```
# Use seaborn for pairplot.
!pip install -q seaborn
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
# Make NumPy printouts easier to read.
np.set_printoptions(precision=3, suppress=True)
```

```
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers

print(tf.__version__)
```

```
2.15.0
```

```
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data'
column_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight',
                'Acceleration', 'Model Year', 'Origin']
raw_dataset = pd.read_csv(url, 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

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

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

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

```
dataset['Origin'] = dataset['Origin'].map({1: 'USA', 2: 'Europe', 3: 'Japan'})
```

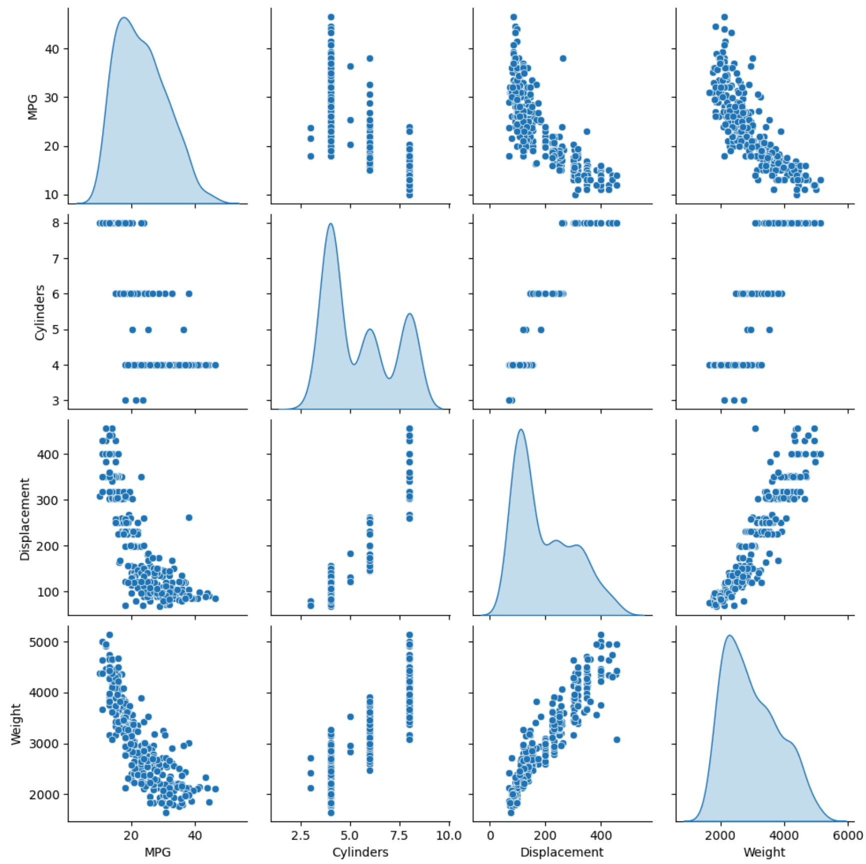
```
dataset = pd.get_dummies(dataset, columns=['Origin'], prefix='', prefix_sep='')
dataset.tail()
```

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Europe	J
393	27.0	4	140.0	86.0	2790.0	15.6	82	0	
394	44.0	4	97.0	52.0	2130.0	24.6	82	1	
395	32.0	4	135.0	84.0	2295.0	11.6	82	0	
396	28.0	4	120.0	79.0	2625.0	18.6	82	0	

```
train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)
```

```
sns.pairplot(train_dataset[['MPG', 'Cylinders', 'Displacement', 'Weight']], diag_kind=
```

```
<seaborn.axisgrid.PairGrid at 0x7c11002ffee0>
```




```
train_dataset.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
MPG	314.0	23.310510	7.728652	10.0	17.00	22.0	28.95	46.6
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
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
USA	314.0	0.624204	0.485101	0.0	0.00	1.0	1.00	1.0

```
train_features = train_dataset.copy()
test_features = test_dataset.copy()

train_labels = train_features.pop('MPG')
test_labels = test_features.pop('MPG')
```

```
train_dataset.describe().transpose()[['mean', 'std']]
```

