About Titanic Dataset

The Titanic dataset is a famous dataset that contains information about passengers aboard the Titanic ship, which sank in 1912 after colliding with an iceberg. The dataset is often used in data science and machine learning education and competitions as a starting point for exploring data analysis and predictive modeling techniques.

The Titanic dataset contains information about **1309** passengers, including their age, gender, ticket class, cabin, port of embarkation, and whether they survived or not. The goal of many analyses and models built on the Titanic dataset is to predict whether a given passenger would have survived the disaster.

The variables in the Titanic dataset are as follows: **PassengerId: Unique identifier for each passenger Survived: Whether the passenger survived (0 = No, 1 = Yes) Pclass: Ticket class (1 = 1st, 2 = 2nd, 3 = 3rd) Name: Passenger name Sex: Passenger gender Age: Passenger age SibSp: Number of siblings/spouses aboard the Titanic Parch: Number of parents/children aboard the Titanic Ticket: Ticket number Fare: Passenger fare Cabin: Cabin number Embarked: Port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton)** As mentioned earlier, the main objective of many analyses and models built on the Titanic dataset is to predict whether a given passenger would have survived the disaster, based on their demographic and travel information. This is a **binary classification problem**, where the target variable is **Survived** and the predictors are the other variables in the dataset.

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

Choose Files No file chosen Upload widget is only available when the cell has been executed ir this cell to enable.

```
Saving titanic.csv to titanic.csv
User uploaded file "titanic.csv" with length 90587 bytes
```

Importing Libraries

```
import pandas as pd
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

Data Loading

```
data=pd.read_csv('/content/titanic.csv')
data.head(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925
				Futrelle,						

Data shuffling

```
data = data.sample(frac=1, random_state=42)
```

```
data.tail(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
1095	1096	0	2	Andrew, Mr. Frank Thomas	male	25.0	0	0	C.A. 34050	10.500
1130	1131	1	1	Douglas, Mrs. Walter Donald (Mahala Dutton)	female	48.0	1	0	PC 17761	106.425

Data Dimention:- No. of Rows and Columns

```
data.shape
```

```
(1309, 12)
```

Double-click (or enter) to edit

Double-click (or enter) to edit

```
print("Number of Rows",data.shape[0])
print("Number of Columns",data.shape[1])
```

```
Number of Rows 1309
Number of Columns 12
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 1148 to 1126
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  1309 non-null   int64
 1   Survived     1309 non-null   int64
 2   Pclass       1309 non-null   int64
 3   Name         1309 non-null   object
 4   Sex          1309 non-null   object
 5   Age          1046 non-null   float64
 6   SibSp        1309 non-null   int64
 7   Parch        1309 non-null   int64
 8   Ticket       1309 non-null   object
 9   Fare         1308 non-null   float64
```

```
 10  Cabin        295 non-null    object
 11  Embarked     1307 non-null   object
dtypes: float64(2), int64(5), object(5)
memory usage: 132.9+ KB
```

Get Overall Statistics About The Dataframe

```
data.describe(include='all')
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	
count	1309.000000	1309.000000	1309.000000	1309	1309	1046.000000	1309.000000	13(
unique	NaN	NaN	NaN	1307	2	NaN	NaN	
top	NaN	NaN	NaN	Kelly, Mr. James	male	NaN	NaN	
freq	NaN	NaN	NaN	2	843	NaN	NaN	
mean	655.000000	0.377387	2.294882	NaN	NaN	29.881138	0.498854	
std	378.020061	0.484918	0.837836	NaN	NaN	14.413493	1.041658	
min	1.000000	0.000000	1.000000	NaN	NaN	0.170000	0.000000	
25%	328.000000	0.000000	2.000000	NaN	NaN	21.000000	0.000000	
50%	655.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	
75%	982.000000	1.000000	3.000000	NaN	NaN	39.000000	1.000000	
max	1309.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	

Data Preprocessing & Data Cleaning

Data Filtering

```
data.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

```
data[['Name','Age']]
```

	Name	Age
1148	Niklasson, Mr. Samuel	28.0
1049	Borebank, Mr. John James	42.0
982	Pedersen, Mr. Olaf	NaN
808	Meyer, Mr. August	39.0
1195	McCarthy, Miss. Catherine Katie""	NaN
...	...	...
1095	Andrew, Mr. Frank Thomas	25.0
1130	Douglas, Mrs. Walter Donald (Mahala Dutton)	48.0
1294	Carrau, Mr. Jose Pedro	17.0
860	Hansen, Mr. Claus Peter	41.0
1126	Vendel, Mr. Olof Edvin	20.0

1309 rows × 2 columns

```
sum(data['Sex']=='male')
```

```
843
```

```
data[data['Sex']=='male'].head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
1148	1149	0	3	Niklasson, Mr. Samuel	male	28.0	0	0	363611	8.0500
1049	1050	0	1	Borebank, Mr. John James	male	42.0	0	0	110489	26.5500
982	983	0	3	Pedersen, Mr. Olaf	male	NaN	0	0	345498	7.7750

```
sum(data['Survived']==1)
```

```
494
```

Check Missing (Null) Values In The *Dataset*

```
data.isnull().sum()
```

```
PassengerId       0
Survived          0
Pclass            0
Name              0
Sex               0
Age             263
SibSp             0
Parch             0
Ticket            0
Fare              1
Cabin          1014
Embarked          2
dtype: int64
```

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(data.isnull())
```

```
<Axes: >
```


```
per_missing = data.isnull().sum() * 100 / len(data)
```

Drop the Column

```
data.drop('Cabin', axis=1,inplace=True)
```


```
data.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            263
SibSp            0
Parch            0
Ticket           0
Fare             1
Embarked         2
dtype: int64
```

Handle Missing Values

```
data['Embarked'].mode()
```

```
0    S
Name: Embarked, dtype: object
```

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

```
data.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            263
SibSp            0
Parch            0
Ticket           0
Fare             1
Embarked         0
dtype: int64
```

```
data['Age']
```

```
1148    28.0
1049    42.0
982      NaN
808     39.0
1195     NaN
        ...
1095    25.0
1130    48.0
1294    17.0
860     41.0
1126    20.0
Name: Age, Length: 1309, dtype: float64
```

```
data['Age'].fillna(data['Age'].mean(), inplace = True)
```

```
data.isnull().sum()
```

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

```
data.isnull().sum()
```

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

```
data['Fare'].fillna(data['Fare'].mean(), inplace = True)
```

```
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
1148	1149	0	3	Niklasson, Mr. Samuel	male	28.000000	0	0	363611
1049	1050	0	1	Borebank, Mr. John James	male	42.000000	0	0	110489
982	983	0	3	Pedersen, Mr. Olaf	male	29.881138	0	0	345498

```
data['Sex'].unique()
```

```
array(['male', 'female'], dtype=object)
```

```
data['Gender']=data['Sex'].map({'male':1, 'female':0})
```

```
data.head(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
1148	1149	0	3	Niklasson, Mr. Samuel	male	28.000000	0	0	363611
1049	1050	0	1	Borebank, Mr. John James	male	42.000000	0	0	110489
982	983	0	3	Pedersen, Mr. Olaf	male	29.881138	0	0	345498
808	809	0	2	Meyer, Mr. August	male	39.000000	0	0	248723
1195	1196	1	3	McCarthy, Miss. Catherine	female	29.881138	0	0	383123

Data Encoding

