Corporate credit rating

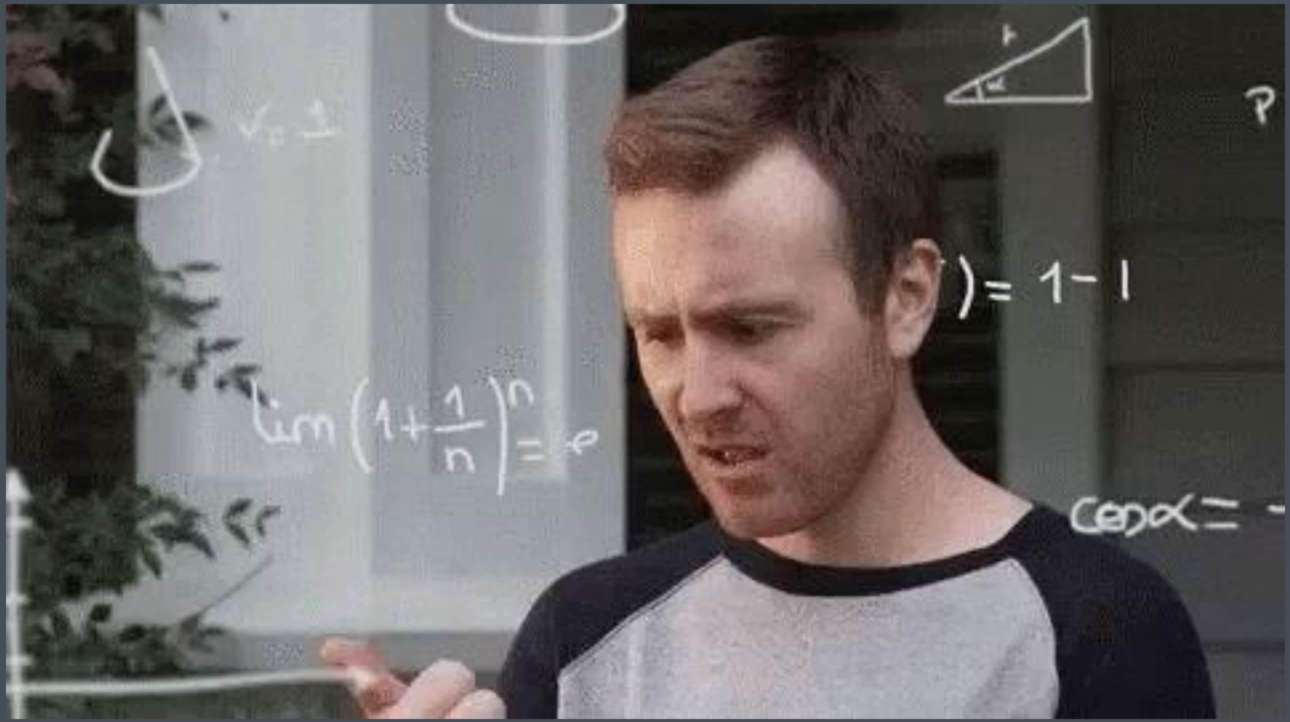
Represent by: John Antony

Nishant Patel

Liam Baker

Question!

Is there any strategies to predict credit rating of unknown company?

