In [9]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

In [10]:
```
df = pd.read_csv('./data/DATA/fake_reg.csv')
```

In [11]:
```
df.head()
```

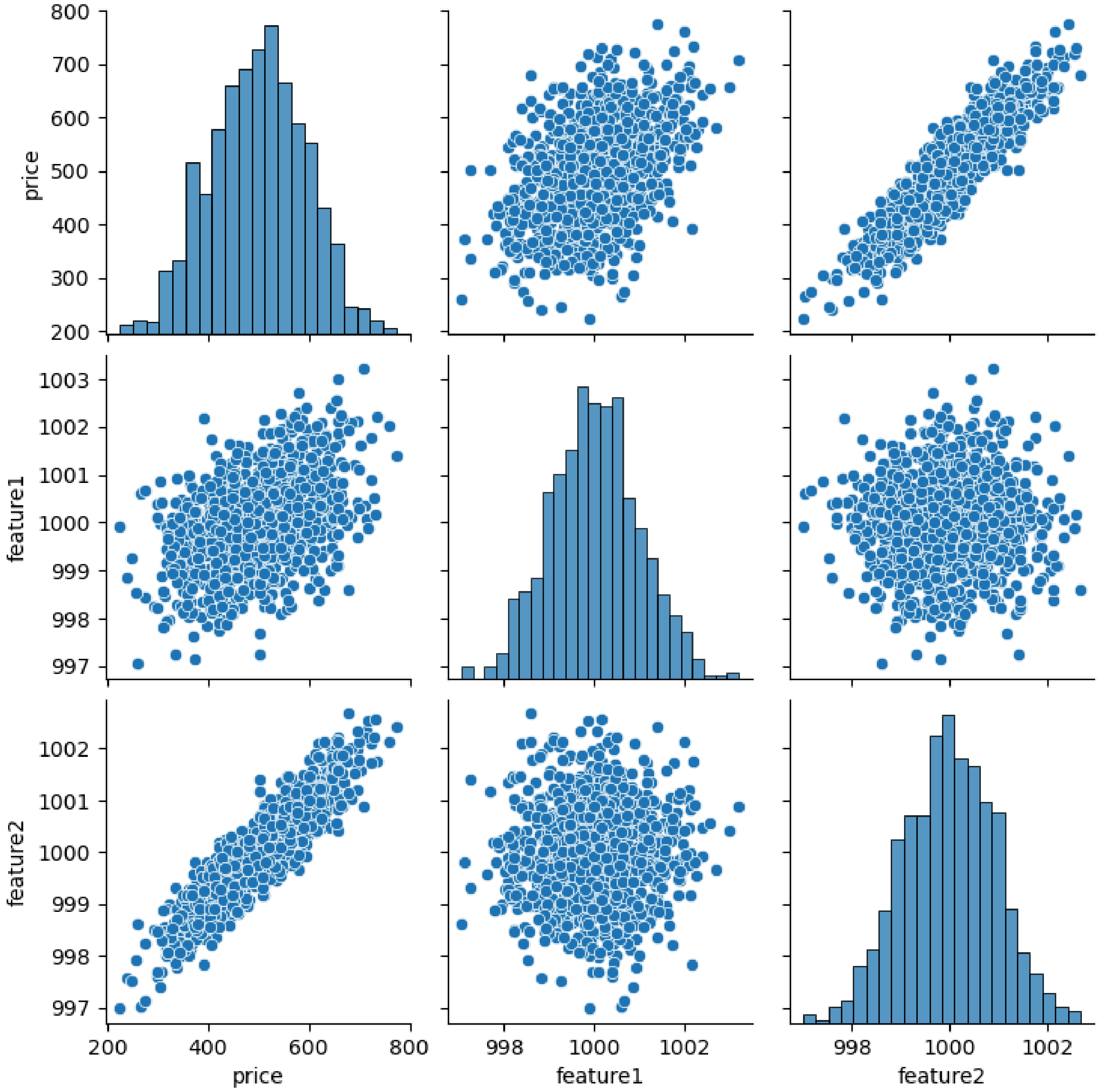
Out[11]:

	price	feature1	feature2
0	461.527929	999.787558	999.766096
1	548.130011	998.861615	1001.042403
2	410.297162	1000.070267	998.844015
3	540.382220	999.952251	1000.440940
4	546.024553	1000.446011	1000.338531