```
x=data['Sex'].map({'male':1, 'female':0})
```

```
data['Embarked'].unique()
```

```
array(['S', 'Q', 'C'], dtype=object)
```

```
pd.get_dummies(data,columns=['Embarked'])
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
1148	1149	0	3	Niklasson, Mr. Samuel	male	28.000000	0	0	363611
1049	1050	0	1	Borebank, Mr. John James	male	42.000000	0	0	110489
982	983	0	3	Pedersen, Mr. Olaf	male	29.881138	0	0	345498
808	809	0	2	Meyer, Mr. August	male	39.000000	0	0	248723
1195	1196	1	3	McCarthy, Miss. Catherine Katie""	female	29.881138	0	0	383123
...	...	...	...	...	...	...	...	...	...
1095	1096	0	2	Andrew, Mr. Frank Thomas	male	25.000000	0	0	C.A. 34050
1130	1131	1	1	Douglas, Mrs. Walter Donald (Mahala Dutton)	female	48.000000	1	0	PC 17761
1294	1295	0	1	Carrau, Mr. Jose Pedro	male	17.000000	0	0	113059
860	861	0	3	Hansen, Mr. Claus Peter Edvin	male	41.000000	2	0	350026

```
data1=pd.get_dummies(data,columns=['Embarked'],drop_first=True)
```

```
data1.head(1)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	G
1148	1149	0	3	Niklasson, Mr. Samuel	male	28.0	0	0	363611	8.05	

Visual Analysis

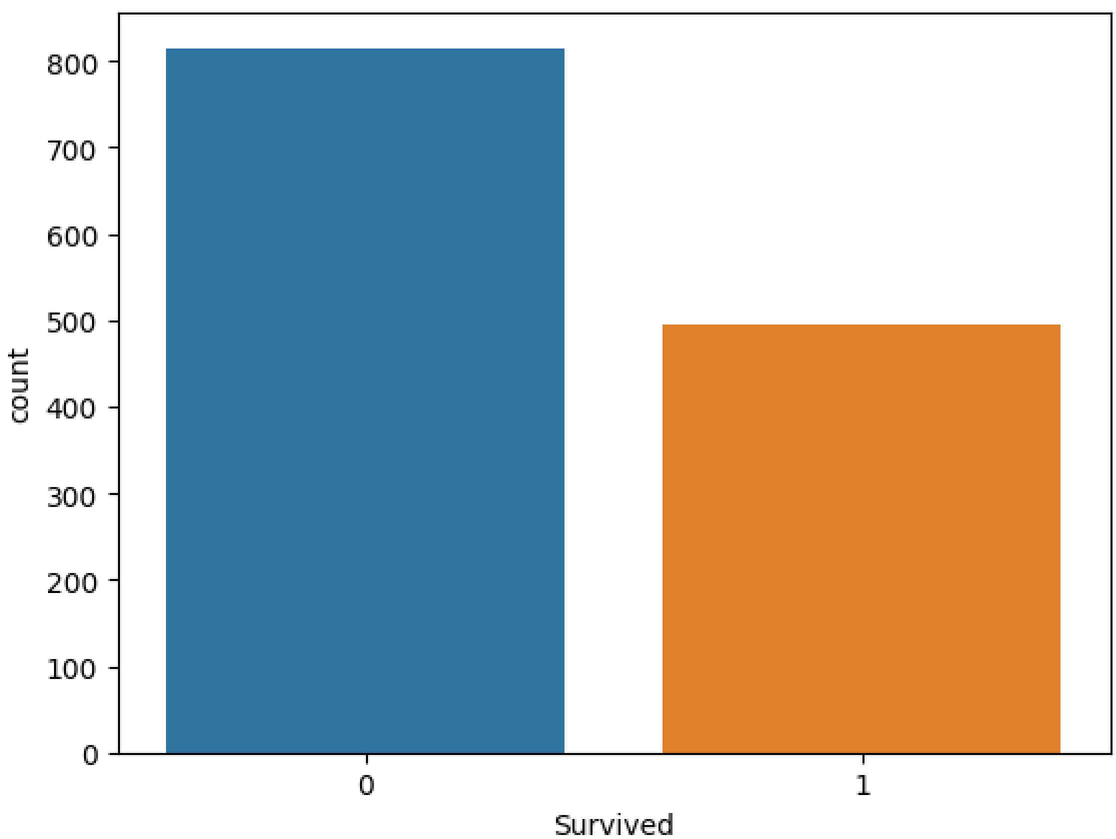
How Many People Survived And How Many Died?

```
data['Survived'].value_counts()
```

```
0    815
1    494
Name: Survived, dtype: int64
```

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x='Survived',data=data)
```

```
<Axes: xlabel='Survived', ylabel='count'>
```



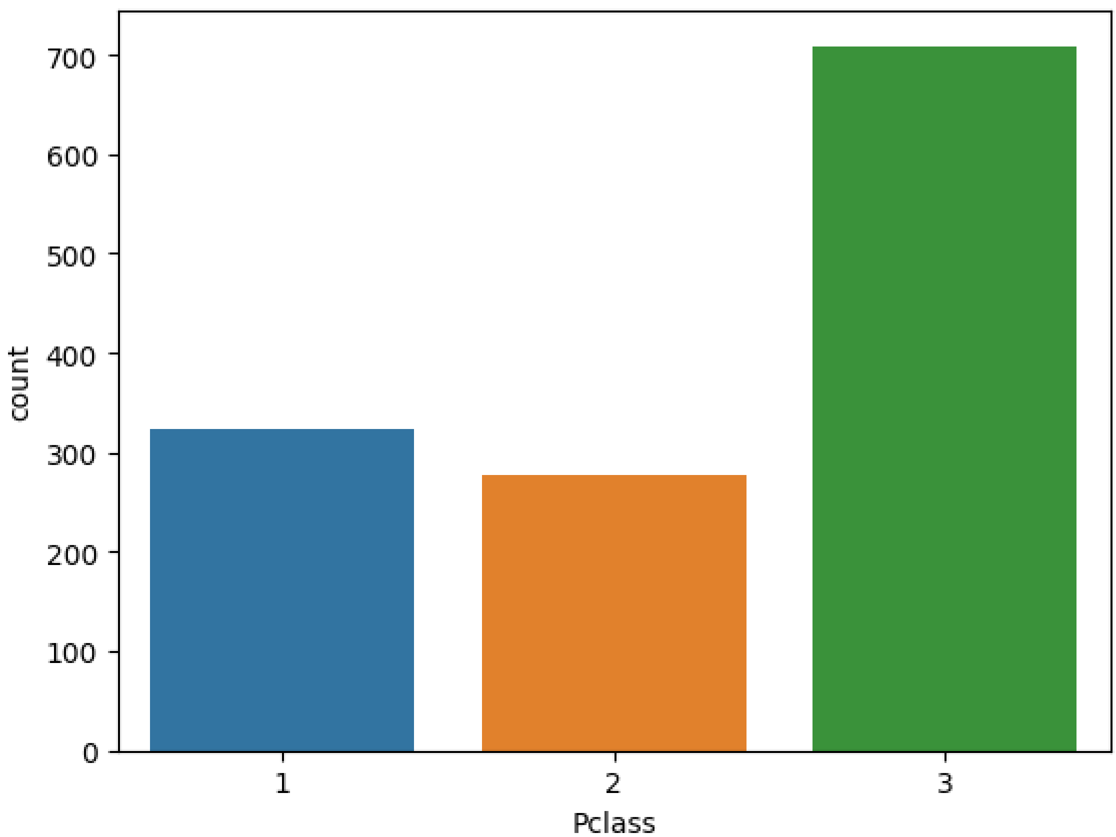
How Many Passengers Were In First Class, Second Class, and Third Class?

```
data['Pclass'].value_counts()
```

```
3    709
1    323
2    277
Name: Pclass, dtype: int64
```

```
sns.countplot(x='Pclass', data=data)
```

```
<Axes: xlabel='Pclass', ylabel='count'>
```



