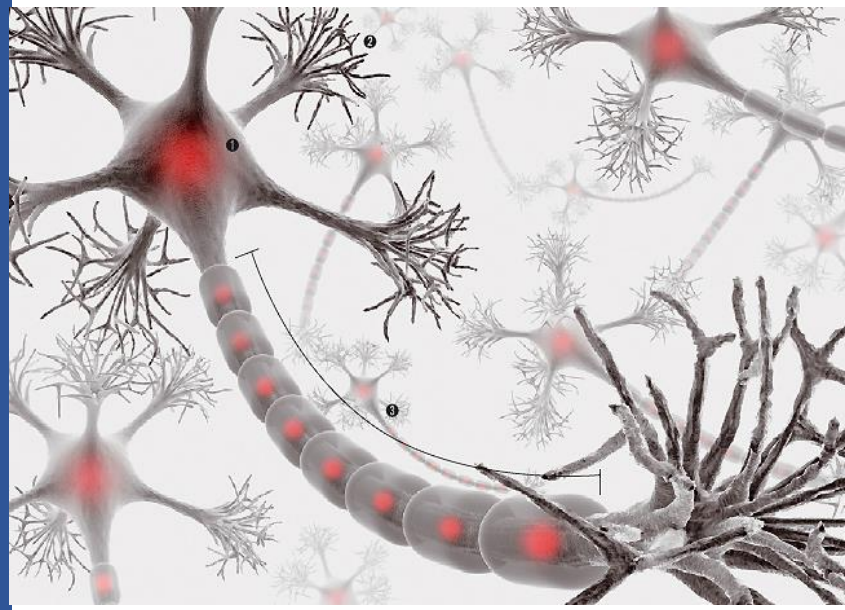
# 자동차 연비 예측

**학습 목표**

- 회귀 예시로 자동차 연비를 예측하는 신경망 모델을 만들어 본다.

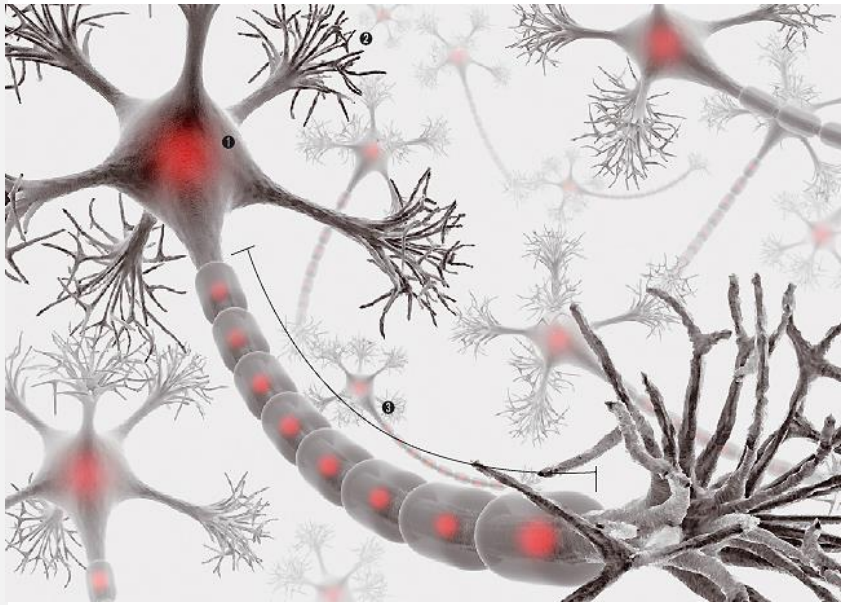
**주요 내용**

- 1. 문제 정의
- 2. 데이터 준비
- 3. 모델 정의 및 훈련, 검증



https://www.tensorflow.org/tutorials/keras/regression

# 1 문제 정의



# 자동차 연비 예측

1970년대 후반과 1980년대 초반의 자동차 연비를 예측하는 모델

**Auto MPG 데이터셋**

- 이 기간에 출시된 자동차 정보 제공
- 398개 인스턴스, 8개 속성

https://archive.ics.uci.edu/ml/datasets/auto+mpg

| 속성 | 타입 | 설명 |
|---|---|---|
| MPG | continuous | 갤런 당 마일수 |
| Cylinders | multi-valued discrete | 실린더 수 |
| Displacement | continuous | 배기량 |
| Horsepower | continuous | 마력 |
| Weight | continuous | 공차 중량 |
| Acceleration | continuous | 가속 |
| Model Year | multi-valued discrete | 모델 연식 |
| Origin | multi-valued discrete | 제조국 |
| Car name | string | 차량 이름 |

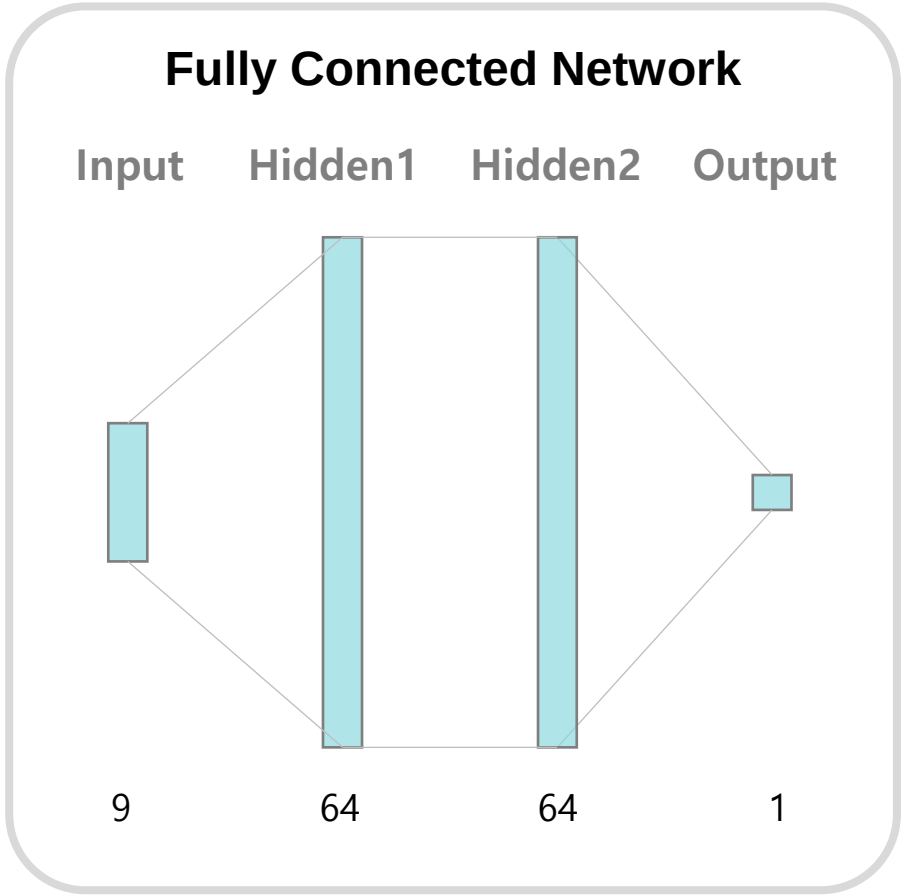
# Network 구성

- Missing Value 제거
- Origin One Hot Vector로 변경
- Normalization

| Cylinders |
| --- |
| Displacement |
| Horsepower |
| Weight |
| Acceleration |
| Model Year |
| USA |
| Europe |
| Japan |

