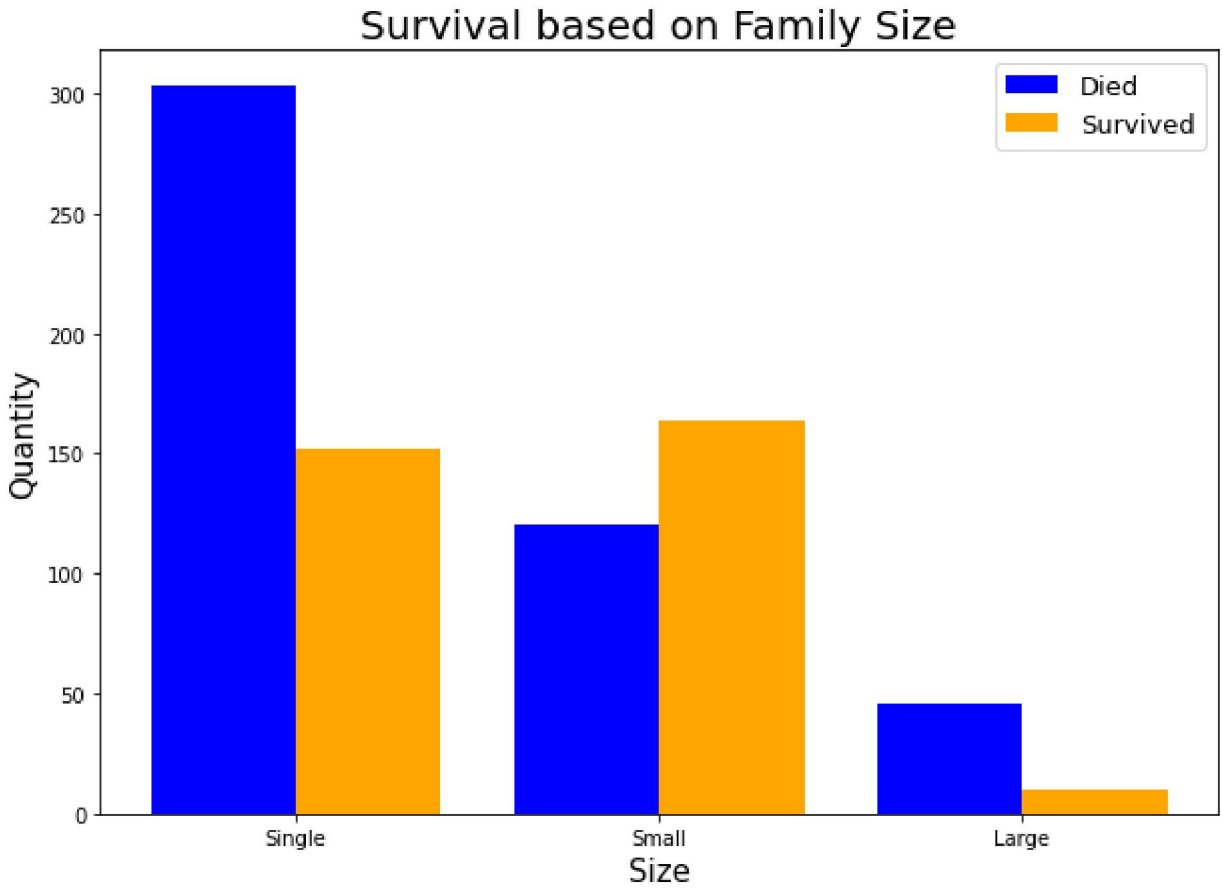
- Single(0): *SmallFamily*=0 and *LargeFamily*=0
- Small(1): *SmallFamily*=1
- Large(2): *LargeFamily*=1

```
fam_size = np.zeros(len(data))

for i in range(len(data)):
  if data['SmallFamily'][i]==1:
    fam_size[i] = 1
  elif data['LargeFamily'][i]==1:
    fam_size[i] = 2

data['fam_size'] = fam_size
print(f"Are single: {data['fam_size'][data['fam_size']==0].count()}")
print(f"Small family: {data['fam_size'][data['fam_size']==1].count()}")
print(f"Large family: {data['fam_size'][data['fam_size']==2].count()}")
```

```
Are single: 455
Small family: 284
Large family: 56
```

```
single = [data['fam_size'][(data['fam_size']==0) & (data['Survived']==0)], data['fam_size'][(
small = [data['fam_size'][(data['fam_size']==1) & (data['Survived']==0)], data['fam_size'][(d
large = [data['fam_size'][(data['fam_size']==2) & (data['Survived']==0)], data['fam_size'][(d

plt.figure()
fig, ax = plt.subplots(figsize=(10,7))
ax.hist(single, color=['blue','orange'], label=['Died','Survived'], bins=1)
ax.hist(small, color=['blue','orange'], bins=1)
ax.hist(large, color=['blue','orange'], bins=1)
ax.set_xticks(np.arange(3))
ax.set_xticklabels(['Single','Small','Large'], size=10)
plt.legend(fontsize=13)
plt.ylabel('Quantity', size=15)
plt.xlabel('Size', size=15)
plt.title('Survival based on Family Size', size=20)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/fromnumeric.py:3208: VisibleDeprecatic
  return asarray(a).size
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprec
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
Text(0.5, 1.0, 'Survival based on Family Size')
<Figure size 432x288 with 0 Axes>
```



Check the odds of survival based on the person's class

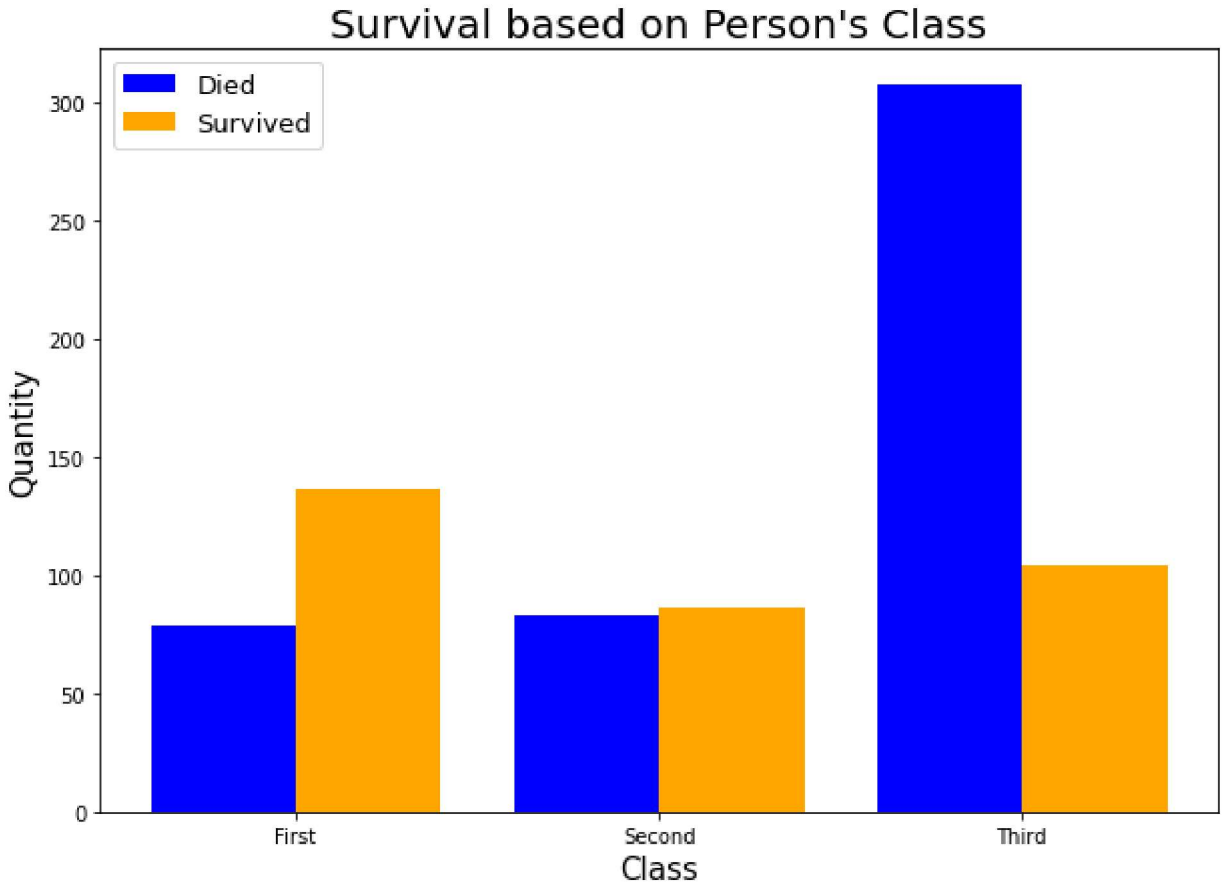
```
classes = np.zeros(len(data))

for i in range(len(data)):
  if data['Pclass_1'][i]==1:
    classes[i] = 1
  elif data['Pclass_2'][i]==1:
    classes[i] = 2
  else:
    classes[i] = 3

data['classes'] = classes
print(f"First class: {data['classes'][data['classes']==1].count()}")
print(f"Second class: {data['classes'][data['classes']==2].count()}")
print(f"Third class: {data['classes'][data['classes']==3].count()}")
```

```
First class: 215
```

```
Second class: 169
Third class: 411
```

```
first = [data['classes'][(data['classes']==1) & (data['Survived']==0)], data['classes'][(data
second = [data['classes'][(data['classes']==2) & (data['Survived']==0)], data['classes'][(dat
third = [data['classes'][(data['classes']==3) & (data['Survived']==0)], data['classes'][(data

plt.figure()
fig, ax = plt.subplots(figsize=(10,7))
ax.hist(first, color=['blue','orange'], label=['Died','Survived'], bins=1)
ax.hist(second, color=['blue','orange'], bins=1)
ax.hist(third, color=['blue','orange'], bins=1)
ax.set_xticks(np.arange(1,4))
ax.set_xticklabels(['First','Second','Third'], size=10)
plt.legend(fontsize=13)
plt.ylabel('Quantity', size=15)
plt.xlabel('Class', size=15)
plt.title("Survival based on Person's Class", size=20)
```