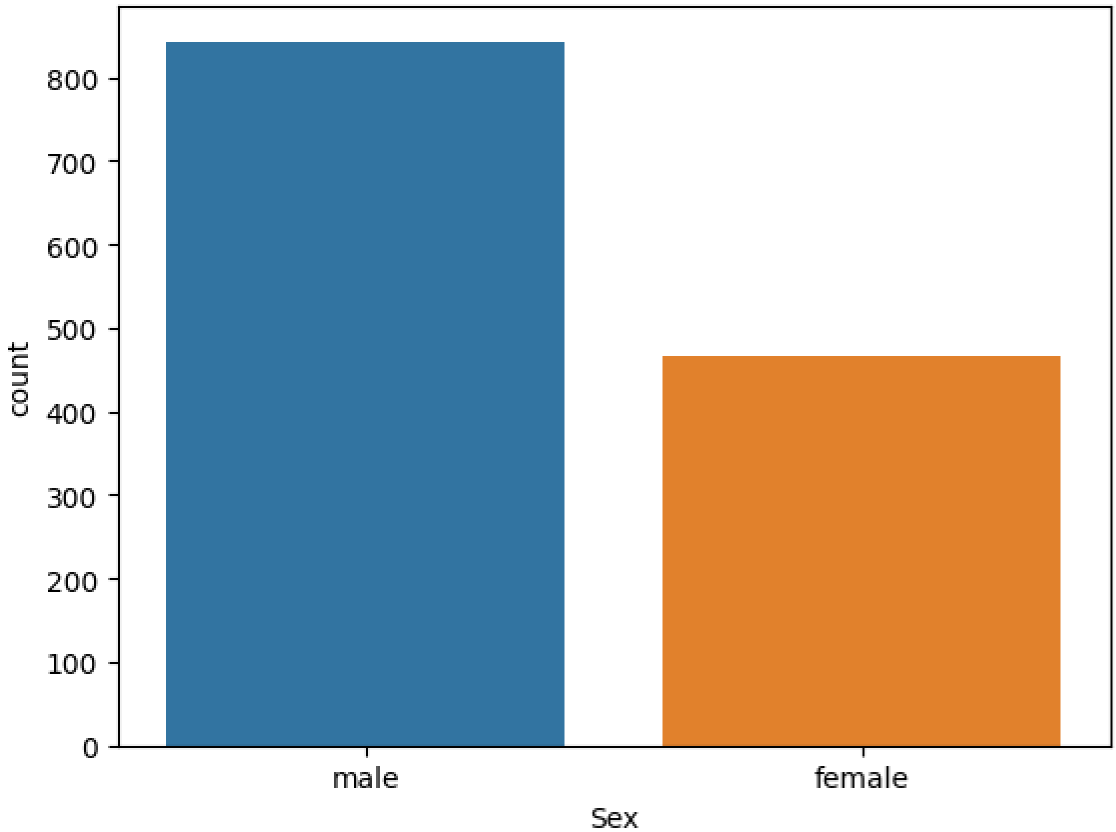
Number of Male And Female Passengers

```
data['Sex'].value_counts()
```

```
male      843
female    466
Name: Sex, dtype: int64
```

```
sns.countplot(x ='Sex', data = data)
```

```
<Axes: xlabel='Sex', ylabel='count'>
```




```
plt.hist(data['Age'])
```

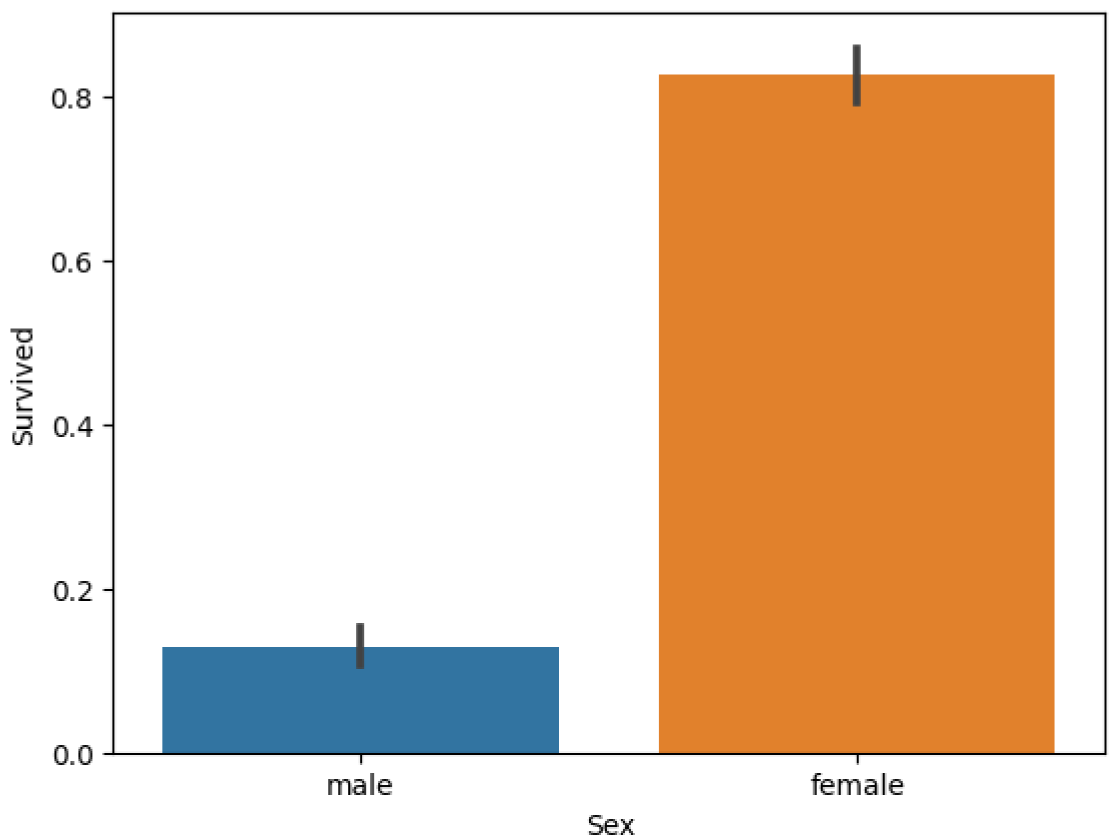
```
(array([ 72.,  62., 274., 513., 161., 108.,  65.,  41.,  10.,   3.]),
 array([ 0.17 ,  8.153, 16.136, 24.119, 32.102, 40.085, 48.068, 56.051,
        64.034, 72.017, 80.   ]),
 <BarContainer object of 10 artists>)
```

