c. Build the 3rd model using 'Tenure', 'Monthly Charges' & 'Total Charges' as the features and 'Churn' as the dependent/target column:

i. The visible/input layer should have 12 nodes with 'Relu' as activation function.

ii. This model would have 1 hidden layer with 8 nodes and 'Relu' as activation function

iii. Use 'Adam' as the optimization algorithm

iv. Fit the model on the train set, with number of epochs to be 150

v. Predict the values on the test set and build a confusion matrix

vi. Plot the 'Accuracy vs Epochs' graph

```
import numpy as np
import pandas as pd
```

```
df= pd.read_csv('/content/customer_churn.csv')
```

```
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport	StreamingTV
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	No	No
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	No	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	No	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	Yes	No
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	No	No

5 rows × 21 columns

```
##Preprocess Dataset
#Now it's time to preprocess the data, firstly we will observe the dataset,  this means we have to see the data types of the columns, other functionalities, and parameters of each


df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

```
##Second, we check the description of the dataset, here we will only visible the num variables functionalities. we will use describe() method.

df.describe()
```

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

```
##Now we drop unwanted features from our dataset because these unwanted features are like the garbage they will affect our model accuracy so we drop it.

# we didn't require customerID so we drop it
df = df.drop('customerID',axis=1)
```

```
##When we note the TotalCharges column then we found that it’s a data type of an object but it even would be float. so we have to typecast this column.

#count of string value into the column.
count=0
for i in df.TotalCharges:
    if i==' ':
        count+=1
print('count of empty string:- ',count)

#we will replace this empty string to nan values
df['TotalCharges'] = df['TotalCharges'].replace(" ",np.nan)

# typecasting of the TotalCharges column
df['TotalCharges'] = df['TotalCharges'].astype(float)
```

```
count of empty string:-  11
```

```
#Now we have to check for null values, for this, we use the pandas IsNull() method which will give True if the null value is present and False when there are no null values.

# checking null value
df.isnull().sum()
```

```
gender               0
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

```
# fill null values with mean
df['TotalCharges'] = df['TotalCharges'].fillna(df['TotalCharges'].mean())
```

```
#Now we will extract the numerical and categorical columns from the dataset for further processes.

#numerical variables

num = list(df.select_dtypes(include=['int64','float64']).keys())
```

```
#categorical variables

cat = list(df.select_dtypes(include='O').keys())

print(cat)

print(num)
```

```
['gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'St
['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges']
```

```
#Now we see the value counts of each category in each categorical column.

# value_counts of the categorical columns
for i in cat:
    print(df[i].value_counts())
```

```
gender
Male      3555
Female    3488
Name: count, dtype: int64
Partner
No     3641
Yes    3402
Name: count, dtype: int64
Dependents
No     4933
Yes    2110
Name: count, dtype: int64
PhoneService
Yes    6361
No      682
Name: count, dtype: int64
MultipleLines
No                  3390
Yes                 2971
No phone service     682
Name: count, dtype: int64
InternetService
Fiber optic    3096
DSL            2421
No             1526
Name: count, dtype: int64
OnlineSecurity
No                     3498
Yes                    2019
No internet service    1526
Name: count, dtype: int64
OnlineBackup
```

```
No                     3088
Yes                    2429
No internet service    1526
Name: count, dtype: int64
DeviceProtection
No                     3095
Yes                    2422
No internet service    1526
Name: count, dtype: int64
TechSupport
No                     3473
Yes                    2044
No internet service    1526
Name: count, dtype: int64
StreamingTV
No                     2810
Yes                    2707
No internet service    1526
Name: count, dtype: int64
StreamingMovies
No                     2785
Yes                    2732
No internet service    1526
Name: count, dtype: int64
Contract
Month-to-month    3875
```

```
# we will convert Yes = 1 and No = 0
df.Churn = df.Churn.replace('Yes',1)
df.Churn = df.Churn.replace('No',0)
```

```
#The handling of categorical columns is over now we have to scale our data because there are some columns present where values are much larger which will affect the runtime of the

scale_cols = ['tenure','MonthlyCharges','TotalCharges']
# now we scling all the data
from sklearn.preprocessing import MinMaxScaler
scale = MinMaxScaler()
df[scale_cols] = scale.fit_transform(df[scale_cols])
#scale_cols contain that columns which are having large numerical values, and with MinMaxScaler we will scale it into values between -1 to 1.
```

```
Start coding or generate with AI.
```

```
##Now we start our model training process, first, we have to divide our dataset into dependent and independent variables.

x=df[['MonthlyCharges','tenure','TotalCharges']]#Features
y=df[['Churn']]#Target
```