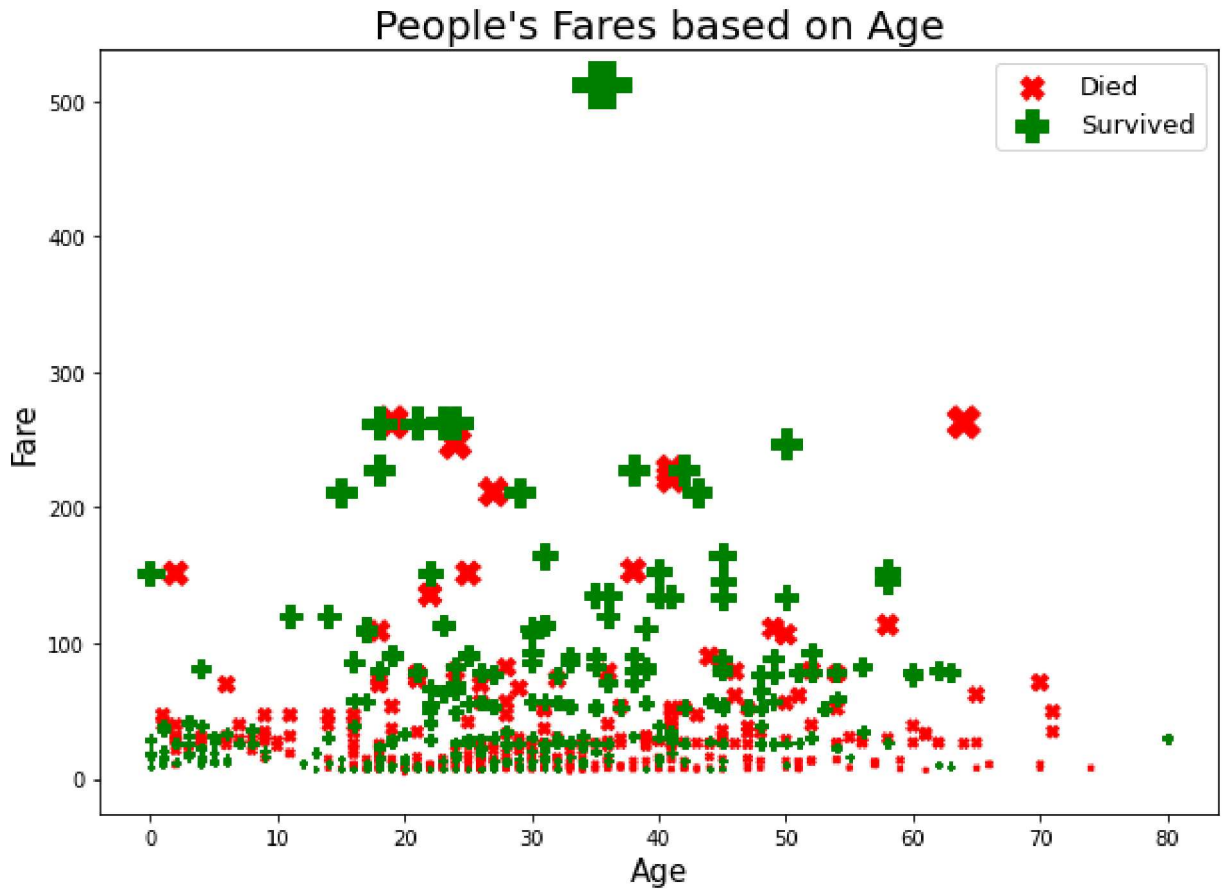
```
/usr/local/lib/python3.7/dist-packages/numpy/core/fromnumeric.py:3208: VisibleDeprecatic
  return asarray(a).size
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprec
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
Text(0.5, 1.0, "Survival based on Person's Class")
<Figure size 432x288 with 0 Axes>
```



Suppose that *Fare* represents how much does each person is valued during the Hunger Games. Analyze the fares behave based on the previous features.

```
plt.figure(figsize=(10,7))
plt.scatter(data['Age'][data['Survived']==0], data['Fare'][data['Survived']==0], marker='X',
plt.scatter(data['Age'][data['Survived']==1], data['Fare'][data['Survived']==1], marker='P',
plt.ylabel('Fare', size=15)
plt.xlabel('Age', size=15)
plt.title("People's Fares based on Age", size=20)
plt.legend(fontsize=13)
```