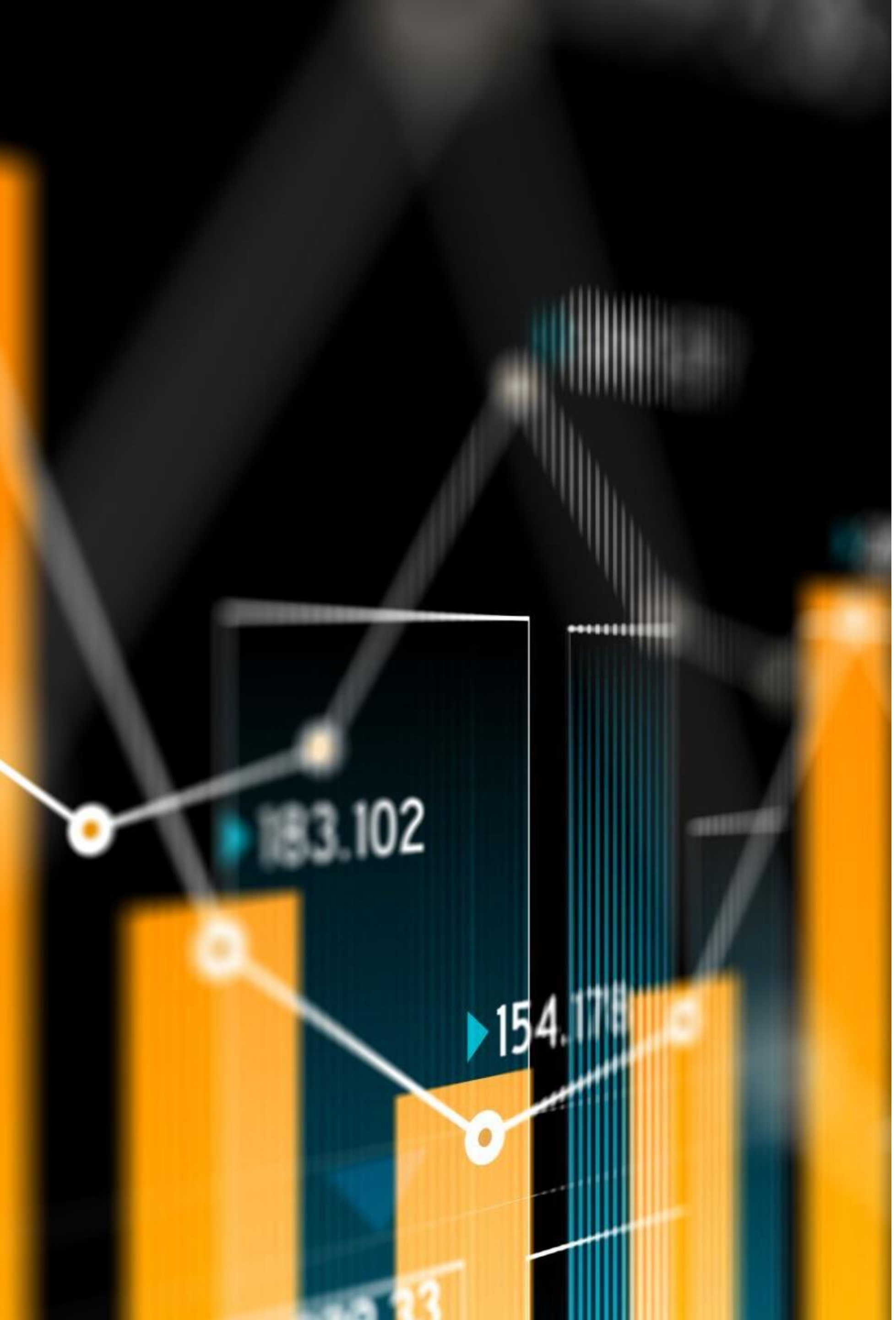


Objectives

- Data cleaning and loading data in database using postgres
- Import library functions
- Preliminary analysis
- Outputs with graphs
- Neural network model and keras optimizer to tune hyperparameter
- Why did we opt for Deep Learning over other models?
- Base line model accuracy
- Hyperparameter tuning for better model accuracy

Import functions

- Train-test-split, pandas, keras, tensarflow
- Adam, standard scaler, matplotlib
- Dense, dropout, sequential
- Mean squared log arithmetic error
- Create engine, skleran, model, config, seaborn
- Psycopg, one hot encoder



Data cleaning and database loading

- Use Rating as independent variable
- Combined sectors with less than 100 counts as others
- Replace ratings with AAA to 1, AA to 1, B to 0 and so on.
- Save dependent variables in cleaned_placeholder_X.csv
- Save independent variable in cleaned_y.csv

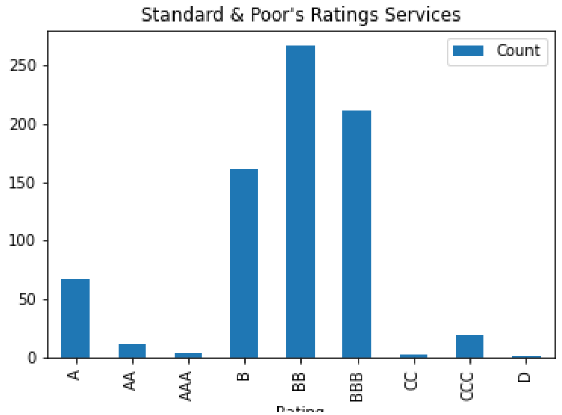
Connecting and storing data in a SQL database