```
##Now we have to split our dataset into train and test sets, where the training set is used to train the model, and the testing set is used for testing the values of targeted colur

from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.30,random_state=42)
print(xtrain.shape)
print(xtest.shape)
```

```
(4930, 3)
(2113, 3)
```

```
# now we create our artificial neural net.
# import tensorflow
import tensorflow as tf
#import keras
from tensorflow import keras
```

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import confusion_matrix
```

```
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
model.fit(xtrain, ytrain, epochs=150,validation_data=(xtest,ytest))
```

```
Epoch 135/150
155/155 [==============================] - 0s 3ms/step - loss: 0.4568 - accuracy: 0.7795 - val_loss: 0.4461 - val_accuracy: 0.7885
Epoch 136/150
155/155 [==============================] - 1s 3ms/step - loss: 0.4573 - accuracy: 0.7836 - val_loss: 0.4459 - val_accuracy: 0.7880
Epoch 137/150
155/155 [==============================] - 1s 4ms/step - loss: 0.4569 - accuracy: 0.7870 - val_loss: 0.4457 - val_accuracy: 0.7870
Epoch 138/150
155/155 [==============================] - 1s 3ms/step - loss: 0.4569 - accuracy: 0.7826 - val_loss: 0.4456 - val_accuracy: 0.7866
Epoch 139/150
155/155 [==============================] - 1s 3ms/step - loss: 0.4569 - accuracy: 0.7846 - val_loss: 0.4457 - val_accuracy: 0.7894
Epoch 140/150
155/155 [==============================] - 0s 3ms/step - loss: 0.4569 - accuracy: 0.7854 - val_loss: 0.4452 - val_accuracy: 0.7889
Epoch 141/150
155/155 [==============================] - 0s 3ms/step - loss: 0.4566 - accuracy: 0.7848 - val_loss: 0.4471 - val_accuracy: 0.7899
Epoch 142/150
155/155 [==============================] - 1s 3ms/step - loss: 0.4570 - accuracy: 0.7856 - val_loss: 0.4463 - val_accuracy: 0.7903
Epoch 143/150
155/155 [==============================] - 0s 3ms/step - loss: 0.4567 - accuracy: 0.7870 - val_loss: 0.4452 - val_accuracy: 0.7870
Epoch 144/150
155/155 [==============================] - 0s 3ms/step - loss: 0.4567 - accuracy: 0.7842 - val_loss: 0.4464 - val_accuracy: 0.7913
Epoch 145/150
155/155 [==============================] - 0s 3ms/step - loss: 0.4570 - accuracy: 0.7848 - val_loss: 0.4455 - val_accuracy: 0.7847
Epoch 146/150
155/155 [==============================] - 0s 3ms/step - loss: 0.4573 - accuracy: 0.7836 - val_loss: 0.4453 - val_accuracy: 0.7866
Epoch 147/150
155/155 [==============================] - 1s 3ms/step - loss: 0.4568 - accuracy: 0.7842 - val_loss: 0.4472 - val_accuracy: 0.7894
Epoch 148/150
155/155 [==============================] - 0s 3ms/step - loss: 0.4572 - accuracy: 0.7868 - val_loss: 0.4452 - val_accuracy: 0.7894
Epoch 149/150
155/155 [==============================] - 1s 3ms/step - loss: 0.4573 - accuracy: 0.7844 - val_loss: 0.4453 - val_accuracy: 0.7889
Epoch 150/150
155/155 [==============================] - 1s 4ms/step - loss: 0.4569 - accuracy: 0.7836 - val_loss: 0.4456 - val_accuracy: 0.7903
<keras.src.callbacks.History at 0x7e7df42afa30>
```

```
#Now we evaluate our model by this we can observe the summary of the model.

# evalute the model
model.evaluate(xtest,ytest)
```

```
67/67 [==============================] - 0s 3ms/step - loss: 0.4456 - accuracy: 0.7903
[0.44564154744148254, 0.7903454899787903]
```

```
#As above we are performing scaling on the data, that’s why our predicted values are scaled so we have to unscale it into normal form for this we write the following program.

# predict the churn values
ypred = model.predict(xtest)
print(ypred)
# unscaling the ypred values
ypred_lis = []
for i in ypred:
    if i>0.5:
        ypred_lis.append(1)
    else:
        ypred_lis.append(0)
print(ypred_lis)
```

```
67/67 [==============================] - 0s 1ms/step
[[0.27675477]
 [0.04049738]
 [0.02234837]
 ...
 [0.04435795]
 [0.4954468 ]
 [0.19287437]]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 
```

```
from sklearn.metrics import confusion_matrix, classification_report
from matplotlib import pyplot as plt
import seaborn as sns
```

```
print(classification_report(ytest,ypred_lis))
```

