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.

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
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 titanic.csv

- **titanic.csv**(text/csv) - 90587 bytes, last modified: 4/26/2023 - 100% done

```
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	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Data shuffling

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

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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1095	1096	0	2	Andrew, Mr. Frank Thomas	male	25.0	0	0	C.A. 34050	10.5000	NaN	S
1130	1131	1	1	Douglas, Mrs. Walter Donald (Mahala Dutton)	female	48.0	1	0	PC 17761	106.4250	C86	C
1294	1295	0	1	Carrau, Mr. Jose Pedro	male	17.0	0	0	113059	47.1000	NaN	S
860	861	0	3	Hansen, Mr. Claus Peter	male	41.0	2	0	350026	14.1083	NaN	S

Data Dimention:- No. of Rows and Columns

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
(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	Parch	Ticket	Fare	Cabin
count	1309.000000	1309.000000	1309.000000	1309	1309	1046.000000	1309.000000	1309.000000	1309	1308.000000	295
unique	NaN	NaN	NaN	1307	2	NaN	NaN	NaN	929	NaN	186
top	NaN	NaN	NaN	Kelly, Mr. James	male	NaN	NaN	NaN	CA. 2343	NaN	C23 C25 C27
freq	NaN	NaN	NaN	2	843	NaN	NaN	NaN	11	NaN	6
mean	655.000000	0.377387	2.294882	NaN	NaN	29.881138	0.498854	0.385027	NaN	33.295479	NaN
std	378.020061	0.484918	0.837836	NaN	NaN	14.413493	1.041658	0.865560	NaN	51.758668	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.170000	0.000000	0.000000	NaN	0.000000	NaN
25%	328.000000	0.000000	2.000000	NaN	NaN	21.000000	0.000000	0.000000	NaN	7.895800	NaN
50%	655.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN

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	Cabin	Embarked
1148	1149	0	3	Niklasson, Mr. Samuel	male	28.0	0	0	363611	8.0500	NaN	S

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

```
494
```

Check Missing (Null) Values In The *Dataset*

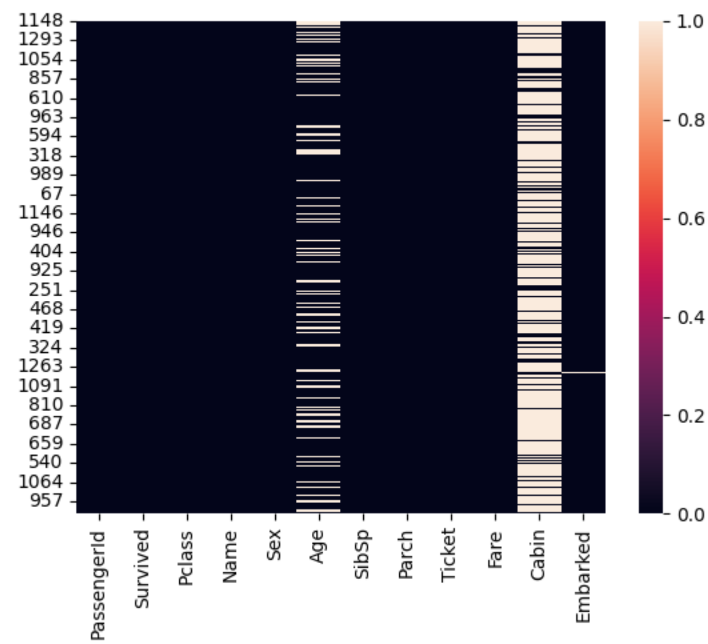
```
data.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            263
SibSp            0
Parch            0
Ticket           0
Fare             1
```