	mean	std
MPG	23.310510	7.728652
Cylinders	5.477707	1.699788
Displacement	195.318471	104.331589
Horsepower	104.869427	38.096214
Weight	2990.251592	843.898596
Acceleration	15.559236	2.789230
Model Year	75.898089	3.675642
Europe	0.178344	0.383413
Japan	0.197452	0.398712
USA	0.624204	0.485101

```
normalizer = tf.keras.layers.Normalization(axis=-1)
```

```
normalizer.adapt(np.array(train_features))
```

```
print(normalizer.mean.numpy())
```

```
[[   5.478  195.318  104.869 2990.252   15.559   75.898    0.178    0.197
     0.624]]
```

```
first = np.array(train_features[:1])

with np.printoptions(precision=2, suppress=True):
  print('First example:', first)
  print()
  print('Normalized:', normalizer(first).numpy())
```

```
First example: [[   4.    90.    75.  2125.    14.5   74.     0.     0.     1. ]]

Normalized: [[-0.87 -1.01 -0.79 -1.03 -0.38 -0.52 -0.47 -0.5   0.78]]
```

```
horsepower = np.array(train_features['Horsepower'])

horsepower_normalizer = layers.Normalization(input_shape=[1,], axis=None)
horsepower_normalizer.adapt(horsepower)
```

```
horsepower_model = tf.keras.Sequential([
    horsepower_normalizer,
    layers.Dense(units=1)
])

horsepower_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 normalization_1 (Normaliza  (None, 1)                 3         
 tion)                                                           
                                                                 
 dense (Dense)               (None, 1)                 2         
                                                                 
=================================================================
Total params: 5 (24.00 Byte)
Trainable params: 2 (8.00 Byte)
Non-trainable params: 3 (16.00 Byte)
_________________________________________________________________
```

```
horsepower_model.predict(horsepower[:10])
```

```
1/1 [==============================] - 0s 289ms/step
array([[-0.787],
       [-0.445],
       [ 1.453],
       [-1.103],
       [-0.998],
       [-0.392],
       [-1.182],
       [-0.998],
```

```
        [-0.26 ],
        [-0.445]], dtype=float32)
```

```
horsepower_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.1),
    loss='mean_absolute_error')
```

```
%%time
history = horsepower_model.fit(
    train_features['Horsepower'],
    train_labels,
    epochs=100,
    # Suppress logging.
    verbose=0,
    # Calculate validation results on 20% of the training data.
    validation_split = 0.2)
```

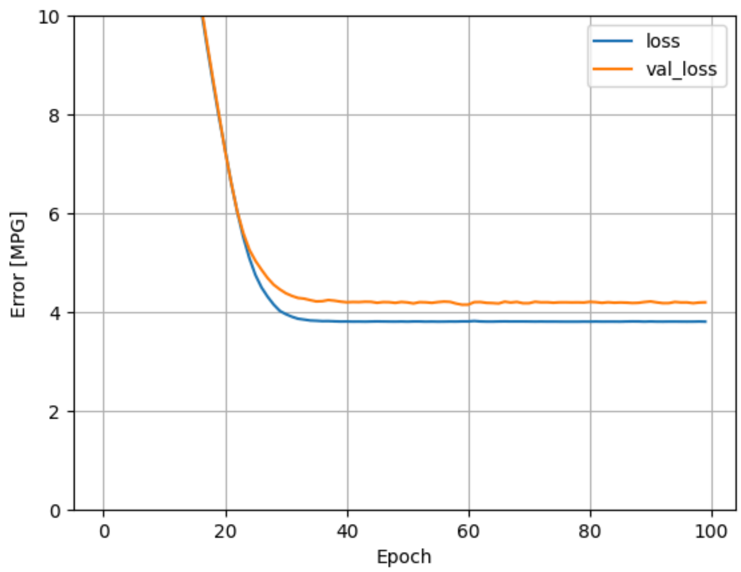
```
     CPU times: user 6.05 s, sys: 210 ms, total: 6.26 s
     Wall time: 10.7 s
```

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

	loss	val_loss	epoch
95	3.803167	4.190689	95
96	3.802571	4.191434	96
97	3.803101	4.175960	97
98	3.805445	4.189009	98
99	3.802643	4.193171	99

```
def plot_loss(history):
  plt.plot(history.history['loss'], label='loss')
  plt.plot(history.history['val_loss'], label='val_loss')
  plt.ylim([0, 10])
  plt.xlabel('Epoch')
  plt.ylabel('Error [MPG]')
  plt.legend()
  plt.grid(True)
```

```
plot_loss(history)
```




```
test_results = {}

test_results['horsepower_model'] = horsepower_model.evaluate(
    test_features['Horsepower'],
    test_labels, verbose=0)
```

```
x = tf.linspace(0.0, 250, 251)
y = horsepower_model.predict(x)
```

```
8/8 [==============================] - 0s 3ms/step
```

