SCREENSHOTS

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.python.framework import ops
from tensorflow.python.framework import tensor_shape
from tensorflow.python.framework import tensor_util
from tensorflow.python.ops import math_ops
from tensorflow.python.ops import random_ops
from tensorflow.python.ops import array_ops
from tensorflow.python.layers import utils
from collections import namedtuple

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import KFold
import time
import operator

print(tf.__version__)
```

```
1.13.1
```

```
train=pd.read_csv("train.csv")
test=pd.read_csv("test.csv")
```

```
train.head()
```

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1	0	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	0	0	0	0

5 rows × 378 columns

```
test.head()
```

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	0	0	0	0
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	0	0	0	0
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	0	0	0	0
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	0	0	0	0
4	5	w	s	as	c	d	y	i	m	0	...	1	0	0	0	0	0	0	0	0	0

5 rows × 377 columns

```
print(train.shape)
print(test.shape)
```

```
(4209, 378)
(4209, 377)
```

```
train.describe()
```

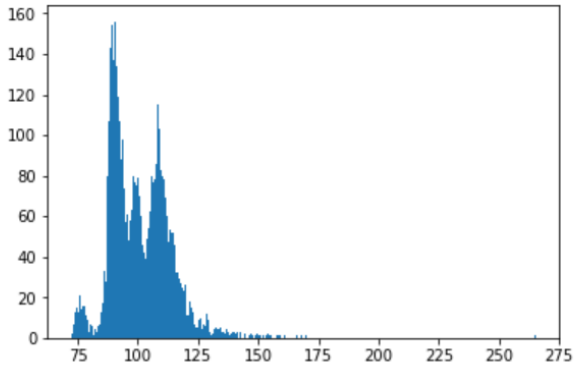
	ID	y	X10	X11	X12	X13	X14	X15	X16	
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.428130	0.000475	0.002613	0.
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867	0.021796	0.051061	0.
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.000000	0.000000	0.000000	0.
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.

8 rows × 370 columns

[7]:
```
unh_features=[]
for fea in train:
    if max(train[fea])==min(train[fea]):
        print(fea)
        unh_features.append(fea)
```

```
X11
X93
X107
X233
X235
X268
X289
X290
X293
X297
X330
X347
```

[8]:
```
plt.hist(train.y,bins=300)
plt.show()
```



]:
```
train[train.y >= 170]
```

]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
883	1770	265.32	y	r	ai	f	d	ag	l	t	...	0	0	0	0	0	0	0	0	0	0

[10]:
```
train=train[train.y < 170]
y=train.y
train=train.drop('y',1)
df=pd.concat([train,test])
df=df.drop(unh_features,1)
```

[11]:
```
df.shape
```

[11]:
```
(8417, 365)
```

[12]:
```
dummies=[]
for col in df:
    if max(df[col]) != 1:
        print(col)
        dummies.append(col)
print(dummies)
```

```
ID
X0
X1
X2
```

```
X2
X3
X4
X5
X6
X8
['ID', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

[13]:
```
dummies=dummies[1:]
```

[14]:
```
for fea in dummies:
    dummy_fea=pd.get_dummies(df[fea], prefix=fea)
    for dummy in dummy_fea:
        df[dummy] = dummy_fea[dummy]
    df = df.drop([fea], 1)
df = df.drop(['ID'],1)
```

[15]:
```
df.shape
```

[15]:
```
(8417, 567)
```

[16]:
```
df.head()
```

[16]:

	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20	...	X8_p	X8_q	X8_r	X8_s	X8_t	X8_u	X8_v	X8_w	X8_x	X8_y
0	0	0	1	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 567 columns

[17]:
```
trainFinal=df[:len(train)]
testFinal=df[len(train):]
yFinal=pd.DataFrame(y)
```

[18]:
```
def create_weights_biases(num_layers, n_inputs, multiplier, max_nodes):
    '''Use the inputs to create the weights and biases for a network'''

    weights = {}
    biases = {}

    for layer in range(1,num_layers):
        if layer == 1:
            weights["h"+str(layer)] = tf.Variable(tf.random_normal([num_features, n_inputs],
                                                                   stddev=np.sqrt(1/num_features)))
            biases["b"+str(layer)] = tf.Variable(tf.random_normal([n_inputs],stddev=0))
            n_previous = n_inputs

        else:
            n_current = int(n_previous * multiplier)

            if n_current >= max_nodes:
                n_current = max_nodes

            weights["h"+str(layer)] = tf.Variable(tf.random_normal([n_previous, n_current],
                                                                   stddev=np.sqrt(1/n_previous)))
            biases["b"+str(layer)] = tf.Variable(tf.random_normal([n_current],stddev=0))
            n_previous = n_current
```

```
    n_current = int(n_previous * multiplier)
    if n_current >= max_nodes:
        n_current = max_nodes

    weights["out"] = tf.Variable(tf.random_normal([n_previous, 1], stddev=np.sqrt(1/n_previous)))
    biases["out"] = tf.Variable(tf.random_normal([1],stddev=0))

    return weights, biases
```