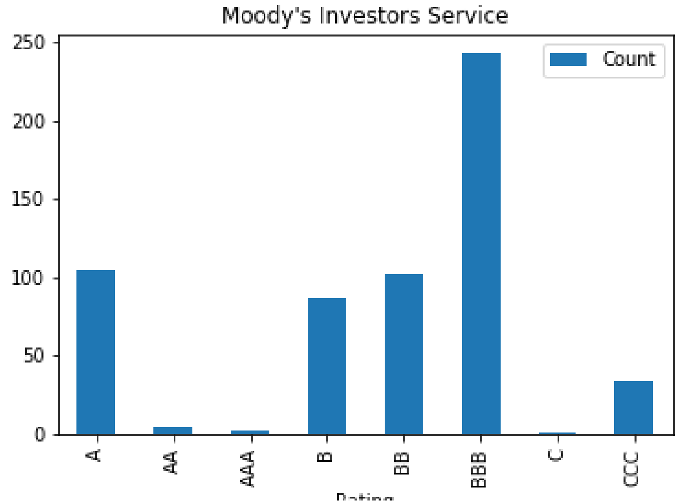
```
In [10]: #Save outputs to CSV - PLACEHOLDER UNTIL DATABASE IS SETUP
         encoded_df.to_csv('./Resources/cleaned_placeholder_X.csv')
         target.to_csv('./Resources/cleaned_y.csv')
         alternate_target.to_csv('./Resources/alternate_y.csv')

In [11]: #Connect and save to a local Postgres server
         #Requires a server to be running on your machine
         protocol = 'postgresql'
         username = config.username
         password = config.password
         host = 'localhost'
         port = 5432
         database_name = config.database_name
         rds_connection_string = f'{protocol}://{username}:{password}@{host}:{port}/{database_name}'
         engine = create_engine(rds_connection_string)
         con = engine.connect()

In [12]: encoded_df.to_sql('X',con,if_exists='replace', index = False)
         target.to_sql('y',con,if_exists='replace', index = False)
         alternate_target.to_sql('alternate_y',con,if_exists='replace', index = False)
         pd.read_csv("./Resources/corporate_rating.csv").to_sql('original',con,if_exists='replace', index = False)

Out[12]: 29
```

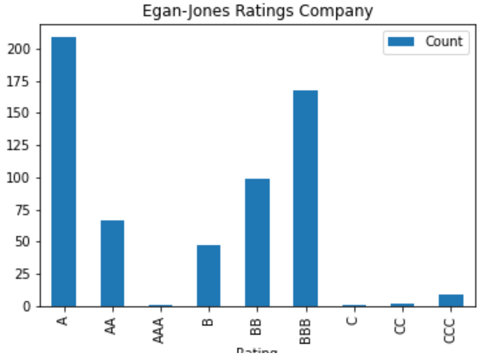
BBB	671
BB	490
A	398
B	302
AA	89
CCC	64
AAA	7
CC	5
C	2
D	1

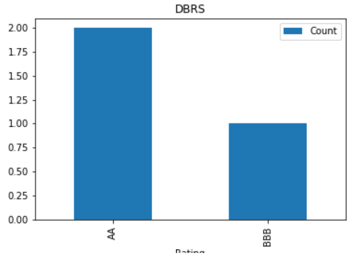
Preliminary Analysis

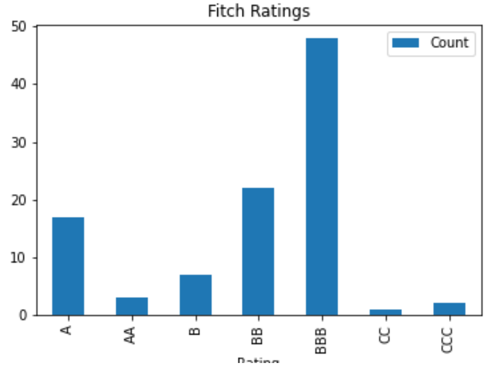
- Biased dataset – very few samples for some ratings
- Cannot build an effective classifier for each individual rating
- Cannot lump together an 'other' category – we need a difference between AAA and C/D ratings
- So, build a binary classifier for 'A' categories (A, AA, AAA) or any other category







