```
%tensorflow_version 2.x
import pandas as pd
import tensorflow as tf
from matplotlib import pyplot as plt
import os
from google.colab import files
uploaded = files.upload()

from numpy import argmax
import re
import numpy as np
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Seasons_Stats.csv to Seasons_Stats.csv
```

```
training_df = pd.read_csv("Seasons_Stats.csv")
```

```
# Print the first rows of the pandas DataFrame.
training_df.head()
```

	Unnamed: 0	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	FTr
0	0	1950.0	Curly Armstrong	G-F	31.0	FTW	63.0	NaN	NaN	NaN	0.368	NaN	0.467
1	1	1950.0	Cliff Barker	SG	29.0	INO	49.0	NaN	NaN	NaN	0.435	NaN	0.387
2	2	1950.0	Leo Barnhorst	SF	25.0	CHS	67.0	NaN	NaN	NaN	0.394	NaN	0.259
3	3	1950.0	Ed Bartels	F	24.0	TOT	15.0	NaN	NaN	NaN	0.312	NaN	0.395
4	4	1950.0	Ed Bartels	F	24.0	DNN	13.0	NaN	NaN	NaN	0.308	NaN	0.378

```
training_df["PTS"] /= 1000.0   # Scale the label.

pd.options.display.max_rows = 10
pd.options.display.float_format = "{:.1f}".format
```

```
test_df = pd.read_csv("Seasons_Stats.csv")
```

```
test_df=test_df[test_df.PTS > 50]
test_df=test_df[test_df.Year >= 2016]   #Test for years since 2016
test_df.head()
```

```
training_df = pd.read_csv("Seasons_Stats.csv")
training_df=training_df[training_df.Year < 2016]
training_df=training_df[training_df.Year > 2012]

training_df.head()

training_df[training_df.PTS > 50]
```

	Unnamed: 0	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	F
21679	21679	2013.0	Quincy Acy	PF	22.0	TOR	29.0	0.0	342.0	15.9	0.6	0.0	C
21680	21680	2013.0	Jeff Adrien	PF	26.0	CHA	52.0	5.0	713.0	13.4	0.5	0.0	C
21681	21681	2013.0	Arron Afflalo	SF	27.0	ORL	64.0	64.0	2307.0	13.0	0.5	0.3	C
21683	21683	2013.0	Cole Aldrich	C	24.0	TOT	45.0	0.0	388.0	11.1	0.6	0.0	C
21686	21686	2013.0	LaMarcus Aldridge	PF	27.0	POR	74.0	74.0	2790.0	20.4	0.5	0.0	C
...	...	...	...	...	...	...	...	...	...	...	...	...	
23511	23511	2015.0	Thaddeus Young	PF	26.0	TOT	76.0	68.0	2434.0	15.7	0.5	0.1	C
23512	23512	2015.0	Thaddeus Young	PF	26.0	MIN	48.0	48.0	1605.0	15.0	0.5	0.1	C
23513	23513	2015.0	Thaddeus Young	PF	26.0	BRK	28.0	20.0	829.0	17.1	0.5	0.2	C
23514	23514	2015.0	Cody Zeller	C	22.0	CHO	62.0	45.0	1487.0	14.1	0.5	0.0	C
23515	23515	2015.0	Tyler Zeller	C	25.0	BOS	82.0	59.0	1731.0	18.9	0.6	0.0	C

1489 rows × 53 columns

```
del training_df['Year']
```

```
del training_df['Unnamed: 0']
del training_df['Player']
del training_df['Pos']
del training_df['Tm']
del training_df['Age']
del training_df['blanl']
del training_df['blank2']

del test_df['Year']
del test_df['Unnamed: 0']
del test_df['Player']
del test_df['Pos']
del test_df['Tm']
del test_df['Age']
del test_df['blanl']
del test_df['blank2']

#inefficient way to delete all non numeric variables from test and training data fr
#For now I don't want to need at player names, but in the future
#I would use an integer enocding to learn more about player specific performance
```

```
# Normalize
train_df_mean = training_df.mean()
train_df_std = training_df.std()
train_df_norm = (training_df - train_df_mean)/train_df_std

test_df_mean = test_df.mean()