```
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)
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
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()
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

					x	Age	SibSp	Parch	Ticket	Fare	Embarked
					e	28.000000	0	0	363611	8.050	S
1049	1050	0	1	Borebank, Mr. John James	male	42.000000	0	0	110489	26.550	S
982	983	0	3	Pedersen, Mr. Olaf	male	29.881138	0	0	345498	7.775	S
808	809	0	2	Meyer, Mr. August	male	39.000000	0	0	248723	13.000	S
1195	1196	1	3	McCarthy, Miss. Catherine Katie""	female	29.881138	0	0	383123	7.750	Q

```
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	Fare	Embarked	Gender
1148	1149	0	3	Niklasson, Mr. Samuel	male	28.000000	0	0	363611	8.050	S	1
1049	1050	0	1	Borebank, Mr. John James	male	42.000000	0	0	110489	26.550	S	1
982	983	0	3	Pedersen, Mr. Olaf	male	29.881138	0	0	345498	7.775	S	1
808	809	0	2	Meyer, Mr. August	male	39.000000	0	0	248723	13.000	S	1
1195	1196	1	3	McCarthy, Miss.	female	29.881138	0	0	383123	7.750	Q	0

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	Fare	Gender	Embarked_C	Embarke
1148	1149	0	3	Niklasson, Mr. Samuel	male	28.000000	0	0	363611	8.0500	1	0	
1049	1050	0	1	Borebank, Mr. John James	male	42.000000	0	0	110489	26.5500	1	0	
982	983	0	3	Pedersen, Mr. Olaf	male	29.881138	0	0	345498	7.7750	1	0	
808	809	0	2	Meyer, Mr. August	male	39.000000	0	0	248723	13.0000	1	0	
1195	1196	1	3	McCarthy, Miss. Catherine Katie""	female	29.881138	0	0	383123	7.7500	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1095	1096	0	2	Andrew, Mr. Frank Thomas	male	25.000000	0	0	C.A. 34050	10.5000	1	0	
[illegible]	[illegible]	[illegible]	[illegible]	Douglas, [illegible] Donald (Mahala Dutton)	[illegible]	[illegible]	[illegible]	0	PC 17761	106.4250	0	1	


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

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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Gender	Embarked_Q	Embarked_S
1148	1149	0	3	Niklasson, Mr. Samuel	male	28.0	0	0	363611	8.05	1	0	1

Visual Analysis

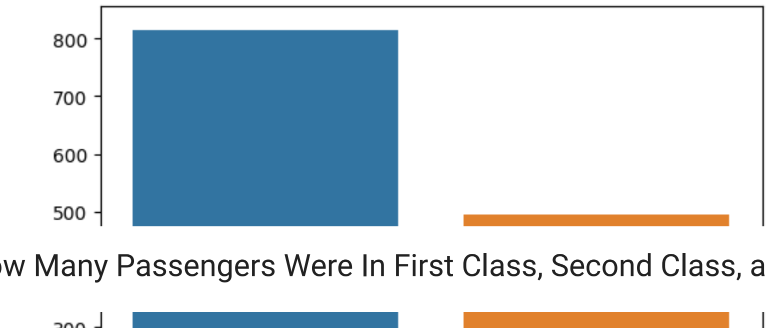
How Many People Survived And How Many Died?

