Step 1: Import Libraries

```
!pip install tensorflow
```

Show hidden output

```
%tensorflow_version X.X
from numpy.random import seed
seed(2)
#from tensorflow import set_random_seed
#set_random_seed(2)
import tensorflow as tf
from tensorflow import keras
from IPython import display
from matplotlib import cm
from matplotlib import gridspec
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
from tensorflow.python.data import Dataset
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
print(tf.__version__)
```

```
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
2.17.0
```

Step 2: Import Data

```
# prompt: read excel file to df

#df = pd.read_excel('Assignment1-OnlineUse.xlsx') # Replace with your file path
```

```
df = pd.read_csv('Assignment1-OnlineUse.csv') # Replace with your file path
print(df.head())
```

```
   ClinicID  Online_use-patient   malepct     unemp  age16to24  age25to34  \
0         1            0.059697  0.478453  0.108287   0.058701   0.181771
1         2            0.133762  0.474550  0.106176   0.146846   0.139272
2         3            0.060985  0.404530  0.036369   0.067028   0.125814
3         4            0.098957  0.398067  0.020966   0.101616   0.073968
4         5            0.099779  0.467494  0.034454   0.108035   0.181779

   age35to44  age45to54  age55to64  age65to74  ...  phoneeasy  onlineasy  \
0   0.099166   0.182478   0.142989   0.154677  ...   0.465492   0.767291
1   0.072946   0.218530   0.165720   0.139868  ...   0.717218   0.941951
2   0.195154   0.170479   0.181745   0.140921  ...   0.857945   0.976829
3   0.205223   0.143792   0.132499   0.200629  ...   0.688784   0.941920
4   0.114250   0.120209   0.230567   0.127471  ...   0.682212   0.848249

       race  longstdhealth  canmngownhealth  reducedability  prefgpalways  \
0  0.824691       0.526022         0.820753        0.651606      0.534662
1  0.956473       0.571299         0.796031        0.696665      0.723858
2  0.922501       0.426156         0.955261        0.652033      0.563364
3  0.975631       0.497154         0.912809        0.515677      0.568220
4  0.921327       0.644262         0.862050        0.598967      0.535829

   bcaaware vendor  numpats
0  0.180921    TPP     4088
1  0.297806    TPP    19599
2  0.231334    TPP    10606
3  0.547644    TPP     8047
4  0.405589    TPP    14585

[5 rows x 22 columns]
```

```
df.describe()
```

```
df.columns
```

```
Index(['ClinicID', 'Online_use-patient', 'malepct', 'unemp', 'age16to24',
       'age25to34', 'age35to44', 'age45to54', 'age55to64', 'age65to74',
       'age75to84', 'age85plus', 'phoneeasy', 'onlineasy', 'race',
       'longstdhealth', 'canmngownhealth', 'reducedability', 'prefgpalways',
       'bcaaware', 'vendor', 'numpats'],
      dtype='object')
```

```
# prompt: unique values in df['Category']

unique_values = df['vendor'].unique()
print(unique_values)
```

```
['TPP' 'EMIS' 'EMIS (I)' 'VISION' 'MICROTEST' 'VISION (I)']
```

```
df.isnull().sum()
```

Visualise distributions

```
# import matplotlib.pyplot as plt

# # extra code – the next 5 lines define the default font sizes
# plt.rc('font', size=12)
# plt.rc('axes', labelsize=12, titlesize=12)
# plt.rc('legend', fontsize=12)
# plt.rc('xtick', labelsize=10)
# plt.rc('ytick', labelsize=10)

# df.hist(bins=50, figsize=(20, 12))

# plt.show()
```

Step 3: Data *Preprocessing*

```
!pip install category_encoders
```

Show hidden output

```
!pip install sklearn
```

Show hidden output

```
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PowerTransformer, OneHotEncoder,  QuantileTransformer, FunctionTransformer, StandardSc
from sklearn.compose import ColumnTransformer
from category_encoders import TargetEncoder
from sklearn.impute import KNNImputer, SimpleImputer
from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean_absolute_error

from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
```

Categorical: One Hot Encoding: 'vendor'

```
df_one_hot = pd.get_dummies(df, columns=['vendor'])
```