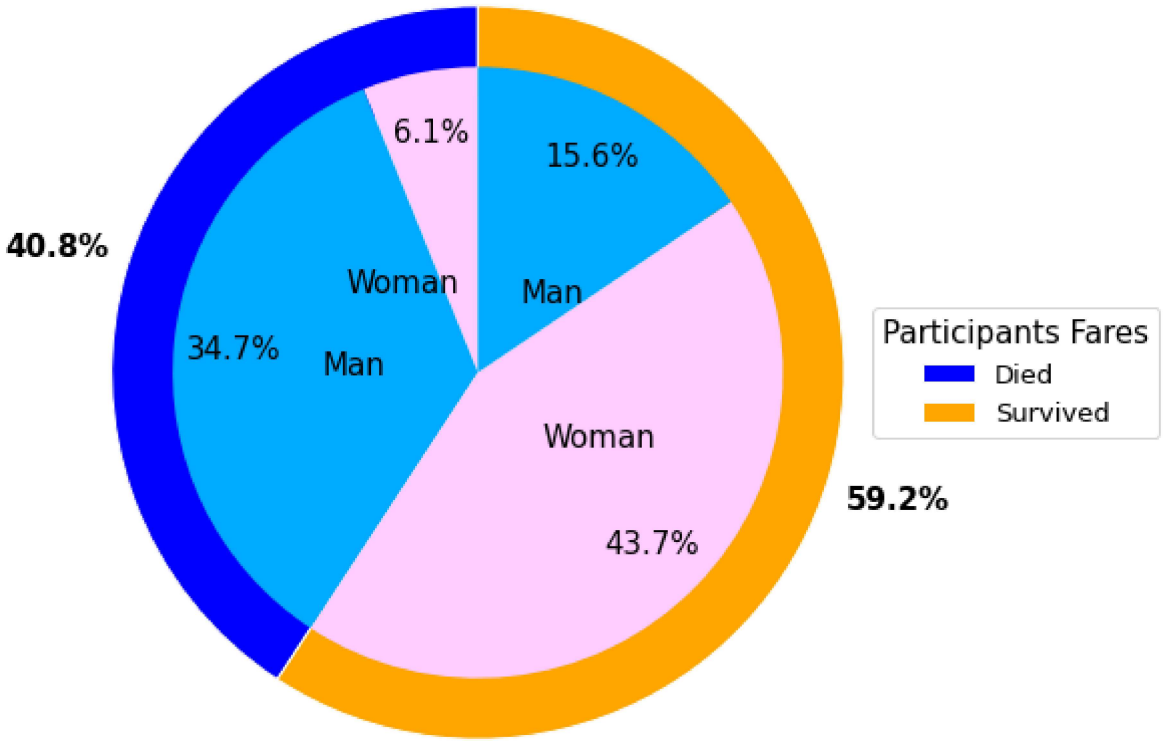
```
<matplotlib.legend.Legend at 0x7ff4f91be450>
```




```
# plt.figure(figsize=(10,7))
# plt.hist(data['Fare'][(data['Survived']==0) & (data['Sex']==1)], stacked=True)
# plt.hist(data['Fare'][(data['Survived']==1) & (data['Sex']==1)], stacked=True)
# plt.hist(data['Fare'][(data['Survived']==0) & (data['Sex']==0)], stacked=True)
# plt.hist(data['Fare'][(data['Survived']==1) & (data['Sex']==0)], stacked=True)

fare_count = [sum(data['Fare'][data['Survived']==0]), sum(data['Fare'][data['Survived']==1])]
fare_dead = [sum(data['Fare'][(data['Survived']==0) & (data['Sex']==0)]), sum(data['Fare'][(d
fare_survived = [sum(data['Fare'][(data['Survived']==1) & (data['Sex']==0)]), sum(data['Fare'
print(fare_dead, fare_survived)

plt.figure(figsize=(10,7))
```

```
wedges, texts, autotexts = plt.pie(fare_count, colors=['blue','orange'], radius=1.2, autopct=
plt.pie(fare_dead+fare_survived, colors=['#FFCCFF','#00AAFF']*2, radius=1, autopct='%1.1f%%',
plt.legend(wedges, ['Died','Survived'],
          title="Participants Fares",
          title_fontsize=15,
          loc="center left",
          bbox_to_anchor=(1, 0.5),
          fontsize=13)
plt.setp(autotexts, size=15, weight="bold")
plt.show()
```

```
[1674.9418000000005, 9498.915999999997] [11962.570899999993, 4264.808600000001]
```



Creating the datasets

Create two separate datasets: one for classifying the survival odds, and another for predicting the person's fare.

For both cases, a data distribution of 70/10/15 is followed for training, validating, and testing, respectively.

```
features = ['Sex','Age','fam_size','classes','Survived','Fare']

#Classification dataset
X_class = data[features[:5]]
X_class = X_class.sample(frac=1, replace=False, random_state=1) #Shuffle the data
X_class = X_class.to_numpy()
```

```
#Regression dataset
X_reg = data[features]
X_reg = X_reg.sample(frac=1, replace=False, random_state=1) #Shuffle the data
X_reg = X_reg.to_numpy()

#Get the indices for every set
train_idx = int(len(X_class) * 0.70)
val_idx = int(len(X_class) * 0.15)
print(f"Training set goes from 0 to {train_idx}")
print(f"Validation set goes from {train_idx+1} to {train_idx+1+val_idx}")
print(f"Test set goes from {train_idx+2+val_idx} to {len(X_class)}")
```

```
Training set goes from 0 to 556
Validation set goes from 557 to 676
Test set goes from 677 to 795
```

For the classification task two models will be used: SimpleLogisticRegression and ANN conditioned to binary classification.

```
slr_y_train = X_class[:train_idx,-1]
slr_x_train = X_class[:train_idx,:-1]
slr_y_val = X_class[train_idx:train_idx+1+val_idx,-1]
slr_x_val = X_class[train_idx:train_idx+1+val_idx,:-1]
slr_y_test = X_class[train_idx+1+val_idx:,-1]
slr_x_test = X_class[train_idx+1+val_idx:,:-1]

ann_y_train = X_class[:train_idx,-1]
ann_x_train = X_class[:train_idx,:-1]
ann_y_val = X_class[train_idx:train_idx+1+val_idx,-1]
ann_x_val = X_class[train_idx:train_idx+1+val_idx,:-1]
ann_y_test = X_class[train_idx+1+val_idx:,-1]
ann_x_test = X_class[train_idx+1+val_idx:,:-1]

print("%%%%%%%%%%%%%   Simple Logistic Regression Model   %%%%%%%%%%%%%")
print(f"Train set: {slr_x_train.shape}, {slr_y_train.shape}")
print(f"Validation set: {slr_x_val.shape}, {slr_y_val.shape}")
print(f"Test set: {slr_x_test.shape}, {slr_y_test.shape}\n")

print("%%%%%%%%%%%%%   ANN Model   %%%%%%%%%%%%%")
print(f"Train set: {ann_x_train.shape}, {ann_y_train.shape}")
print(f"Validation set: {ann_x_val.shape}, {ann_y_val.shape}")
print(f"Test set: {ann_x_test.shape}, {ann_y_test.shape}")
```

```
%%%%%%%%%%%%%   Simple Logistic Regression Model   %%%%%%%%%%%%%
Train set: (556, 4), (556,)
Validation set: (120, 4), (120,)
Test set: (119, 4), (119,)

%%%%%%%%%%%%%   ANN Model   %%%%%%%%%%%%%
Train set: (556, 4), (556,)
```

```
Validation set: (120, 4), (120,)
Test set: (119, 4), (119,)
```

The prediction of the people's fares will be only through ANN. Since there's no direct linear behavior observed on the selected features, the neural network is capable of finding a multidimensional relationship between those features.

```
reg_y_train = X_reg[:train_idx,-1].reshape((train_idx,1))
reg_x_train = X_reg[:train_idx,:-1]
reg_y_val = X_reg[train_idx:train_idx+1+val_idx,-1].reshape((120,1))
reg_x_val = X_reg[train_idx:train_idx+1+val_idx,:-1]
reg_y_test = X_reg[train_idx+1+val_idx:,-1].reshape((119,1))
reg_x_test = X_reg[train_idx+1+val_idx:,:-1]

print(f"Train set: {reg_x_train.shape}, {reg_y_train.shape}")
print(f"Validation set: {reg_x_val.shape}, {reg_y_val.shape}")
print(f"Test set: {reg_x_test.shape}, {reg_y_test.shape}")
```

```
Train set: (556, 5), (556, 1)
Validation set: (120, 5), (120, 1)
Test set: (119, 5), (119, 1)
```

Classification of the survival odds

Evaluation of the Simple Linear Regression Model

The training set will be used for selecting the adequate number of epochs, while the validation set will be used to evaluate the size of the learning rate.