```
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'>
```



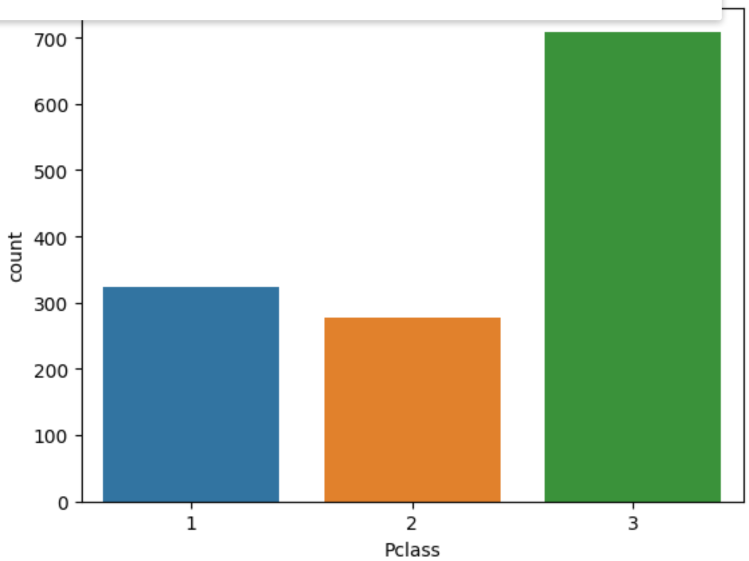
How Many Passengers Were In First Class, Second Class, and Third Class?

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

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

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

Automatic saving failed. This file was updated remotely or in another tab. Show diff



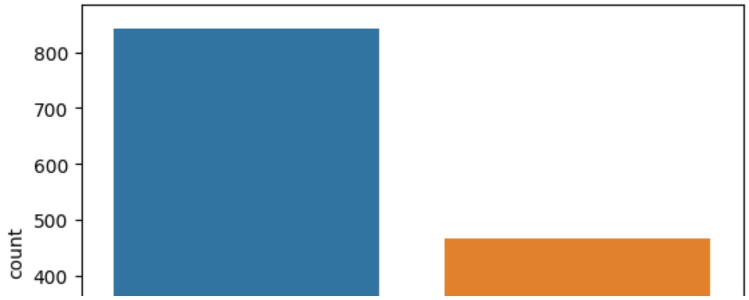
Number of Male And Female Passengers

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

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

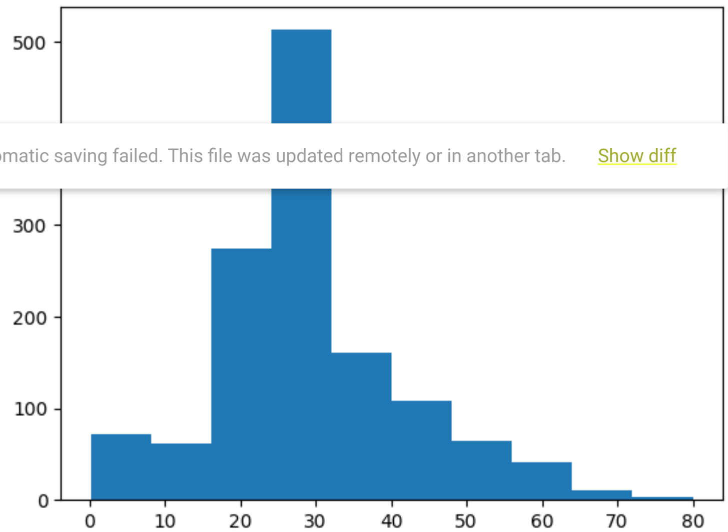
```
sns.countplot(x ='Sex', data = data)
```

```
<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>)
```



Automatic saving failed. This file was updated remotely or in another tab. Show diff

12. Bivariate Analysis

How Has Better Chance of Survival Male or Female?

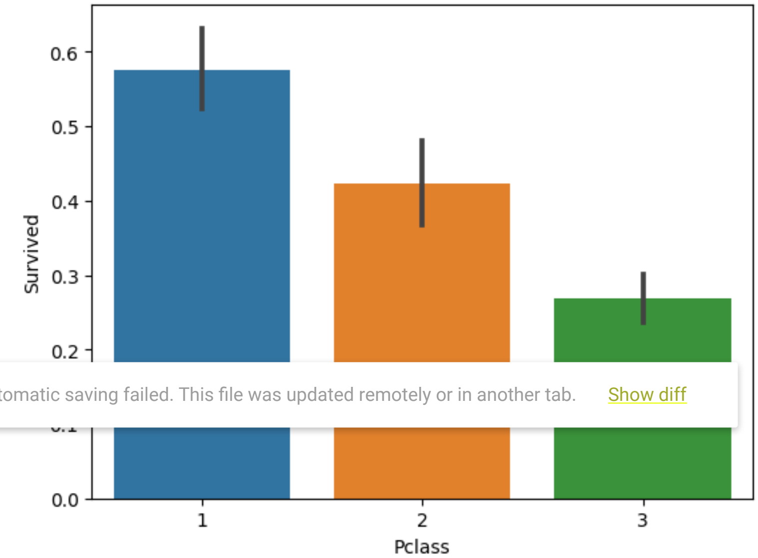
```
sns.barplot(x='Sex',y='Survived',data=data)
```