12. Bivariate Analysis

How Has Better Chance of Survival Male or Female?

```
sns.barplot(x='Sex',y='Survived',data=data)
```

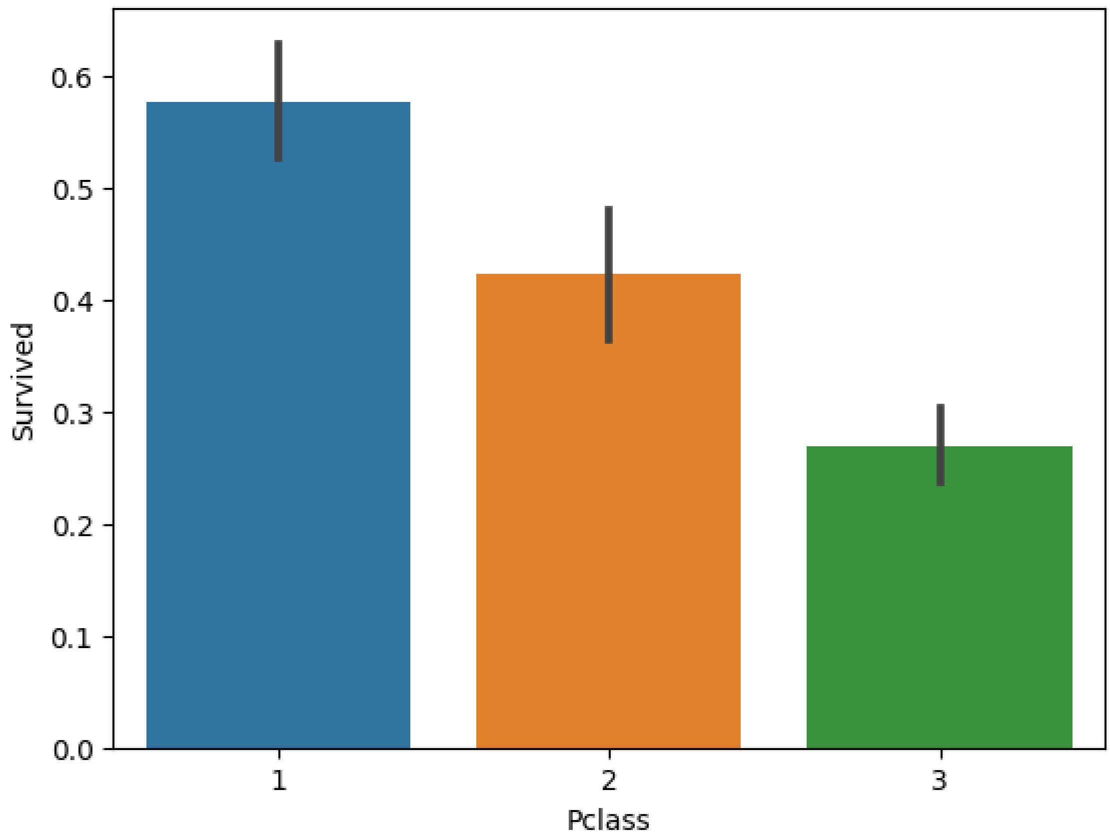
```
<Axes: xlabel='Sex', ylabel='Survived'>
```



Which Passenger Class Has Better Chance of Surviva(First, Second, Or Third Class)?

```
sns.barplot(x="Pclass", y="Survived",data=data)
```

<Axes: xlabel='Pclass', ylabel='Survived'>




```
# Convert categorical variables to numeric
data = pd.get_dummies(data, columns=['Sex', 'Embarked'])
data.head(5)
```

	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare
1148	1149	0	3	Niklasson, Mr. Samuel	28.000000	0	0	363611	8.050
1049	1050	0	1	Borebank, Mr. John James	42.000000	0	0	110489	26.550
982	983	0	3	Pedersen, Mr. Olaf	29.881138	0	0	345498	7.775

```
data=data.drop(['PassengerId', 'Name', 'Ticket'], axis=1)
```

Dataset Splitting into test and train

```
# Split the data into training and testing sets
X = data.drop('Survived', axis=1)
y = data['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Data Scaling

```
# Scale the numeric features
scaler = StandardScaler()
X_train[['Age', 'Fare']] = scaler.fit_transform(X_train[['Age', 'Fare']])
X_test[['Age', 'Fare']] = scaler.transform(X_test[['Age', 'Fare']])
```

Model 1: Logistic regression using ANN

```
# Define the model
model = Sequential()
model.add(Dense(1, input_shape=(X_train.shape[1],), activation='sigmoid'))

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

```
# Model Fitting with 100 epochs and 32 batch_size
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1)
```

```
Epoch 1/100
33/33 [==============================] - 1s 2ms/step - loss: 0.6077 - accuracy: 0.70
Epoch 2/100
33/33 [==============================] - 0s 2ms/step - loss: 0.5873 - accuracy: 0.72
Epoch 3/100
33/33 [==============================] - 0s 2ms/step - loss: 0.5688 - accuracy: 0.72
Epoch 4/100
33/33 [==============================] - 0s 2ms/step - loss: 0.5522 - accuracy: 0.73
Epoch 5/100
33/33 [==============================] - 0s 2ms/step - loss: 0.5369 - accuracy: 0.74
Epoch 6/100
33/33 [==============================] - 0s 2ms/step - loss: 0.5232 - accuracy: 0.75
Epoch 7/100
33/33 [==============================] - 0s 2ms/step - loss: 0.5104 - accuracy: 0.76
Epoch 8/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4988 - accuracy: 0.77
Epoch 9/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4884 - accuracy: 0.78
Epoch 10/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4787 - accuracy: 0.78
Epoch 11/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4700 - accuracy: 0.79
Epoch 12/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4625 - accuracy: 0.80
Epoch 13/100
33/33 [==============================] - 0s 3ms/step - loss: 0.4550 - accuracy: 0.81
Epoch 14/100
33/33 [==============================] - 0s 3ms/step - loss: 0.4484 - accuracy: 0.82
Epoch 15/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4426 - accuracy: 0.83
Epoch 16/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4371 - accuracy: 0.83
Epoch 17/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4322 - accuracy: 0.83
Epoch 18/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4277 - accuracy: 0.83
Epoch 19/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4236 - accuracy: 0.84
Epoch 20/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4200 - accuracy: 0.85
Epoch 21/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4166 - accuracy: 0.85
Epoch 22/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4136 - accuracy: 0.85
Epoch 23/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4108 - accuracy: 0.85
Epoch 24/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4081 - accuracy: 0.85
Epoch 25/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4059 - accuracy: 0.85
Epoch 26/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4038 - accuracy: 0.85
```

```
Epoch 27/100
33/33 [==============================] - 0s 2ms/step - loss: 0.4017 - accuracy: 0.85
Epoch 28/100
```

```
# Evaluate the model on the test data
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Logistics regressionn Accuracy: %.2f' % (accuracy*100))
```