[19]:
```
def network(num_layers, n_inputs, weights, biases, rate, is_training, activation_function):
    '''Add the required number of layers to the network'''

    for layer in range(1, num_layers):
        if layer == 1:
            current_layer = eval(activation_function + "(tf.matmul(n_inputs, weights['h1']) + biases['b1'
            current_layer = tf.nn.dropout(current_layer, 1-rate)
            previous_layer = current_layer
        else:
            current_layer = eval(activation_function + "(tf.matmul(previous_layer,\
            weights['h'+str(layer)]) + biases['b'+str(layer)])")
            current_layer = tf.nn.dropout(current_layer, 1-rate)
            previous_layer = current_layer
    out_layer = tf.matmul(previous_layer, weights['out']) + biases['out']
    return out_layer
```

[20]:
```
def model_inputs():
    '''Create placeholders for model's inputs '''

    inputs = tf.placeholder(tf.float32, [None, None], name='inputs')
    targets = tf.placeholder(tf.float32, [None, 1], name='targets')
    learning_rate = tf.placeholder(tf.float32, name='learning_rate')
    dropout_rate = tf.placeholder(tf.float32, name='dropout_rate')
    is_training = tf.placeholder(tf.bool, name='is_training')

    return inputs, targets, learning_rate, dropout_rate, is_training
```

[21]:
```
def build_graph(num_layers,n_inputs,weights_multiplier,dropout_rate,learning_rate,max_nodes,activation_fu
    '''Use inputs to build the graph and export the required features for training'''

    tf.reset_default_graph()

    inputs, targets, learning_rate, dropout_rate, is_training = model_inputs()

    weights, biases = create_weights_biases(num_layers, n_inputs, weights_multiplier, max_nodes)

    preds = network(num_layers, inputs, weights, biases, dropout_rate, is_training, activation_function)

    with tf.name_scope("cost"):
        cost = tf.losses.mean_squared_error(labels=targets, predictions=preds)
        tf.summary.scalar('cost', cost)

    with tf.name_scope("optimze"):
        optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

    merged = tf.summary.merge_all()

    export_nodes = ['inputs','targets','dropout_rate','is_training','cost','preds','merged',
                    'optimizer','learning_rate']
    Graph = namedtuple('Graph', export_nodes)
    local_dict = locals()
    graph = Graph(*[local_dict[each] for each in export_nodes])

    return graph
```