test_df_std = test_df.std()
test_df_norm = (test_df - test_df_mean)/test_df_std
```

```
test_df.head()
training_df.head()
```

	G	GS	MP	PER	TS%	3PAr	FTr	ORB%	DRB%	TRB%	AST%	STL%	BLK%	TOV%
21679	29.0	0.0	342.0	15.9	0.6	0.0	0.5	10.3	16.6	13.4	5.2	2.0	3.5	15.6
21680	52.0	5.0	713.0	13.4	0.5	0.0	0.6	10.6	21.2	15.7	8.3	1.3	3.1	13.1
21681	64.0	64.0	2307.0	13.0	0.5	0.3	0.2	1.4	10.3	5.8	14.6	0.9	0.4	12.1
21682	3.0	0.0	9.0	15.3	0.6	0.5	0.0	0.0	12.1	6.2	19.4	0.0	0.0	0.0
21683	45.0	0.0	388.0	11.1	0.6	0.0	0.2	8.7	26.7	17.7	3.4	0.7	4.6	20.6

```
# Create an empty list that will eventually hold all created feature columns.
feature_columns = []

resolution_in_Zs = 0.3  # 3/10 of a standard deviation.

# Create a bucket feature column for offensive win shares.

OWS_as_a_numeric_column = tf.feature_column.numeric_column("OWS")

OWS_boundaries = list(np.arange(int(min(train_df_norm['OWS'])),
                                 int(max(train_df_norm['OWS'])),
                                 resolution_in_Zs))

OWS = tf.feature_column.bucketized_column(OWS_as_a_numeric_column, OWS_boundaries)

#Create a bucket feature column for defensive win shares.


DWS_as_a_numeric_column = tf.feature_column.numeric_column("DWS")

DWS_boundaries = list(np.arange(int(min(train_df_norm['DWS'])),
                                  int(max(train_df_norm['DWS'])),
                                  resolution_in_Zs))

DWS = tf.feature_column.bucketized_column(DWS_as_a_numeric_column,
                                            DWS_boundaries)

# Create a feature cross of OWS and DWS.
OWS_x_DWS = tf.feature_column.crossed_column([OWS,DWS],hash_bucket_size=30)
crossed_feature_1 = tf.feature_column.indicator_column(OWS_x_DWS)
feature_columns.append(crossed_feature_1)

# Create a bucket feature column for offensive rebounds.

ORB_as_a_numeric_column = tf.feature_column.numeric_column("ORB")
```

```
ORB_boundaries = list(np.arange(int(min(train_df_norm['ORB'])),
                                     int(max(train_df_norm['ORB'])),
                                     resolution_in_Zs))

ORB = tf.feature_column.bucketized_column(ORB_as_a_numeric_column, ORB_boundaries)

#Create a bucket feature column for defensive rebounds.


DRB_as_a_numeric_column = tf.feature_column.numeric_column("DRB")

DRB_boundaries = list(np.arange(int(min(train_df_norm['DRB'])),
                                      int(max(train_df_norm['DRB'])),
                                      resolution_in_Zs))

DRB = tf.feature_column.bucketized_column(DRB_as_a_numeric_column,
                                                DRB_boundaries)

# Create a feature cross of ORB and DRB.
ORB_x_DRB = tf.feature_column.crossed_column([ORB,DRB],hash_bucket_size=30)
crossed_feature_2 = tf.feature_column.indicator_column(ORB_x_DRB)
feature_columns.append(crossed_feature_2)

# Represent Turnovers as a floating-point value.
Turnovers = tf.feature_column.numeric_column("TOV")
feature_columns.append(Turnovers)

# Represent Steals as a floating-point value.
Steals = tf.feature_column.numeric_column("STL")
feature_columns.append(Steals)