In [12]:
```
sns.pairplot(df)
```

```
C:\learnings\envs\deeplearning\lib\site-packages\seaborn\axisgrid.py:123: UserWar
ning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[12]:
```
<seaborn.axisgrid.PairGrid at 0x27b29bacfa0>
```



In [14]:
```
from sklearn.model_selection import train_test_split
```

In [15]:
```
X = df[['feature1', 'feature2']].values
```

In [16]:
```
y = df['price'].values
```

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

In [19]:
```
X_train.shape
```

Out[19]: (700, 2)

In [20]:
```
X_test.shape
```

Out[20]: (300, 2)

In [21]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [25]:
```
#help(MinMaxScaler)
```

In [22]:
```
scaler = MinMaxScaler()
```

In [23]:
```
scaler.fit(X_train)
```

Out[23]:

▾ MinMaxScaler

MinMaxScaler()

In [24]:
```
X_train = scaler.transform(X_train)
```

In [25]:
```
X_test = scaler.transform(X_test)
```

In [26]:
```
X_train.max()
```

Out[26]: 1.0

In [28]:
```
from tensorflow.keras.models import Sequential
```

In [29]:
```
from tensorflow.keras.layers import Dense
```

In [30]:
```
help(Sequential)
```

```
Help on class Sequential in module keras.engine.sequential:

class Sequential(keras.engine.functional.Functional)
 |  Sequential(*args, **kwargs)
 |
 |  `Sequential` groups a linear stack of layers into a `tf.keras.Model`.
 |
 |  `Sequential` provides training and inference features on this model.
 |
 |  Examples:
 |
 |  ```python
 |  # Optionally, the first layer can receive an `input_shape` argument:
 |  model = tf.keras.Sequential()
 |  model.add(tf.keras.layers.Dense(8, input_shape=(16,)))
 |  # Afterwards, we do automatic shape inference:
 |  model.add(tf.keras.layers.Dense(4))
 |
 |  # This is identical to the following:
 |  model = tf.keras.Sequential()
 |  model.add(tf.keras.Input(shape=(16,)))
 |  model.add(tf.keras.layers.Dense(8))
 |
 |  # Note that you can also omit the `input_shape` argument.
 |  # In that case the model doesn't have any weights until the first call
 |  # to a training/evaluation method (since it isn't yet built):
 |  model = tf.keras.Sequential()
 |  model.add(tf.keras.layers.Dense(8))
 |  model.add(tf.keras.layers.Dense(4))
 |  # model.weights not created yet
 |
 |  # Whereas if you specify the input shape, the model gets built
 |  # continuously as you are adding layers:
 |  model = tf.keras.Sequential()
 |  model.add(tf.keras.layers.Dense(8, input_shape=(16,)))
 |  model.add(tf.keras.layers.Dense(4))
 |  len(model.weights)
 |  # Returns "4"
 |
 |  # When using the delayed-build pattern (no input shape specified), you can
 |  # choose to manually build your model by calling
 |  # `build(batch_input_shape)`:
 |  model = tf.keras.Sequential()
 |  model.add(tf.keras.layers.Dense(8))
 |  model.add(tf.keras.layers.Dense(4))
 |  model.build((None, 16))
 |  len(model.weights)
 |  # Returns "4"
 |
 |  # Note that when using the delayed-build pattern (no input shape specified),
 |  # the model gets built the first time you call `fit`, `eval`, or `predict`,
 |  # or the first time you call the model on some input data.
 |  model = tf.keras.Sequential()
 |  model.add(tf.keras.layers.Dense(8))
 |  model.add(tf.keras.layers.Dense(1))
 |  model.compile(optimizer='sgd', loss='mse')
 |  # This builds the model for the first time:
 |  model.fit(x, y, batch_size=32, epochs=10)
 |  ```
 |
```