[22]:
```
def train(model, epochs, log_string, learning_rate):
    '''Train the Network and return the average MSE for each iteration of the model'''

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        testing_loss_summary = []

        iteration = 0
        stop_early = 0
        stop = 10

        learning_rate_decay_threshold = np.random.choice([2,3,4,5])
        n_splits = 5
        original_learning_rate = learning_rate

        print()
        print("Training Model: {}".format(log_string))

        train_writer = tf.summary.FileWriter('./logs/1/train/{}'.format(log_string), sess.graph)
        test_writer = tf.summary.FileWriter('./logs/1/test/{}'.format(log_string))

      kf = KFold(n_splits=n_splits, shuffle=True, random_state=2)
      split = 0
      sum_loss_testing = 0

      for train_index, test_index in kf.split(trainFinal):

          x_train = trainFinal.iloc[train_index]
          y_train = yFinal.iloc[train_index]
          x_test = trainFinal.iloc[test_index]
          y_test = yFinal.iloc[test_index]

          training_check = (len(x_train)//batch_size)-1 # Check training progress after this many batche
          testing_check = training_check # Check testing results

          split += 1
          print('Start KFold number {} from {}'.format(split, n_splits))

          for epoch_i in range(1, epochs+1):
              batch_loss = 0
              batch_time = 0

             for batch in range(int(len(x_train)/batch_size)):
                 batch_x = x_train[batch*batch_size:(1+batch)*batch_size]
                 batch_y = y_train[batch*batch_size:(1+batch)*batch_size]

                 start_time = time.time()

                 summary, loss, _ = sess.run([model.merged,
                                              model.cost,
                                              model.optimizer],
                                              {model.inputs: batch_x,
                                               model.targets: batch_y,
                                               model.learning_rate: learning_rate,
                                               model.dropout_rate: dropout_rate,
                                               model.is_training: True})


                 batch_loss += loss
                 end_time = time.time()
                 batch_time += end_time - start_time

                 train_writer.add_summary(summary, iteration)
```

```
                iteration += 1

                if batch % training_check == 0 and batch > 0:
                    print('Epoch {:>3}/{} Batch {:>4}/{} - MSE: {:>6.3f}, Seconds: {:>4.2f}'
                          .format(epoch_i,
                                  epochs,
                                  batch,
                                  len(x_train) // batch_size,
                                  (batch_loss / training_check),
                                  batch_time))
                    batch_loss = 0
                    batch_time = 0

                if batch % testing_check == 0 and batch > 0:
                    batch_loss_testing = 0
                    batch_time_testing = 0
                    for batch in range(int(len(x_test)/batch_size)):
                        batch_x = x_test[batch*batch_size:(1+batch)*batch_size]
                        batch_y = y_test[batch*batch_size:(1+batch)*batch_size]

                        start_time_testing = time.time()
                          summary, loss = sess.run([model.merged,
                                                    model.cost],
                                                       {model.inputs: batch_x,
                                                        model.targets: batch_y,
                                                        model.learning_rate: learning_rate,
                                                        model.dropout_rate: 0,
                                                        model.is_training: False})

                          batch_loss_testing += loss
                          end_time_testing = time.time()
                          batch_time_testing += end_time_testing - start_time_testing

                          test_writer.add_summary(summary, iteration)

                       n_batches_testing = batch + 1
                       print('Testing MSE: {:>6.3f}, Seconds: {:>4.2f}'
                             .format(batch_loss_testing / n_batches_testing,
                                     batch_time_testing))

                       batch_time_testing = 0

                       testing_loss_summary.append(batch_loss_testing)
                       if batch_loss_testing <= min(testing_loss_summary):
                           print('New Record!')
                           lowest_loss_testing = batch_loss_testing/n_batches_testing
                           stop_early = 0 # Reset stop_early if new minimum loss is found
                           checkpoint = "./{}.ckpt".format(log_string)
                           saver = tf.train.Saver()
                           saver.save(sess, checkpoint)

                       else:
                           print("No Improvement.")
                           stop_early += 1 # Increase stop_early if no new minimum loss is found
                           if stop_early % learning_rate_decay_threshold == 0:
                               learning_rate *= learning_rate_decay
                               print("New learning rate = ", learning_rate)
                           elif stop_early == stop:
                               break

            if stop_early == stop:
                print("Stopping training for this fold.")
                print("Lowest MSE =", lowest_loss_testing)
                print()
                sum_loss_testing += lowest_loss_testing
```

```
                    early_stop = 0
                    testing_loss_summary = []
                    learning_rate = original_learning_rate
                    break

        average_testing_loss = sum_loss_testing/n_splits
        print("Stopping training for this iteration.")
        print("Average MSE for this iteration: ", average_testing_loss)
        print()

    return average_testing_loss
```

[23]:
```
num_iterations = 25
results = {}
for i in range(num_iterations):
    num_features = trainFinal.shape[1]
    epochs = 50
    learning_rate = np.random.uniform(0.001, 0.1)
    learning_rate_decay = np.random.uniform(0.1,0.5)
    weights_multiplier = np.random.uniform(0.5,2)
    n_inputs = np.random.randint(int(num_features)*0.1,int(num_features)*2)
    num_layers = np.random.choice([2,3,4])
    dropout_rate = np.random.uniform(0,0.3)
    batch_size = np.random.choice([64,128,256])
    max_nodes = np.random.choice([32,64,128,256,512,1024,2048,4096])
    activation_function = np.random.choice(['tf.nn.sigmoid',
                                            'tf.nn.relu',
                                            'tf.nn.elu'])

    print("Starting iteration #",i+1)
    log_string = 'LR={},LRD={},WM={},NI={},NL={},DR={},BS={},MN={},AF={}'.format(learning_rate,
                                                                                learning_rate_decay,
                                                                                weights_multiplier,
                                                                                n_inputs,
                                                                                num_layers,
                                                                                dropout_rate,
                                                                                batch_size,
                                                                                max_nodes,
                                                                                activation_function)

    model = build_graph(num_layers, n_inputs, weights_multiplier,
                        dropout_rate,learning_rate,max_nodes,activation_function)
    result = train(model, epochs, log_string, learning_rate)
     results[log_string] = result
```

```
Starting iteration # 1
WARNING:tensorflow:From /opt/anaconda3/lib/python3.7/site-packages/tensorflow/python/framework/op_def_lib
rary.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From <ipython-input-19-f1d0724fa693>:7: calling dropout (from tensorflow.python.ops.nn
_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
WARNING:tensorflow:From /opt/anaconda3/lib/python3.7/site-packages/tensorflow/python/ops/losses/losses_im
pl.py:667: to_float (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future v
ersion.
Instructions for updating:
Use tf.cast instead.

Training Model: LR=0.004890773749952225,LRD=0.35431805771808944,WM=1.5946404035599373,NI=756,NL=4,DR=0.13
858139670331282,BS=128,MN=1024,AF=tf.nn.sigmoid
Start KFold number 1 from 5
Epoch   1/50 Batch   25/26 - MSE: 3972.145, Seconds: 1.36
```

```
No improvement.
Stopping training for this fold.
Lowest MSE = 62.24092395489033

Stopping training for this iteration.
Average MSE for this iteration:  59.9557729280912
```