# Represent population as a floating-point value.
Three_Point_Attempts = tf.feature_column.numeric_column("3PA")
feature_columns.append(Three_Point_Attempts)

# Represent population as a floating-point value.
Minutes_Played = tf.feature_column.numeric_column("MP")
feature_columns.append(Minutes_Played)


# Represent population as a floating-point value.
Two_Point_Attempts = tf.feature_column.numeric_column("2PA")
feature_columns.append(Two_Point_Attempts)

# Represent Field Goal Attempts as a floating-point value.
Field_Goal_Attempts = tf.feature_column.numeric_column("FGA")
feature_columns.append(Field_Goal_Attempts)

my_feature_layer = tf.keras.layers.DenseFeatures(feature_columns)
```

```
def plot_the_loss_curve(epochs, mse):
  """Plot a curve of loss vs. epoch."""

  plt.figure()
  plt.xlabel("Epoch")
  plt.ylabel("Mean Squared Error")

  plt.plot(epochs, mse, label="Loss")
  plt.legend()
  plt.ylim([mse.min()*0.95, mse.max() * 1.03])
  plt.show()

print("Defined the plot_the_loss_curve function.")
```

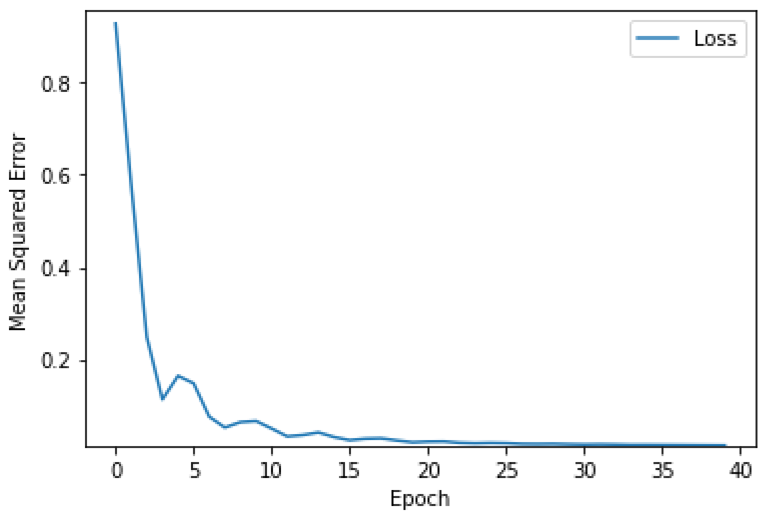
```
Defined the plot_the_loss_curve function.
```

```
def train_model(model, dataset, epochs, label_name,
                batch_size=None):

  # Split the dataset into features and label.
  features = {name:np.array(value) for name, value in dataset.items()}
  label = np.array(features.pop(label_name))
  history = model.fit(x=features, y=label, batch_size=batch_size,
                      epochs=epochs, shuffle=True)

  #store epochs separately from the rest of history
  epochs = history.epoch

  # To track the progression of training, gather a snapshot
  # of the model's mean squared error at each epoch.
  hist = pd.DataFrame(history.history)
  mse = hist["mean_squared_error"]

  return epochs, mse


def create_model(my_learning_rate, my_feature_layer):
  #Create and compile a simple linear model
  model = tf.keras.models.Sequential()

  model.add(my_feature_layer)

# Define the first hidden layer with 32 nodes.
  model.add(tf.keras.layers.Dense(units=32,
                                  activation='relu',
                                  name='Hidden1'))

# Define the second hidden layer with 20 nodes.
  model.add(tf.keras.layers.Dense(units=20,
                                  activation='relu',
                                  name='Hidden2'))
```

```
  # Define the third hidden layer with 10 nodes.
  model.add(tf.keras.layers.Dense(units=10,
                                  activation='relu',
                                  name='Hidden3'))



  # Define the output layer.
  model.add(tf.keras.layers.Dense(units=1,
                                  name='Output'))

  model.compile(optimizer=tf.keras.optimizers.Adam(lr=my_learning_rate),
                loss="mean_squared_error",
                metrics=[tf.keras.metrics.MeanSquaredError()])

  return model

