```
pip install scikeras
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scikeras in /usr/local/lib/python3.8/dist-packages (0.10.0)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.8/dist-packages (from scikeras) (23.0)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.8/dist-packages (from scikeras) (1.0.2)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.7.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=1.0.0->scikeras) (3.1.
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.22.4)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.2.0)
```

```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from scikeras.wrappers import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
```

```
from sklearn.datasets import load_boston
boston = load_boston()
dataset = pd.DataFrame(boston.data)
dataset.shape
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function load_boston is deprecated; `load_boston`

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
(506, 13)
```

```
#Adding the feature names to the dataframe
dataset.columns = boston.feature_names
dataset.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94

```
#Adding target variable to dataframe
dataset['PRICE'] = boston.target
dataset.shape
```

```
(506, 14)
```

```
#dataframe = pd.read_csv("housing.csv", delim_whitespace=True, header=None)
#dataset = boston.values
# split into input (X) and output (Y) variables
X =boston.data# dataset.iloc[0:12]
Y = dataset['PRICE']
print(X)
print(X.shape)
print(Y)
```

```
[[6.3200e-03 1.8000e+01 2.3100e+00 ... 1.5300e+01 3.9690e+02 4.9800e+00]
 [2.7310e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9690e+02 9.1400e+00]
 [2.7290e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9283e+02 4.0300e+00]
 ...
 [6.0760e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 5.6400e+00]
 [1.0959e-01 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9345e+02 6.4800e+00]
 [4.7410e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 7.8800e+00]]
(506, 13)
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: PRICE, Length: 506, dtype: float64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.3, random_state = 4)
```

```
# define base model
def baseline_model():
 # create model
 model = Sequential()
 model.add(Dense(13, input_shape=(13,), kernel_initializer='normal', activation='relu'))
 model.add(Dense(1, kernel_initializer='normal'))
 # Compile model
 model.compile(loss='mean_squared_error', optimizer='adam')
 return model
```

```
estimator = KerasRegressor(model=baseline_model, epochs=100, batch_size=5, verbose=0)
kfold = KFold(n_splits=10)
```

```
results = cross_val_score(estimator, X_train, y_train, cv=kfold, scoring='neg_mean_squared_error')
```

```
print("Baseline: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Baseline: -25.00 (10.39) MSE
```

```
from sklearn.metrics import mean_squared_error
estimator.fit(X, Y)
prediction = estimator.predict(X_test)
errors = mean_squared_error(y_test, prediction)
print("Mean_Square_error",errors)
```

```
Mean_Square_error 20.136239140496127
```

```
root_m_errors = mean_squared_error(y_test, prediction, squared=False)
print(root_m_errors)
```

```
4.487342102012741
```

```
from sklearn.metrics import mean_absolute_error
abs_errors = mean_absolute_error(y_test, prediction)
print(abs_errors)
```

