Project 2: Solution Document

A)a ->

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

This would give you the count of total number of male customers

A)b ->

```
sum(customer['InternetService']=="DSL")
```

This would give you the total number of customers whose Internet Service is 'DSL'

A)c ->

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

new_customer.head()
```

With this command, you can extract all the female senior citizens whose payment method is 'Mailed check'

A)d->

```
new_customer=customer[(customer['tenure']<10) | (customer['TotalCharges']<500)]

new_customer.head()
```

With this command, you can extract all those records where either the tenure is less than 10 or Total charges is less than 500.

B)a->

```
names = customer["Churn"].value_counts().keys().tolist()

sizes=  customer["Churn"].value_counts().tolist()
```

We are starting off by extracting the names of the levels in the churn column, then we extracting the counts of the levels in the churn column.

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

plt.show()
```

Using plt.pie(), we are making the pie-chart. 'autopct' parameter is used to add the percentage distribution in the plot.

B)b->

```
plt.bar(customer['InternetService'].value_counts().keys().tolist(),customer['InternetService'].value_counts().tolist(),color='orange')
```

We are creating the bar-plot using plt.bar()

```
plt.xlabel('Categories of Internet Service')

plt.ylabel('Count of categories')

plt.title('Distribution of Internet Service')

plt.show()
```

Going ahead, we are assigning the x-label, y-label and title to the plot.

C)a->

```
x=customer[['tenure']]

y=customer[['Churn']]
```

We are starting off by extracting the target and feature columns.

```
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=42)
```

Going ahead, we are dividing the data into train and test sets using train_test_split().

Here, we are setting the test_size to be 0.30, which means 30% of the records go into the test set, while 70% of the records go into the train set.

```
from keras.models import Sequential

from keras.layers import Dense


model = Sequential()

model.add(Dense(12, input_dim=1, activation='relu'))

model.add(Dense(8, activation='relu'))

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

After that we create an instance of a sequential model by using Sequential().

Going ahead we will add the input layer to our model. This input layer would comprise of 12 nodes and would have 'relu' as the activation function. After that we'll add a hidden layer with 8 nodes and 'relu' as activation function. Finally, we'll add the output layer which would comprise of just one node and 'sigmoid' as activation function.

We are using 'sigmoid' here because this is a binary classification problem and 'sigmoid' gives us a probability between 0 & 1.

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

Further, we'll tune the model. Here, we are using 'binary_crossentropy' as our loss function because this is a binary classification problem.

Optimizer used is 'adam' and we would want to calculate the accuracy.

```
model.fit(x_train, y_train, epochs=150,validation_data=(x_test,y_test))
```

Going ahead, we will fit the model on the train set and evaluate it on top of the test set. The number of epochs given over here is 150.

```
4922/4922 [==============================] - 1s 111us/step - loss: 0.5080 - acc: 0.7534 - val_loss: 0.5105 - val_acc: 0.7564
Epoch 145/150
4922/4922 [==============================] - 1s 107us/step - loss: 0.5080 - acc: 0.7531 - val_loss: 0.5151 - val_acc: 0.7564
Epoch 146/150
4922/4922 [==============================] - 0s 77us/step - loss: 0.5083 - acc: 0.7534 - val_loss: 0.5139 - val_acc: 0.7564
Epoch 147/150
4922/4922 [==============================] - 0s 82us/step - loss: 0.5093 - acc: 0.7548 - val_loss: 0.5119 - val_acc: 0.7564
Epoch 148/150
4922/4922 [==============================] - 0s 95us/step - loss: 0.5096 - acc: 0.7536 - val_loss: 0.5202 - val_acc: 0.7564
Epoch 149/150
4922/4922 [==============================] - 0s 87us/step - loss: 0.5095 - acc: 0.7529 - val_loss: 0.5105 - val_acc: 0.7564
Epoch 150/150
4922/4922 [==============================] - 0s 77us/step - loss: 0.5081 - acc: 0.7542 - val_loss: 0.5114 - val_acc: 0.7564
```

This gives us a final validation accuracy of 75.64%. But this is not the average accuracy across 150 epochs, so let's also find that:

```
import numpy as np

np.mean(model.history.history['val_acc'])
```

```
Out[68]: 0.7562053710841832
```

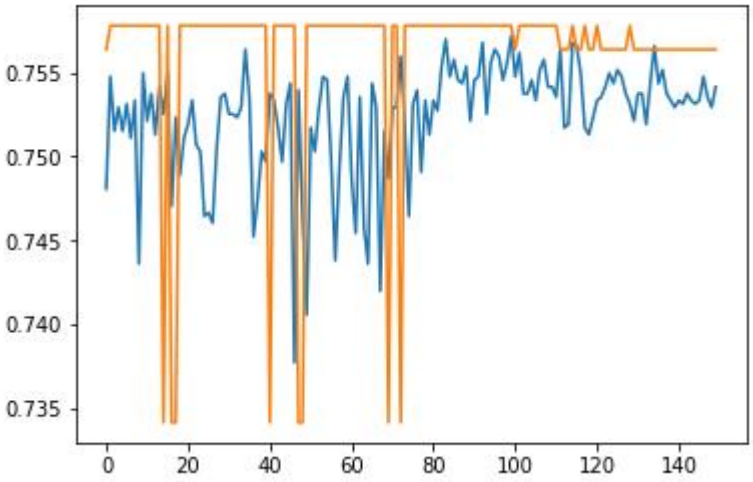
So, the mean accuracy comes out to be 75.62%.

```
y_pred=model.predict_classes(x_test)

from sklearn.metrics import confusion_matrix

confusion_matrix(y_test,y_pred)
```

Further, we will, predict the values on 'x_test' and build a confusion matrix with the actual values and the predicted values.

```
from matplotlib import pyplot as plt

plt.plot(model.history.history['acc'])

plt.plot(model.history.history['val_acc'])

plt.show()
```

Finally, we will make the 'Accuracy vs Epochs' plot:



C)b->

```
model = Sequential()
model.add(Dense(12, input_dim=1, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(8, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
```

Now, we are building our 2nd model, where we are adding a drop-out layer after the input layer and the hidden layer.

Drop-out value of 0.3 means that 70% of the nodes in the input layer will be dropped out.

Drop-out value of 0.2 means that 80% of the nodes in the hidden layer will be dropped out.

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=150,validation_data=(x_test,y_test))
y_pred = model.predict_classes(x_test)
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

```
from matplotlib import pyplot as plt

plt.plot(model.history.history['acc'])

plt.plot(model.history.history['val_acc'])

plt.show()
```

After this, we have fit the model and predicted the values.

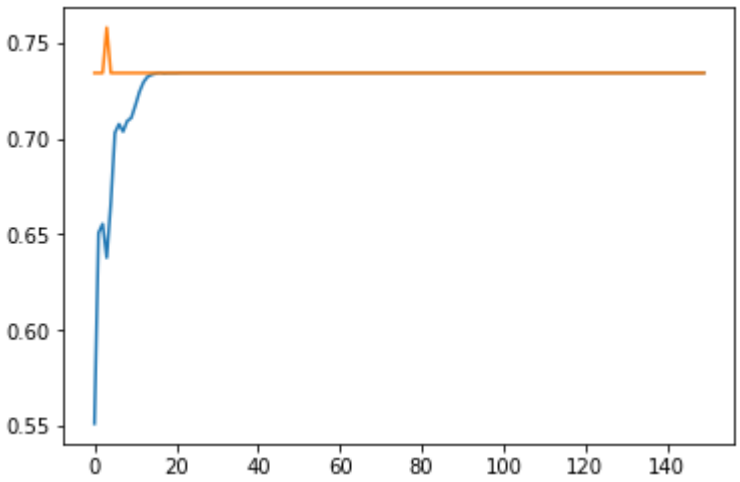
```
4922/4922 [==============================] - 0s 64us/step - loss: 0.5198 - acc: 0.7343 - val_loss: 0.5182 - val_acc: 0.7341
Epoch 144/150
4922/4922 [==============================] - 0s 78us/step - loss: 0.5204 - acc: 0.7343 - val_loss: 0.5198 - val_acc: 0.7341
Epoch 145/150
4922/4922 [==============================] - 0s 84us/step - loss: 0.5203 - acc: 0.7343 - val_loss: 0.5278 - val_acc: 0.7341
Epoch 146/150
4922/4922 [==============================] - 0s 86us/step - loss: 0.5207 - acc: 0.7343 - val_loss: 0.5176 - val_acc: 0.7341
Epoch 147/150
4922/4922 [==============================] - 0s 63us/step - loss: 0.5226 - acc: 0.7343 - val_loss: 0.5202 - val_acc: 0.7341
Epoch 148/150
4922/4922 [==============================] - 0s 65us/step - loss: 0.5195 - acc: 0.7343 - val_loss: 0.5229 - val_acc: 0.7341
Epoch 149/150
4922/4922 [==============================] - 0s 80us/step - loss: 0.5194 - acc: 0.7343 - val_loss: 0.5243 - val_acc: 0.7341
Epoch 150/150
4922/4922 [==============================] - 0s 75us/step - loss: 0.5181 - acc: 0.7343 - val_loss: 0.5218 - val_acc: 0.7341
```

So, we see that the 2nd model gives us a final validation accuracy of 73.41%. Now, let's calculate the mean validation accuracy across 150 epochs:

```
import numpy as np

np.mean(model.history.history['val_acc'])
```

```
Out[75]: 0.7342812007422695
```

So, the mean accuracy comes out to be 73.42%.



By looking at this graph, we can infer that the validation accuracy is constantly 73.41%. Now, this tells us that something is wrong with our model.

The most probable explanation for this is the drop-out percentage is very high for the input layer and the hidden layer and thus the model which we have built might be underfitting the data.

C)c->

```
x=customer[['MonthlyCharges','tenure','TotalCharges']]#Features

y=customer[['Churn']]#Target
```

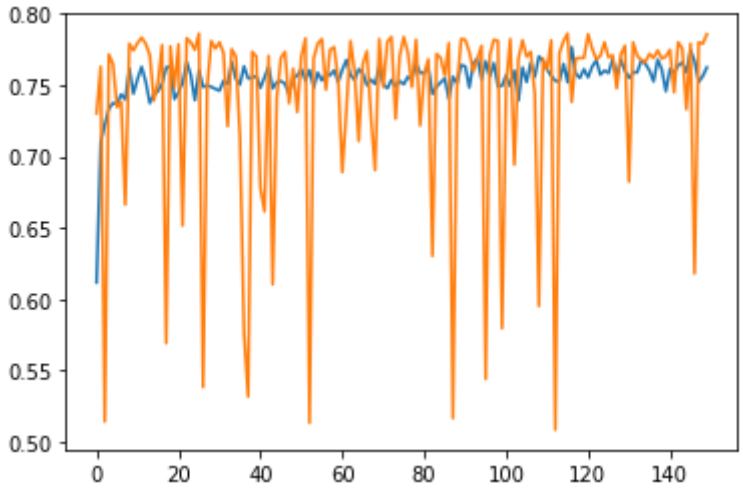
This time, we are taking 'Monthly Charges', 'Total Charges' and 'Tenure' as the features and 'Churn' as the target.

```
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=42)

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(x_train, y_train, epochs=150,validation_data=(x_test,y_test))

y_pred = model.predict_classes(x_test)

from sklearn.metrics import confusion_matrix

confusion_matrix(y_test,y_pred)


from matplotlib import pyplot as plt

plt.plot(model.history.history['acc'])

plt.plot(model.history.history['val_acc'])

plt.show()
```

After this, we divide the data into train and test sets and build the model on train test and predict the values on the test set.

```
Epoch 146/150
4922/4922 [==============================] - 0s 60us/step - loss: 0.5133 - acc: 0.7747 - val_loss: 0.4689 - val_acc: 0.7791
Epoch 147/150
4922/4922 [==============================] - 0s 60us/step - loss: 0.5271 - acc: 0.7670 - val_loss: 0.6488 - val_acc: 0.6180
Epoch 148/150
4922/4922 [==============================] - 0s 59us/step - loss: 0.7094 - acc: 0.7523 - val_loss: 0.6085 - val_acc: 0.7801
Epoch 149/150
4922/4922 [==============================] - 0s 64us/step - loss: 0.6599 - acc: 0.7564 - val_loss: 0.7243 - val_acc: 0.7791
Epoch 150/150
4922/4922 [==============================] - 0s 83us/step - loss: 0.5704 - acc: 0.7625 - val_loss: 0.6593 - val_acc: 0.7858
```

So, we see that we get a final validation accuracy of 78.58%.



But, when we look at this graph, we see that there is a constant fluctuation in the validation accuracy.

So, let's find out the mean validation accuracy across 150 epochs:

```
import numpy as np

np.mean(model.history.history['val_acc'])
```

And this gives a mean validation accuracy of 74.24%

Conclusion:

The first model gave us a mean validation accuracy of 75.62%, the second model had accuracy of 73.42 and the third model had a mean validation accuracy of 74.24%.

The second model gave us the least accuracy because we added two dropout layers with high probabilities of dropout.

Now, there could be many factors why third model's accuracy was less than that of first model. Most probably one or more of the features used during the model building could be of less significance leading to the reduction in accuracy.

It should also be kept in mind that these accuracy values are very specific to the hyperparameters used during the model building process such as optimizers, activation functions and number of epochs. If we were to tweak these hyperparameters we would get completely different accuracy values for all the three models.