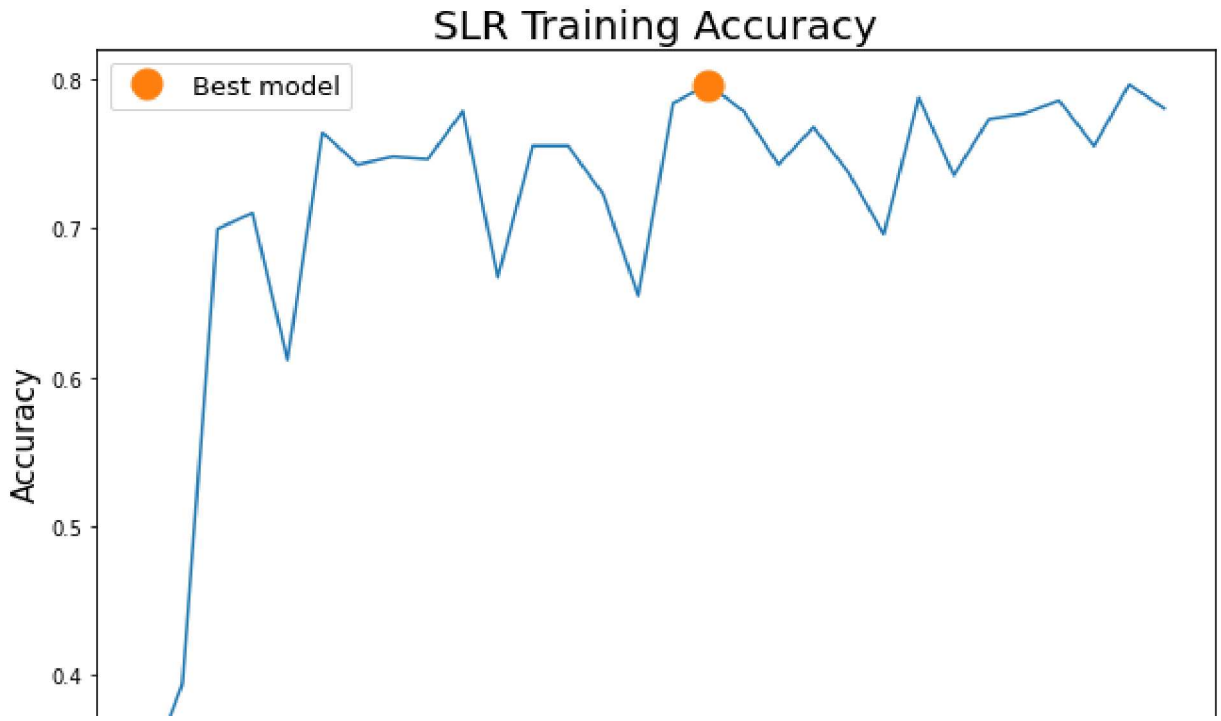
```
epochs = [i for i in range(1000,31000,1000)]
accuracies = []
log_reg = SimpleLogisticRegression()

#Evaluate the training accuracy based on the number of epochs
for epoch in epochs:
  print(f"Epoch {epochs.index(epoch)+1} of {len(epochs)}")
  log_reg.fit(slr_x_train, slr_y_train, epochs=epoch, show_curve=False)
  y_hat = log_reg.predict(slr_x_train)
  accuracies.append(accuracy(slr_y_train,y_hat))

#Save the best number of epochs
best_acc = max(accuracies)
if isinstance(best_acc, np.ndarray):
    best_acc = best_acc[0]
best_epoch = epochs[accuracies.index(best_acc)]
```

```
if isinstance(best_epoch, np.ndarray):
    best_epoch = best_epoch[0]
print(f"Best epochs: {best_epoch} with {best_acc} of accuracy")
```

```
Epoch 1 of 30
Epoch 2 of 30
Epoch 3 of 30
Epoch 4 of 30
Epoch 5 of 30
Epoch 6 of 30
/content/general.py:58: RuntimeWarning: divide by zero encountered in log
  return -(1 / len(y)) * np.sum(y * np.log(p_hat) + (1-y)*np.log(1-p_hat))
Epoch 7 of 30
Epoch 8 of 30
Epoch 9 of 30
Epoch 10 of 30
Epoch 11 of 30
/content/general.py:58: RuntimeWarning: invalid value encountered in multiply
  return -(1 / len(y)) * np.sum(y * np.log(p_hat) + (1-y)*np.log(1-p_hat))
Epoch 12 of 30
Epoch 13 of 30
Epoch 14 of 30
Epoch 15 of 30
Epoch 16 of 30
Epoch 17 of 30
Epoch 18 of 30
Epoch 19 of 30
Epoch 20 of 30
Epoch 21 of 30
Epoch 22 of 30
Epoch 23 of 30
Epoch 24 of 30
Epoch 25 of 30
Epoch 26 of 30
Epoch 27 of 30
Epoch 28 of 30
Epoch 29 of 30
Epoch 30 of 30
Best epochs: 17000 with 0.7967625899280576 of accuracy
```

```
plt.figure(figsize=(10,7))
plt.plot(epochs,accuracies)
plt.plot(best_epoch, best_acc,'o', markersize=15, label='Best model')
plt.legend(fontsize=13)
plt.xlabel('Epochs', size=15)
plt.ylabel('Accuracy', size=15)
plt.title('SLR Training Accuracy', size=20)
```

```
Text(0.5, 1.0, 'SLR Training Accuracy')
```



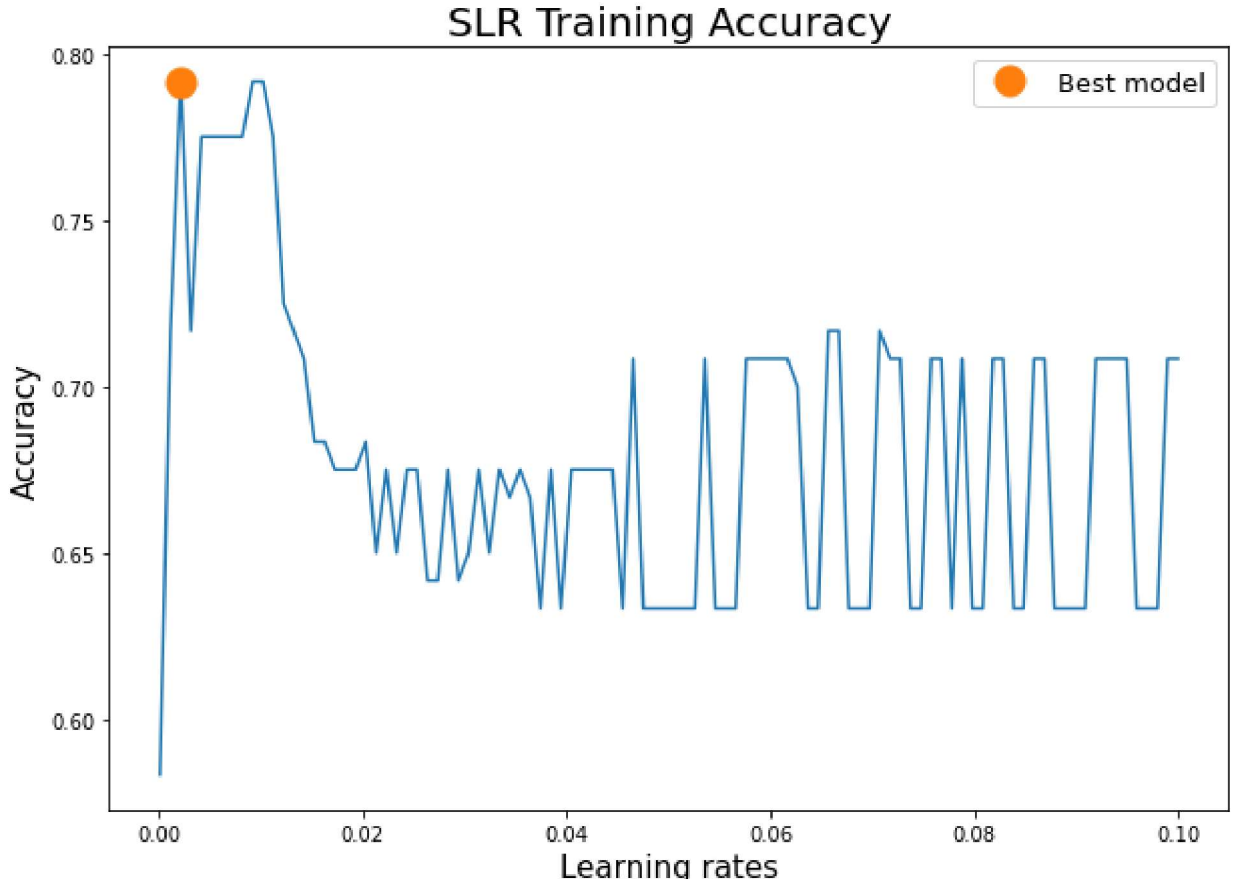

```
rates = np.linspace(1e-4, 0.1, 100)
accuracies = []

#Evaluate the validation accuracy based on the learning rate
for i in range(len(rates)):
  print(f"Learning rate {i+1} of {len(rates)}")
  log_reg.fit(slr_x_train, slr_y_train, lr=rates[i], epochs=best_epoch, show_curve=False)
  y_hat = log_reg.predict(slr_x_val)
  accuracies.append(accuracy(slr_y_val,y_hat))

#Save the best number of epochs
best_acc = max(accuracies)
if isinstance(best_acc, np.ndarray):
    best_acc = best_acc[0]
best_lr = rates[accuracies.index(best_acc)]
if isinstance(best_lr, np.ndarray):
    best_lr = best_lr[0]
print(f"Best learning rate: {best_lr} with {best_acc} of accuracy")
```

```
Learning rate 44 of 100
Learning rate 45 of 100
Learning rate 46 of 100
Learning rate 47 of 100
Learning rate 48 of 100
Learning rate 49 of 100
Learning rate 50 of 100
Learning rate 51 of 100
Learning rate 52 of 100
Learning rate 53 of 100
Learning rate 54 of 100
Learning rate 55 of 100
Learning rate 56 of 100
```

```
Learning rate 57 of 100
Learning rate 58 of 100
Learning rate 59 of 100
Learning rate 60 of 100
Learning rate 61 of 100
Learning rate 62 of 100
Learning rate 63 of 100
Learning rate 64 of 100
Learning rate 65 of 100
Learning rate 66 of 100
Learning rate 67 of 100
Learning rate 68 of 100
Learning rate 69 of 100
Learning rate 70 of 100
Learning rate 71 of 100
Learning rate 72 of 100
Learning rate 73 of 100
Learning rate 74 of 100
Learning rate 75 of 100
Learning rate 76 of 100
Learning rate 77 of 100
Learning rate 78 of 100
Learning rate 79 of 100
Learning rate 80 of 100
Learning rate 81 of 100
Learning rate 82 of 100
Learning rate 83 of 100
Learning rate 84 of 100
Learning rate 85 of 100
Learning rate 86 of 100
Learning rate 87 of 100
Learning rate 88 of 100
Learning rate 89 of 100
Learning rate 90 of 100
Learning rate 91 of 100
Learning rate 92 of 100
Learning rate 93 of 100
Learning rate 94 of 100
Learning rate 95 of 100
Learning rate 96 of 100
Learning rate 97 of 100
Learning rate 98 of 100
Learning rate 99 of 100
Learning rate 100 of 100
Best learning rate: 0.002118181818181818 with 0.7916666666666666 of accuracy
```