Train/Validation Split

```
# Features are all columns except 'Target'
X = df_one_hot.drop(columns=['ClinicID', 'Online_use-patient'], axis=1)
# Target variable is 'Target'
y = df_one_hot['Online_use-patient']
```

```
# Split the data: 80% for training, 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Print shapes to verify the split
print(f'X_train shape: {X_train.shape}')
print(f'X_test shape: {X_test.shape}')
print(f'y_train shape: {y_train.shape}')
print(f'y_test shape: {y_test.shape}')
```

```
X_train shape: (5485, 25)
X_test shape: (1372, 25)
y_train shape: (5485,)
y_test shape: (1372,)
```

```
X.columns
```

```
Index(['malepct', 'unemp', 'age16to24', 'age25to34', 'age35to44', 'age45to54',
       'age55to64', 'age65to74', 'age75to84', 'age85plus', 'phoneeasy',
       'onlineasy', 'race', 'longstdhealth', 'canmngownhealth',
       'reducedability', 'prefgpalways', 'bcaaware', 'numpats', 'vendor_EMIS',
       'vendor_EMIS (I)', 'vendor_MICROTEST', 'vendor_TPP', 'vendor_VISION',
       'vendor_VISION (I)'],
      dtype='object')
```

```
df_one_hot.columns
```

```
Index(['ClinicID', 'Online_use-patient', 'malepct', 'unemp', 'age16to24',
       'age25to34', 'age35to44', 'age45to54', 'age55to64', 'age65to74',
       'age75to84', 'age85plus', 'phoneeasy', 'onlineasy', 'race',
       'longstdhealth', 'canmngownhealth', 'reducedability', 'prefgpalways',
       'bcaaware', 'numpats', 'vendor_EMIS', 'vendor_EMIS (I)',
       'vendor_MICROTEST', 'vendor_TPP', 'vendor_VISION', 'vendor_VISION (I)'],
      dtype='object')
```

Step 4: Build Model

Baseline models

Baseline model 1

```
baseline_model = keras.Sequential([
    keras.layers.Dense(26, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(1)  # linear/relu in the output layer
])

#optimizer = tf.keras.optimizers.RMSprop(0.001) # Gradient Descent algorithm
#optimizer = tf.keras.optimizers.Adam()
```

```
#optimizer = tf.keras.optimizers.Adam()

# Compile the model with categorical_crossentropy
baseline_model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_error'])

# Display the model's architecture
baseline_model.summary()
```

Fit Model

```
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 100 == 0: print('')
    print('.', end='')

EPOCHS = 200
b_history = baseline_model.fit(X_train, y_train, epochs=EPOCHS,
                    validation_data= (X_test, y_test), verbose=0,
                    callbacks=[PrintDot()])
```

```
....................................................................................................
```

```
.........................................................................................................
```

```
# Evaluate the model on the test set
test_loss, test_accuracy = baseline_model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}')
```

```
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 28.1137 - mean_absolute_error: 3.5310
Test Accuracy: 3.61
```

Lowest Validation Error

```
# Print the lowest validation error
lowest_validation_error = min(b_history.history['val_loss'])
print(f'Lowest Validation Error: {lowest_validation_error:.4f}')
```

```
Lowest Validation Error: 21.0886
```

Baseline model 2

```
# Regularized model

baseline_model = keras.Sequential()
baseline_model.add(keras.layers.Dense(26, activation=tf.nn.relu,
                       input_shape=(X_train.shape[1],))),
baseline_model.add(keras.layers.Dense(16, activation=tf.nn.relu,)),
baseline_model.add(keras.layers.Dense(8, activation=tf.nn.relu,)),
baseline_model.add(keras.layers.Dense(8, activation=tf.nn.relu,)),

baseline_model.add(keras.layers.Dense(1))

optimizer = tf.keras.optimizers.RMSprop() # Gradient Descent algorithm
optimizer = tf.keras.optimizers.Adam()

baseline_model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_error'])
```

```
baseline_model.summary()
```

Fit Model

```
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 100 == 0: print('')
    print('.', end='')

class TrackValidationLoss(keras.callbacks.Callback):
    def __init__(self):
        super(TrackValidationLoss, self).__init__()
        self.lowest_val_loss = np.inf  # Start with infinity

    def on_epoch_end(self, epoch, logs):
        current_val_loss = logs.get('val_loss')
        if current_val_loss < self.lowest_val_loss:
            self.lowest_val_loss = current_val_loss
```

```
            print(f'\nLowest validation loss updated: {self.lowest_val_loss:.4f}')

# Create an instance of the custom callback
track_val_loss = TrackValidationLoss()

EPOCHS = 200
b_history = baseline_model.fit(X_train, y_train, epochs=EPOCHS,
                    validation_data= (X_test, y_test), verbose=0,
                    callbacks=[PrintDot(), track_val_loss])
```

Show hidden output

```
# Evaluate the model on the test set
test_loss, test_accuracy = baseline_model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}')
```

```
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 10.4783 - mean_absolute_error: 0.3098
Test Accuracy: 0.42
```