$$x = (x_1, x_2, x_3, \dots, x_9)$$

- 9차원 입력 벡터

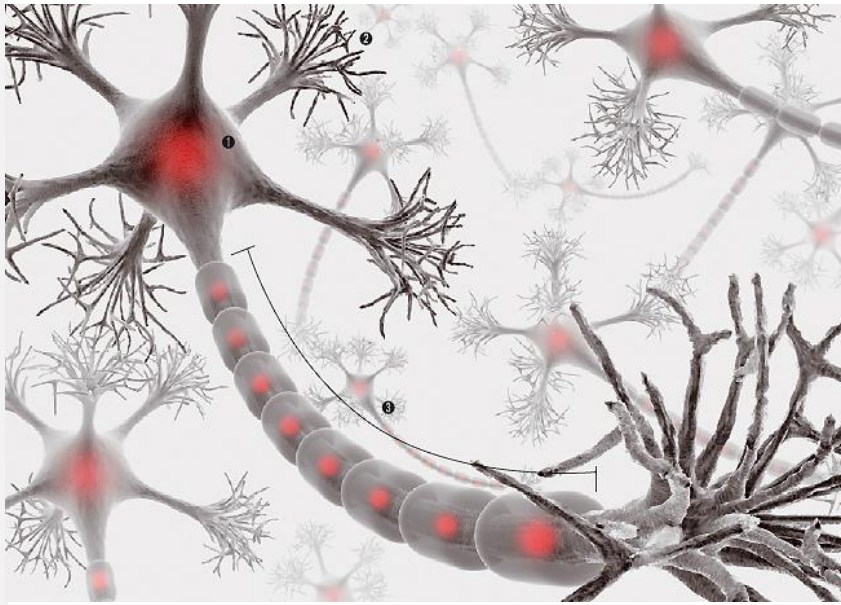



- 2-hidden layers
- 64 neurons
- Hidden unit의 activation은 relu

$y$

- 연비 (MPG)
- Real-valued Scalar

# 2 데이터 준비



# 패키지 설치

```
# 산점도 행렬을 그리기 위해 seaborn 패키지를 설치합니다
!pip install -q seaborn
```

# 텐서플로와 다른 라이브러리 임포트

```
from __future__ import absolute_import, division, print_function, unicode_literals, unicode_literals

import pathlib

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

print(tf.__version__)
```

```
1.14.1-dev20190524
```

# Auto MPG 데이터셋

```
dataset_path = keras.utils.get_file("auto-mpg.data", "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data")

dataset_path
```

```
'/home/kbuilder/.keras/datasets/auto-mpg.data'
```

# Pandas로 데이터 읽기

```
column_names = ['MPG','Cylinders','Displacement','Horsepower','Weight',
                'Acceleration', 'Model Year', 'Origin']
raw_dataset = pd.read_csv(dataset_path, names=column_names,
                          na_values = "?", comment='\t',
                          sep=" ", skipinitialspace=True)

dataset = raw_dataset.copy()
dataset.tail()
```

| | MPG | Cylinders | Displacement | Horsepower | Weight | Acceleration | Model Year | Origin |
|---|---|---|---|---|---|---|---|---|
| 393 | 27.0 | 4 | 140.0 | 86.0 | 2790.0 | 15.6 | 82 | 1 |
| 394 | 44.0 | 4 | 97.0 | 52.0 | 2130.0 | 24.6 | 82 | 2 |
| 395 | 32.0 | 4 | 135.0 | 84.0 | 2295.0 | 11.6 | 82 | 1 |
| 396 | 28.0 | 4 | 120.0 | 79.0 | 2625.0 | 18.6 | 82 | 1 |
| 397 | 31.0 | 4 | 119.0 | 82.0 | 2720.0 | 19.4 | 82 | 1 |

# pandas 데이터 구조

## 테이블 형태의 데이터 구조

이름이 있는 Column

Index를 갖는 Row

| | MPG | Cylinders | Displacement | Horsepower | Weight | Acceleration | Model Year | Origin |
|---|---|---|---|---|---|---|---|---|
| 393 | 27.0 | 4 | 140.0 | 86.0 | 2790.0 | 15.6 | 82 | 1 |
| 394 | 44.0 | 4 | 97.0 | 52.0 | 2130.0 | 24.6 | 82 | 2 |
| 395 | 32.0 | 4 | 135.0 | 84.0 | 2295.0 | 11.6 | 82 | 1 |
| 396 | 28.0 | 4 | 120.0 | 79.0 | 2625.0 | 18.6 | 82 | 1 |
| 397 | 31.0 | 4 | 119.0 | 82.0 | 2720.0 | 19.4 | 82 | 1 |

Column 이름으로 Query가 가능 (data['Horsepower'])

**DataFrame**

- Column들로 이뤄진 데이터 구조로 Column마다 데이터 타입이 다를 수 있음
  (Column 또는 Row는 **Series** 객체로 관리)
- Column은 Name으로 Row는 Index로 접근

# pandas Column 이름으로 조회

## 컬럼 이름으로 조회

```
dataset['MPG']
```

```
0      18.0
1      15.0
2      18.0
3      16.0
4      17.0
      ...
393    27.0
394    44.0
395    32.0
396    28.0
397    31.0

Name: MPG, Length: 398, dtype: float64
```

## 여러 컬럼 조회

칼럼 이름을 리스트로 명시

```
dataset[['MPG', 'Weight']]
```

| | MPG | Weight |
|---|---|---|
| 1 | 18.0 | 3504.0 |
| 2 | 15.0 | 3693.0 |
| 3 | 18.0 | 3436.0 |
| 4 | 16.0 | 3433.0 |
| ... | ... | ... |
| 393 | 27.0 | 2790.0 |
| 394 | 44.0 | 2130.0 |
| 395 | 32.0 | 2295.0 |
| 396 | 28.0 | 2625.0 |
| 397 | 31.0 | 2720.0 |

