1.) Preprocess your data into scaled input variables and an output variable

```
import pandas as pd
from google.colab import drive
import matplotlib.pyplot as plt
import numpy as np
import datetime
```

```
drive.mount('/content/gdrive/', force_remount = True)
```

```
Mounted at /content/gdrive/
```

```
df = pd.read_csv("/content/CLV.csv")
df.drop(columns = ["Unnamed: 0"])
```

	Customer Lifetime Value	Income	Number of Policies	Total Claim Amount	Months Since Last Claim	Vehicle Size_Large	Vehicle Size_Medsize	Gender_M	EmploymentStatus_Employed	EmploymentStatu
0	2763.519279	56274	1	384.811147	32	0	1	0	1	
1	6979.535903	0	8	1131.464935	13	0	1	0	0	
2	12887.431650	48767	2	566.472247	18	0	1	0	1	
3	7645.861827	0	7	529.881344	18	0	1	1	0	
4	2813.692575	43836	1	138.130879	12	0	1	1	1	
...	...	...	...	...	...	...	...	...	...	
9129	23405.987980	71941	2	198.234764	18	0	1	1	1	
9130	3096.511217	21604	1	379.200000	14	0	1	0	1	
9131	8163.890428	0	2	790.784983	9	0	1	1	0	
9132	7524.442436	21941	3	691.200000	34	1	0	1	1	
9133	2611.836866	0	1	369.600000	3	0	1	1	0	

9134 rows × 17 columns

```
df2 = df.copy()
X = df2.drop(columns = ["Customer Lifetime Value"])
X = pd.get_dummies(X)
y = df["Customer Lifetime Value"]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3)
```

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

2.) Run a GridSearch CV on at least 10 possible combinations of hyper parameters

```
clf = MLPRegressor()
from sklearn.model_selection import GridSearchCV


params = {'hidden_layer_sizes': [(10,), (5,20,), (10,10,)],
          'activation': ["relu", "tanh"],
          'alpha': [0.0001, 0.01]
}


grid  = GridSearchCV(clf, params, cv=5)
grid.fit(X_train, y_train)
```

```
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimi
  warnings.warn(
GridSearchCV(cv=5, estimator=MLPRegressor(),
             param_grid={'activation': ['relu', 'tanh'],
                         'alpha': [0.0001, 0.01],
```

```
print('best parameters:', grid.best_params_)
print('best score:'), grid.best_score_
```

```
best parameters: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (10, 10)}
best score:
(None, 0.04043918550360488)
```

3.) Train a model with the optimal solution from GridSearch

```
regressor = MLPRegressor(hidden_layer_sizes=(10,10), activation='relu', alpha=0.01, solver='adam', max_iter=1000)
regressor.fit(X_train, y_train)
```

```
ages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached a

yer_sizes=(10, 10), max_iter=1000)
```

```
y_pred_in = regressor.predict(X_train)
y_pred = regressor.predict(X_test)
```

4.) What are the in-sample and out of sample MSEs

```
out_mse = mean_squared_error(y_test, y_pred)
out_r2 = r2_score(y_test, y_pred)

print("out of sample mse:", out_mse)
print("out of sample r^2:", out_r2)
```

```
out of sample mse: 39171135.99614613
out of sample r^2: 0.07355299033565355
```

5.) Build a Keras with the architecture defined by GridSearchCV

```
import keras.models
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
```

```
model = Sequential()
#hidden layer of 10,10
model.add(Dense(10, input_dim=X_train.shape[1], activation="relu"))
model.add(Dense(10, activation="relu"))


model.compile(loss='mse', optimizer=Adam(lr=0.01))
model.fit(X_train, y_train, batch_size=32, epochs=10)
```

```
Epoch 1/10
/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:117: UserWarning: The `lr` argument is deprecated, use `le
  super().__init__(name, **kwargs)
200/200 [==============================] - 1s 3ms/step - loss: 113739736.0000
Epoch 2/10
200/200 [==============================] - 1s 3ms/step - loss: 107925904.0000
```

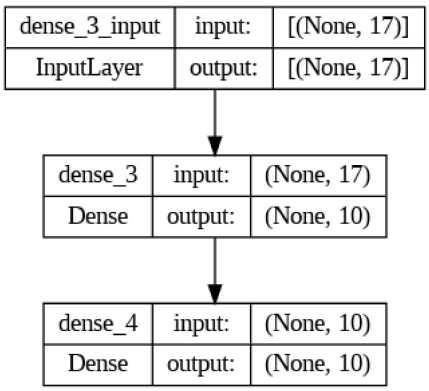
```
Epoch 3/10
200/200 [==============================] - 1s 3ms/step - loss: 97403192.0000
Epoch 4/10
200/200 [==============================] - 1s 3ms/step - loss: 85009408.0000
Epoch 5/10
200/200 [==============================] - 1s 3ms/step - loss: 73273352.0000
Epoch 6/10
200/200 [==============================] - 1s 2ms/step - loss: 63819156.0000
Epoch 7/10
200/200 [==============================] - 0s 2ms/step - loss: 57142896.0000
Epoch 8/10
200/200 [==============================] - 0s 2ms/step - loss: 52905992.0000
Epoch 9/10
200/200 [==============================] - 1s 3ms/step - loss: 50435564.0000
Epoch 10/10
200/200 [==============================] - 1s 3ms/step - loss: 49073856.0000
<keras.callbacks.History at 0x7fc525f40430>
```

6.) Make two visualizations of your NN using “plot_model” and “ann_viz”

```
pip install ann-visualizer
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting ann-visualizer
  Downloading ann_visualizer-2.5.tar.gz (4.7 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: ann-visualizer
  Building wheel for ann-visualizer (setup.py) ... done
  Created wheel for ann-visualizer: filename=ann_visualizer-2.5-py3-none-any.whl size=4168 sha256=845d5ae35e3a62cc96c3b5fc2e94605c21e57
  Stored in directory: /root/.cache/pip/wheels/4b/ef/77/9b8c4ae2f9a11de19957b80bc5c684accd99114bb8dc6b374c
Successfully built ann-visualizer
Installing collected packages: ann-visualizer
Successfully installed ann-visualizer-2.5
```

```
from tensorflow.keras.utils import plot_model
plot_model(model,show_shapes = True)
```




```
from ann_visualizer.visualize import ann_viz;
ann_viz(model, title = "CLV NN Viz Quentin", filename = "/folder/nn_model.gz")
```

✓ 0s completed at 9:52 AM