[24]:
```
def find_inputs(model):
    '''Use the log_string from the model to extract the values for all of the model's inputs'''

    learning_rate_start = model.find('LR=') + 3
    learning_rate_end = model.find(',LRD', learning_rate_start)
    learning_rate = float(model[learning_rate_start:learning_rate_end])

    learning_rate_decay_start = model.find('LRD=') + 4
    learning_rate_decay_end = model.find(',WM', learning_rate_decay_start)
    learning_rate_decay = float(model[learning_rate_decay_start:learning_rate_decay_end])

    weights_multiplier_start = model.find('WM=') + 3
    weights_multiplier_end = model.find(',NI', weights_multiplier_start)
    weights_multiplier = float(model[weights_multiplier_start:weights_multiplier_end])

    n_inputs_start = model.find('NI=') + 3
    n_inputs_end = model.find(',NL', n_inputs_start)
    n_inputs = int(model[n_inputs_start:n_inputs_end])

    num_layers_start = model.find('NL=') + 3
    num_layers_end = model.find(',DR', num_layers_start)
    num_layers = int(model[num_layers_start:num_layers_end])

    dropout_rate_start = model.find('DR=') + 3
    dropout_rate_end = model.find(',BS', dropout_rate_start)
    dropout_rate = float(model[dropout_rate_start:dropout_rate_end])

    batch_size_start = model.find('BS=') + 3
    batch_size_end = model.find(',MN', batch_size_start)
    batch_size = int(model[batch_size_start:batch_size_end])

    max_nodes_start = model.find('MN=') + 3
    max_nodes_end = model.find(',AF', max_nodes_start)
    max_nodes = int(model[max_nodes_start:max_nodes_end])

    activation_function_start = model.find('AF=') + 3
    activation_function = str(model[activation_function_start:])

    return (learning_rate, learning_rate_decay, weights_multiplier, n_inputs,
            num_layers, dropout_rate, batch_size, max_nodes, activation_function)
```

[25]:
```
sorted_results = sorted(results.items(), key=operator.itemgetter(1))
```

[26]:
```
results_df = pd.DataFrame(columns=["learning_rate",
                                   "learning_rate_decay",
                                   "weights_multiplier",
                                   "n_inputs",
                                   "num_layers",
                                   "dropout_rate",
                                   "batch_size",
                                   "max_nodes",
                                   "activation_function"])

for result in sorted_results:
    learning_rate, learning_rate_decay, weights_multiplier, n_inputs,\
        num_layers, dropout_rate, batch_size, max_nodes, activation_function = find_inputs(result[0])
```

```
MSE = result[1]

new_row = pd.DataFrame([[MSE,
                         learning_rate,
                         learning_rate_decay,
                         weights_multiplier,
                         n_inputs,
                         num_layers,
                         dropout_rate,
                         batch_size,
                         max_nodes,
                         activation_function]],
                       columns = ["MSE",
                                  "learning_rate",
                                  "learning_rate_decay",
                                  "weights_multiplier",
                                  "n_inputs",
                                  "num_layers",
                                  "dropout_rate",
                                  "batch_size",
                                  "max_nodes",
                                  "activation_function"])
```

```
    results_df = results_df.append(new_row, ignore_index=True,sort=False)
```

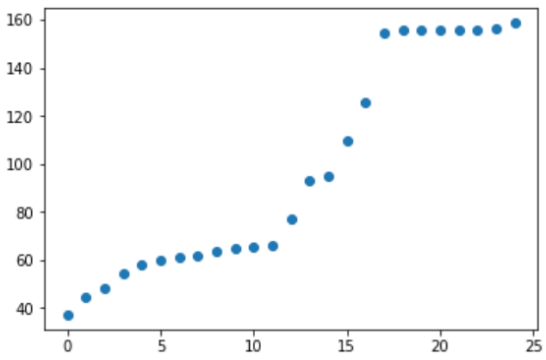
[27]:
```
results_df.head()
```

[27]:

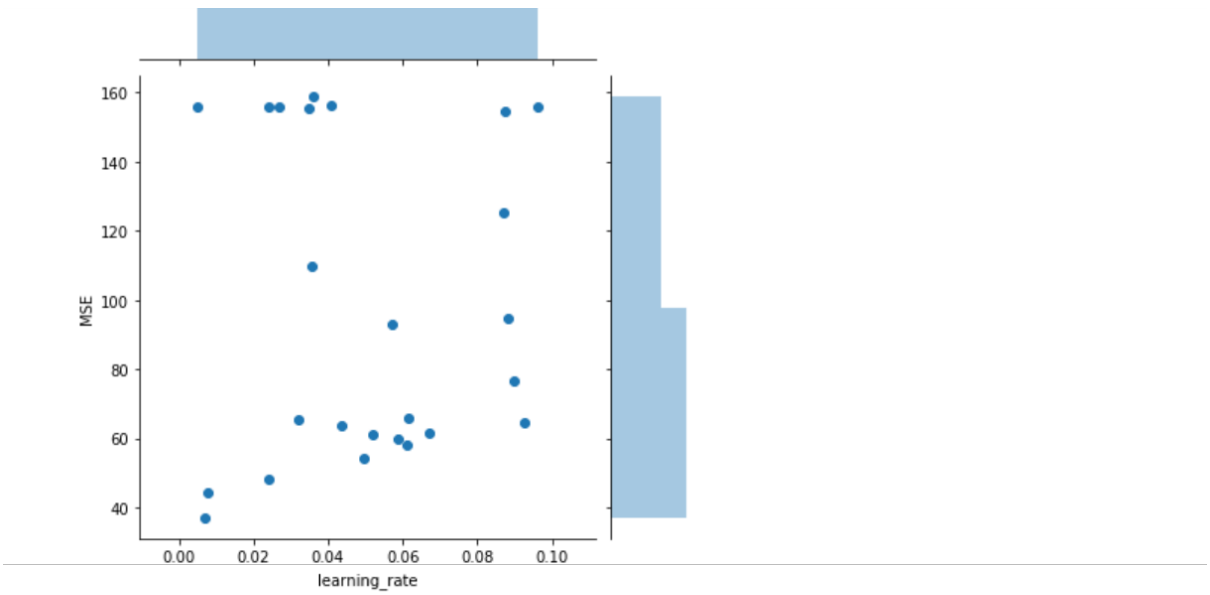
	learning_rate	learning_rate_decay	weights_multiplier	n_inputs	num_layers	dropout_rate	batch_size	max_nodes	activati
0	0.006848	0.175169	0.505300	1032	2	0.217217	128	1024	
1	0.007817	0.409927	1.509780	271	2	0.133306	256	1024	
2	0.023933	0.230210	0.643985	295	2	0.136398	256	256	
3	0.049629	0.171783	1.384525	888	2	0.144926	256	2048	
4	0.061219	0.382154	0.746649	112	3	0.097582	128	128	

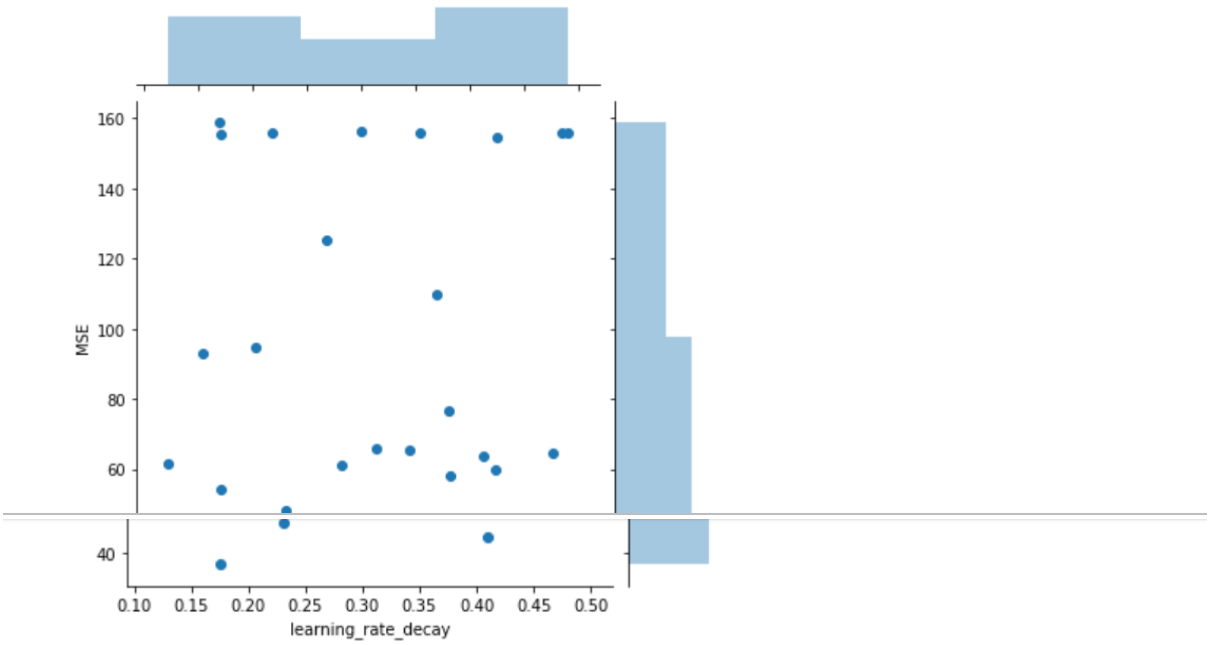
[28]:
```
plt.scatter(results_df.index, results_df.MSE)
```