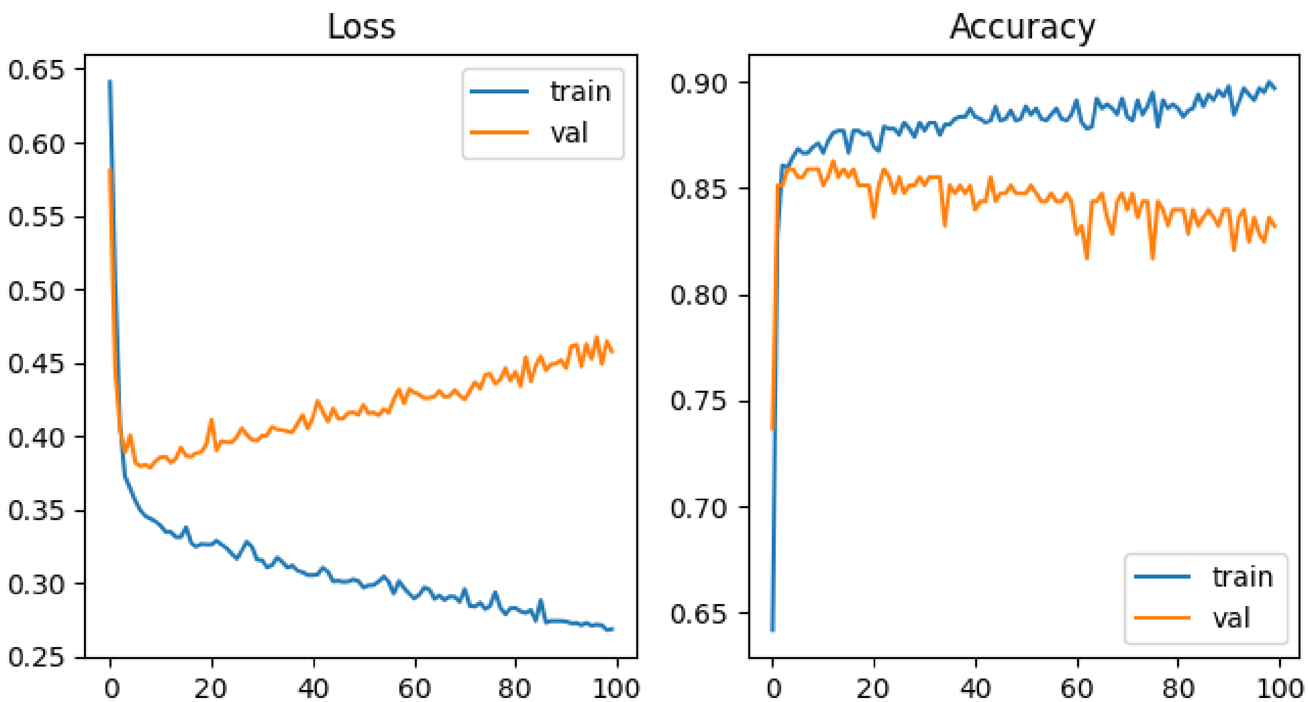
```
Logistics regressionn Accuracy: 85.11
```

Model 2: 64-32-16-8-1 using ANN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define a more complex model with more layers and neurons
model1 = Sequential()
model1.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model1.add(Dense(32, activation='relu'))
model1.add(Dense(16, activation='relu'))
model1.add(Dense(8, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))

# Compile the model
model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model for a large number of epochs
history = model1.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_
```

```
import matplotlib.pyplot as plt

# Plot the training and validation loss and accuracy
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(train_loss, label='train')
plt.plot(val_loss, label='val')
plt.legend()
plt.title('Loss')

plt.subplot(1, 2, 2)
```

```
plt.plot(train_acc, label='train')
plt.plot(val_acc, label='val')
plt.legend()
plt.title('Accuracy')
plt.show()
```



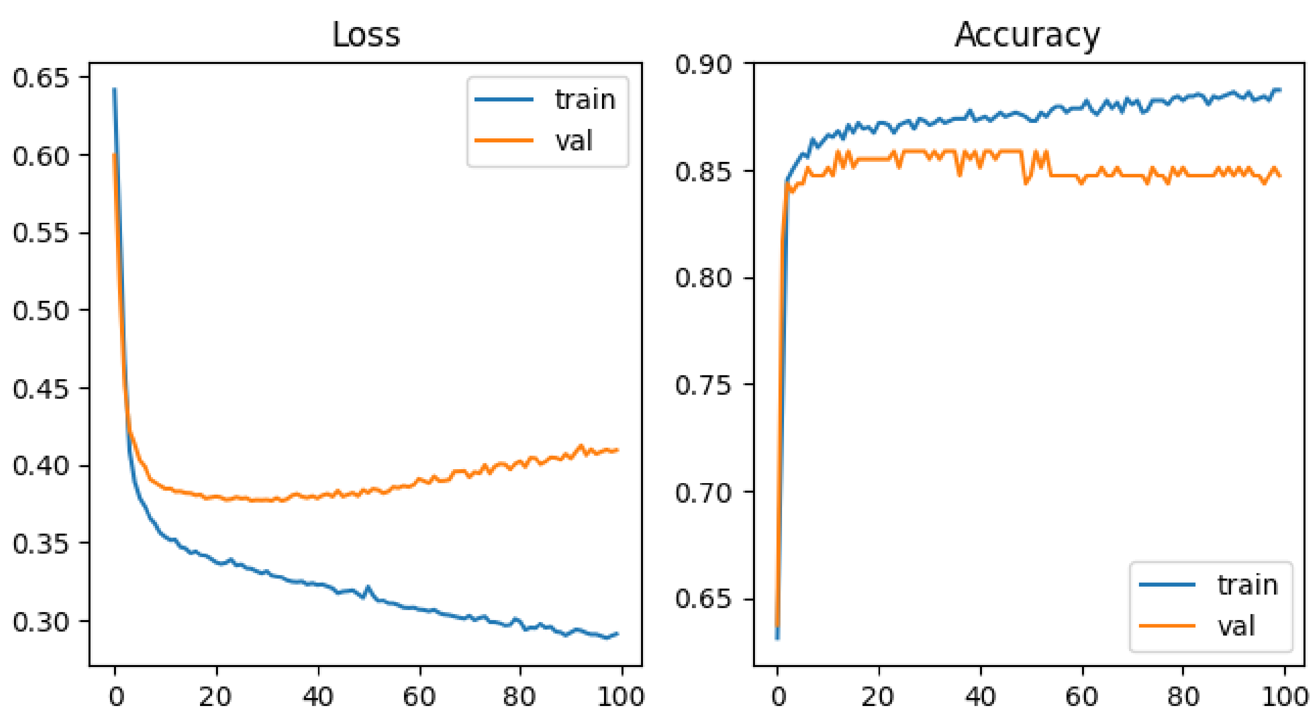

```
# Evaluate the model on the test data
loss, accuracy = model1.evaluate(X_test, y_test, verbose=0)
print('Model 2 Accuracy: %.2f' % (accuracy*100))
```