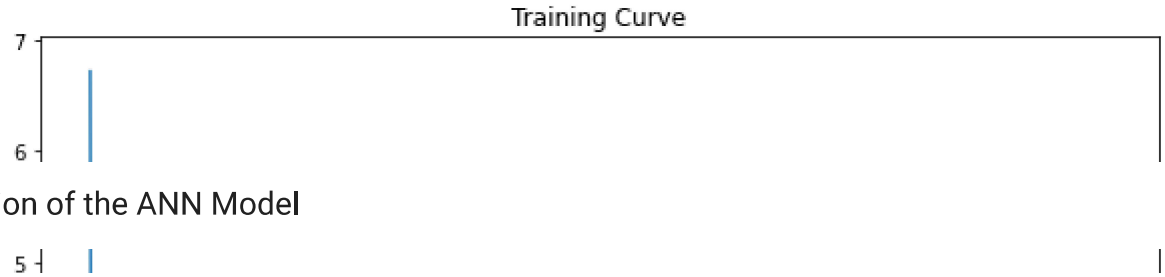
```
plt.figure(figsize=(10,7))
plt.plot(rates,accuracies)
plt.plot(best_lr, best_acc,'o', markersize=15, label='Best model')
plt.legend(fontsize=13)
plt.xlabel('Learning rates', size=15)
plt.ylabel('Accuracy', size=15)
plt.title('SLR Validation Accuracy', size=20)
```

```
Text(0.5, 1.0, 'SLR Training Accuracy')
```




```
#Train the model with the best number of epochs and learning rate for the test set
log_reg.fit(slr_x_test, slr_y_test, lr=best_lr, epochs=best_epoch, show_curve=True)
y_hat = log_reg.predict(slr_x_test)
acc = accuracy(slr_y_test,y_hat)

print(f"Testing accuracy: {acc}")
```

```
Testing accuracy: 0.7647058823529411
```



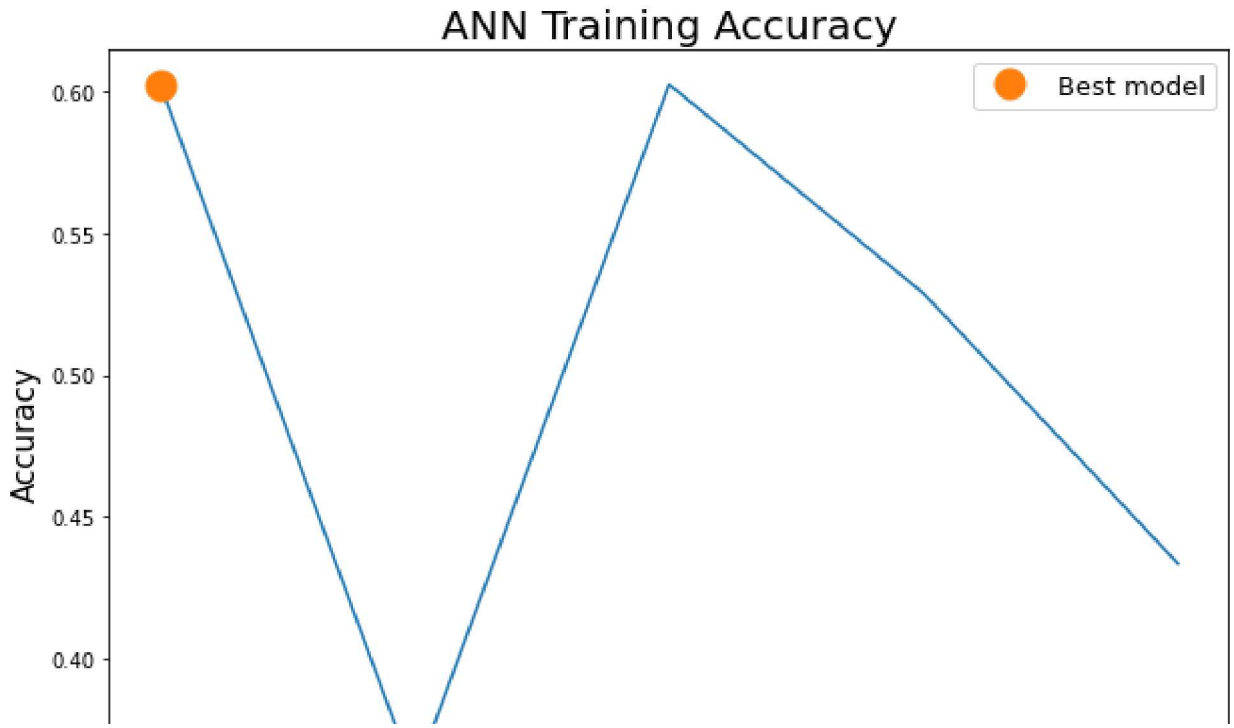
Evaluation of the ANN Model

```
models = [[6],[4],[4,6,4],[4,6,6,4],[6,4]]
acts = [[sigmoid],[sigmoid],[sigmoid,sigmoid,sigmoid],[sigmoid,sigmoid,sigmoid,sigmoid],[sigm
accuracies = []

#Train the model with the best architecture
for i in range(len(models)):
  print(f"Model {i+1} of {len(models)}")
  my_ann = ANN(models[i], activations=acts[i])
  my_ann.fit(ann_x_train, ann_y_train, show_curve=False, binary=True)
  y_hat = my_ann.predict(ann_x_train)
  accuracies.append(accuracy(ann_y_train, y_hat))

#Save the best architecture
best_acc = max(accuracies)
if isinstance(best_acc, np.ndarray):
    best_acc = best_acc[0]
best_model = models[accuracies.index(best_acc)]
print(f"Best architecture: {best_model} with {best_acc} of accuracy")
```

```
Model 1 of 5
Model 2 of 5
Model 3 of 5
Model 4 of 5
Model 5 of 5
Best architecture: [6] with 0.6025179856115108 of accuracy
```

```
plt.figure()
fig, ax = plt.subplots(figsize=(10,7))
ax.plot([i+1 for i in range(len(models))],accuracies)
ax.plot(models.index(best_model)+1, best_acc,'o', markersize=15, label='Best model')
ax.set_xticks([1,2,3,4,5])
ax.set_xticklabels(['[6]','[4]','[4,6,4]','[4,6,6,4]','[6,4]'], size=10)
plt.legend(fontsize=13)
plt.xlabel('Architecture', size=15)
plt.ylabel('Accuracy', size=15)
plt.title('ANN Training Accuracy', size=20)
```

```
Text(0.5, 1.0, 'ANN Training Accuracy')
<Figure size 432x288 with 0 Axes>
```