398 rows × 2 columns

# pandas Column 조건 검색

**여러 조건으로 행 선택**

```
dataset[(dataset['Origin']==2) & (dataset['Horsepower']>70)].head(4)
```

| | MPG | Cylinders | Displacement | Horsepower | Weight | Acceleration | Model Year | Origin |
|---|---|---|---|---|---|---|---|---|
| 20 | 25.0 | 4 | 110.0 | 87.0 | 2672.0 | 17.5 | 70 | 2 |
| 21 | 24.0 | 4 | 107.0 | 90.0 | 2430.0 | 14.5 | 70 | 2 |
| 22 | 25.0 | 4 | 104.0 | 95.0 | 2375.0 | 17.5 | 70 | 2 |
| 23 | 26.0 | 4 | 121.0 | 113.0 | 2234.0 | 12.5 | 70 | 2 |

- 차량 제조국이 Europe이고 마력이 70이상인 차량 선택
- head()는 default로 처음 5개 항목을 반환, tail()은 default로 마지막 5개 항목을 반환

# pandas Row Index로 조회

**인덱스 위치를 지정해서 행 선택 (iloc 함수)**

인덱스 위치

```
dataset.iloc[:5]
```

| | MPG | Cylinders | Displacement | Horsepower | Weight | Acceleration | Model Year | Origin |
|---|---|---|---|---|---|---|---|---|
| 0 | 18.0 | 8 | 307.0 | 130.0 | 3504.0 | 12.0 | 70 | 1 |
| 1 | 15.0 | 8 | 350.0 | 165.0 | 3693.0 | 11.5 | 70 | 1 |
| 2 | 18.0 | 8 | 318.0 | 150.0 | 3436.0 | 11.0 | 70 | 1 |
| 3 | 16.0 | 8 | 304.0 | 150.0 | 3433.0 | 12.0 | 70 | 1 |
| 4 | 17.0 | 8 | 302.0 | 140.0 | 3449.0 | 10.5 | 70 | 1 |

- iloc는 인덱스 위치에 대해 실행
- 그외 ix(), loc() : 인덱스 레이블을 지정할 수도 있음

# pandas 행 & 열 추가/삭제, 누락 데이터 제거

**새로운 이름으로 열 추가**

```
dataset['USA’] = 1
```

**drop 함수로 행 삭제**

```
dataset = dataset.drop(‘Origin’, axis=1)
```

axis=0은 row, 1은 column

**drop 함수로 열 삭제**

```
dataset = dataset.drop(Row_Index, axis=0)
```

axis=0은 row, 1은 column

**누락 데이터 조회**

```
dataset.isna()
```

**누락 데이터 개수 합산**

```
dataset.isna().sum()
```

**누락 데이터 제거**

```
dataset = dataset.dropna()
```

# pandas 데이터 샘플링

**샘플 개수로 지정**

```
dataset.sample(n=5000, random_state=0)
```

**샘플 비율로 지정**

```
dataset.sample(frac=0.8, random_state=0)
```

Seed for the random number generator (if int), or numpy RandomState object.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sample.html

# pandas Series 연산

Series는 같은 이름을 갖는 항목끼리 연산, 같은 원리도 DataFrame도 이름을 매칭해서 연산

```
student1 = pd.Series({'국어':100, '영어':80,'수학':90})
student2 = pd.Series({'수학':80, '국어':90, '영어':80})
print(student1)
print('\n')
print(student2)
```

```
[Output]
국어   100
영어    80
수학    90
dtype: int64

수학   80
국어   90
영어   80
dtype: int64
```

```
add_data = student1 + student2 # 덧셈
sub_data = student1 - student2 # 뺄셈
mul_data = student1 * student2 # 곱셈
div_data = student1 / student2 # 나눗셉
# 결과를 데이터 프레임으로 합치기 (시리즈 -> 데이터프레임)
result = pd.DataFrame([add_data,sub_data,mul_data,div_data],
                      index = ['덧셈','뺄셈','곱셈','나눗셈'])
print(result)
```

```
[Output]
          국어        수학     영어
덧셈   190.000000  170.000  160.0
뺄셈    10.000000   10.000    0.0
곱셈  9000.000000 7200.000 6400.0
나눗셈    1.111111    1.125    1.0
```



https://yganalyst.github.io/study/Pd_3/

# 데이터 전처리 (문제)

누락 데이터 확인 및 제거

누락 데이터 확인 : dataset.isna()
누락 데이터 제거 : dataset.drop()

Origin을 One-Hot Vector로 변환

Origin 1은 USA, 2는 Europe, 3은 Japan

데이터셋 분할

dataset.sample(), dataset.drop() 사용

레이블 분리

dataset.pop() 사용

데이터 정규화

표준정규분포 정규화 $Z = \frac{X - \mu}{\sigma} \sim \mathcal{N}(0,1)$

# 누락 데이터 확인 및 제거 (문제)

누락된 데이터 확인

```
# 누락 데이터 확인은 dataset.isna()로 확인
# your code
```

```
MPG             0
Cylinders       0
Displacement    0
Horsepower      6
Weight          0
Acceleration    0
Model Year      0
Origin          0
dtype: int64
```

누락된 행을 삭제

```
# your code
```

# Origin을 One-Hot Vector로 변환

누락된 데이터 열은 수치형이 아니고 범주형이므로 원-핫 인코딩(one-hot encoding)으로 변환

```
origin = dataset.pop('Origin')
dataset['USA'] = (origin == 1)*1.0
dataset['Europe'] = (origin == 2)*1.0
dataset['Japan'] = (origin == 3)*1.0
dataset.tail()
```

| | MPG | Cylinders | Displacement | Horsepower | Weight | Acceleration | Model Year | USA | Europe | Japan |
|---|---|---|---|---|---|---|---|---|---|---|
| 393 | 27.0 | 4 | 140.0 | 86.0 | 2790.0 | 15.6 | 82 | 1.0 | 0.0 | 0.0 |
| 394 | 44.0 | 4 | 97.0 | 52.0 | 2130.0 | 24.6 | 82 | 0.0 | 1.0 | 0.0 |
| 395 | 32.0 | 4 | 135.0 | 84.0 | 2295.0 | 11.6 | 82 | 1.0 | 0.0 | 0.0 |
| 396 | 28.0 | 4 | 120.0 | 79.0 | 2625.0 | 18.6 | 82 | 1.0 | 0.0 | 0.0 |
| 397 | 31.0 | 4 | 119.0 | 82.0 | 2720.0 | 19.4 | 82 | 1.0 | 0.0 | 0.0 |

# 데이터셋 분할 (문제)

데이터셋을 훈련 세트와 테스트 세트로 분할 (Hint : dataset.sample(), dataset drop() 사용)

