Problem Statement:

You are the Data Scientist at a telecom company "Leo" whose customers are churning out to its competitors. You have to analyse the data of your company and find insights and stop your customers from churning out to other telecom companies.

```
# Import necessary Liabraries:

import numpy as np
import pandas as pd
```

```
# Loading Dataset:
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

```
# A) Data Manipulation:
##a. Find the total number of male customers.
df= pd.read_csv('/content/customer_churn.csv')
len(df[df.gender == "Male"])
```

3555

```
sum(df['gender']=="Male")
```

3555

```
##b. Find the total number of customers whose Internet Service is 'DSL'
sum(df['InternetService']=="DSL")
```

2421

```
##c. Extract all the Female senior citizens whose Payment Method is Mailed check & store the result in 'new_customer'

new_customer=df[(df['gender']=='Female') &
(df['SeniorCitizen']==1) & (df['PaymentMethod']=='Mailed check')]
new_customer.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtectio
139	0390-DCFDQ	Female	1	Yes	No	1	Yes	No	Fiber optic	No	...	N
176	2656-FMOKZ	Female	1	No	No	15	Yes	Yes	Fiber optic	No	...	N
267	3197-ARFOY	Female	1	No	No	19	Yes	No	Fiber optic	Yes	...	N
451	5760-WRAHC	Female	1	No	No	22	Yes	No	DSL	Yes	...	Ye
470	4933-IKULF	Female	1	No	No	17	Yes	No	No	No internet service	...	No internet servic

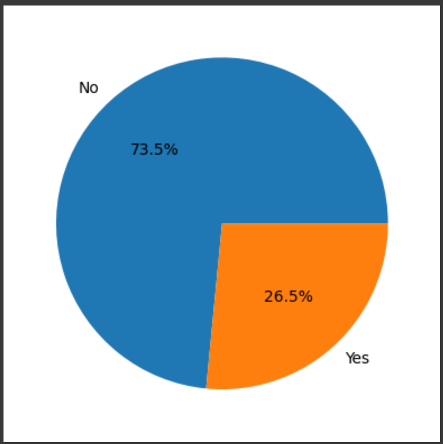
5 rows × 21 columns

```
##d. Extract all those customers whose tenure is less than 10 months or their Total charges is less than 500$ & store the result in 'new_customer'

new_customer2 = df[(df['tenure'] < 10) | (pd.to_numeric(df['TotalCharges'], errors='coerce') < 500)]
new_customer2.head()
```

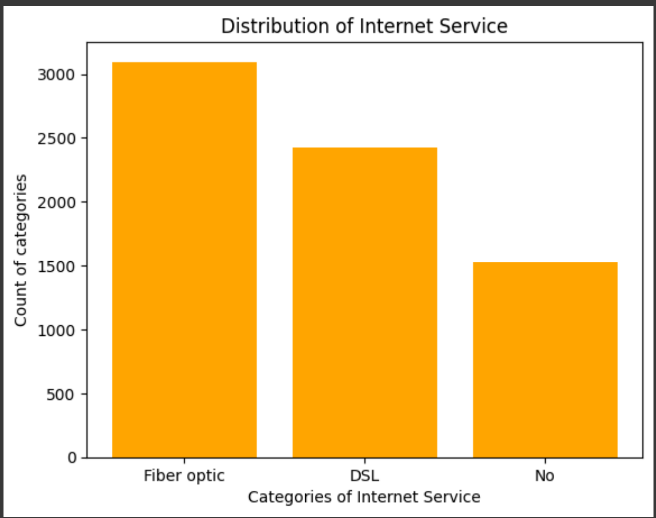
	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No
5	9305-CDSKC	Female	0	No	No	8	Yes	Yes	Fiber optic	No	...	Yes
7	6713-OKOMC	Female	0	No	No	10	No	No phone service	DSL	Yes	...	No