# The following variables are the hyperparameters.
learning_rate = 0.01
epochs = 40
batch_size = 1000

# Specify the label
label_name = "PTS"

# Establish the model's topography.
my_model = create_model(learning_rate, my_feature_layer)

# Train the model on the normalized training set. We're passing the entire
# normalized training set, but the model will only use the features
# defined by the feature_layer.
epochs, mse = train_model(my_model, train_df_norm, epochs,
                          label_name, batch_size)
plot_the_loss_curve(epochs, mse)

# After building a model against the training set, test that model
# against the test set.
test_features = {name:np.array(value) for name, value in test_df_norm.items()}
test_label = np.array(test_features.pop(label_name)) # isolate the label
print("\n Evaluate the new model against the test set:")
my_model.evaluate(x = test_features, y = test_label, batch_size=batch_size)
```

```
Epoch 1/40
2/2 [==============================] - 0s 4ms/step - loss: 0.9237 - mean_squared_error:
Epoch 2/40
2/2 [==============================] - 0s 4ms/step - loss: 0.5695 - mean_squared_error:
Epoch 3/40
2/2 [==============================] - 0s 3ms/step - loss: 0.2429 - mean_squared_error:
Epoch 4/40
2/2 [==============================] - 0s 4ms/step - loss: 0.1132 - mean_squared_error:
Epoch 5/40
2/2 [==============================] - 0s 3ms/step - loss: 0.1652 - mean_squared_error:
Epoch 6/40
2/2 [==============================] - 0s 3ms/step - loss: 0.1463 - mean_squared_error:
Epoch 7/40
2/2 [==============================] - 0s 2ms/step - loss: 0.0737 - mean_squared_error:
Epoch 8/40
2/2 [==============================] - 0s 2ms/step - loss: 0.0526 - mean_squared_error:
Epoch 9/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0636 - mean_squared_error:
Epoch 10/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0664 - mean_squared_error:
Epoch 11/40
2/2 [==============================] - 0s 2ms/step - loss: 0.0487 - mean_squared_error:
Epoch 12/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0325 - mean_squared_error:
Epoch 13/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0357 - mean_squared_error:
Epoch 14/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0413 - mean_squared_error:
Epoch 15/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0306 - mean_squared_error:
Epoch 16/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0243 - mean_squared_error:
Epoch 17/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0277 - mean_squared_error:
Epoch 18/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0284 - mean_squared_error:
Epoch 19/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0233 - mean_squared_error:
Epoch 20/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0196 - mean_squared_error:
Epoch 21/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0211 - mean_squared_error:
Epoch 22/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0213 - mean_squared_error:
Epoch 23/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0185 - mean_squared_error:
Epoch 24/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0176 - mean_squared_error:
Epoch 25/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0183 - mean_squared_error:
Epoch 26/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0179 - mean_squared_error:
Epoch 27/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0163 - mean_squared_error:
Epoch 28/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0159 - mean_squared_error:
Epoch 29/40
```

```
2/2 [==============================] - 0s 3ms/step - loss: 0.0163 - mean_squared_error:
Epoch 30/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0153 - mean_squared_error:
Epoch 31/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0148 - mean_squared_error:
Epoch 32/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0153 - mean_squared_error:
Epoch 33/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0147 - mean_squared_error:
Epoch 34/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0142 - mean_squared_error:
Epoch 35/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0141 - mean_squared_error:
Epoch 36/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0138 - mean_squared_error:
Epoch 37/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0136 - mean_squared_error:
Epoch 38/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0135 - mean_squared_error:
Epoch 39/40
2/2 [==============================] - 0s 3ms/step - loss: 0.0131 - mean_squared_error:
Epoch 40/40
2/2 [==============================] - 0s 4ms/step - loss: 0.0129 - mean_squared_error:
```




```
 Evaluate the new model against the test set:
1/1 [==============================] - 0s 1ms/step - loss: 0.0261 - mean_squared_error:
[0.02606414072215557, 0.02606414072215557]
```