```
<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	Gender	Sex_female	Sex_male	Embarke
1148	1149	0	3	Niklasson, Mr. Samuel	28.000000	0	0	363611	8.050	1	0	1	
1049	1050	0	1	Borebank, Mr. John James	42.000000	0	0	110489	26.550	1	0	1	
982	983	0	3	Pedersen, Mr. Olaf	29.881138	0	0	345498	7.775	1	0	1	
808	809	0	2	Meyer, Mr. August	39.000000	0	0	248723	13.000	1	0	1	
1195	1196	1	3	McCarthy, Miss. Catherine Katie""	29.881138	0	0	383123	7.750	0	1	0	

```
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
cy'])

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

```
33/33 [==============================] - 0s 2ms/step - loss: 0.3779 - accuracy: 0.8567
Epoch 98/100
33/33 [==============================] - 0s 2ms/step - loss: 0.3778 - accuracy: 0.8567
Epoch 99/100
33/33 [==============================] - 0s 2ms/step - loss: 0.3776 - accuracy: 0.8567
Epoch 100/100
33/33 [==============================] - 0s 2ms/step - loss: 0.3774 - accuracy: 0.8567
<keras.callbacks.History at 0x7fe628525b40>
```

```
# 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
```

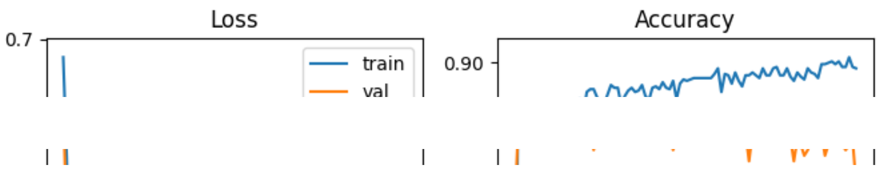
Model 2: 64-32-16-8-1 using ANN

```
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'))
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
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_test), verbose=0)
```

```
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: 81.68
```

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'))
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
# 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_test), verbose=0)


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))
```

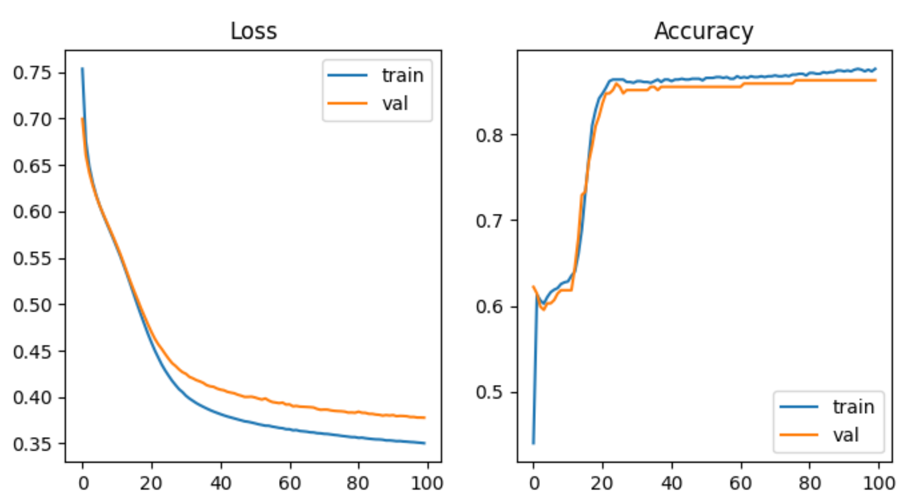
    Model 3 Accuracy: 84.35

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'])
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
n_data=(X_test, y_test), verbose=0)