```
 |  Method resolution order:
 |      Sequential
 |      keras.engine.functional.Functional
 |      keras.engine.training.Model
 |      keras.engine.base_layer.Layer
 |      tensorflow.python.module.module.Module
 |      tensorflow.python.trackable.autotrackable.AutoTrackable
 |      tensorflow.python.trackable.base.Trackable
 |      keras.utils.version_utils.LayerVersionSelector
 |      keras.utils.version_utils.ModelVersionSelector
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, layers=None, name=None)
 |      Creates a `Sequential` model instance.
 |
 |      Args:
 |        layers: Optional list of layers to add to the model.
 |        name: Optional name for the model.
 |
 |  add(self, layer)
 |      Adds a layer instance on top of the layer stack.
 |
 |      Args:
 |          layer: layer instance.
 |
 |      Raises:
 |          TypeError: If `layer` is not a layer instance.
 |          ValueError: In case the `layer` argument does not
 |              know its input shape.
 |          ValueError: In case the `layer` argument has
 |              multiple output tensors, or is already connected
 |              somewhere else (forbidden in `Sequential` models).
 |
 |  build(self, input_shape=None)
 |      Builds the model based on input shapes received.
 |
 |      This is to be used for subclassed models, which do not know at
 |      instantiation time what their inputs look like.
 |
 |      This method only exists for users who want to call `model.build()` in a
 |      standalone way (as a substitute for calling the model on real data to
 |      build it). It will never be called by the framework (and thus it will
 |      never throw unexpected errors in an unrelated workflow).
 |
 |      Args:
 |       input_shape: Single tuple, `TensorShape` instance, or list/dict of
 |         shapes, where shapes are tuples, integers, or `TensorShape`
 |         instances.
 |
 |      Raises:
 |        ValueError:
 |          1. In case of invalid user-provided data (not of type tuple,
 |             list, `TensorShape`, or dict).
 |          2. If the model requires call arguments that are agnostic
 |             to the input shapes (positional or keyword arg in call
 |             signature).
 |          3. If not all layers were properly built.
 |          4. If float type inputs are not supported within the layers.
```

```
 |
 |        In each of these cases, the user should build their model by calling
 |        it on real tensor data.
 |
 |  call(self, inputs, training=None, mask=None)
 |      Calls the model on new inputs.
 |
 |      In this case `call` just reapplies
 |      all ops in the graph to the new inputs
 |      (e.g. build a new computational graph from the provided inputs).
 |
 |      Args:
 |          inputs: A tensor or list of tensors.
 |          training: Boolean or boolean scalar tensor, indicating whether to
 |              run the `Network` in training mode or inference mode.
 |          mask: A mask or list of masks. A mask can be
 |              either a tensor or None (no mask).
 |
 |      Returns:
 |          A tensor if there is a single output, or
 |          a list of tensors if there are more than one outputs.
 |
 |  compute_mask(self, inputs, mask)
 |      Computes an output mask tensor.
 |
 |      Args:
 |          inputs: Tensor or list of tensors.
 |          mask: Tensor or list of tensors.
 |
 |      Returns:
 |          None or a tensor (or list of tensors,
 |              one per output tensor of the layer).
 |
 |  compute_output_shape(self, input_shape)
 |      Computes the output shape of the layer.
 |
 |      This method will cause the layer's state to be built, if that has not
 |      happened before. This requires that the layer will later be used with
 |      inputs that match the input shape provided here.
 |
 |      Args:
 |          input_shape: Shape tuple (tuple of integers)
 |              or list of shape tuples (one per output tensor of the layer).
 |              Shape tuples can include None for free dimensions,
 |              instead of an integer.
 |
 |      Returns:
 |          An input shape tuple.
 |
 |  get_config(self)
 |      Returns the config of the `Model`.
 |
 |      Config is a Python dictionary (serializable) containing the
 |      configuration of an object, which in this case is a `Model`. This allows
 |      the `Model` to be be reinstantiated later (without its trained weights)
 |      from this configuration.
 |
 |      Note that `get_config()` does not guarantee to return a fresh copy of
 |      dict every time it is called. The callers should make a copy of the
 |      returned dict if they want to modify it.
```