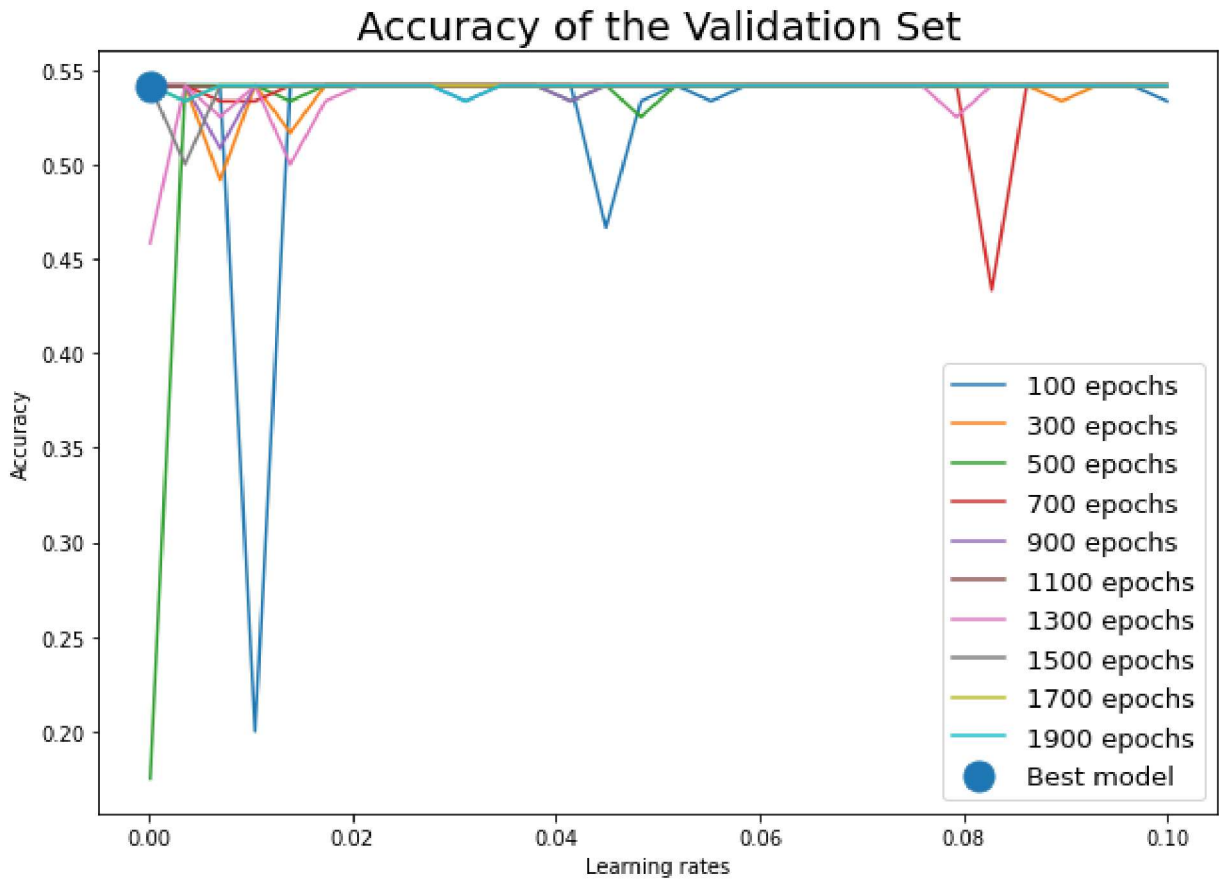

```
model = models[accuracies.index(best_acc)]
activation = acts[accuracies.index(best_acc)]
my_ann = ANN(model, activations=activation)
epochs = [i for i in range(100,2100,200)]
rates = np.linspace(1e-4, 0.1, 30)
hyperparameters = []
accuracies = []


for i in range(len(epochs)):
  print(f"Epochs {i+1} of {len(epochs)}")
  for j in range(len(rates)):
    print(f"Learning rate {j+1} of {len(rates)}")
    my_ann.fit(ann_x_train, ann_y_train, lr=rates[j], epochs=epochs[i], show_curve=False, bin
    y_hat = my_ann.predict(ann_x_val)
    accuracies.append(accuracy(ann_y_val, y_hat))
    hyperparameters.append([epochs[i],rates[j]])
  print()

#Save the best architecture
best_acc = max(accuracies)
if isinstance(best_acc, np.ndarray):
    best_acc = best_acc[0]
best_epoch, best_rate = hyperparameters[accuracies.index(best_acc)]
print(f"Best epoch: {best_epoch} Best learning rate: {best_rate} with {best_acc} of accuracy"
```

```
Learning rate 14 of 30
Learning rate 15 of 30
Learning rate 16 of 30
Learning rate 17 of 30
```

```
        Learning rate 18 of 30
        Learning rate 19 of 30
        Learning rate 20 of 30
        Learning rate 21 of 30
        Learning rate 22 of 30
        Learning rate 23 of 30
        Learning rate 24 of 30
        Learning rate 25 of 30
        Learning rate 26 of 30
        Learning rate 27 of 30
        Learning rate 28 of 30
        Learning rate 29 of 30
        Learning rate 30 of 30

        Epochs 9 of 10
        Learning rate 1 of 30
        Learning rate 2 of 30
        Learning rate 3 of 30
        Learning rate 4 of 30
        Learning rate 5 of 30
        Learning rate 6 of 30
        Learning rate 7 of 30
        Learning rate 8 of 30
        Learning rate 9 of 30
        Learning rate 10 of 30
        Learning rate 11 of 30
        Learning rate 12 of 30
        Learning rate 13 of 30
        Learning rate 14 of 30
        Learning rate 15 of 30
        Learning rate 16 of 30
        Learning rate 17 of 30
        Learning rate 18 of 30
        Learning rate 19 of 30
        Learning rate 20 of 30
        Learning rate 21 of 30
        Learning rate 22 of 30
        Learning rate 23 of 30
        Learning rate 24 of 30
        Learning rate 25 of 30
        Learning rate 26 of 30
        Learning rate 27 of 30
        Learning rate 28 of 30
        Learning rate 29 of 30
        Learning rate 30 of 30

        Epochs 10 of 10
        Learning rate 1 of 30
        Learning rate 2 of 30
        Learning rate 3 of 30
        Learning rate 4 of 30
        Learning rate 5 of 30
        Learning rate 6 of 30
        Learning rate 7 of 30

    init = 0
```

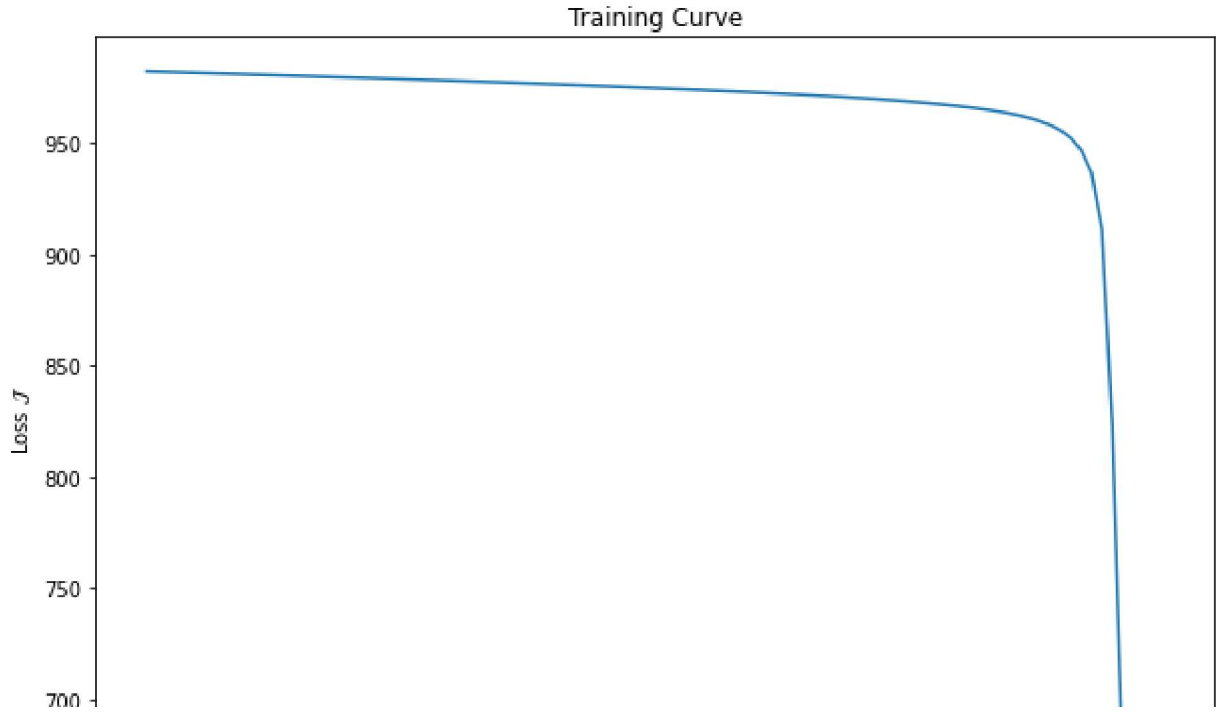
```
end = 30

plt.figure(figsize=(10,7))
for epoch in epochs:
  plt.plot(rates, accuracies[init:end], label=f'{epoch} epochs')
  init = end
  end += 30

plt.plot(best_rate, best_acc,'o', markersize=15, label='Best model')
plt.xlabel("Learning rates")
plt.ylabel('Accuracy')
plt.legend(fontsize=13)
plt.title('Accuracy of the Validation Set', size=20)
```

```
Text(0.5, 1.0, 'Accuracy of the Validation Set')
```




```
#Test the model
my_ann.fit(ann_x_train, ann_y_train, lr=best_rate, epochs=best_epoch, show_curve=True, binary
y_hat = my_ann.predict(ann_x_test)
acc = accuracy(ann_y_test, y_hat)
print(f"Test accuracy of {acc}")
```

```
Test accuracy of 0.5798319327731093
```