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: 86.26
```

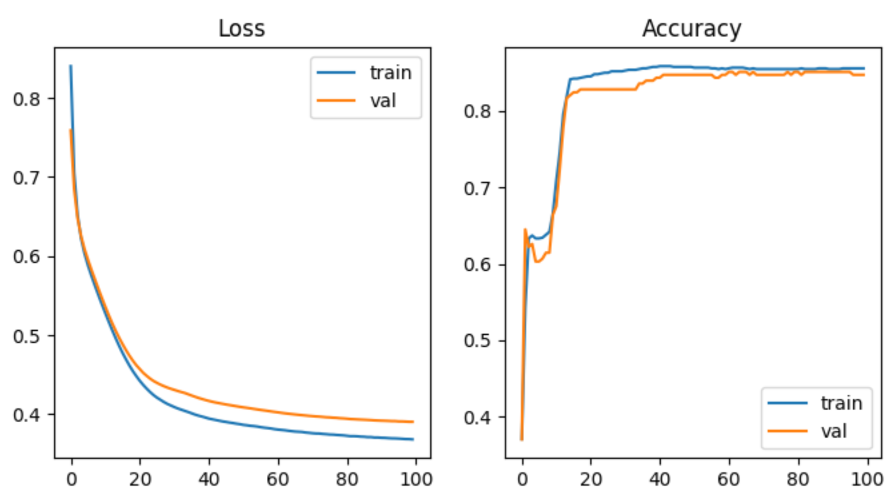
Model 5: 8-1 ANN

```
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_test), verbose=0)
```

```
import matplotlib.pyplot as plt

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



```
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: 84.73
```

Model 6: (4-1) ANN

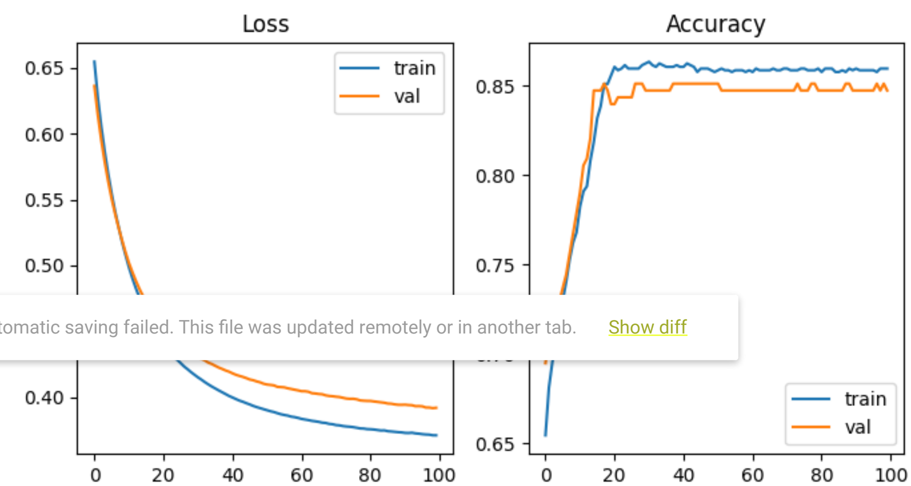
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# 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_test), verbose=1)
```