```
 |
 |      Developers of subclassed `Model` are advised to override this method,
 |      and continue to update the dict from `super(MyModel, self).get_config()`
 |      to provide the proper configuration of this `Model`. The default config
 |      is an empty dict. Optionally, raise `NotImplementedError` to allow Keras
 |      to attempt a default serialization.
 |
 |      Returns:
 |          Python dictionary containing the configuration of this `Model`.
 |
 |  pop(self)
 |      Removes the last layer in the model.
 |
 |      Raises:
 |          TypeError: if there are no layers in the model.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  from_config(config, custom_objects=None) from builtins.type
 |      Creates a layer from its config.
 |
 |      This method is the reverse of `get_config`,
 |      capable of instantiating the same layer from the config
 |      dictionary. It does not handle layer connectivity
 |      (handled by Network), nor weights (handled by `set_weights`).
 |
 |      Args:
 |          config: A Python dictionary, typically the
 |              output of get_config.
 |
 |      Returns:
 |          A layer instance.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties defined here:
 |
 |  layers
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  input_spec
 |      `InputSpec` instance(s) describing the input format for this layer.
 |
 |      When you create a layer subclass, you can set `self.input_spec` to
 |      enable the layer to run input compatibility checks when it is called.
 |      Consider a `Conv2D` layer: it can only be called on a single input
 |      tensor of rank 4. As such, you can set, in `__init__()`:
 |
 |      ```python
 |      self.input_spec = tf.keras.layers.InputSpec(ndim=4)
 |      ```
 |
 |      Now, if you try to call the layer on an input that isn't rank 4
 |      (for instance, an input of shape `(2,)`, it will raise a
 |      nicely-formatted error:
 |
 |      ```
 |      ValueError: Input 0 of layer conv2d is incompatible with the layer:
```

```
 |      expected ndim=4, found ndim=1. Full shape received: [2]
 |      ```
 |
 |      Input checks that can be specified via `input_spec` include:
 |      - Structure (e.g. a single input, a list of 2 inputs, etc)
 |      - Shape
 |      - Rank (ndim)
 |      - Dtype
 |
 |      For more information, see `tf.keras.layers.InputSpec`.
 |
 |      Returns:
 |        A `tf.keras.layers.InputSpec` instance, or nested structure thereof.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from keras.engine.functional.Functional:
 |
 |  get_weight_paths(self)
 |      Retrieve all the variables and their paths for the model.
 |
 |      The variable path (string) is a stable key to indentify a `tf.Variable`
 |      instance owned by the model. It can be used to specify variable-specific
 |      configurations (e.g. DTensor, quantization) from a global view.
 |
 |      This method returns a dict with weight object paths as keys
 |      and the corresponding `tf.Variable` instances as values.
 |
 |      Note that if the model is a subclassed model and the weights haven't
 |      been initialized, an empty dict will be returned.
 |
 |      Returns:
 |          A dict where keys are variable paths and values are `tf.Variable`
 |           instances.
 |
 |      Example:
 |
 |      ```python
 |      class SubclassModel(tf.keras.Model):
 |
 |        def __init__(self, name=None):
 |          super().__init__(name=name)
 |          self.d1 = tf.keras.layers.Dense(10)
 |          self.d2 = tf.keras.layers.Dense(20)
 |
 |        def call(self, inputs):
 |          x = self.d1(inputs)
 |          return self.d2(x)
 |
 |      model = SubclassModel()
 |      model(tf.zeros((10, 10)))
 |      weight_paths = model.get_weight_paths()
 |      # weight_paths:
 |      # {
 |      #    'd1.kernel': model.d1.kernel,
 |      #    'd1.bias': model.d1.bias,
 |      #    'd2.kernel': model.d2.kernel,
 |      #    'd2.bias': model.d2.bias,
 |      # }
 |
 |      # Functional model
```