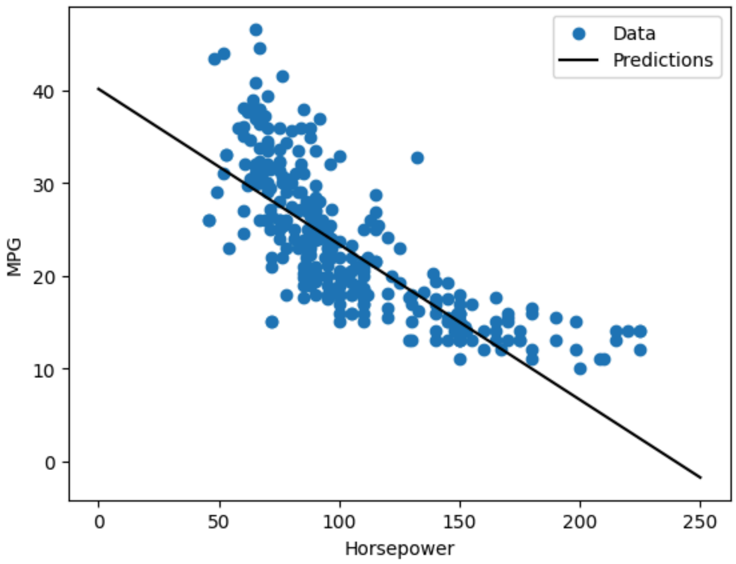
```
def plot_horsepower(x, y):
  plt.scatter(train_features['Horsepower'], train_labels, label='Data')
  plt.plot(x, y, color='k', label='Predictions')
  plt.xlabel('Horsepower')
  plt.ylabel('MPG')
  plt.legend()
```


```
x = tf.linspace(0.0, 250, 251)
y = horsepower_model.predict(x)
```


```
8/8 [==============================] - 0s 5ms/step
```


```
def plot_horsepower(x, y):
  plt.scatter(train_features['Horsepower'], train_labels, label='Data')
  plt.plot(x, y, color='k', label='Predictions')
  plt.xlabel('Horsepower')
  plt.ylabel('MPG')
  plt.legend()
```


```
plot_horsepower(x, y)
```




```
linear_model = tf.keras.Sequential([
    normalizer,
    layers.Dense(units=1)
])
```

```
linear_model.predict(train_features[:10])
```

```
1/1 [==============================] - 0s 85ms/step
array([[-0.215],
       [-0.007],
       [-0.929],
       [ 0.524],
       [ 1.24 ],
       [-0.715],
       [ 0.976],
       [ 0.967],
       [-0.991],
       [ 0.294]], dtype=float32)
```

```
linear_model.layers[1].kernel
```

```
<tf.Variable 'dense_1/kernel:0' shape=(9, 1) dtype=float32, numpy=
array([[-0.68 ],
       [-0.119],
       [-0.418],
       [ 0.739],
       [-0.68 ],
       [ 0.309],
       [ 0.627],
```

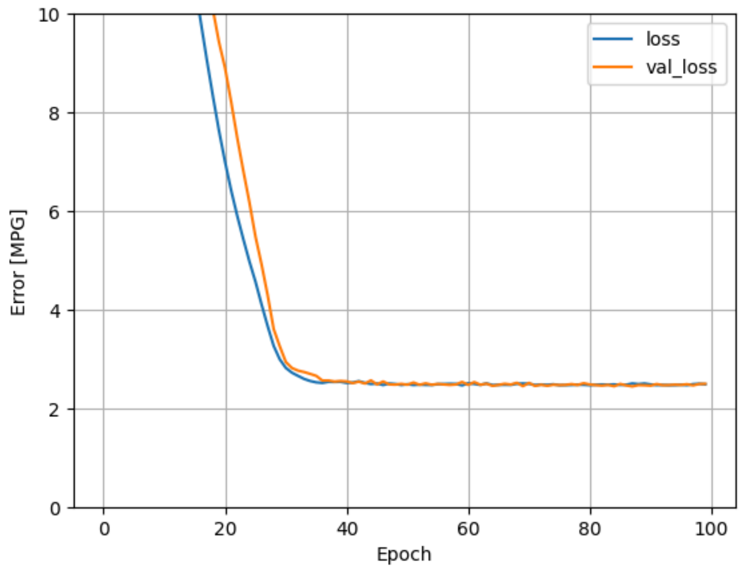
```
           [ 0.754],
           [ 0.09 ]], dtype=float32)>
```

```
linear_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.1),
    loss='mean_absolute_error')
```

```
%%time
history = linear_model.fit(
    train_features,
    train_labels,
    epochs=100,
    # Suppress logging.
    verbose=0,
    # Calculate validation results on 20% of the training data.
    validation_split = 0.2)
```

```
CPU times: user 5.68 s, sys: 171 ms, total: 5.85 s
Wall time: 7.59 s
```

```
plot_loss(history)
```




```
test_results['linear_model'] = linear_model.evaluate(
    test_features, test_labels, verbose=0)
```