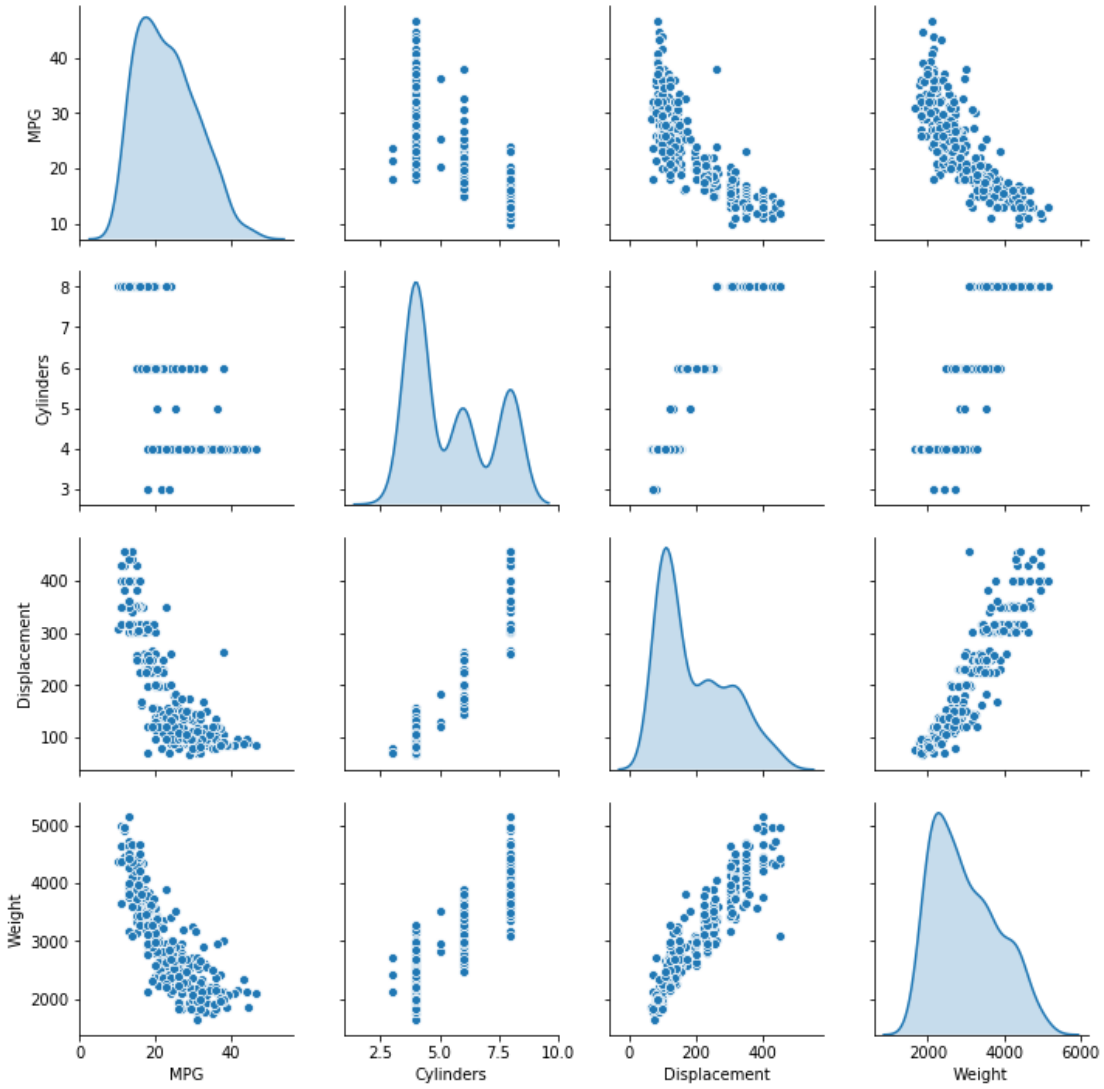
```
train_dataset = # your code
test_dataset = # your code
```

# 데이터 조사하기

산점도 행렬로 데이터 분포 및 상관성 조사

```
sns.pairplot(train_dataset[["MPG",
"Cylinders", "Displacement", "Weight"]],
diag_kind="kde")
```

kde : Use kernel density estimates for univariate plots:



# 데이터 통계

```
train_stats = train_dataset.describe()
train_stats.pop("MPG")
train_stats = train_stats.transpose()
train_stats
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| Cylinders | 314.0 | 5.477707 | 1.699788 | 3.0 | 4.00 | 4.0 | 8.00 | 8.0 |
| Displacement | 314.0 | 195.318471 | 104.331589 | 68.0 | 105.50 | 151.0 | 265.75 | 455.0 |
| Horsepower | 314.0 | 104.869427 | 38.096214 | 46.0 | 76.25 | 94.5 | 128.00 | 225.0 |
| Weight | 314.0 | 2990.251592 | 843.898596 | 1649.0 | 2256.50 | 2822.5 | 3608.00 | 5140.0 |
| Acceleration | 314.0 | 15.559236 | 2.789230 | 8.0 | 13.80 | 15.5 | 17.20 | 24.8 |
| Model Year | 314.0 | 75.898089 | 3.675642 | 70.0 | 73.00 | 76.0 | 79.00 | 82.0 |
| USA | 314.0 | 0.624204 | 0.485101 | 0.0 | 0.00 | 1.0 | 1.00 | 1.0 |
| Europe | 314.0 | 0.178344 | 0.383413 | 0.0 | 0.00 | 0.0 | 0.00 | 1.0 |
| Japan | 314.0 | 0.197452 | 0.398712 | 0.0 | 0.00 | 0.0 | 0.00 | 1.0 |

# 레이블 분리 (문제)

레이블 분리

```
# dataset.pop() 사용
train_labels = # your code
test_labels = # your code
```

# 데이터 정규화

표준정규분포 데이터 정규화

```
def norm(x):
  return (x - train_stats['mean']) / train_stats['std’]

normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)
```

# 참고 DataFrame 연산

**DataFrame과 Series는 같은 이름을 가진 항목끼리 연산**

train_dataset = {'MPG' : {...}, 'Cylinders' : {....}, ..., 'Weight' " {...}, ...}  DataFrame은 Column들의 Series를 갖는 Series이다!

| | MPG | Cylinders | Displacement | Horsepower | Weight | Acceleration | Model Year | Origin |
|---|---|---|---|---|---|---|---|---|
| 393 | 27.0 | 4 | 140.0 | 86.0 | 2790.0 | 15.6 | 82 | 1 |
| 394 | 44.0 | 4 | 97.0 | 52.0 | 2130.0 | 24.6 | 82 | 2 |
| 395 | 32.0 | 4 | 135.0 | 84.0 | 2295.0 | 11.6 | 82 | 1 |
| 396 | 28.0 | 4 | 120.0 | 79.0 | 2625.0 | 18.6 | 82 | 1 |
| 397 | 31.0 | 4 | 119.0 | 82.0 | 2720.0 | 19.4 | 82 | 1 |

$$Z = \frac{X - \mu}{\sigma}$$

$$= \frac{\text{train\_dataset} - \text{train\_stats}['\text{mean}']}{\text{train\_stats}['\text{std}']}$$

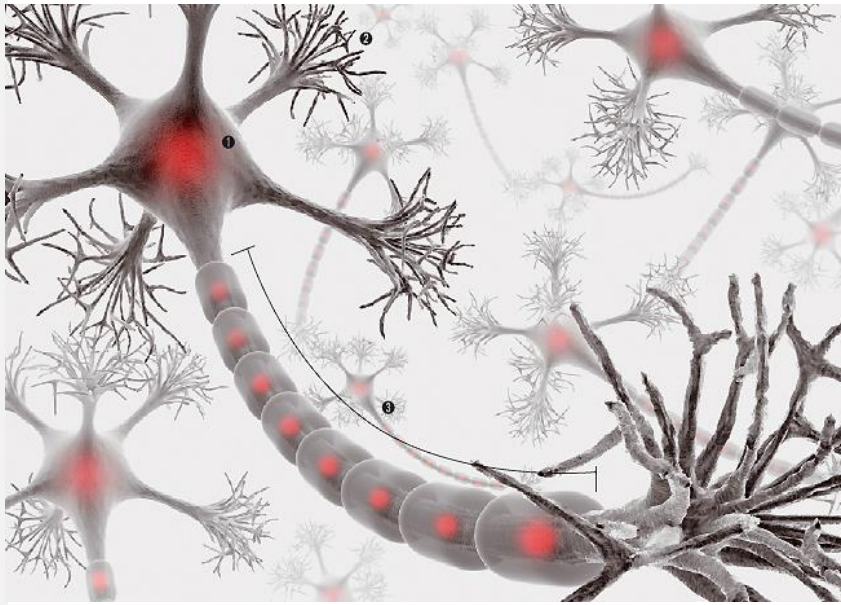
train_stats = {'count' : {...}, mean' : {....}, ...}

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| Cylinders | 314.0 | 5.477707 | 1.699788 | 3.0 | 4.00 | 4.0 | 8.00 | 8.0 |
| Displacement | 314.0 | 195.318471 | 104.331589 | 68.0 | 105.50 | 151.0 | 265.75 | 455.0 |
| Horsepower | 314.0 | 104.869427 | 38.096214 | 46.0 | 76.25 | 94.5 | 128.00 | 225.0 |
| Weight | 314.0 | 2990.251592 | 843.898596 | 1649.0 | 2256.50 | 2822.5 | 3608.00 | 5140.0 |
| Acceleration | 314.0 | 15.559236 | 2.789230 | 8.0 | 13.80 | 15.5 | 17.20 | 24.8 |
| Model Year | 314.0 | 75.898089 | 3.675642 | 70.0 | 73.00 | 76.0 | 79.00 | 82.0 |
| USA | 314.0 | 0.624204 | 0.485101 | 0.0 | 0.00 | 1.0 | 1.00 | 1.0 |
| Europe | 314.0 | 0.178344 | 0.383413 | 0.0 | 0.00 | 0.0 | 0.00 | 1.0 |
| Japan | 314.0 | 0.197452 | 0.398712 | 0.0 | 0.00 | 0.0 | 0.00 | 1.0 |

train_stats['mean'] = {''Cylinders' : {...}, 'Displacement' : {....}, ..., 'Weight' " {...}, ...}

train_stats['std'] = {''Cylinders' : {...}, 'Displacement' : {....}, ..., 'Weight' " {...}, ...}

# 3 모델 정의 및 훈련, 검증



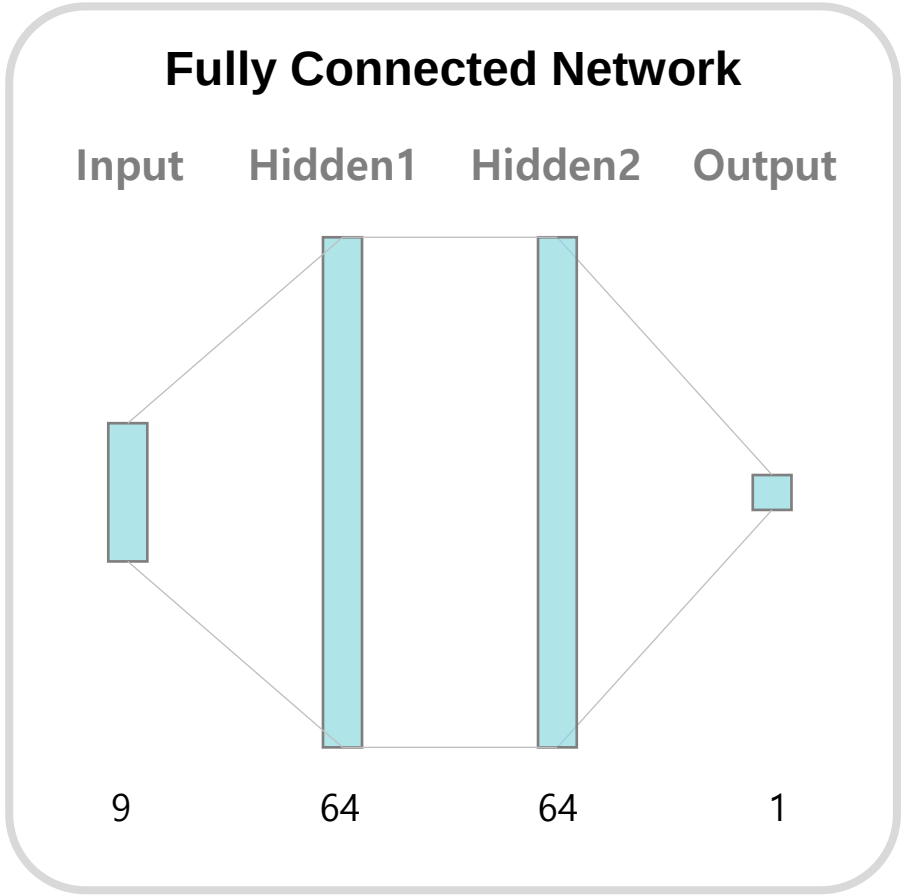
# Network 구성

- Missing Value 제거
- Origin One Hot Vector로 변경
- Normalization

| Cylinders |
| --- |
| Displacement |
| Horsepower |
| Weight |
| Acceleration |
| Model Year |
| USA |
| Europe |
| Japan |

$$x = (x_1, x_2, x_3, \dots, x_9)$$

- 9차원 입력 벡터




- 2-hidden layers
- 64 neurons
- Hidden unit의 activation은 relu

$y$

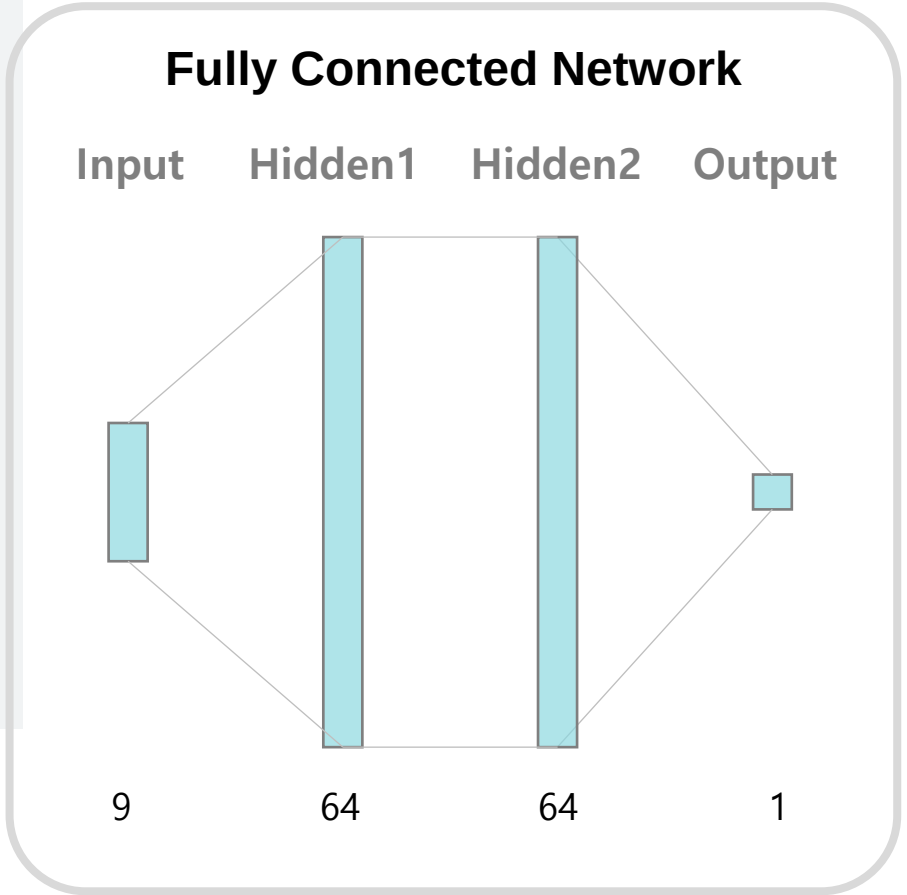
- 연비 (MPG)
- Real-valued Scalar

# 모델 (문제)

## 모델 정의

If the pandas object is dataframe then it returns columns