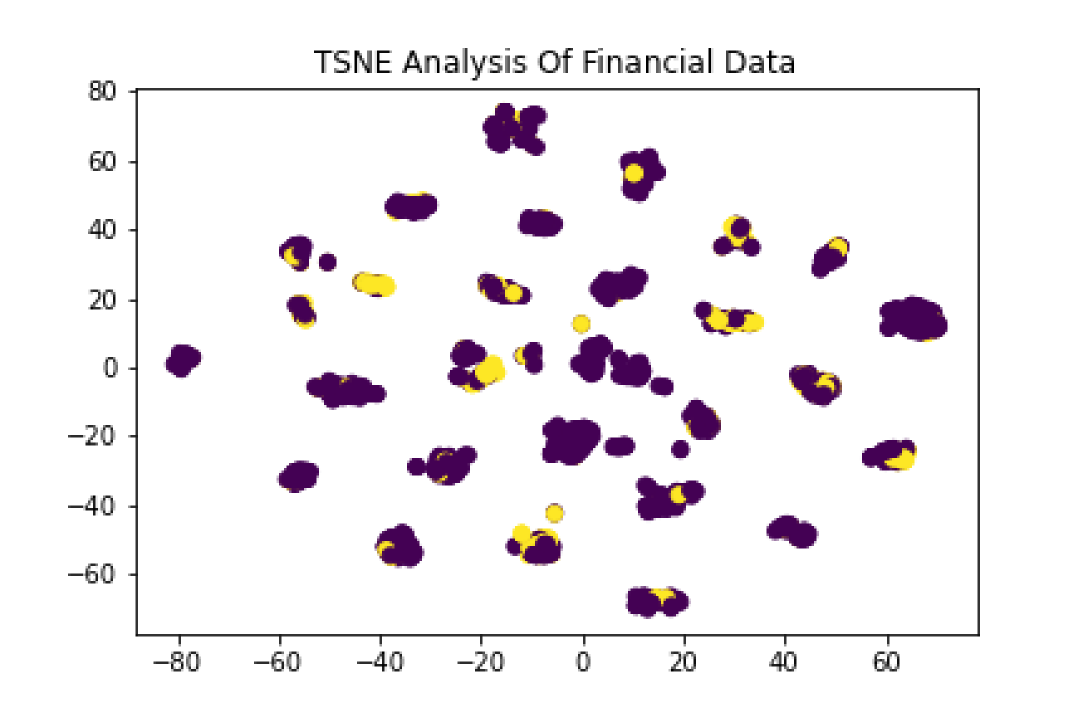




- We chose to remove the rating agency from the classifier since we want to focus on financial trends to pick out real information