5 rows × 21 columns

```
#B) Data Visualization:

import matplotlib.pyplot as plt
##a. Build a pie-chart to show the distribution of customers would be churning out
names = df["Churn"].value_counts().keys().tolist()
sizes= df["Churn"].value_counts().tolist()

plt.pie(sizes,labels=names,autopct="%0.1f%%")
plt.show()
```




```
##b. Build a bar-plot to show the distribution of ‘Internet Service’
plt.bar(df['InternetService'].value_counts().keys().tolist(),df['InternetService'].value_counts().tolist(),color='orange')

plt.xlabel('Categories of Internet Service')
plt.ylabel('Count of categories')
plt.title('Distribution of Internet Service')
plt.show()
```




```
#C) Model Building:
##a. Build a sequential model using Keras, to find out if the customerwouldchurn or not, using ‘tenure’ as the feature 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

x=df[['tenure']]
y=df[['Churn']]

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=45)
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import confusion_matrix
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(x_train)
X_test_scaled = scaler.transform(x_test)
```

```
model = Sequential()
model.add(Dense(12, input_dim=1, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

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

```
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder instance
label_encoder = LabelEncoder()

# Fit the LabelEncoder on the unique values of y_train
label_encoder.fit(np.unique(y_train))

# Transform the string labels in y_train to numerical values
y_train_encoded = label_encoder.transform(y_train)


# Train the model using the encoded y_train data
history = model.fit(X_train_scaled, y_train_encoded, epochs=150, batch_size=10, verbose=1)
```

```
Epoch 36/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5114 - accuracy: 0.7462
Epoch 37/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5111 - accuracy: 0.7481
Epoch 38/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5109 - accuracy: 0.7477
Epoch 39/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5114 - accuracy: 0.7467
Epoch 40/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5109 - accuracy: 0.7483
Epoch 41/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5111 - accuracy: 0.7479
Epoch 42/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5111 - accuracy: 0.7499
Epoch 43/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5112 - accuracy: 0.7489
Epoch 44/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5112 - accuracy: 0.7477
Epoch 45/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5108 - accuracy: 0.7487
Epoch 46/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5115 - accuracy: 0.7467
Epoch 47/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5111 - accuracy: 0.7467
Epoch 48/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5109 - accuracy: 0.7477
Epoch 49/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5111 - accuracy: 0.7460
Epoch 50/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5109 - accuracy: 0.7491
Epoch 51/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5109 - accuracy: 0.7458
Epoch 52/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5111 - accuracy: 0.7505
Epoch 53/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5107 - accuracy: 0.7483
Epoch 54/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5110 - accuracy: 0.7456
Epoch 55/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5111 - accuracy: 0.7475
Epoch 56/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5110 - accuracy: 0.7479
Epoch 57/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5108 - accuracy: 0.7487
Epoch 58/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5113 - accuracy: 0.7477
Epoch 59/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5109 - accuracy: 0.7475
Epoch 60/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5110 - accuracy: 0.7489
Epoch 61/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5109 - accuracy: 0.7465
Epoch 62/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5112 - accuracy: 0.7491
Epoch 63/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5107 - accuracy: 0.7493
Epoch 64/150
```

```
##This gives us a final validation accuracy of 74.83%. But this is not the average accuracyacross 150 epochs, so let's also find that:

import numpy as np
np.mean(history.history['accuracy'])
```

```
0.7482961471875509
```

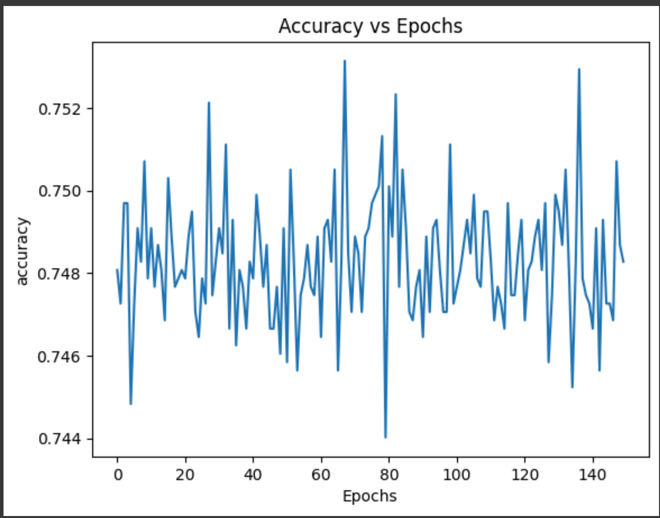
```
# So, the mean accuracy comes out to be 74.82%.
y_pred = model.predict(X_test_scaled)
```

```
67/67 [==============================] - 0s 2ms/step
```

```
from sklearn.metrics import confusion_matrix

# Encode the y_test data
y_test_encoded = label_encoder.transform(y_test)

# Calculate the confusion matrix
confusion_matrix(y_test_encoded, y_pred_binary)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:134: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  y = column_or_1d(y, dtype=self.classes_.dtype, warn=True)
array([[1433,  140],
       [ 356,  184]])
```