```
Model 2 Accuracy: 83.21
```

Model 3: (32-16-8-1) ANN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define a more complex model with more layers and neurons
model_3 = Sequential()
model_3.add(Dense(32, input_dim=X_train.shape[1], activation='relu'))
model_3.add(Dense(16, activation='relu'))
model_3.add(Dense(8, activation='relu'))
model_3.add(Dense(1, activation='sigmoid'))
```

```
# Compile the model
model_3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model for a large number of epochs
history = model_3.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y


import matplotlib.pyplot as plt

# Plot the training and validation loss and accuracy
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(train_loss, label='train')
plt.plot(val_loss, label='val')
plt.legend()
plt.title('Loss')

plt.subplot(1, 2, 2)
plt.plot(train_acc, label='train')
plt.plot(val_acc, label='val')
plt.legend()
plt.title('Accuracy')
plt.show()
```



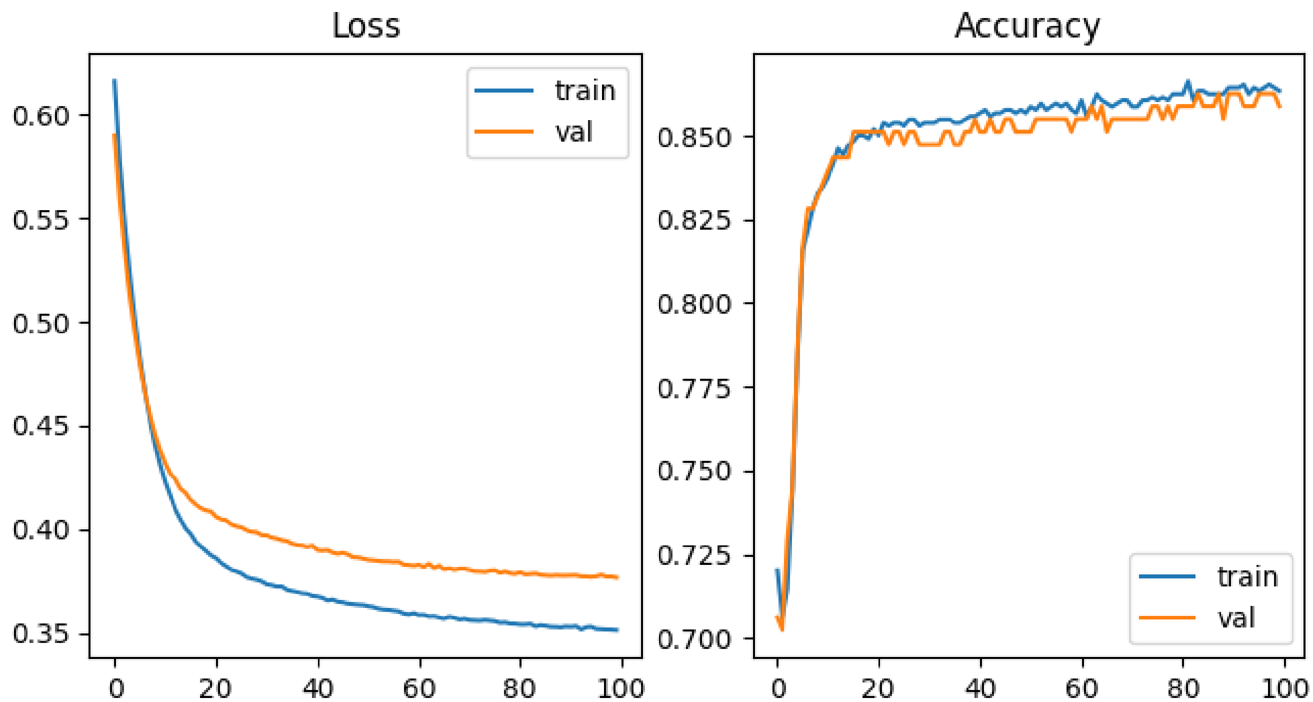
```
# Evaluate the model on the test data
loss, accuracy = model_3.evaluate(X_test, y_test, verbose=0)
print('Model 3 Accuracy: %.2f' % (accuracy*100))
```

```
Model 3 Accuracy: 84.73
```

Model 4:(16-8-1) ANN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define a more complex model with more layers and neurons
model_4 = Sequential()
model_4.add(Dense(16, input_dim=X_train.shape[1], activation='relu'))
model_4.add(Dense(8, activation='relu'))
model_4.add(Dense(1, activation='sigmoid'))

# Compile the model
model_4.compile(optimizer='SGD', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model for a large number of epochs
history = model_4.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y
```

```
import matplotlib.pyplot as plt

# Plot the training and validation loss and accuracy
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(train_loss, label='train')
plt.plot(val_loss, label='val')
plt.legend()
plt.title('Loss')

plt.subplot(1, 2, 2)
plt.plot(train_acc, label='train')
plt.plot(val_acc, label='val')
plt.legend()
plt.title('Accuracy')
plt.show()
```




```
# Evaluate the model on the test data
loss, accuracy = model_4.evaluate(X_test, y_test, verbose=0)
print('Model 4 Accuracy: %.2f' % (accuracy*100))
```

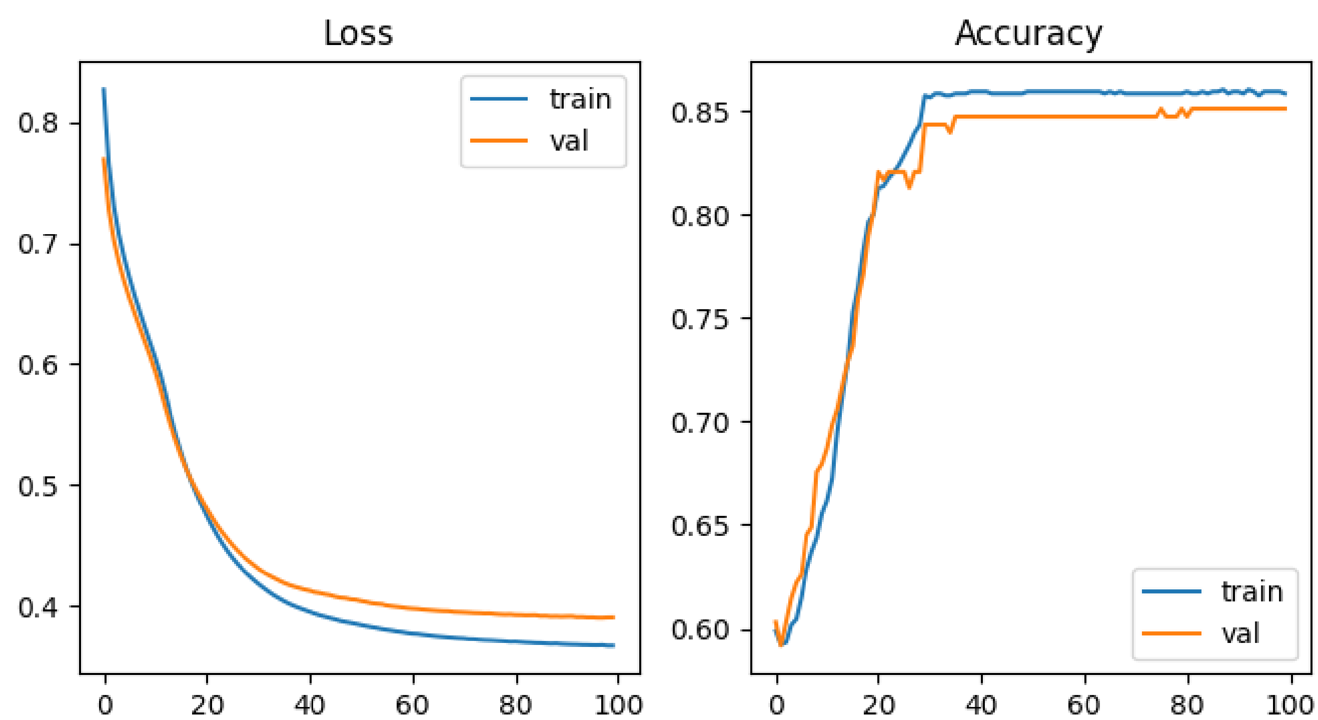
```
Model 4 Accuracy: 85.88
```

Model 5: 8-1 ANN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define a more complex model with more layers and neurons
model_5 = Sequential()
model_5.add(Dense(8, input_dim=X_train.shape[1], activation='relu'))
model_5.add(Dense(1, activation='sigmoid'))

# Compile the model
model_5.compile(optimizer='SGD', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model for a large number of epochs
history = model_5.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y
```

```
import matplotlib.pyplot as plt

# Plot the training and validation loss and accuracy
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
```

```
val_acc = history.history['val_accuracy']

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(train_loss, label='train')
plt.plot(val_loss, label='val')
plt.legend()
plt.title('Loss')

plt.subplot(1, 2, 2)
plt.plot(train_acc, label='train')
plt.plot(val_acc, label='val')
plt.legend()
plt.title('Accuracy')
plt.show()
```




```
# Evaluate the model on the test data
loss, accuracy = model_5.evaluate(X_test, y_test, verbose=0)
print('Model 5 Accuracy: %.2f' % (accuracy*100))
```

```
Model 5 Accuracy: 85.11
```