```
 |      inputs = tf.keras.Input((10,), batch_size=10)
 |      x = tf.keras.layers.Dense(20, name='d1')(inputs)
 |      output = tf.keras.layers.Dense(30, name='d2')(x)
 |      model = tf.keras.Model(inputs, output)
 |      d1 = model.layers[1]
 |      d2 = model.layers[2]
 |      weight_paths = model.get_weight_paths()
 |      # weight_paths:
 |      # {
 |      #    'd1.kernel': d1.kernel,
 |      #    'd1.bias': d1.bias,
 |      #    'd2.kernel': d2.kernel,
 |      #    'd2.bias': d2.bias,
 |      # }
 |      ```
 |  
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from keras.engine.functional.Functional:
 |  
 |  input
 |      Retrieves the input tensor(s) of a layer.
 |      
 |      Only applicable if the layer has exactly one input,
 |      i.e. if it is connected to one incoming layer.
 |      
 |      Returns:
 |          Input tensor or list of input tensors.
 |      
 |      Raises:
 |        RuntimeError: If called in Eager mode.
 |        AttributeError: If no inbound nodes are found.
 |  
 |  input_shape
 |      Retrieves the input shape(s) of a layer.
 |      
 |      Only applicable if the layer has exactly one input,
 |      i.e. if it is connected to one incoming layer, or if all inputs
 |      have the same shape.
 |      
 |      Returns:
 |          Input shape, as an integer shape tuple
 |          (or list of shape tuples, one tuple per input tensor).
 |      
 |      Raises:
 |          AttributeError: if the layer has no defined input_shape.
 |          RuntimeError: if called in Eager mode.
 |  
 |  output
 |      Retrieves the output tensor(s) of a layer.
 |      
 |      Only applicable if the layer has exactly one output,
 |      i.e. if it is connected to one incoming layer.
 |      
 |      Returns:
 |        Output tensor or list of output tensors.
 |      
 |      Raises:
 |        AttributeError: if the layer is connected to more than one incoming
 |          layers.
 |        RuntimeError: if called in Eager mode.
```

```
 |
 |  output_shape
 |      Retrieves the output shape(s) of a layer.
 |
 |      Only applicable if the layer has one output,
 |      or if all outputs have the same shape.
 |
 |      Returns:
 |          Output shape, as an integer shape tuple
 |          (or list of shape tuples, one tuple per output tensor).
 |
 |      Raises:
 |          AttributeError: if the layer has no defined output shape.
 |          RuntimeError: if called in Eager mode.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from keras.engine.training.Model:
 |
 |  __call__(self, *args, **kwargs)
 |
 |  __copy__(self)
 |
 |  __deepcopy__(self, memo)
 |
 |  __reduce__(self)
 |      Helper for pickle.
 |
 |  __setattr__(self, name, value)
 |      Support self.foo = trackable syntax.
 |
 |  compile(self, optimizer='rmsprop', loss=None, metrics=None, loss_weights=Non
e, weighted_metrics=None, run_eagerly=None, steps_per_execution=None, jit_compile
=None, **kwargs)
 |      Configures the model for training.
 |
 |      Example:
 |
 |      ```python
 |      model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
 |                    loss=tf.keras.losses.BinaryCrossentropy(),
 |                    metrics=[tf.keras.metrics.BinaryAccuracy(),
 |                             tf.keras.metrics.FalseNegatives()])
 |      ```
 |
 |      Args:
 |          optimizer: String (name of optimizer) or optimizer instance. See
 |            `tf.keras.optimizers`.
 |          loss: Loss function. May be a string (name of loss function), or
 |            a `tf.keras.losses.Loss` instance. See `tf.keras.losses`. A loss
 |            function is any callable with the signature `loss = fn(y_true,
 |            y_pred)`, where `y_true` are the ground truth values, and
 |            `y_pred` are the model's predictions.
 |            `y_true` should have shape
 |            `(batch_size, d0, .. dN)` (except in the case of
 |            sparse loss functions such as
 |            sparse categorical crossentropy which expects integer arrays of
 |            shape `(batch_size, d0, .. dN-1)`).
 |            `y_pred` should have shape `(batch_size, d0, .. dN)`.
 |            The loss function should return a float tensor.
 |            If a custom `Loss` instance is
```