```
from matplotlib import pyplot as plt

plt.plot(history.history['accuracy'])
plt.title('Accuracy vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('accuracy')
plt.show()
```




```
#b. Build the 2nd model using same target and feature variables:
##i. Add a drop-out layer after the input layer with drop-out value of 0.3
##ii. Add a drop-out layer after the hidden layer with drop-out value of 0.2
##iii. Predict the values on the test set and build a confusion matrix
##iv. Plot the ‘Accuracy vs Epochs’ graph
```

```
from keras.layers import Dropout
model2 = Sequential()
model2.add(Dense(12, input_dim=1, activation='relu'))
model2.add(Dropout(0.3))
model2.add(Dense(8, activation='relu'))
model2.add(Dropout(0.2))
model2.add(Dense(1, activation='sigmoid'))
```

```
model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
history2 = model2.fit(X_train_scaled, y_train_encoded, epochs=150, batch_size=10, verbose=1)
```

```
Epoch 1/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5597 - accuracy: 0.7288
Epoch 2/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5340 - accuracy: 0.7323
Epoch 3/150
493/493 [==============================] - 2s 3ms/step - loss: 0.5327 - accuracy: 0.7349
Epoch 4/150
493/493 [==============================] - 2s 4ms/step - loss: 0.5275 - accuracy: 0.7369
Epoch 5/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5282 - accuracy: 0.7288
Epoch 6/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5284 - accuracy: 0.7341
Epoch 7/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5226 - accuracy: 0.7381
Epoch 8/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5244 - accuracy: 0.7414
Epoch 9/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5239 - accuracy: 0.7381
Epoch 10/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5230 - accuracy: 0.7371
Epoch 11/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5226 - accuracy: 0.7420
Epoch 12/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5218 - accuracy: 0.7383
Epoch 13/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5244 - accuracy: 0.7351
Epoch 14/150
493/493 [==============================] - 2s 5ms/step - loss: 0.5252 - accuracy: 0.7377
Epoch 15/150
493/493 [==============================] - 2s 4ms/step - loss: 0.5220 - accuracy: 0.7373
Epoch 16/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5195 - accuracy: 0.7391
Epoch 17/150
```

```
493/493 [==============================] - 1s 3ms/step - loss: 0.5197 - accuracy: 0.7452
Epoch 18/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5217 - accuracy: 0.7377
Epoch 19/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5200 - accuracy: 0.7385
Epoch 20/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5197 - accuracy: 0.7454
Epoch 21/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5223 - accuracy: 0.7369
Epoch 22/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5206 - accuracy: 0.7387
Epoch 23/150
493/493 [==============================] - 1s 3ms/step - loss: 0.5227 - accuracy: 0.7349
Epoch 24/150
493/493 [==============================] - 2s 4ms/step - loss: 0.5203 - accuracy: 0.7387
Epoch 25/150
493/493 [==============================] - 2s 4ms/step - loss: 0.5185 - accuracy: 0.7383
Epoch 26/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5174 - accuracy: 0.7398
Epoch 27/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5223 - accuracy: 0.7371
Epoch 28/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5216 - accuracy: 0.7323
Epoch 29/150
493/493 [==============================] - 1s 2ms/step - loss: 0.5206 - accuracy: 0.7377
```

```
##This gives us a final validation accuracy of 74.44%. But this is not the average accuracyacross 150 epochs, so let’s also find that:

import numpy as np
np.mean(history2.history['accuracy'])
```

```
0.7407640286286672
```

```
# So, the mean accuracy comes out to be 74.07%.
y_pred = model2.predict(X_test_scaled)
```

```
67/67 [==============================] - 0s 2ms/step
```

```
from sklearn.metrics import confusion_matrix

# Encode the y_test data
y_test_encoded = label_encoder.transform(y_test)

# Calculate the confusion matrix
confusion_matrix(y_test_encoded, y_pred_binary)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:134: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  y = column_or_1d(y, dtype=self.classes_.dtype, warn=True)
array([[1433,  140],
       [ 356,  184]])
```

```
plt.plot(history2.history['accuracy'])
plt.title('Accuracy vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('accuracy')
plt.show()
```