```
3.2927065096403423
```

```
prediction.shape
```

```
(152,)
```

```
prediction
```

```
array([16.080547 , 22.951525 , 21.116524 , 19.49501  , 46.437763 ,
       24.30467  , 33.79852  , 17.03501  , 13.63224  , 17.521805 ,
       33.218544 , 28.231752 , 18.698633 , 33.69988  , 22.396074 ,
       13.8283   , 22.221275 ,  9.675329 , 10.176145 , 12.831154 ,
        8.09062  , 14.936548 , 20.306232 , 21.415878 , 20.278969 ,
       20.776043 , 19.598969 , 18.051981 , 16.76956  , 19.197493 ,
       11.316172 , 24.186146 , 31.479431 , 22.253542 , 16.50034  ,
       14.341301 , 34.107735 , 39.282177 , 28.294527 , 25.736681 ,
       41.54558  , 37.67275  , 18.17292  , 32.60875  , 31.222017 ,
       23.801994 , 46.64964  , 20.257114 , 19.34218  , 23.236074 ,
       39.263527 , 18.449451 , 14.26607  , 26.903168 , 14.908658 ,
       22.005798 , 23.229996 , 41.26844  , 13.856382 , 42.426014 ,
       18.422667 , 19.459988 , 36.145557 , 18.193764 , 48.239056 ,
       28.961435 , 48.921776 , 10.352896 , 17.515774 , 22.111872 ,
       23.616808 , 22.81732  , 23.985485 , 26.82099  , 17.3279   ,
       24.190248 , 18.519358 , 25.485592 ,  9.9624405, 13.942787 ,
       23.702862 , 15.487465 , 31.411198 , 20.81942  , 29.37409  ,
       25.14272  , 32.111164 , 21.084415 , 22.324701 , 43.243034 ,
       38.154095 , 44.956665 , 21.061428 , 47.151108 , 22.329617 ,
       23.250784 , 23.9479   , 45.003727 , 15.524502 , 23.772465 ,
       11.491859 , 24.011028 , 39.925957 , 17.092772 , 29.951275 ,
       22.779469 , 41.00313  , 31.159893 , 34.669044 , 25.549906 ,
       26.995817 , 18.612461 , 13.753554 , 34.395393 , 35.47121  ,
       22.641436 , 48.800697 , 17.249874 , 13.736362 , 20.260414 ,
       19.697369 , 14.273516 , 27.650787 , 25.190863 , 21.694798 ,
       18.440872 , 42.424026 , 11.621473 , 16.58459  , 16.559675 ,
       43.599747 , 19.86341  , 48.504276 , 35.477966 , 34.0842   ,
       21.659342 , 12.105693 , 26.789127 , 22.57792  , 20.648537 ,
       17.525415 , 11.834238 , 21.426456 , 23.383629 ,  9.3852215,
       22.265203 , 15.105053 , 15.368527 , 44.12031  , 23.106133 ,
       35.73562  , 15.032433 ], dtype=float32)
```

```
y_test
```

```
8      16.5
289    24.8
68     17.4
211    19.3
226    37.6
       ...
446    14.9
364    21.9
337    18.5
39     30.8
478    14.6
Name: PRICE, Length: 152, dtype: float64
```

```
# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=baseline_model, epochs=50, batch_size=5, verbose=0)))
```

```
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline,  X_train, y_train, cv=kfold, scoring='neg_mean_squared_error')
print("Standardized: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Standardized: -21.48 (11.12) MSE
```

```
from sklearn.metrics import mean_squared_error
estimator.fit(X, Y)
prediction = estimator.predict(X_test)
errors = mean_squared_error(y_test, prediction)
print("Mean_Square_error",errors)
```

```
Mean_Square_error 19.86905654719887
```

```
root_m_errors = mean_squared_error(y_test, prediction, squared=False)
print(root_m_errors)
```

```
4.457471990624155
```

```
from sklearn.metrics import mean_absolute_error
abs_errors = mean_absolute_error(y_test, prediction)
print(abs_errors)
```

```
3.1274011072359587
```

Evaluate a Deeper Network Topology

```
# define the model
def larger_model():
 # create model
 model = Sequential()
 model.add(Dense(13, input_shape=(13,), kernel_initializer='normal', activation='relu'))
 model.add(Dense(6, kernel_initializer='normal', activation='relu'))
 model.add(Dense(1, kernel_initializer='normal'))
 # Compile model
 model.compile(loss='mean_squared_error', optimizer='adam')
 return model
```

```
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=larger_model, epochs=50, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold, scoring='neg_mean_squared_error')
print("Larger: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Larger: -23.16 (26.34) MSE
```

Evaluate a Wider Network Topology

```
# Regression Example With Boston Dataset: Standardized and Wider

def wider_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_shape=(13,), kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=wider_model, epochs=100, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold, scoring='neg_mean_squared_error')
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Wider: -20.96 (21.17) MSE
```

```
from sklearn.metrics import mean_squared_error
estimator.fit(X, Y)
prediction = estimator.predict(X_test)
errors = mean_squared_error(y_test, prediction)
print("Mean_Square_error",errors)
```

```
Mean_Square_error 26.822935333426734
```

```
root_m_errors = mean_squared_error(y_test, prediction, squared=False)
print(root_m_errors)
```

```
5.179086341569016
```

```
from sklearn.metrics import mean_absolute_error
abs_errors = mean_absolute_error(y_test, prediction)
print(abs_errors)
```

```
4.036824513109107
```