Model 6: (4-1) ANN

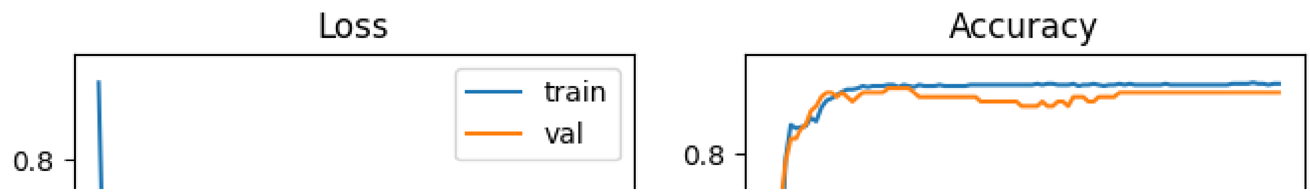
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
# Define a more complex model with more layers and neurons
model_6 = Sequential()
model_6.add(Dense(4, input_dim=X_train.shape[1], activation='relu'))
model_6.add(Dense(1, activation='sigmoid'))

# Compile the model
model_6.compile(optimizer='SGD', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model for a large number of epochs
history = model_6.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y
```

```
Epoch 1/100
33/33 [==============================] - 1s 8ms/step - loss: 0.8714 - accuracy: 0.40
Epoch 2/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6894 - accuracy: 0.47
Epoch 3/100
33/33 [==============================] - 0s 3ms/step - loss: 0.6516 - accuracy: 0.70
Epoch 4/100
33/33 [==============================] - 0s 3ms/step - loss: 0.6285 - accuracy: 0.79
Epoch 5/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6111 - accuracy: 0.82
Epoch 6/100
33/33 [==============================] - 0s 3ms/step - loss: 0.5961 - accuracy: 0.82
Epoch 7/100
33/33 [==============================] - 0s 3ms/step - loss: 0.5824 - accuracy: 0.82
Epoch 8/100
33/33 [==============================] - 0s 4ms/step - loss: 0.5688 - accuracy: 0.82
Epoch 9/100
33/33 [==============================] - 0s 4ms/step - loss: 0.5551 - accuracy: 0.83
Epoch 10/100
33/33 [==============================] - 0s 3ms/step - loss: 0.5417 - accuracy: 0.82
Epoch 11/100
33/33 [==============================] - 0s 3ms/step - loss: 0.5285 - accuracy: 0.83
Epoch 12/100
33/33 [==============================] - 0s 3ms/step - loss: 0.5153 - accuracy: 0.84
Epoch 13/100
33/33 [==============================] - 0s 3ms/step - loss: 0.5025 - accuracy: 0.84
Epoch 14/100
33/33 [==============================] - 0s 3ms/step - loss: 0.4898 - accuracy: 0.84
Epoch 15/100
33/33 [==============================] - 0s 3ms/step - loss: 0.4774 - accuracy: 0.85
Epoch 16/100
33/33 [==============================] - 0s 4ms/step - loss: 0.4655 - accuracy: 0.85
Epoch 17/100
33/33 [==============================] - 0s 3ms/step - loss: 0.4543 - accuracy: 0.85
Epoch 18/100
33/33 [==============================] - 0s 3ms/step - loss: 0.4438 - accuracy: 0.85
Epoch 19/100
33/33 [==============================] - 0s 3ms/step - loss: 0.4341 - accuracy: 0.85
Epoch 20/100
33/33 [==============================] - 0s 4ms/step - loss: 0.4253 - accuracy: 0.85
Epoch 21/100
33/33 [==============================] - 0s 4ms/step - loss: 0.4176 - accuracy: 0.85
Epoch 22/100
```

```
33/33 [==============================] - 0s 4ms/step - loss: 0.4109 - accuracy: 0.85
Epoch 23/100
33/33 [==============================] - 0s 3ms/step - loss: 0.4053 - accuracy: 0.85
Epoch 24/100
33/33 [==============================] - 0s 3ms/step - loss: 0.4006 - accuracy: 0.85
Epoch 25/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3965 - accuracy: 0.85
Epoch 26/100
33/33 [==============================] - 0s 6ms/step - loss: 0.3931 - accuracy: 0.85
Epoch 27/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3902 - accuracy: 0.85
Epoch 28/100
```

```
import matplotlib.pyplot as plt

# Plot the training and validation loss and accuracy
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(train_loss, label='train')
plt.plot(val_loss, label='val')
plt.legend()
plt.title('Loss')

plt.subplot(1, 2, 2)
plt.plot(train_acc, label='train')
plt.plot(val_acc, label='val')
plt.legend()
plt.title('Accuracy')
plt.show()
```




```
# Evaluate the model on the test data
loss, accuracy = model_6.evaluate(X_test, y_test, verbose=0)
print('Model 6 Accuracy: %.2f' % (accuracy*100))
```

