```
pip install psycopg2-binary
```

```
    Collecting psycopg2-binary
      Downloading https://files.pythonhosted.org/packages/6d/45/c519a5cfac05e14b1ccb
         |████████████████████████████████| 3.0MB 13.9MB/s
    Installing collected packages: psycopg2-binary
    Successfully installed psycopg2-binary-2.8.6
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,OneHotEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
import tensorflow as tf
import sqlalchemy as db
from sqlalchemy.ext.automap import automap_base
from sklearn.metrics import mean_squared_error
from keras.metrics import RootMeanSquaredError
```

```
engine = db.create_engine('postgresql://postgres:finalproject@finalproject.cfnecwioim
with engine.connect() as conn, conn.begin():
    df = pd.read_sql("""select * from all_cleaned_redfin_data;""", conn)
```

```
ml_df = df
ml_df = ml_df.drop([
    'address',
    'state_or_province',
    #'city',
    'url',
    'mls_number',
    'latitude',
    'longitude',
    'neighborhood',
    'price_per_square_feet'
    ], axis =1)
ml_df
```

	index	property_type	city	zip_code	price	beds	baths	s
0	0	Single Family Residential	Nashville	37215	1425000	4.0	5.0	
1	1	Single Family Residential	Nashville	37220	1975000	5.0	4.5	
2	2	Single Family Residential	Nashville	37215	1100000	4.0	5.0	
3	3	Single Family Residential	Nashville	37215	1600000	4.0	4.0	
4	4	Single Family Residential	Nashville	37215	1775000	4.0	5.0	
...	...	...	...	...	...	...	...	...
5918	340	Single Family Residential	Madison	37115	235950	1.0	1.0	
5919	341	Single Family Residential	Madison	37115	230950	1.0	1.0	
5920	342	Single Family Residential	Madison	37115	200950	1.0	1.0	

age = today - year built zip code price per square feet try scaling everything loss function or roc curve

```
convert_dict = {'zip_code': object,
                'beds': int,
                'square_feet': int,
                'lot_size': int,
                'year_built': object,
               }

ml_df = ml_df.astype(convert_dict)
object1 = ml_df.dtypes[ml_df.dtypes == "object"].index.to_list()
object1.remove('status')
print(object1)
```

```
['property_type', 'city', 'zip_code', 'year_built']
```

```
enc = OneHotEncoder(sparse=False)

encode_df = pd.DataFrame(enc.fit_transform(ml_df[object1]))
encode_df.columns = enc.get_feature_names(object1)
encode_df

ml_encode_df = ml_df.merge(encode_df,left_index=True, right_index=True)
ml_encode_df = ml_encode_df.drop(object1,1)
```

```
ml_encode_df = ml_encode_df.drop(columns=[1])
ml_encode_df

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# Scaling the Price column of the created dataFrame and storing
# the result in ScaledPrice Column
ml_encode_df[["scaledSF", "scaledLS"]] = scaler.fit_transform(ml_encode_df[["square_f
ml_encode_df = ml_encode_df.drop(["square_feet","lot_size"], axis= 1)
ml_encode_df
```