```
Epoch 72/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3787 - accuracy: 0.8586 - val_loss: 0.4001 - val_accuracy: 0.8473
Epoch 73/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3781 - accuracy: 0.8586 - val_loss: 0.3996 - val_accuracy: 0.8473
Epoch 74/100
33/33 [==============================] - 0s 5ms/step - loss: 0.3778 - accuracy: 0.8586 - val_loss: 0.3990 - val_accuracy: 0.8511
Epoch 75/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3776 - accuracy: 0.8596 - val_loss: 0.3988 - val_accuracy: 0.8473
Epoch 76/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3773 - accuracy: 0.8596 - val_loss: 0.3988 - val_accuracy: 0.8473
Epoch 77/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3768 - accuracy: 0.8586 - val_loss: 0.3984 - val_accuracy: 0.8473
Epoch 78/100
ccuracy: 0.8586 - val_loss: 0.3976 - val_accuracy: 0.8511
ccuracy: 0.8586 - val_loss: 0.3973 - val_accuracy: 0.8511
Epoch 80/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3758 - accuracy: 0.8596 - val_loss: 0.3972 - val_accuracy: 0.8473
Epoch 81/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3757 - accuracy: 0.8577 - val_loss: 0.3972 - val_accuracy: 0.8473
Epoch 82/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3755 - accuracy: 0.8596 - val_loss: 0.3968 - val_accuracy: 0.8473
Epoch 83/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3751 - accuracy: 0.8596 - val_loss: 0.3965 - val_accuracy: 0.8473
Epoch 84/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3748 - accuracy: 0.8596 - val_loss: 0.3961 - val_accuracy: 0.8473
Epoch 85/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3748 - accuracy: 0.8577 - val_loss: 0.3958 - val_accuracy: 0.8473
Epoch 86/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3742 - accuracy: 0.8577 - val_loss: 0.3952 - val_accuracy: 0.8473
Epoch 87/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3740 - accuracy: 0.8586 - val_loss: 0.3951 - val_accuracy: 0.8473
Epoch 88/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3738 - accuracy: 0.8577 - val_loss: 0.3946 - val_accuracy: 0.8511
Epoch 89/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3735 - accuracy: 0.8596 - val_loss: 0.3943 - val_accuracy: 0.8511
Epoch 90/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3733 - accuracy: 0.8586 - val_loss: 0.3943 - val_accuracy: 0.8473
Epoch 91/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3729 - accuracy: 0.8596 - val_loss: 0.3943 - val_accuracy: 0.8473
Epoch 92/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3729 - accuracy: 0.8586 - val_loss: 0.3941 - val_accuracy: 0.8473
Epoch 93/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3731 - accuracy: 0.8586 - val_loss: 0.3939 - val_accuracy: 0.8473
Epoch 94/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3726 - accuracy: 0.8586 - val_loss: 0.3932 - val_accuracy: 0.8473
Epoch 95/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3723 - accuracy: 0.8586 - val_loss: 0.3933 - val_accuracy: 0.8473
Epoch 96/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3720 - accuracy: 0.8586 - val_loss: 0.3931 - val_accuracy: 0.8473
Epoch 97/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3717 - accuracy: 0.8577 - val_loss: 0.3922 - val_accuracy: 0.8511
Epoch 98/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3716 - accuracy: 0.8596 - val_loss: 0.3922 - val_accuracy: 0.8473
Epoch 99/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3711 - accuracy: 0.8596 - val_loss: 0.3917 - val_accuracy: 0.8511
Epoch 100/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3711 - accuracy: 0.8596 - val_loss: 0.3920 - val_accuracy: 0.8473
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
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: 84.73
```

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_test), verbose=1)
```

```
Epoch 81/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3821 - accuracy: 0.8586 - val_loss: 0.4045 - val_accuracy: 0.8473
Epoch 82/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3820 - accuracy: 0.8577 - val_loss: 0.4043 - val_accuracy: 0.8473
Epoch 83/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3816 - accuracy: 0.8577 - val_loss: 0.4043 - val_accuracy: 0.8473
Epoch 84/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3814 - accuracy: 0.8586 - val_loss: 0.4044 - val_accuracy: 0.8473
Epoch 85/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3810 - accuracy: 0.8586 - val_loss: 0.4042 - val_accuracy: 0.8473
Epoch 86/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3808 - accuracy: 0.8586 - val_loss: 0.4040 - val_accuracy: 0.8473
Epoch 87/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3806 - accuracy: 0.8577 - val_loss: 0.4037 - val_accuracy: 0.8473
Epoch 88/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3803 - accuracy: 0.8586 - val_loss: 0.4034 - val_accuracy: 0.8473
Epoch 89/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3800 - accuracy: 0.8586 - val_loss: 0.4031 - val_accuracy: 0.8473
Epoch 90/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3798 - accuracy: 0.8586 - val_loss: 0.4031 - val_accuracy: 0.8473
Epoch 91/100
33/33 [==============================] - 0s 4ms/step - loss: 0.3796 - accuracy: 0.8586 - val_loss: 0.4032 - val_accuracy: 0.8473
Epoch 92/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3793 - accuracy: 0.8586 - val_loss: 0.4029 - val_accuracy: 0.8473
Epoch 93/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3792 - accuracy: 0.8586 - val_loss: 0.4028 - val_accuracy: 0.8473
Epoch 94/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3787 - accuracy: 0.8586 - val_loss: 0.4027 - val_accuracy: 0.8473
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
ccuracy: 0.8596 - val_loss: 0.4026 - val_accuracy: 0.8473
33/33 [==============================] - 0s 3ms/step - loss: 0.3785 - accuracy: 0.8606 - val_loss: 0.4025 - val_accuracy: 0.8473
Epoch 97/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3782 - accuracy: 0.8596 - val_loss: 0.4023 - val_accuracy: 0.8473
Epoch 98/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3778 - accuracy: 0.8596 - val_loss: 0.4019 - val_accuracy: 0.8473
Epoch 99/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3775 - accuracy: 0.8596 - val_loss: 0.4019 - val_accuracy: 0.8473
Epoch 100/100
33/33 [==============================] - 0s 3ms/step - loss: 0.3772 - accuracy: 0.8596 - val_loss: 0.4019 - val_accuracy: 0.8473
```

```
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: 84.73
```