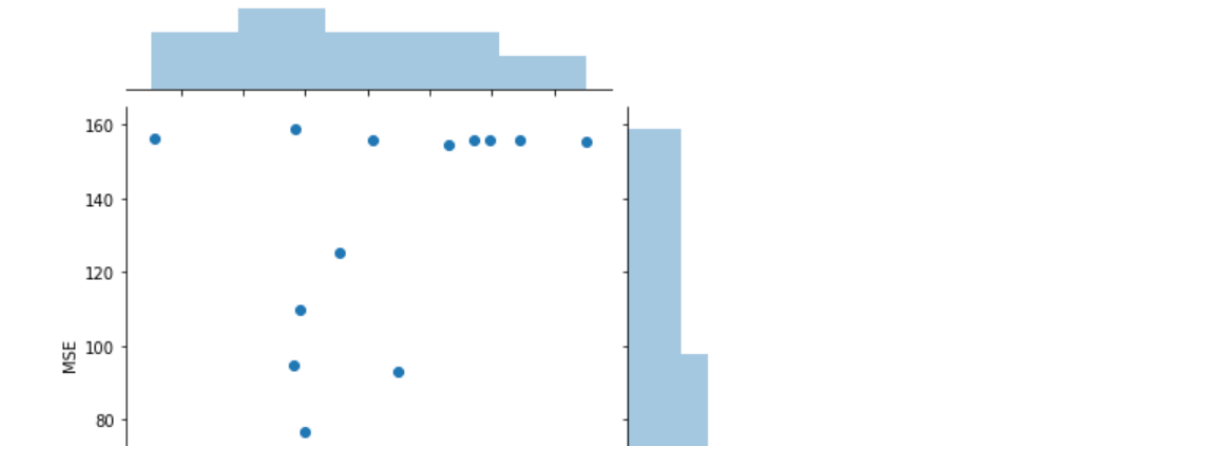
[28]:
```
<matplotlib.collections.PathCollection at 0x7f6714197710>
```

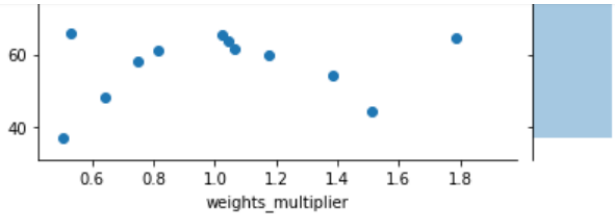


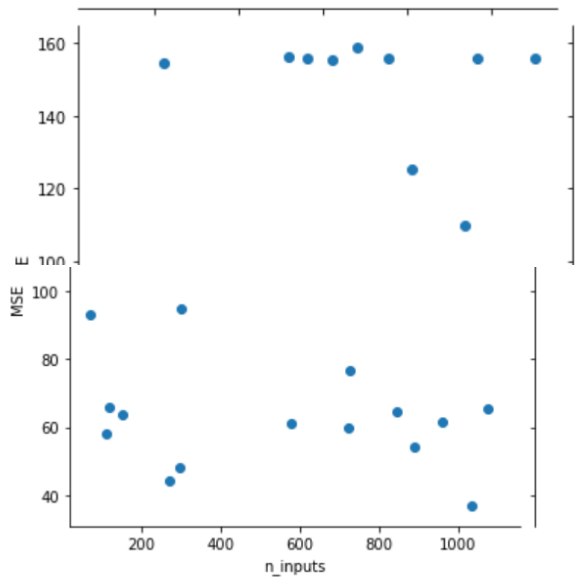
[29]:
```
import seaborn as sns
for feature in results_df:
    if feature == "MSE":
        continue
    elif feature == "activation_function":
        sns.stripplot(x=feature, y="MSE", data=results_df)
        sns.stripplot(x=feature, y="MSE", data=results_df)
    else:
        sns.jointplot(x=feature, y="MSE", data=results_df)
```