Prediction of the fare

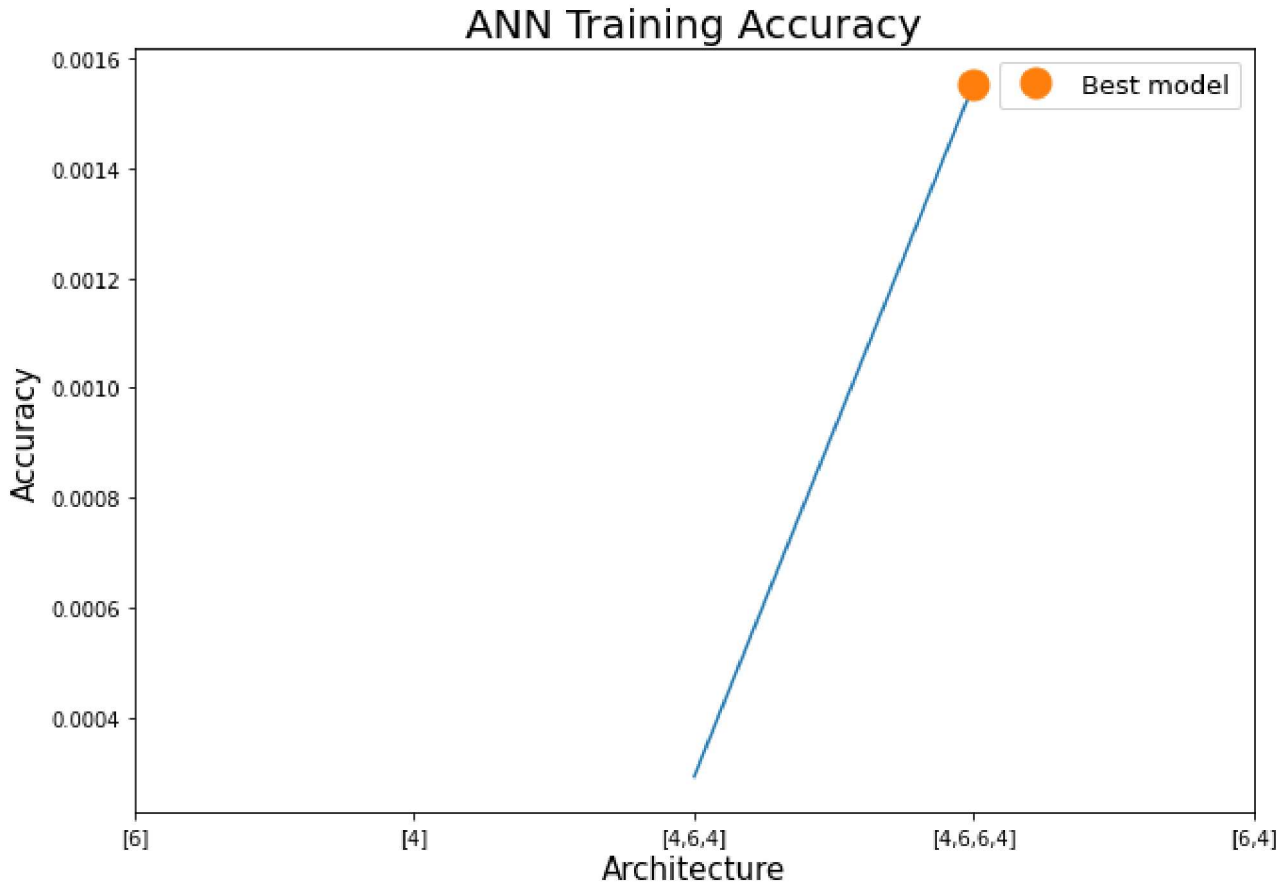
```
models = [[6],[4],[4,6,4],[4,6,6,4],[6,4]]
acts = [[relu],[relu],[relu,np.tanh,relu],[relu,np.tanh,np.tanh,relu],[relu,relu]]
r2s = np.empty(0)

#Train the model with the best architecture
for i in range(len(models)):
  print(f"Model {i+1} of {len(models)}")
  my_ann = ANN(architecture=models[i], activations=acts[i], mode=1)
  my_ann.fit(reg_x_train, reg_y_train, show_curve=False, binary=False)
  y_hat = my_ann.predict(reg_x_train)
  r2s = np.append(r2s,R2(reg_y_train, y_hat))

#Save the best architecture
best_acc = np.nanmax(r2s)
if isinstance(best_acc, np.ndarray):
    best_acc = best_acc[0]
best_model = models[np.where(r2s==best_acc)[0][0]]
print(f"Best architecture: {best_model} with {best_acc} of R2")
```

```
Model 1 of 5
Model 2 of 5
/content/general.py:33: RuntimeWarning: overflow encountered in square
  return (1/(2*len(y))) * np.sum((y-y_hat)**2)
/content/general.py:449: RuntimeWarning: overflow encountered in matmul
  dZ = dH @ self.W[l].T
/content/general.py:438: RuntimeWarning: invalid value encountered in matmul
  dW = self.Z[l-1].T @ dH
```

```
/content/general.py:438: RuntimeWarning: overflow encountered in matmul
  dW = self.Z[l-1].T @ dH
Model 3 of 5
Model 4 of 5
Model 5 of 5
Best architecture: [4, 6, 6, 4] with 0.0015542366259390095 of R2
```

```
plt.figure()
fig, ax = plt.subplots(figsize=(10,7))
ax.plot([i+1 for i in range(len(models))],r2s)
ax.plot(models.index(best_model)+1, best_acc,'o', markersize=15, label='Best model')
ax.set_xticks([1,2,3,4,5])
ax.set_xticklabels(['[6]','[4]','[4,6,4]','[4,6,6,4]','[6,4]'], size=10)
plt.legend(fontsize=13)
plt.xlabel('Architecture', size=15)
plt.ylabel('Accuracy', size=15)
plt.title('ANN Training Accuracy', size=20)
```

```
Text(0.5, 1.0, 'ANN Training Accuracy')
<Figure size 432x288 with 0 Axes>
```




```
model = models[np.where(r2s==best_acc)[0][0]]
activation = acts[np.where(r2s==best_acc)[0][0]]
my_ann = ANN(model, activations=activation, mode=1)
epochs = [i for i in range(100,2100,200)]
rates = np.linspace(1e-4, 0.1, 30)
hyperparameters = []
```