Lowest Validation Error

```
# Print the lowest validation error
lowest_validation_error = min(b_history.history['val_loss'])
print(f'Lowest Validation Error: {lowest_validation_error:.4f}')
```

```
Lowest Validation Error: 21.0182
```

Plot Results

```
# Plotting results
train_loss = b_history.history['loss']
val_loss = b_history.history['val_loss']
train_accuracy = b_history.history['mean_absolute_error']
```

```
val_accuracy = b_history.history['val_mean_absolute_error']

# Set the number of epochs for x-axis
epochs_range = range(1, EPOCHS + 1)

# Create subplots for loss and accuracy
plt.figure(figsize=(12, 5))

# Plot training and validation loss
plt.subplot(1, 2, 1)
plt.plot(epochs_range, train_loss, 'bo-', label='Training Loss', linewidth=1, markersize=1)
plt.plot(epochs_range, val_loss, 'ro-', label='Validation Loss', linewidth=1, markersize=1)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plot training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs_range, train_accuracy, 'bo-', label='Training Accuracy', linewidth=1, markersize=1)
plt.plot(epochs_range, val_accuracy, 'ro-', label='Validation Accuracy', linewidth=1, markersize=1)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Show the plots
plt.tight_layout()
plt.show()
```

Predictions

```
# valpreds = baseline_model.predict_on_batch(X_test)
# print(valpreds)
```

```
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    print(y_test)
```

Show hidden output

```
# Plot Weights
nfw = baseline_model.get_weights()[0][0]
y_pos = np.arange(len(nfw))

plt.bar(y_pos, nfw, align='center', alpha=0.5)
```

Regularized Model

```
l1_model = keras.Sequential([
    keras.layers.Dense(32, kernel_regularizer=keras.regularizers.l1(0.1), activation=tf.nn.relu,
                       input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, use_bias=True, kernel_regularizer=keras.regularizers.l1(0.01), activation=tf.nn.relu),
    keras.layers.Dense(32, use_bias=True, kernel_regularizer=keras.regularizers.l1(0.01), activation=tf.nn.relu),
    keras.layers.Dense(32, use_bias=True, kernel_regularizer=keras.regularizers.l1(0.01), activation=tf.nn.relu),
    keras.layers.Dense(1)

  ])

l1_model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_error'])
```

```
l1_model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_error'])

l2_model = keras.Sequential([
    keras.layers.Dense(32, kernel_regularizer=keras.regularizers.l2(0.1), activation=tf.nn.relu,
                       input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, kernel_regularizer=keras.regularizers.l2(0.1), activation=tf.nn.relu),
    keras.layers.Dense(32, kernel_regularizer=keras.regularizers.l2(0.1), activation=tf.nn.relu),
    keras.layers.Dense(32, kernel_regularizer=keras.regularizers.l2(0.1), activation=tf.nn.relu),
    keras.layers.Dense(1)
  ])

l2_model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_error'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
l1_history = l1_model.fit(X_train, y_train, epochs=EPOCHS,
                    validation_data= (X_test, y_test), verbose=0,
                    callbacks=[PrintDot()])
l2_history = l2_model.fit(X_train, y_train, epochs=EPOCHS,
                    validation_data= (X_test, y_test), verbose=0,
                    callbacks=[PrintDot()])
```

```
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
```

```
# Evaluate the model on the test set
test_loss, test_accuracy = l1_model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}')
```

```
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 10.8884 - mean_absolute_error: 0.3490
Test Accuracy: 0.47
```

```
# Evaluate the model on the test set
test_loss, test_accuracy = l2_model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}')
```

```
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 10.6656 - mean_absolute_error: 0.2960
Test Accuracy: 0.41
```

Plot Results

```
def plot_history(histories):
    plt.figure(figsize=(14, 6))

    # Plotting loss
    plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
    for label, history in histories:
        plt.plot(history.history['loss'], label=f'{label} Loss')
        plt.plot(history.history['val_loss'], linestyle='--', label=f'{label} Val Loss')
    plt.title('Model Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid()

    # Plotting accuracy
    plt.subplot(1, 2, 2)  # 1 row, 2 columns, 2nd subplot
    for label, history in histories:
        plt.plot(history.history['mean_absolute_error'], label=f'{label} MAE')
        plt.plot(history.history['val_mean_absolute_error'], linestyle='--', label=f'{label} Val MAE')
    plt.title('Model Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid()

    plt.tight_layout()  # Adjust layout to prevent overlap
    plt.show()
```

```
# Example usage
plot_history([('baseline', b_history),
              ('L1', l1_history),
              ('L2', l2_history)])
```

Conclusion

- The accuracy of the model using a naïve approach- 0.42

- The accuracy of the best model: 0.47 using L1- regularization approach