MSE
160
140
120
100
80
60
40
0.00
0.02
0.04
0.06
0.08
0.10
learning_rate


MSE
160
140
120
100
80
60
40
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
learning_rate_decay


MSE
160
140
120
100
80





[30]:
```
best_models = [] # contains the log_strings of the best iterations, to be used for the final predictions
best_R2 = 0 # records the best R2 score
best_predictions = pd.DataFrame([0]*len(trainFinal)) # records the best predictions for each row
current_model = 1 # Used to equally weight the predictions from each iteration
testing_limit = 3 # If 3 consectutive iterations do not improve the best R2, stop predicting

for model, result in sorted_results:
    checkpoint = str(model) + ".ckpt"
```

```
    _, _, weights_multiplier, n_inputs, num_layers, _, _, max_nodes, activation_function = find_inputs(mod

    model = build_graph(num_layers,n_inputs,weights_multiplier,dropout_rate,
                        learning_rate,max_nodes,activation_function)

    # Predict one row at a time
    batch_size = 1

    with tf.Session() as sess:
        saver = tf.train.Saver()
        saver.restore(sess, checkpoint)
        predictions = [] # record the predictions

        for batch in range(int(len(trainFinal)/batch_size)):
            batch_x = trainFinal[batch*batch_size:(1+batch)*batch_size]

            batch_predictions = sess.run([model.preds],
                                         {model.inputs: batch_x,
                                          model.learning_rate: learning_rate,
                                          model.dropout_rate: 0,
                                          model.is_training: False})
```

```
            for prediction in batch_predictions[0]:
                predictions.append(prediction)

    predictions = pd.DataFrame(predictions)

    R2 = r2_score(y, predictions)
    print("R2 Score = ", R2)

    # Equally weight each prediction
    combined_predictions = (best_predictions*(current_model-1) + predictions) / current_model

    # Find the r2 score with the new predictions
    new_R2 = r2_score(y, combined_predictions)
    print("New R2 score = ", new_R2)

    if new_R2 >= best_R2:
        best_predictions = combined_predictions
        best_R2 = new_R2
        best_models.append(checkpoint)
        limit = 0
        current_model += 1
        print("Improvement!")
        print()
    else:
        print("No improvement.")
        limit += 1
        if limit == testing_limit:
            print("Stopping predictions.")
            break
```

```
WARNING:tensorflow:From /opt/anaconda3/lib/python3.7/site-packages/tensorflow/python/training/saver.py:126
6: checkpoint_exists (from tensorflow.python.training.checkpoint_management) is deprecated and will be rem
oved in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from LR=0.006840379898597865,LRD=0.17516904929685626,WM=0.5053003614
547112,NI=1032,NL=2,DR=0.21721676724454952,BS=128,MN=1024,AF=tf.nn.relu.ckpt
R2 Score =  0.6289213546775874
New R2 score =  0.6289213546775874
Improvement!

INFO:tensorflow:Restoring parameters from LR=0.00781732390082656,LRD=0.40992728225770814,WM=1.509779640465
8745,NI=271,NL=2,DR=0.13330576797679408,BS=256,MN=1024,AF=tf.nn.sigmoid.ckpt
R2 Score =  0.6660128099903835

INFO:tensorflow:Restoring parameters from LR=0.0518475775948857,LRD=0.2823241657182445,WM=0.814038722518321
5,NI=576,NL=2,DR=0.2495012193140341,BS=256,MN=512,AF=tf.nn.sigmoid.ckpt
R2 Score =  0.6627405909664613
New R2 score =  0.6978602810391386
No improvement.
INFO:tensorflow:Restoring parameters from LR=0.06721761689863513,LRD=0.12307521561981148,WM=1.065323196195
3413,NI=958,NL=2,DR=0.15110124137972852,BS=128,MN=64,AF=tf.nn.sigmoid.ckpt
R2 Score =  0.6272154307884952
New R2 score =  0.6962118421201016
No improvement.
INFO:tensorflow:Restoring parameters from LR=0.043469772168736924,LRD=0.41208299100043333,WM=1.04327522493
63166,NI=150,NL=2,DR=0.06527166637721249,BS=128,MN=128,AF=tf.nn.elu.ckpt
R2 Score =  0.6586452299697911
New R2 score =  0.6954826550351285
No improvement.
Stopping predictions.
```