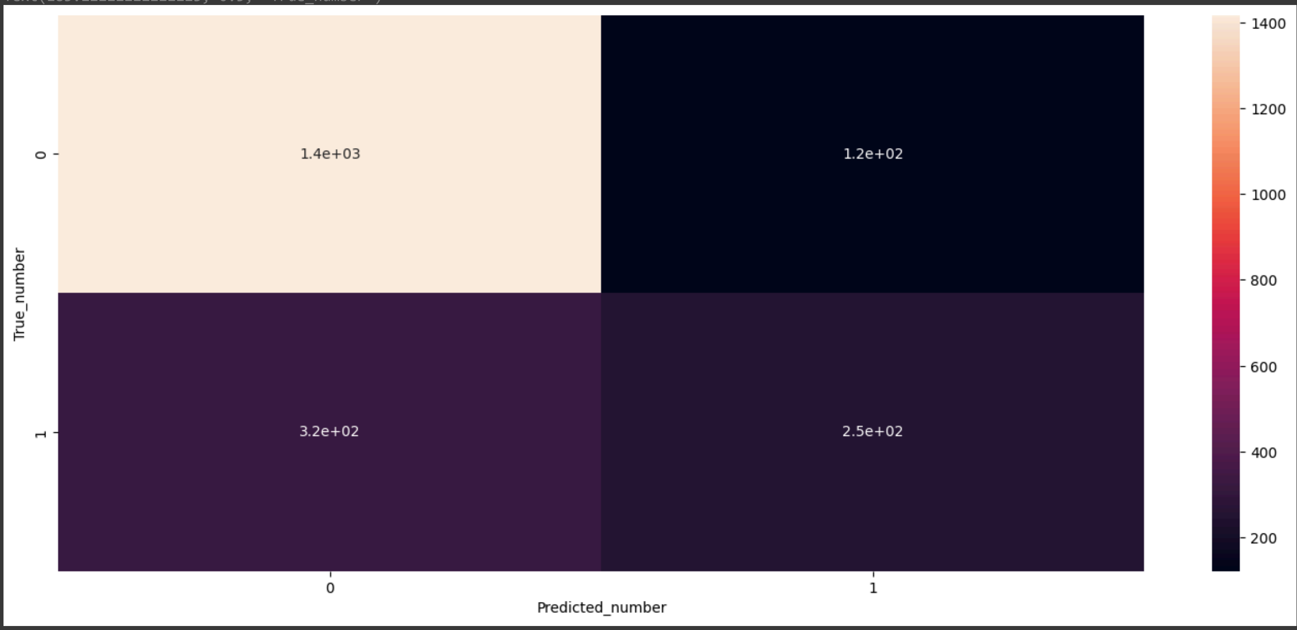
```
              precision    recall  f1-score   support

           0       0.82      0.92      0.86      1539
           1       0.67      0.44      0.53       574

    accuracy                           0.79      2113
   macro avg       0.74      0.68      0.70      2113
weighted avg       0.78      0.79      0.77      2113
```

```
conf_mat = tf.math.confusion_matrix(labels=ytest,predictions=ypred_lis)
plt.figure(figsize = (17,7))
sns.heatmap(conf_mat, annot=True)
plt.xlabel('Predicted_number')
plt.ylabel('True_number')
```

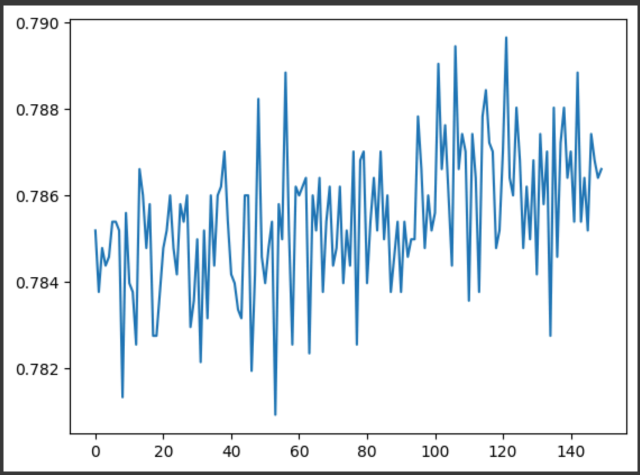
Text(183.22222222222223, 0.5, 'True_number')




```
from sklearn.metrics import confusion_matrix
confusion_matrix(ytest,ypred_lis)
```

```
array([[1417,  122],
       [ 321,  253]])
```

```
plt.plot(model.history.history['accuracy'])
#plt.plot(model.history.history['val_accuracy'])
plt.show()
```




```
#So, let’s find out the mean validation accuracy across 150 epochs:
import numpy as np
np.mean(model.history.history['val_accuracy'])
```

```
0.7878876884778341
```