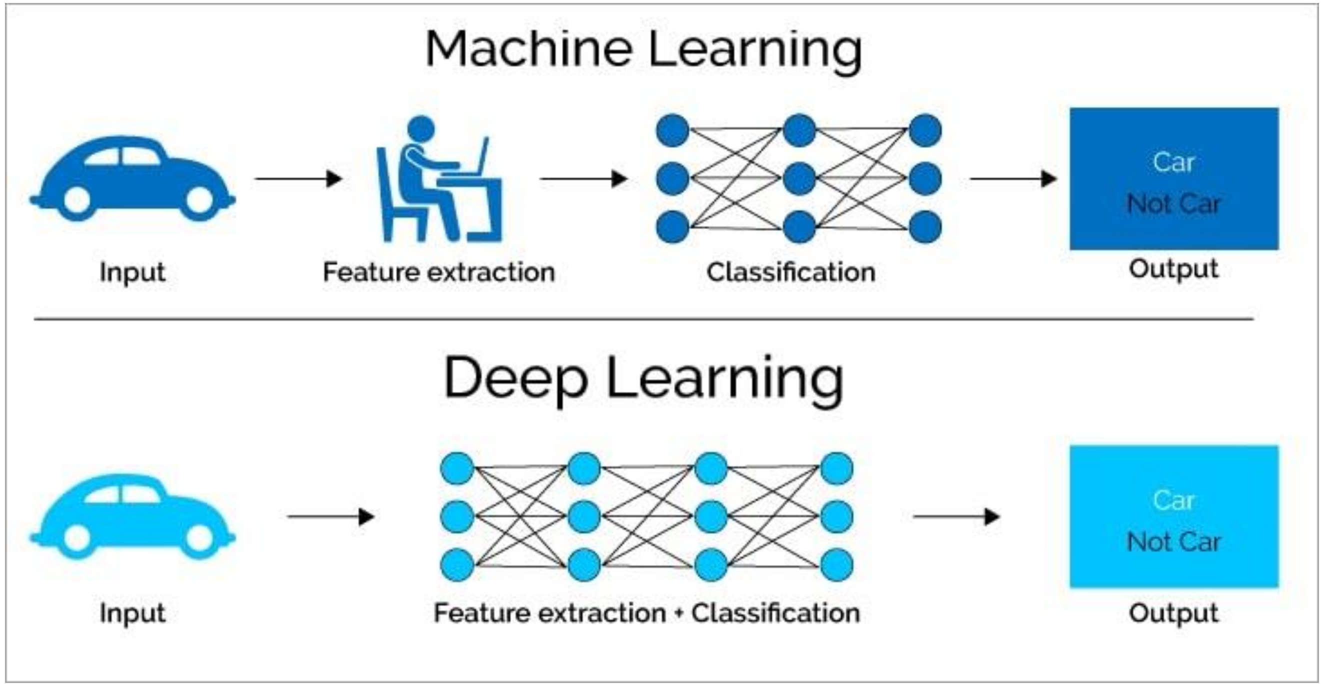
Clustering

- An alternative pathway for solving this problem is clustering
- We chose not to use clustering as it is more suited to unsupervised learning, and we have our target data
- TSNE Component Analysis shows that clustering may be useful for solving this problem with an unlabelled dataset



Why did we opt for Deep Learning over other models?

- Use case for this model was to build a tool to assist financial analysts to screen the data to see if they have assessed credit rating of companies well or poorly.
- The dataset we were working with is relatively complex therefore deep learning was easier to predict the accuracy of the model.
- Minimal maintenance required once model is set up.
- Quality of accuracy likely to increase overtime as more data becomes available.



Baseline Model Accuracy

- Split and scale data set
- Compile, train and evaluate model
- Achieved accuracy of 78.15%

```
In [6]: # Split our preprocessed data into our features and target arrays
y = cleaned_y_df['Rating'].values
y

# declare x variable
X = cleaned_placeholder_df.values
X

# Split the preprocessed data into a training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state = 55)
```

```
In [7]: # Create a StandardScaler instances
scaler = StandardScaler()

# Fit the StandardScaler
X_scaler = scaler.fit(X_train)

# Scale the data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

Compile, Train and Evaluate the Model

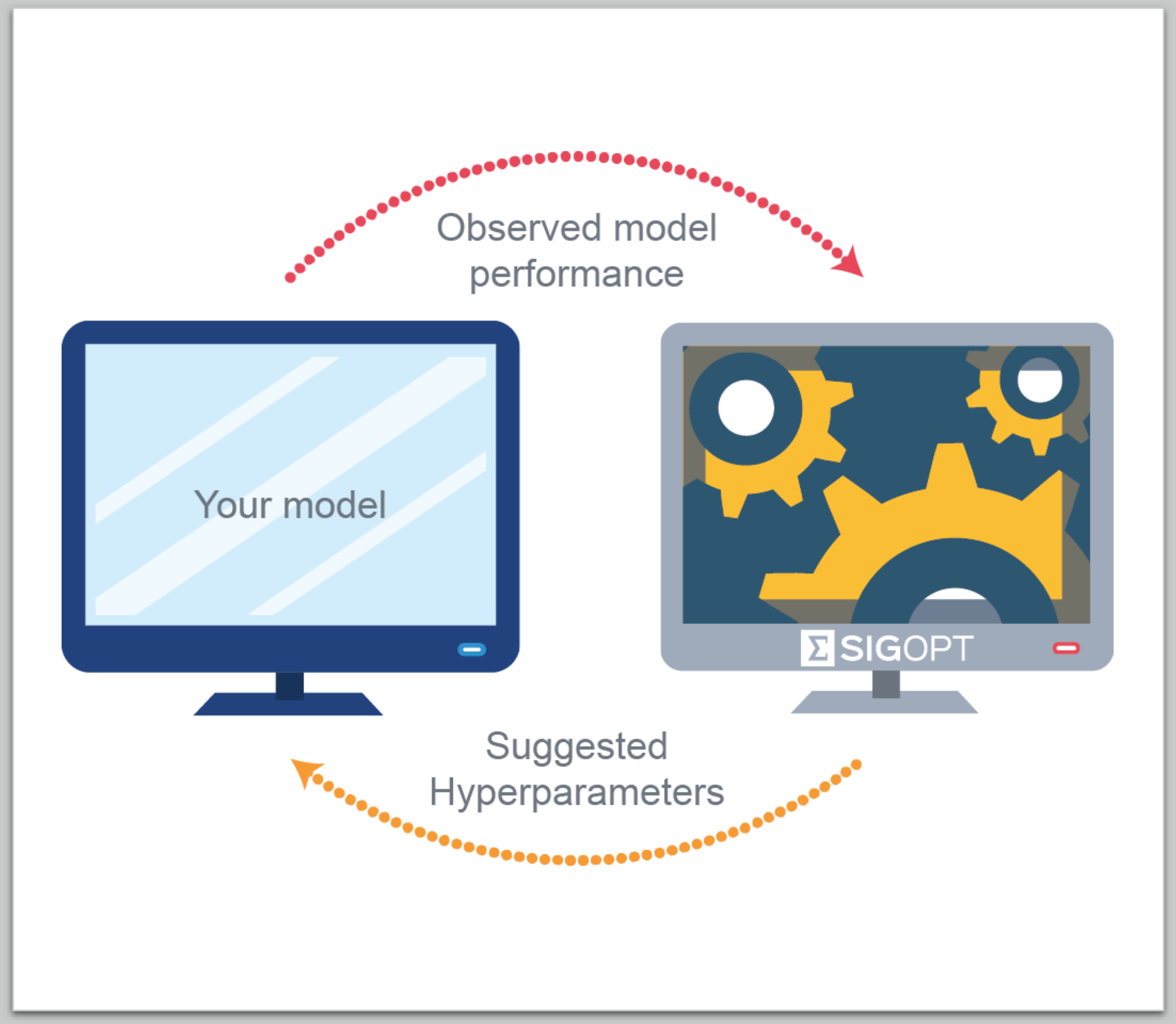
```
In [8]: # Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_layer = len(X_train_scaled[0])
hidden_nodes_L1= 60
hidden_nodes_L2 = 30
hidden_nodes_L3 = 10