Regression with a deep neural network (DNN)**bold text**

```
def build_and_compile_model(norm):
  model = keras.Sequential([
      norm,
      layers.Dense(64, activation='relu'),
      layers.Dense(64, activation='relu'),
      layers.Dense(1)
  ])

  model.compile(loss='mean_absolute_error',
                optimizer=tf.keras.optimizers.Adam(0.001))
  return model
```

Regression using a DNN and a single input

```
dnn_horsepower_model = build_and_compile_model(horsepower_normalizer)
```

```
dnn_horsepower_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_1 (Normaliza  (None, 1)                 3
```

```
 tion)

 dense_2 (Dense)             (None, 64)                128

 dense_3 (Dense)             (None, 64)                4160

 dense_4 (Dense)             (None, 1)                 65

=================================================================
Total params: 4356 (17.02 KB)
Trainable params: 4353 (17.00 KB)
Non-trainable params: 3 (16.00 Byte)
_________________________________________________________________
```

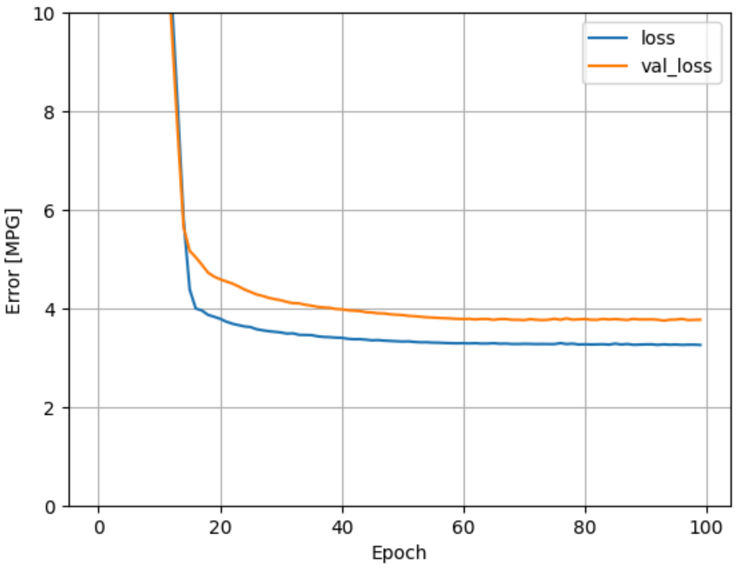
```
dnn_horsepower_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_1 (Normaliza  (None, 1)                 3
 tion)

 dense_2 (Dense)             (None, 64)                128

 dense_3 (Dense)             (None, 64)                4160

 dense_4 (Dense)             (None, 1)                 65

=================================================================
Total params: 4356 (17.02 KB)
Trainable params: 4353 (17.00 KB)
Non-trainable params: 3 (16.00 Byte)
_________________________________________________________________
```

```
%%time
history = dnn_horsepower_model.fit(
    train_features['Horsepower'],
    train_labels,
    validation_split=0.2,
    verbose=0, epochs=100)
```

```
CPU times: user 5.42 s, sys: 176 ms, total: 5.59 s
Wall time: 5.98 s
```

```
plot_loss(history)
```




```
test_results['dnn_horsepower_model'] = dnn_horsepower_model.evaluate(
    test_features['Horsepower'], test_labels,
    verbose=0)
```

Regression using a DNN and multiple inputs

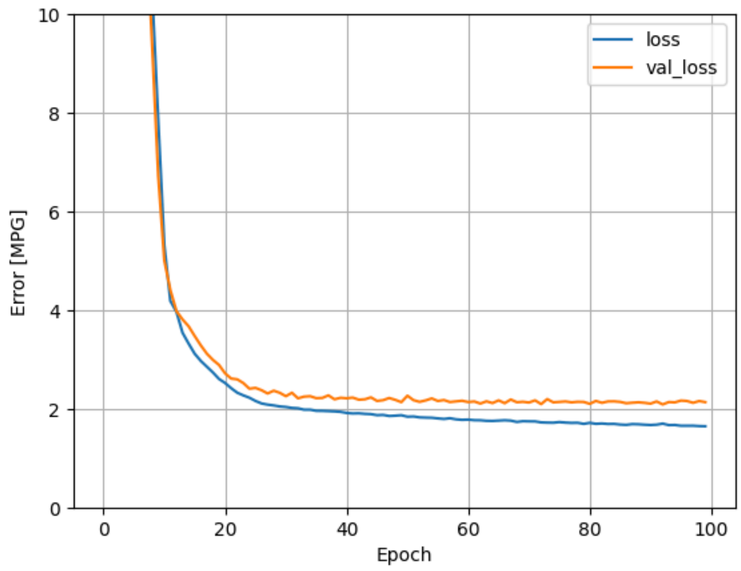
```
dnn_model = build_and_compile_model(normalizer)
dnn_model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizati  (None, 9)                 19
 on)

 dense_8 (Dense)             (None, 64)                640

 dense_9 (Dense)             (None, 64)                4160

 dense_10 (Dense)            (None, 1)                 65

=================================================================
Total params: 4884 (19.08 KB)
Trainable params: 4865 (19.00 KB)
Non-trainable params: 19 (80.00 Byte)
_________________________________________________________________
```

```
%%time
history = dnn_model.fit(
    train_features,
    train_labels,
    validation_split=0.2,
    verbose=0, epochs=100)
```

```
CPU times: user 5.21 s, sys: 197 ms, total: 5.4 s
Wall time: 5.85 s
```

```
plot_loss(history)
```