```
def build_model():
  # input len(train_dataset.keys())
  # hidden 64, hidden 64, output 1
  model = keras.Sequential(#your code)

  # RMSProp으로 Optimizer 생성, learning rate = 0.001
  optimizer = #your code

  # loss, optimizer, metric 설정
  # metric은 'mean_absolute_error', 'mean_squared_error'로 설정
  model.compile(#your code)

return model
```




## 모델 생성

```
model = build_model()
```

# 모델 확인

모델 구조 출력

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                640
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 4,865
Trainable params: 4,865
Non-trainable params: 0
_________________________________________________________________
```

# 모델 실행

훈련 전 10개 데이터에 대해 예측

```
example_batch = normed_train_data[:10]
example_result = model.predict(example_batch)
example_result
```

```
array([[ 0.45123982],
       [ 0.26063395],
       [-0.04244372],
       [ 0.65448815],
       [ 0.26905507],
       [ 0.12702759],
       [ 0.31641036],
       [ 0.1571908 ],
       [ 0.12432042],
       [ 0.20841858]], dtype=float32)
```

# 모델 훈련 (문제)

```
# 에포크가 끝날 때마다 점(.)을 출력해 훈련 진행 과정을 표시합니다
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 100 == 0: print('')
    print('.', end='')

EPOCHS = 1000

# 1. PrintDot() Callback 추가
# 2. validation_split을 이용해서 validation set은 training set의 20%로 설정
# 3. verbose=0로 설정해서 출력 없애기
history = model.fit(#your code)
```