Model 9: Random Forest using ANN

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from keras.models import Sequential
from keras.layers import Dense, Dropout

# Create an ANN model
model_RF = Sequential()
model_RF.add(Dense(32, input_dim=X.shape[1], activation='relu'))
model_RF.add(Dropout(0.5))
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

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

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

# Fit the model to the training data
model_RF.fit(X_train, y_train, epochs=50, batch_size=32, verbose=1)

# Predict the target variable for the test data
y_pred = np.round(model_RF.predict(X_test))

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy*100)

# Print the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion matrix:')
print(cm)
```

```
33/33 [==============================] - 0s 2ms/step - loss: 0.4000 - accuracy: 0.8453
Epoch 41/50
33/33 [==============================] - 0s 3ms/step - loss: 0.3943 - accuracy: 0.8510
Epoch 42/50
33/33 [==============================] - 0s 2ms/step - loss: 0.4022 - accuracy: 0.8443
Epoch 43/50
33/33 [==============================] - 0s 2ms/step - loss: 0.4084 - accuracy: 0.8558
Epoch 44/50
33/33 [==============================] - 0s 2ms/step - loss: 0.4049 - accuracy: 0.8548
Epoch 45/50
33/33 [==============================] - 0s 2ms/step - loss: 0.4141 - accuracy: 0.8510
Epoch 46/50
33/33 [==============================] - 0s 2ms/step - loss: 0.4082 - accuracy: 0.8491
Epoch 47/50
33/33 [==============================] - 0s 2ms/step - loss: 0.3946 - accuracy: 0.8577
Epoch 48/50
33/33 [==============================] - 0s 2ms/step - loss: 0.3861 - accuracy: 0.8462
Epoch 49/50
33/33 [==============================] - 0s 2ms/step - loss: 0.4087 - accuracy: 0.8529
Epoch 50/50
33/33 [==============================] - 0s 2ms/step - loss: 0.4137 - accuracy: 0.8510
9/9 [==============================] - 0s 3ms/step
Accuracy: 85.1145038167939
Confusion matrix:
[[148  13]
 [ 26  75]]
```

```
X_train.shape
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
X_train.head(10)
```

	Pclass	Age	SibSp	Parch	Fare	Gender	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
1006	3	-0.936393	1	0	-0.363662	1	0	1	1	0	0
1090	3	-0.007506	0	0	-0.480436	0	1	0	0	0	1
915	1	1.409057	1	3	4.201449	0	1	0	1	0	0
1176	3	0.470877	0	0	-0.496318	1	0	1	0	0	1
1234	1	2.190873	0	1	8.804002	0	1	0	1	0	0
553	3	-0.623666	0	0	-0.496778	1	0	1	1	0	0
664	3	-0.780030	1	0	-0.483888	1	0	1	0	0	1
114	3	-1.014574	0	0	-0.363587	0	1	0	1	0	0
420	3	-0.007506	0	0	-0.484426	1	0	1	1	0	0
696	3	1.096330	0	0	-0.481587	1	0	1	0	0	1