```
Model 6 Accuracy: 85.11
```

Model 7:2-1 ANN

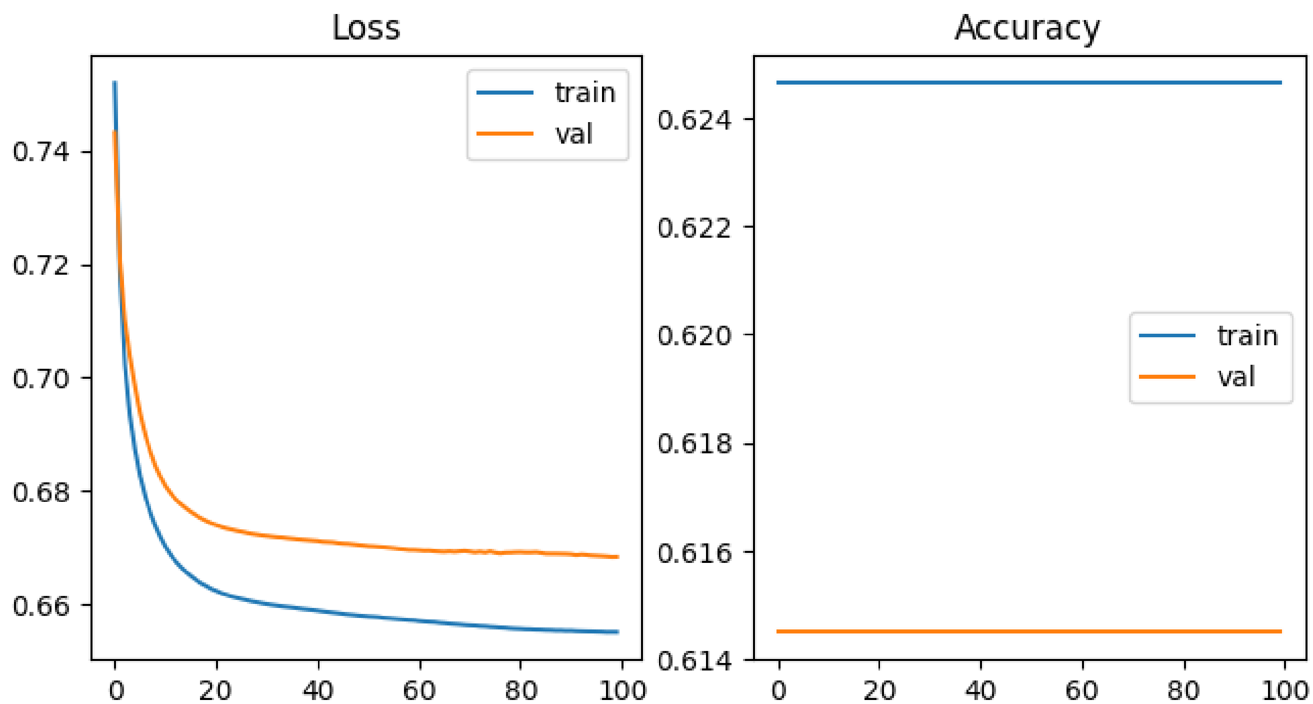

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define a more complex model with more layers and neurons
model_7 = Sequential()
model_7.add(Dense(2, input_dim=X_train.shape[1], activation='relu'))
model_7.add(Dense(1, activation='sigmoid'))

# Compile the model
model_7.compile(optimizer='SGD', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model for a large number of epochs
history = model_7.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y
```

```
Epoch 1/100
33/33 [==============================] - 1s 9ms/step - loss: 0.7519 - accuracy: 0.62
Epoch 2/100
33/33 [==============================] - 0s 3ms/step - loss: 0.7185 - accuracy: 0.62
Epoch 3/100
33/33 [==============================] - 0s 4ms/step - loss: 0.7023 - accuracy: 0.62
Epoch 4/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6934 - accuracy: 0.62
Epoch 5/100
33/33 [==============================] - 0s 3ms/step - loss: 0.6873 - accuracy: 0.62
Epoch 6/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6826 - accuracy: 0.62
Epoch 7/100
33/33 [==============================] - 0s 3ms/step - loss: 0.6791 - accuracy: 0.62
Epoch 8/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6763 - accuracy: 0.62
Epoch 9/100
33/33 [==============================] - 0s 3ms/step - loss: 0.6739 - accuracy: 0.62
Epoch 10/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6719 - accuracy: 0.62
Epoch 11/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6702 - accuracy: 0.62
```

```
Epoch 12/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6688 - accuracy: 0.62
Epoch 13/100
33/33 [==============================] - 0s 3ms/step - loss: 0.6676 - accuracy: 0.62
Epoch 14/100
33/33 [==============================] - 0s 3ms/step - loss: 0.6667 - accuracy: 0.62
Epoch 15/100
33/33 [==============================] - 0s 3ms/step - loss: 0.6658 - accuracy: 0.62
Epoch 16/100
33/33 [==============================] - 0s 3ms/step - loss: 0.6650 - accuracy: 0.62
Epoch 17/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6644 - accuracy: 0.62
Epoch 18/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6638 - accuracy: 0.62
Epoch 19/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6633 - accuracy: 0.62
Epoch 20/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6628 - accuracy: 0.62
Epoch 21/100
33/33 [==============================] - 0s 3ms/step - loss: 0.6624 - accuracy: 0.62
Epoch 22/100
33/33 [==============================] - 0s 4ms/step - loss: 0.6620 - accuracy: 0.62
Epoch 23/100
33/33 [==============================] - 0s 5ms/step - loss: 0.6617 - accuracy: 0.62
Epoch 24/100
33/33 [==============================] - 0s 5ms/step - loss: 0.6614 - accuracy: 0.62
Epoch 25/100
33/33 [==============================] - 0s 5ms/step - loss: 0.6612 - accuracy: 0.62
Epoch 26/100
33/33 [==============================] - 0s 5ms/step - loss: 0.6610 - accuracy: 0.62
Epoch 27/100
33/33 [==============================] - 0s 5ms/step - loss: 0.6608 - accuracy: 0.62
Epoch 28/100
```

```
import matplotlib.pyplot as plt

# Plot the training and validation loss and accuracy
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(train_loss, label='train')
plt.plot(val_loss, label='val')
plt.legend()
plt.title('Loss')

plt.subplot(1, 2, 2)
plt.plot(train_acc, label='train')
plt.plot(val_acc, label='val')
plt.legend()
```

```
plt.title('Accuracy')
plt.show()
```




```
# Evaluate the model on the test data
loss, accuracy = model_7.evaluate(X_test, y_test, verbose=0)
print('Model 7 Accuracy: %.2f' % (accuracy*100))
```

```
Model 7 Accuracy: 61.45
```

Model 8: Logistic regression using SKLEARN

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import pandas as pd


# Create a logistic regression model
model_8 = LogisticRegression()

# Fit the model to the training data
model_8.fit(X_train, y_train)

# Predict the target variable for the test data
y_pred = model_8.predict(X_test)
```

```
# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Logistic Regression Accuracy:%.2f'% (accuracy*100))
```

```
Logistic Regression Accuracy:85.11
```

Model 9: Random Forest

```
from sklearn.ensemble import RandomForestClassifier
# Create a random forest classifier
model_9 = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
# Fit the model to the training data
model_9.fit(X_train, y_train)

# Predict the target variable for the test data
y_pred = model_9.predict(X_test)

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Random forest Accuracy:%.2f'% (accuracy*100))
```

```
Random forest Accuracy:85.50
```