nn1 = tf.keras.models.Sequential()

# First hidden layer
nn1.add(tf.keras.layers.Dense(units=hidden_nodes_L1, activation="relu", input_dim=input_layer))

# Second hidden layer
nn1.add(tf.keras.layers.Dense(units=hidden_nodes_L2, activation="relu"))

# Third hidden layer
nn1.add(tf.keras.layers.Dense(units=hidden_nodes_L3, activation="sigmoid"))

# Output layer
nn1.add(tf.keras.layers.Dense(units=1, activation="selu"))

# Check the structure of the model
nn1.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 60)                2400

 dense_1 (Dense)             (None, 30)                1830

 dense_2 (Dense)             (None, 10)                310

 dense_3 (Dense)             (None, 1)                 11
```

```
In [11]: # Evaluate the model using the test data
model_loss, model_accuracy = nn1.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

16/16 - 0s - loss: 0.7596 - accuracy: 0.7815 - 153ms/epoch - 10ms/step
Loss: 0.7596246600151062, Accuracy: 0.7814960479736328
```

```
In [12]: # Export our model to HDF5 file
nn1.save("Baseline_model_Credit_rating.h5")
```

Can hyperparameter tuning increase our models accuracy?

- Hyperparameter tuning is used to search for the optimum set hyperparameters (Number of hidden layers, neurons and learning rate)
- Works by running multiple trials in a single training job. Each trial is a complete implementation of your model with values for your chosen hyperparameters.



Optimized model- Hyperparameter tuning

- Set up the build_model function which is the model builder function that creates, compiles, and returns a neural network model.
- Obtain the best model with those hyperparameters using the get_best_models method of the tuner instance.