[31]:
```
best_models
```

[31]:
```
['LR=0.0068480379898597865,LRD=0.17516904929685626,WM=0.5053003614547112,NI=1032,NL=2,DR=0.217216767244549
52,BS=128,MN=1024,AF=tf.nn.relu.ckpt',
 'LR=0.00781732390082656,LRD=0.40992728225770814,WM=1.5097796404658745,NI=271,NL=2,DR=0.13330576797679408,
BS=256,MN=1024,AF=tf.nn.sigmoid.ckpt',
 'LR=0.049629404605042365,LRD=0.1717828003094657,WM=1.3845247620511545,NI=888,NL=2,DR=0.14492580384481166,
BS=256,MN=2048,AF=tf.nn.sigmoid.ckpt',
 'LR=0.06121919712054469,LRD=0.3821542267514719,WM=0.7466492797403004,NI=112,NL=3,DR=0.09758179510246075,B
S=128,MN=128,AF=tf.nn.elu.ckpt',
 'LR=0.05890537282055379,LRD=0.42399730663735824,WM=1.1743309126467907,NI=723,NL=2,DR=0.1252452256987556,B
S=64,MN=2048,AF=tf.nn.elu.ckpt']
```

[32]:
```
best_predictions = pd.DataFrame([0]*len(testFinal))
current_model = 1

for model in best_models:
    checkpoint = model

    _, _, weights_multiplier, n_inputs, num_layers, _, _, max_nodes, activation_function = find_inputs(mod

    # Remove '.ckpt' from the activation_function string
    activation_function = activation_function[:activation_function.find('.ckpt')]

    model = build_graph(num_layers,n_inputs,weights_multiplier,dropout_rate,
                        learning_rate,max_nodes,activation_function)

    batch_size = 1

    with tf.Session() as sess:
        saver = tf.train.Saver()
        saver.restore(sess, checkpoint)
        predictions = []

        for batch in range(int(len(testFinal)/batch_size)):
            batch_x = testFinal[batch*batch_size:(1+batch)*batch_size]

            batch_predictions = sess.run([model.preds],
                                         {model.inputs: batch_x,
                                          model.learning_rate: learning_rate,
                                          model.dropout_rate: 0,
                                          model.is_training: False})

            for prediction in batch_predictions[0]:
                predictions.append(prediction)

    predictions = pd.DataFrame(predictions)

    combined_predictions = (best_predictions*(current_model-1) + predictions) / current_model
    best_predictions = combined_predictions
    current_model += 1
```

```
INFO:tensorflow:Restoring parameters from LR=0.0068480379898597865,LRD=0.17516904929685626,WM=0.5053003614
547112,NI=1032,NL=2,DR=0.21721676724454952,BS=128,MN=1024,AF=tf.nn.relu.ckpt
INFO:tensorflow:Restoring parameters from LR=0.00781732390082656,LRD=0.40992728225770814,WM=1.509779640465
8745,NI=271,NL=2,DR=0.13330576797679408,BS=256,MN=1024,AF=tf.nn.sigmoid.ckpt
INFO:tensorflow:Restoring parameters from LR=0.049629404605042365,LRD=0.1717828003094657,WM=1.384524762051
1545,NI=888,NL=2,DR=0.14492580384481166,BS=256,MN=2048,AF=tf.nn.sigmoid.ckpt
INFO:tensorflow:Restoring parameters from LR=0.06121919712054469,LRD=0.3821542267514719,WM=0.7466492797403
004,NI=112,NL=3,DR=0.09758179510246075,BS=128,MN=128,AF=tf.nn.elu.ckpt
INFO:tensorflow:Restoring parameters from LR=0.05890537282055379,LRD=0.42399730663735824,WM=1.174330912646
7907,NI=723,NL=2,DR=0.1252452256987556,BS=64,MN=2048,AF=tf.nn.elu.ckpt
```

[33]:
```
best_predictions['ID'] = test.ID
best_predictions['y'] = best_predictions[0]
best_predictions = best_predictions.drop([0],1)
```

```
best_predictions.to_csv("submission.csv", index=False)
```

[34]:
```
best_predictions.head()
```

[34]:

	ID	y
0	1	78.582414
1	2	96.226047
2	3	81.614616
3	4	76.747839
4	5	105.431487

[35]:
```
best_predictions.describe()
```

[35]:

	ID	y
count	4209.000000	4209.000000
mean	4211.039202	100.324548
std	2423.078926	9.867065
min	1.000000	73.410442
25%	2115.000000	92.836122
50%	4202.000000	96.913811
75%	6310.000000	109.961414
max	8416.000000	125.187039

[36]:
```
yFinal.describe()
```

[36]:

	y
count	4208.000000
mean	100.630190
std	12.424146
min	72.110000
25%	90.817500
50%	99.150000
75%	109.010000
max	169.910000