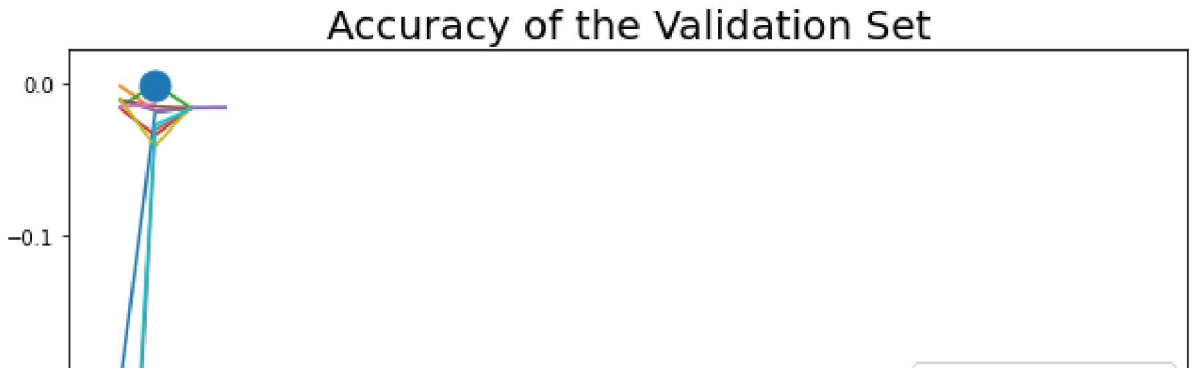
```
r2s = np.empty(0)


for i in range(len(epochs)):
  print(f"Epochs {i+1} of {len(epochs)}")
  for j in range(len(rates)):
    print(f"Learning rate {j+1} of {len(rates)}")
    my_ann.fit(reg_x_train, reg_y_train, lr=rates[j], epochs=epochs[i], show_curve=False, bin
    y_hat = my_ann.predict(reg_x_val)
    r2s = np.append(r2s, R2(reg_y_val, y_hat))
    hyperparameters.append([epochs[i],rates[j]])
  print()

#Save the best architecture
best_acc = np.nanmax(r2s)
if isinstance(best_acc, np.ndarray):
    best_acc = best_acc[0]
best_epoch, best_rate = hyperparameters[np.where(r2s==best_acc)[0][0]]
print(f"Best epoch: {best_epoch} Best learning rate: {best_rate} with {best_acc} of R2")
```

```
    Learning rate 5 of 30
    Learning rate 6 of 30
    Learning rate 7 of 30
    Learning rate 8 of 30
    Learning rate 9 of 30
    Learning rate 10 of 30
    Learning rate 11 of 30
    Learning rate 12 of 30
    Learning rate 13 of 30
    Learning rate 14 of 30
    Learning rate 15 of 30
    Learning rate 16 of 30
    Learning rate 17 of 30
    Learning rate 18 of 30
    Learning rate 19 of 30
    Learning rate 20 of 30
    Learning rate 21 of 30
    Learning rate 22 of 30
    Learning rate 23 of 30
    Learning rate 24 of 30
    Learning rate 25 of 30
    Learning rate 26 of 30
    Learning rate 27 of 30
    Learning rate 28 of 30
    Learning rate 29 of 30
    Learning rate 30 of 30

    Epochs 10 of 10
    Learning rate 1 of 30
    Learning rate 2 of 30
    Learning rate 3 of 30
    Learning rate 4 of 30
    Learning rate 5 of 30
    Learning rate 6 of 30
    Learning rate 7 of 30
    Learning rate 8 of 30
```

```
Learning rate 9 of 30
Learning rate 10 of 30
Learning rate 11 of 30
Learning rate 12 of 30
Learning rate 13 of 30
Learning rate 14 of 30
Learning rate 15 of 30
Learning rate 16 of 30
Learning rate 17 of 30
Learning rate 18 of 30
Learning rate 19 of 30
Learning rate 20 of 30
Learning rate 21 of 30
Learning rate 22 of 30
Learning rate 23 of 30
Learning rate 24 of 30
Learning rate 25 of 30
Learning rate 26 of 30
Learning rate 27 of 30
Learning rate 28 of 30
Learning rate 29 of 30
```

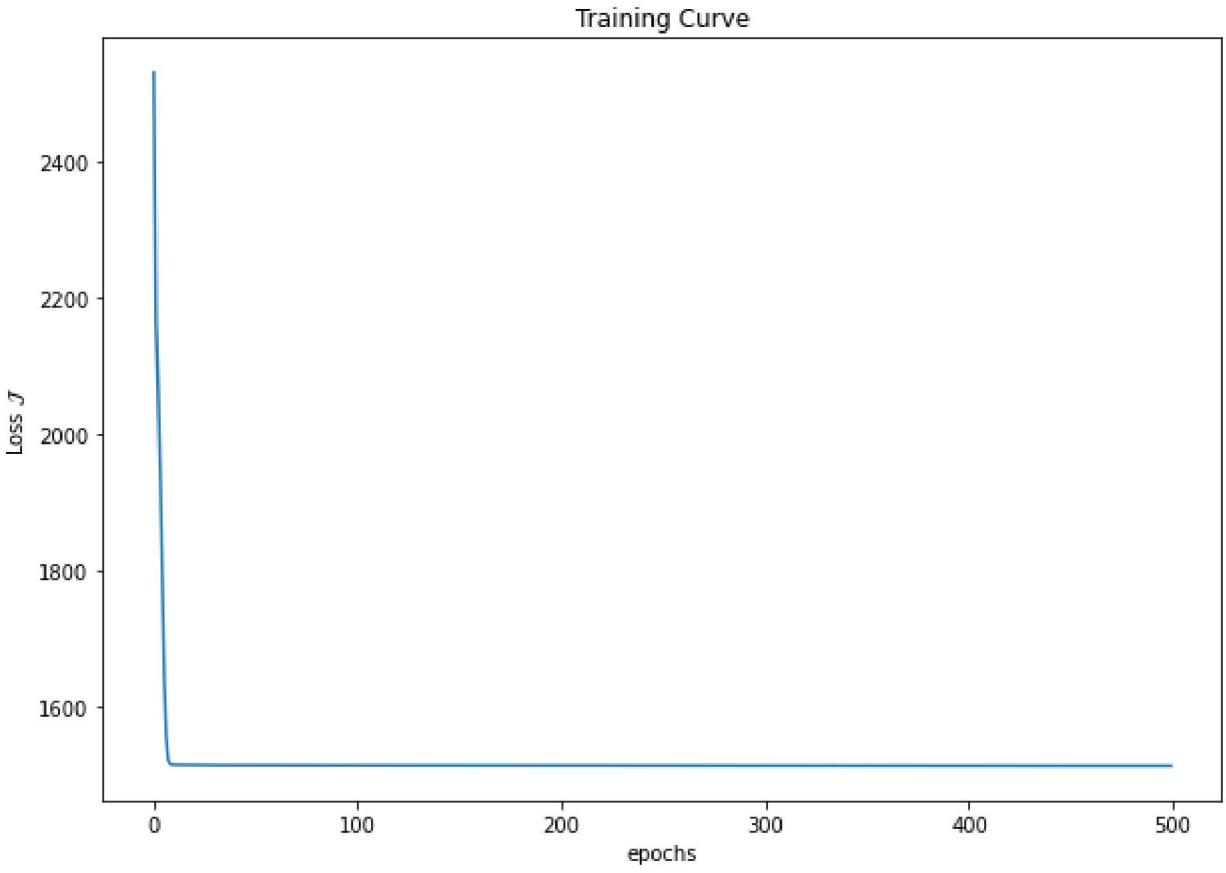
```
init = 0
end = 30

plt.figure(figsize=(10,7))
for epoch in epochs:
  plt.plot(rates, r2s[init:end], label=f'{epoch} epochs')
  init = end
  end += 30

plt.plot(best_rate, best_acc,'o', markersize=15, label='Best model')
plt.xlabel("Learning rates")
plt.ylabel('Accuracy')
plt.legend(fontsize=13)
plt.title('Accuracy of the Validation Set', size=20)
```

```
Text(0.5, 1.0, 'Accuracy of the Validation Set')
```




```
my_ann = ANN(model, activations=activation, mode=1)
my_ann.fit(reg_x_train, reg_y_train, lr=best_rate, epochs=best_epoch, show_curve=True, binary
y_hat = my_ann.predict(reg_x_test)
r2 = R2(reg_y_test, y_hat)
print(f"Test R2 is of {r2}")
```

```
Test R2 is of -0.0038660225386351055
```



✓ 0s completed at 3:44 PM