	index	price	beds	baths	status	property_type_Condo/Co-op	p
0	0	1425000	4	5.0	Sold	0.0	
1	1	1975000	5	4.5	Sold	0.0	
2	2	1100000	4	5.0	Sold	0.0	
3	3	1600000	4	4.0	Sold	0.0	
4	4	1775000	4	5.0	Sold	0.0	
...	...	...	...	...	...	...	
5918	340	235950	1	1.0	Active	0.0	
5919	341	230950	1	1.0	Active	0.0	
5920	342	200950	1	1.0	Active	0.0	
5921	343	225950	1	1.0	Active	0.0	
5922	344	195950	1	1.0	Active	0.0	

```
active_df = ml_encode_df.loc[ml_encode_df['status'] == 'Active'].drop(['status'], axi
sold_df = ml_encode_df.loc[ml_encode_df['status'] != 'Active'].drop(['status'], axis=
y_sold, y_active = sold_df["price"].values, active_df["price"].values
X_sold, X_active = sold_df.drop(["price"],1).values, active_df.drop(["price"],1).valu


X_train, X_test, y_train, y_test = train_test_split(X_sold, y_sold, random_state=78)


#Define the model - deep neural net
number_input_features = len(X_train[0])

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(
```

```
nn.add(
    tf.keras.layers.Dense(units=50, input_dim=number_input_features, activation="relu
)
nn.add(
    tf.keras.layers.Dense(units=40, activation="relu")
)

nn.add(
    tf.keras.layers.Dense(units=30, activation="relu")
)
nn.add(
    tf.keras.layers.Dense(units=20, activation="relu")
)

nn.add(
    tf.keras.layers.Dense(units=10, activation="relu")
)
# Second hidden layer
nn.add(tf.keras.layers.Dense(units=8, kernel_initializer='normal'))



# Output layer
nn.add(tf.keras.layers.Dense(units=1))

# Check the structure of the model
nn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 50)                9150
_________________________________________________________________
dense_8 (Dense)              (None, 40)                2040
_________________________________________________________________
dense_9 (Dense)              (None, 30)                1230
_________________________________________________________________
dense_10 (Dense)             (None, 20)                620
_________________________________________________________________
dense_11 (Dense)             (None, 10)                210
_________________________________________________________________
dense_12 (Dense)             (None, 8)                 88
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 9
=================================================================
Total params: 13,347
Trainable params: 13,347
Non-trainable params: 0
_________________________________________________________________
```

```
# def baseline_model():
#   # create model
#   model = Sequential()
```

```
#   model.add(Dense(13, input_dim= len(X_train[0]), kernel_initializer='he_normal', a
#   model.add(Dense(1, kernel_initializer='normal'))
#   # Compile model
#   model.compile(loss='mean_squared_error', optimizer='Adam')
#   return model
# #evaluate the model
# param_grid = {
# #     'optimizer': ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nad
#      'batch_size': [10, 20, 40, 60, 80, 100],
#      'epochs': [100, 200, 300],
# #     'learn_rate': [0.001, 0.01, 0.1, 0.2, 0.3],
# #     'momentum': [0.0, 0.2, 0.4, 0.6, 0.8, 0.9],
# #     'init_mode': ['uniform', 'lecun_uniform', 'normal', 'zero', 'glorot_normal',
# #     'activation': ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid',
# #     'weight_constraint': [1, 2, 3, 4, 5],
# #     'dropout_rate': [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9],
# #     'neurons': [1, 5, 10, 15, 20, 25, 30]

# }

# model = KerasRegressor(build_fn=baseline_model, epochs=100, batch_size=5, verbose=0
# grid_search = GridSearchCV(estimator = model, param_grid = param_grid,
#                       cv = 3, n_jobs = -1, verbose = 2, return_train_score=True)
# grid_result = grid_search.fit(X_train, y_train)
# print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
# means = grid_result.cv_results_['mean_test_score']
# stds = grid_result.cv_results_['std_test_score']
# params = grid_result.cv_results_['params']
# for mean, stdev, param in zip(means, stds, params):
#     print("%f (%f) with: %r" % (mean, stdev, param))



# Compile the model
nn.compile(loss='mse', optimizer="adam", metrics=[tf.keras.metrics.RootMeanSquaredErr


# Train the model
fit_model = nn.fit(X_train,y_train,epochs=300)


prediction = nn.predict(X_test)
pred = pd.DataFrame({ 'actual': y_test})
pred['prediction'] = prediction
pred['error'] =pred.prediction - pred.actual
pred['error_abs'] = abs(pred['error'])
print(pred.error_abs.mean())
print(mean_squared_error(y_test, prediction))
print(mean_squared_error(y_test, prediction, squared=False))
```

```
152953.2560344061
95537408850.99065
```

```
     309091.262980678

# Train the model
fit_model = nn.fit(X_train,y_train,epochs=100)


prediction = nn.predict(X_test)
pred = pd.DataFrame({ 'actual': y_test})
pred['prediction'] = prediction
pred['error'] =pred.prediction - pred.actual
pred['error_abs'] = abs(pred['error'])
print(pred.error_abs.mean())
print(mean_squared_error(y_test, prediction))
print(mean_squared_error(y_test, prediction, squared=False))

     160716.08573746443
     127584436058.66924
     357189.63599000074


# Train the model
fit_model = nn.fit(X_train,y_train,epochs=100)


prediction = nn.predict(X_test)
pred = pd.DataFrame({ 'actual': y_test})
pred['prediction'] = prediction
pred['error'] =pred.prediction - pred.actual
pred['error_abs'] = abs(pred['error'])
print(pred.error_abs.mean())
print(mean_squared_error(y_test, prediction))
print(mean_squared_error(y_test, prediction, squared=False))

     164930.7456325569
     132396939891.98933
     363863.90298020677


# Train the model
fit_model = nn.fit(X_train,y_train,epochs=100)


prediction = nn.predict(X_test)
pred = pd.DataFrame({ 'actual': y_test})
pred['prediction'] = prediction
pred['error'] =pred.prediction - pred.actual
pred['error_abs'] = abs(pred['error'])
print(pred.error_abs.mean())
print(mean_squared_error(y_test, prediction))
print(mean_squared_error(y_test, prediction, squared=False))

     174071.25446746088
     167726804493.731
     409544.6306493725
```

```
# Train the model
fit_model = nn.fit(X_train,y_train,epochs=100)


prediction = nn.predict(X_test)
pred = pd.DataFrame({ 'actual': y_test})
pred['prediction'] = prediction
pred['error'] =pred.prediction - pred.actual
pred['error_abs'] = abs(pred['error'])
print(pred.error_abs.mean())
print(mean_squared_error(y_test, prediction))
print(mean_squared_error(y_test, prediction, squared=False))
```

```
170055.24040384957
145243192951.37732
381107.8495011318
```

```
 #Train the model
fit_model = nn.fit(X_train,y_train,epochs=100)


prediction = nn.predict(X_test)
pred = pd.DataFrame({ 'actual': y_test})
pred['prediction'] = prediction
pred['error'] =pred.prediction - pred.actual
pred['error_abs'] = abs(pred['error'])
print(pred.error_abs.mean())
print(mean_squared_error(y_test, prediction))
print(mean_squared_error(y_test, prediction, squared=False))
```

```
177060.34091616288
154722813603.87906
393348.20910216315
```

```
 #Train the model
fit_model = nn.fit(X_train,y_train,epochs=100)


prediction = nn.predict(X_test)
pred = pd.DataFrame({ 'actual': y_test})
pred['prediction'] = prediction
pred['error'] =pred.prediction - pred.actual
pred['error_abs'] = abs(pred['error'])
print(pred.error_abs.mean())
print(mean_squared_error(y_test, prediction))
print(mean_squared_error(y_test, prediction, squared=False))
```

```
167081.80589326992
132397988068.93181
363865.3433193821
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import numpy as np
active_df = df.loc[df['status'] == 'Active'].drop(['status'], axis=1)
active_df['predicted_price'] = nn.predict(X_active)
active_df['house_value'] = np.where(active_df['price'] > active_df['predicted_price']
active_df.groupby(['house_value']).count()['mls_number']
active_df.to_json('house_value.json')
!cp house_value.json "drive/My Drive/"
```