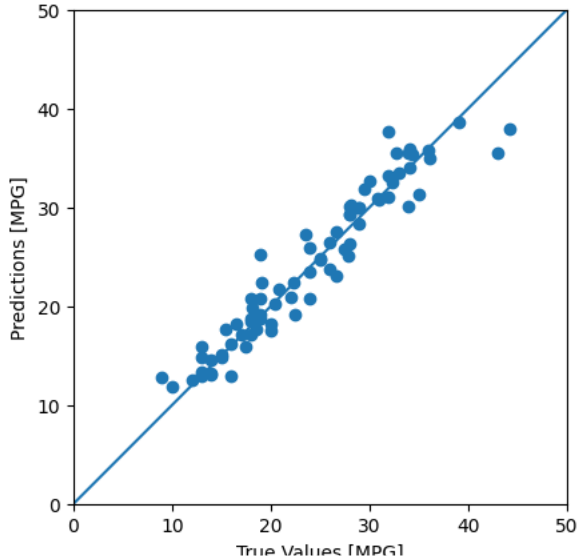

```
test_results['dnn_model'] = dnn_model.evaluate(test_features, test_labels, verbose=0)
```

```
pd.DataFrame(test_results, index=['Mean absolute error [MPG]']).T
```

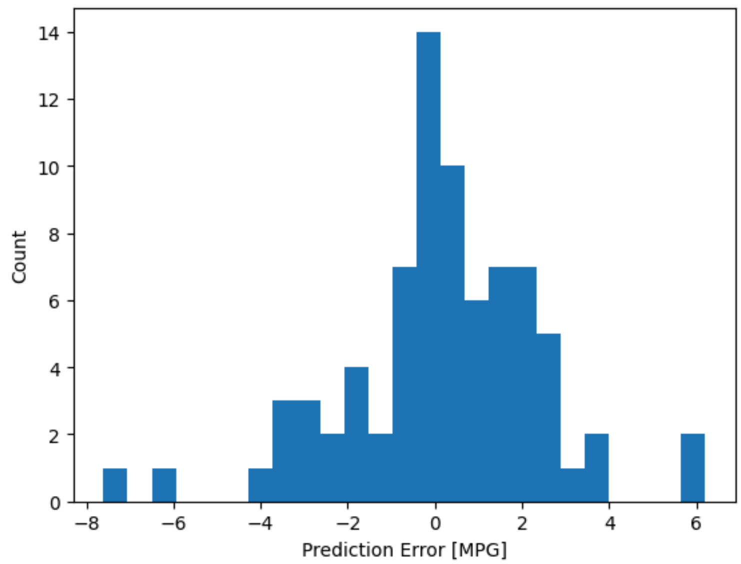
	Mean absolute error [MPG]
horsepower_model	3.649576
linear_model	2.463477
dnn_horsepower_model	2.944389
dnn_model	1.655435

```
test_predictions = dnn_model.predict(test_features).flatten()

a = plt.axes(aspect='equal')
plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values [MPG]')
plt.ylabel('Predictions [MPG]')
lims = [0, 50]
plt.xlim(lims)
plt.ylim(lims)
_ = plt.plot(lims, lims)
```

```
3/3 [==============================] - 0s 5ms/step
```




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




```
dnn_model.save('dnn_model.keras')
```

```
pd.DataFrame(test_results, index=['Mean absolute error [MPG]']).T
```

	Mean absolute error [MPG]
horsepower_model	3.649576
linear_model	2.463477
dnn_horsepower_model	2.944389
dnn_model	1.655435