```
def my_prediction_function(model, data):
    numOfFeatures = 11
    numOfLayers = len(model.layers)
    w = [None]*numOfFeatures
    weights = model.layers[numOfLayers-1].get_weights()[0]
    for i in range(min(numOfFeatures, len(weights))):
        w[i] = weights[i]
    bias = model.layers[numOfLayers-1].get_weights()[1]
    z = 0
    for i in range(min(numOfFeatures, len(weights))):
        z = z + data[:,i]*w[i]
    z = z + bias
    result = 1/(1+np.exp(-z))
    return np.mean(result)
```

```
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_test), verbose=0)
```

```
import tensorflow as tf

# assuming X_train is a numpy array
X_test_tensor = tf.convert_to_tensor(X_test)
result = my_prediction_function(model_5, X_test_tensor.numpy())
```

```
print("Test Result",result*100)
```

```
Test Result 49.48080313461359
```

```
for i in range(X_train.shape[1]):
  print(i)
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
2
3
4
5
6
7
8
9
10
```

```
print(X_train.iloc[:,0].head(10))
X_train.head(10)
```

```
1006    3
1090    3
915     1
1176    3
1234    1
553     3
664     3
114     3
420     3
696     3
Name: Pclass, dtype: int64
```

	Pclass	Age	SibSp	Parch	Fare	Gender	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
1006	3	-0.936393	1	0	-0.363662	1	0	1	1	0	0
1090	3	-0.007506	0	0	-0.480436	0	1	0	0	0	1
915	1	1.409057	1	3	4.201449	0	1	0	1	0	0
1176	3	0.470877	0	0	-0.496318	1	0	1	0	0	1
1234	1	2.190873	0	1	8.804002	0	1	0	1	0	0
553	3	-0.623666	0	0	-0.496778	1	0	1	1	0	0
664	3	-0.780030	1	0	-0.483888	1	0	1	0	0	1
114	3	-1.014574	0	0	-0.363587	0	1	0	1	0	0
420	3	-0.007506	0	0	-0.484426	1	0	1	1	0	0
696	3	1.096330	0	0	-0.481587	1	0	1	0	0	1

```
def features_prediction_function(model, X_test):
    numOfFeatures = 11
    numOfInstances = len(X_test)
    numOfLayers = len(model.layers)
    weights = model.layers[numOfLayers-1].get_weights()[0]
    bias = model.layers[numOfLayers-1].get_weights()[1]
```

```
    feature_wise_results = []
    for i in range(numOfFeatures):
        # Get the i-th feature from the test set
        x = X_test[:, i]
        # Reshape the column to have a single feature dimension
        x = x.reshape(-1, 1)
        # Get the i-th weight
        w_i = weights[i] if i < len(weights) else None
        # Get the prediction result for the i-th feature
        pred = my_prediction_function_for_instance(model, x, w_i, bias)
        # Append the result to the list
        feature_wise_results.append(pred)
    return feature_wise_results

def my_prediction_function_for_instance(model, x, w, bias):
    if w is not None:
        z = np.dot(x, w) + bias
    else:
        z = bias
    result = 1 / (1 + np.exp(-z))
    return np.mean(result)
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
# assuming X_train is a numpy array
X_test_tensor = tf.convert_to_tensor(X_test)
result = features_prediction_function(model_5, X_test_tensor.numpy())
per_results=result*100
# Get the column names from the original test data
col_names = list(X_test.columns)

# Create a pandas DataFrame to display the results
df = pd.DataFrame({'Feature': col_names, 'Prediction_Feature Importance': result})

# Display the results
print(df)
```

```
        Feature  Prediction
0        Pclass    0.250650
1           Age    0.600472
2         SibSp    0.640545
3         Parch    0.537315
4          Fare    0.628655
5        Gender    0.653323
6    Sex_female    0.687481
7      Sex_male    0.687465
8    Embarked_C    0.603027
9    Embarked_Q    0.603027
10   Embarked_S    0.603027
```

Double-click (or enter) to edit

0s completed at 8:15 PM

Automatic saving failed. This file was updated remotely or in another tab. Show diff