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

# 모델 과정 시각화

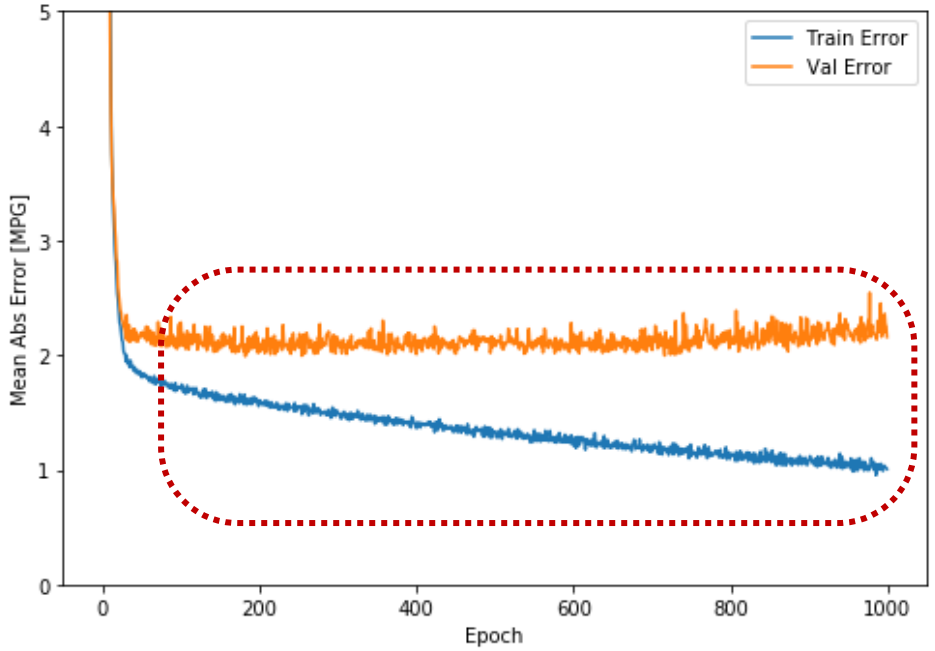
모델 히스토리 테이블 출력

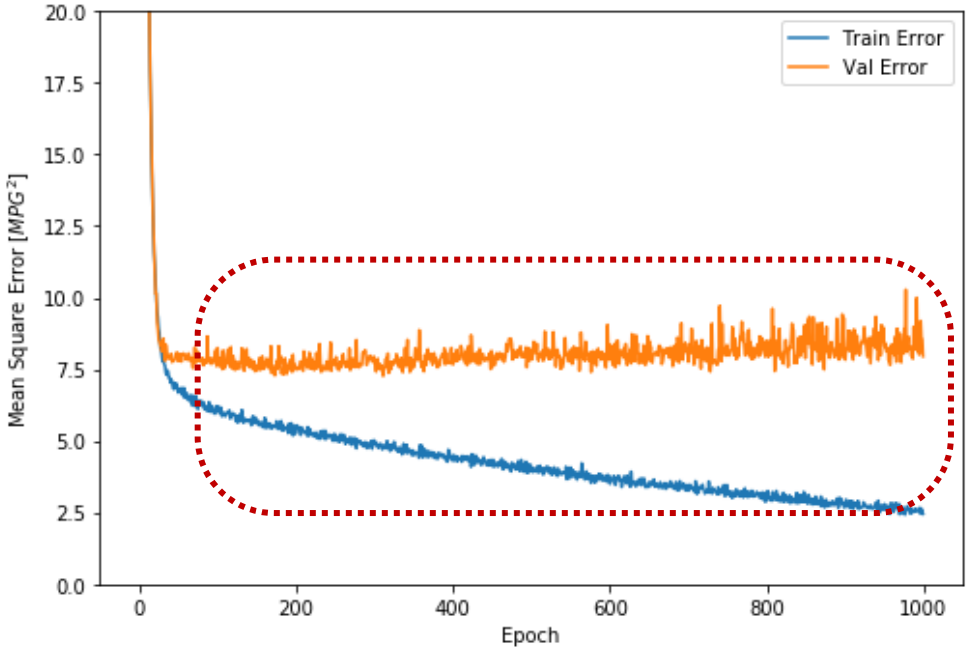
```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

| | loss | mean_absolute_error | mean_squared_error | val_loss | val_mean_absolute_error | val_mean_squared_error | epoch |
|---|---|---|---|---|---|---|---|
| 995 | 2.671277 | 1.040861 | 2.671277 | 8.136572 | 2.205987 | 8.136572 | 995 |
| 996 | 2.453449 | 1.026373 | 2.453449 | 9.190708 | 2.362641 | 9.190708 | 996 |
| 997 | 2.630218 | 1.022048 | 2.630218 | 8.589535 | 2.264779 | 8.589535 | 997 |
| 998 | 2.518478 | 1.019921 | 2.518478 | 8.555749 | 2.256706 | 8.555749 | 998 |
| 999 | 2.453510 | 1.009472 | 2.453510 | 7.952622 | 2.160183 | 7.952622 | 999 |

# 훈련 모니터링

MAE, MSE 오류 그래프 (1000 epochs)




검증 오류가 낮아지지 않으므로 EarlyStopping 콜백(callback)을 사용

# 훈련 모니터링

```
import matplotlib.pyplot as plt

def plot_history(history):
  hist = pd.DataFrame(history.history) # DataFrame으로 변환
  hist['epoch'] = history.epoch # epoch 추가

  plt.figure(figsize=(8,12))

  plt.subplot(2,1,1) # MAE 그래프
  plt.xlabel('Epoch')
  plt.ylabel('Mean Abs Error [MPG]')
  plt.plot(hist['epoch'], hist['mean_absolute_error'],  label='Train Error')
  plt.plot(hist['epoch'], hist['val_mean_absolute_error'], label = 'Val Error')
  plt.ylim([0,5])
  plt.legend()

  plt.subplot(2,1,2) # MSE 그래프
  plt.xlabel('Epoch')
  plt.ylabel('Mean Square Error [$MPG^2$]')
  plt.plot(hist['epoch'], hist['mean_squared_error'],  label='Train Error')
  plt.plot(hist['epoch'], hist['val_mean_squared_error'], label = 'Val Error')
  plt.ylim([0,20])
  plt.legend()
  plt.show()

plot_history(history)
```

# EarlyStopping 콜백 사용 (문제)

Callback 정의 및 설정

```
model = build_model()

# keras.callbacks.EarlyStopping 사용
# patience : 성능 향상을 체크할 에포크 횟수
early_stop = # your code

history = model.fit(normed_train_data, train_labels, epochs=EPOCHS,
                    validation_split = 0.2, verbose=0, callbacks=[early_stop, PrintDot()])

plot_history(history)
```

# 참고 tf.keras.callbacks.EarlyStopping

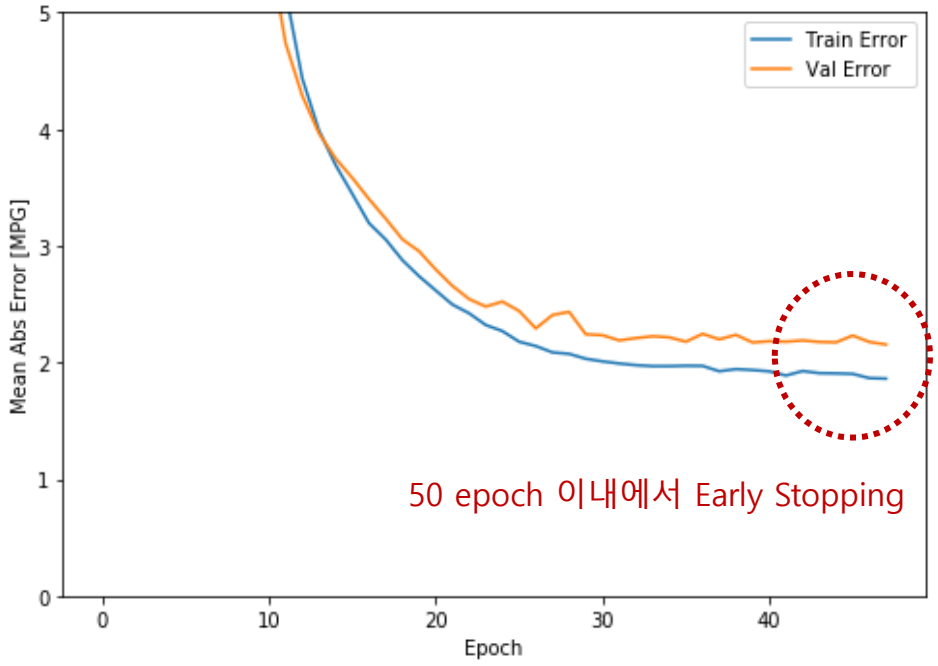
```
tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', min_delta=0, patience=0, verbose=0, mode='auto',
    baseline=None, restore_best_weights=False
)
```

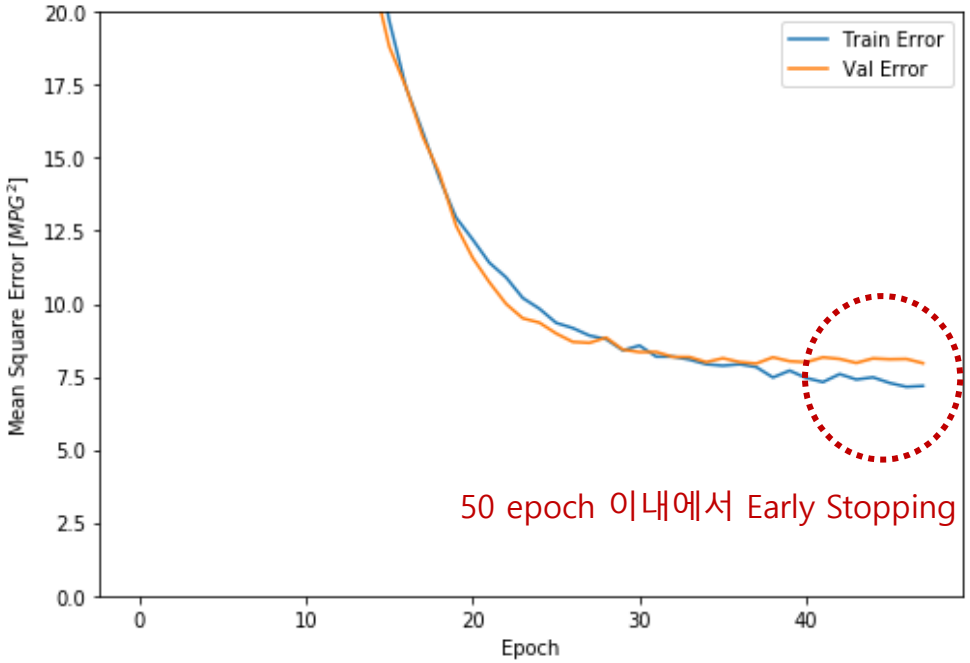
- **monitor**: Quantity to be monitored.
- **min_delta**: Minimum change in the monitored quantity to qualify as an improvement, i.e. an absolute change of less than min_delta, will count as no improvement.
- **patience**: 몇 번 동안 개선되지 않으면 중단할지 회수 지정 Number of epochs with no improvement after which training will be stopped.
- **verbose**: verbosity mode.
- **mode**: One of {"auto", "min", "max"}. In min mode, training will stop when the quantity monitored has stopped decreasing; in max mode it will stop when the quantity monitored has stopped increasing; in auto mode, the direction is automatically inferred from the name of the monitored quantity.
- **baseline**: Baseline value for the monitored quantity. Training will stop if the model doesn't show improvement over the baseline.
- **restore_best_weights**: Whether to restore model weights from the epoch with the best value of the monitored quantity. If False, the model weights obtained at the last step of training are used.

https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

# EarlyStopping 콜백 사용

EarlyStopping 후 MAE, MSE 오류 그래프



# 테스트

**테스트 세트**에서 모델의 성능을 확인