```
In [27]: import kerastuner as kt
         msle = MeanSquaredLogarithmicError()


         def build_model(hp):
           model = tf.keras.Sequential()

           # Tune the number of units in the first Dense layer
           # Choose an optimal value between 32-512
           hp_units1 = hp.Int('units1', min_value=32, max_value=512, step=32)
           hp_units2 = hp.Int('units2', min_value=32, max_value=512, step=32)
           hp_units3 = hp.Int('units3', min_value=32, max_value=512, step=32)
           model.add(Dense(units=hp_units1, activation='relu'))
           model.add(tf.keras.layers.Dense(units=hp_units2, activation='relu'))
           model.add(tf.keras.layers.Dense(units=hp_units3, activation='sigmoid'))
           model.add(Dense(1, kernel_initializer='normal', activation='selu'))

           # Tune the learning rate for the optimizer
           # Choose an optimal value from 0.01, 0.001, or 0.0001
           hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

           model.compile(
               optimizer=tf.keras.optimizers.Adam(learning_rate=hp_learning_rate),
               loss=msle,
               metrics=[msle]
           )

           return model

         # HyperBand algorithm from keras tuner
         tuner = kt.Hyperband(
             build_model,
             objective='val_mean_squared_logarithmic_error',
             max_epochs=10,
             directory='keras_tuner_dir',
             project_name='keras_tuner_demo'
         )

         tuner.search(X_train_scaled, y_train, epochs=10, validation_split=0.2)
```

```
In [28]: for h_param in [f"units{i}" for i in range(1,4)] + ['learning_rate']:
           print(h_param, tuner.get_best_hyperparameters()[0].get(h_param))

units1 64
units2 288
units3 128
learning_rate 0.001
```

```
In [29]: best_model = tuner.get_best_models()[0]
         best_model.build(X_train_scaled.shape)
         best_model.summary()

WARNING:tensorflow:Value in checkpoint could not be found in the restored object: (root).optimizer._variables.15
WARNING:tensorflow:Value in checkpoint could not be found in the restored object: (root).optimizer._variables.16
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (1521, 64)                2560

 dense_1 (Dense)             (1521, 288)               18720

 dense_2 (Dense)             (1521, 128)               36992

 dense_3 (Dense)             (1521, 1)                 129

=================================================================
Total params: 58,401
Trainable params: 58,401
Non-trainable params: 0
_________________________________________________________________
```

Applying best hyperparameters to achieve the accuracy

Hyperparameter tuning increased the accuracy of our model from 78.18% to 79.92%

```
In [30]: # Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_layer = len(X_train_scaled[0])
hidden_nodes_L1= 64
hidden_nodes_L2 = 288
hidden_nodes_L3 = 128


nn2 = tf.keras.models.Sequential()

# First hidden layer
nn2.add(tf.keras.layers.Dense(units=hidden_nodes_L1, activation="relu", input_dim=input_layer))

# Second hidden layer
nn2.add(tf.keras.layers.Dense(units=hidden_nodes_L2, activation="relu"))

# Third hidden layer
nn2.add(tf.keras.layers.Dense(units=hidden_nodes_L3, activation="sigmoid"))

# Output layer
nn2.add(tf.keras.layers.Dense(units=1, activation="selu"))

# Check the structure of the model
nn2.summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 64)                2560

 dense_5 (Dense)             (None, 288)               18720

 dense_6 (Dense)             (None, 128)               36992

 dense_7 (Dense)             (None, 1)                 129

=================================================================
Total params: 58,401
Trainable params: 58,401
```

```
In [33]: # Evaluate the model using the test data
model_loss, model_accuracy = nn2.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

16/16 - 0s - loss: 0.7746 - accuracy: 0.7992 - 136ms/epoch - 9ms/step
Loss: 0.7746039628982544, Accuracy: 0.7992125749588013

In [34]: # Export our model to HDF5 file
nn2.save("Optimized_Credit_rating.h5")
```

A More Specific Classifier?

After assessing performance as a binary classifier, we chose to test it on each individual credit rating

Very quickly reaches excellent accuracy on training set, but poor performance on testing

Overfits due to small sample size for some categories

```
Epoch 1/3
48/48 [==============================] - 0s 1ms/step - loss: 0.0400 - accuracy: 0.9244
Epoch 2/3
48/48 [==============================] - 0s 1ms/step - loss: 0.0342 - accuracy: 0.9408
Epoch 3/3
48/48 [==============================] - 0s 1ms/step - loss: 0.0347 - accuracy: 0.9402
```

```
16/16 - 0s - loss: 0.4565 - accuracy: 0.5276 - 23ms/epoch - 1ms/step
Loss: 0.4564847946166992, Accuracy: 0.5275590419769287
```

Summary

- Successful prototype of a rating assessment network with >75% classification accuracy
- Only useful for screening or confirming – not a placement for skilled humans
- A better, more useful product can likely be created with access to more complete data