```
loss, mae, mse = model.evaluate(normed_test_data, test_labels, verbose=0)

print("테스트 세트의 평균 절대 오차: {:5.2f} MPG".format(mae))
```

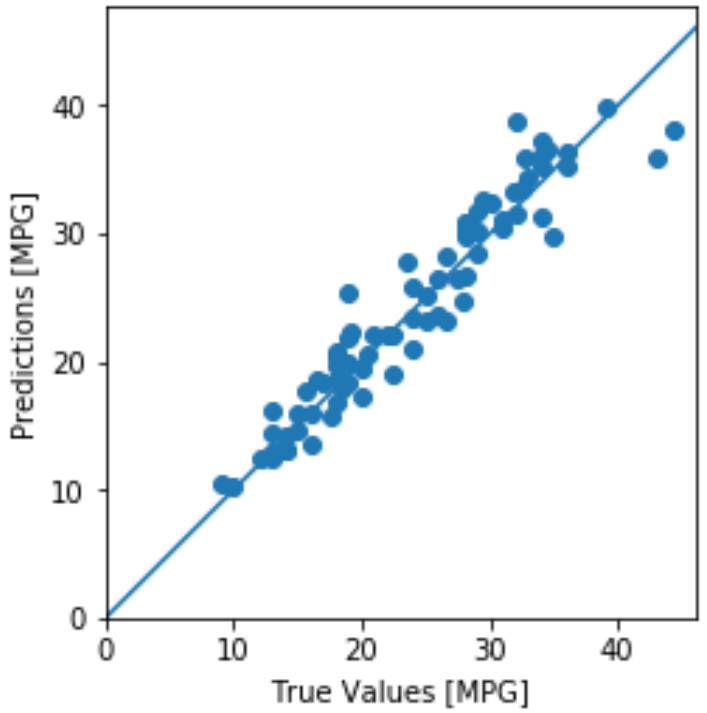
```
테스트 세트의 평균 절대 오차: 1.84 MPG
```

Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch

# 예측

예측 테스트 결과와 실제 레이블과의 상관 관계 그래프

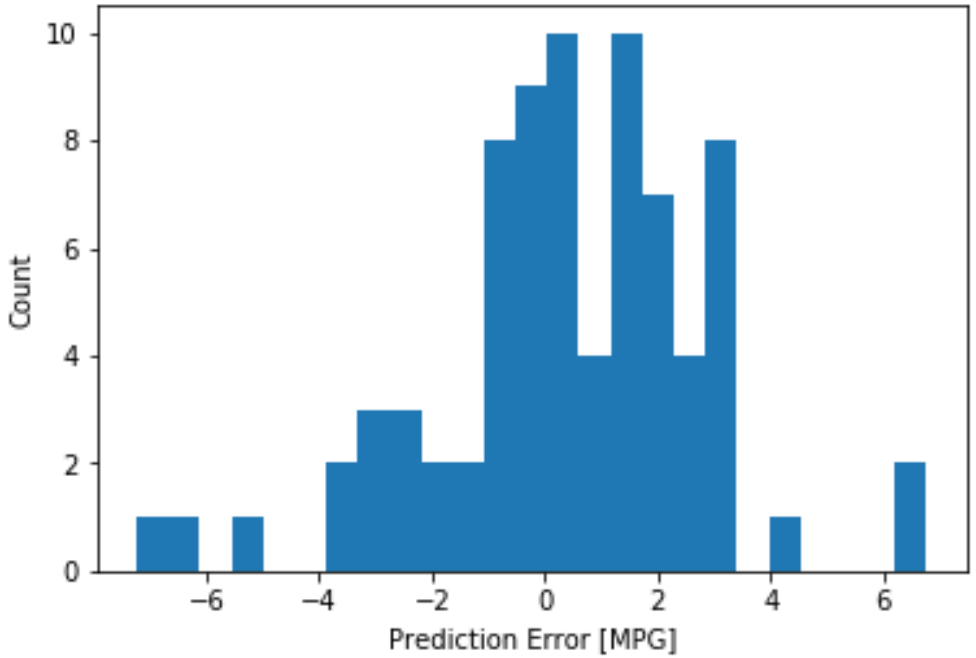
```
test_predictions = model.predict(normed_test_data).flatten()

plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values [MPG]')
plt.ylabel('Predictions [MPG]')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
plt.ylim([0,plt.ylim()[1]])
_ = plt.plot([-100, 100], [-100, 100])
```




plt.axis('equal') : 각 축의 스케일을 동일하게 함
plt.axis('square') : 각 축의 범위가 xmax - xmin = ymax - ymin이 되게 설정

# 오차의 분포

```
error = test_predictions - test_labels
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error [MPG]")
_ = plt.ylabel("Count")
```




정규 분포와 유사한 형태

# Thank you!