```
 |              used and reduction is set to `None`, return value has shape
 |              `(batch_size, d0, .. dN-1)` i.e. per-sample or per-timestep loss
 |              values; otherwise, it is a scalar. If the model has multiple
 |              outputs, you can use a different loss on each output by passing a
 |              dictionary or a list of losses. The loss value that will be
 |              minimized by the model will then be the sum of all individual
 |              losses, unless `loss_weights` is specified.
 |            metrics: List of metrics to be evaluated by the model during
 |              training and testing. Each of this can be a string (name of a
 |              built-in function), function or a `tf.keras.metrics.Metric`
 |              instance. See `tf.keras.metrics`. Typically you will use
 |              `metrics=['accuracy']`.
 |              A function is any callable with the signature `result = fn(y_true,
 |              y_pred)`. To specify different metrics for different outputs of a
 |              multi-output model, you could also pass a dictionary, such as
 |              `metrics={'output_a':'accuracy', 'output_b':['accuracy', 'mse']}`.
 |              You can also pass a list to specify a metric or a list of metrics
 |              for each output, such as
 |              `metrics=[['accuracy'], ['accuracy', 'mse']]`
 |              or `metrics=['accuracy', ['accuracy', 'mse']]`. When you pass the
 |              strings 'accuracy' or 'acc', we convert this to one of
 |              `tf.keras.metrics.BinaryAccuracy`,
 |              `tf.keras.metrics.CategoricalAccuracy`,
 |              `tf.keras.metrics.SparseCategoricalAccuracy` based on the loss
 |              function used and the model output shape. We do a similar
 |              conversion for the strings 'crossentropy' and 'ce' as well.
 |              The metrics passed here are evaluated without sample weighting; if
 |              you would like sample weighting to apply, you can specify your
 |              metrics via the `weighted_metrics` argument instead.
 |            loss_weights: Optional list or dictionary specifying scalar
 |              coefficients (Python floats) to weight the loss contributions of
 |              different model outputs. The loss value that will be minimized by
 |              the model will then be the *weighted sum* of all individual
 |              losses, weighted by the `loss_weights` coefficients.  If a list,
 |              it is expected to have a 1:1 mapping to the model's outputs. If a
 |              dict, it is expected to map output names (strings) to scalar
 |              coefficients.
 |            weighted_metrics: List of metrics to be evaluated and weighted by
 |              `sample_weight` or `class_weight` during training and testing.
 |            run_eagerly: Bool. Defaults to `False`. If `True`, this `Model`'s
 |              logic will not be wrapped in a `tf.function`. Recommended to leave
 |              this as `None` unless your `Model` cannot be run inside a
 |              `tf.function`. `run_eagerly=True` is not supported when using
 |              `tf.distribute.experimental.ParameterServerStrategy`.
 |            steps_per_execution: Int. Defaults to 1. The number of batches to
 |              run during each `tf.function` call. Running multiple batches
 |              inside a single `tf.function` call can greatly improve performance
 |              on TPUs or small models with a large Python overhead. At most, one
 |              full epoch will be run each execution. If a number larger than the
 |              size of the epoch is passed, the execution will be truncated to
 |              the size of the epoch. Note that if `steps_per_execution` is set
 |              to `N`, `Callback.on_batch_begin` and `Callback.on_batch_end`
 |              methods will only be called every `N` batches (i.e. before/after
 |              each `tf.function` execution).
 |            jit_compile: If `True`, compile the model training step with XLA.
 |              [XLA](https://www.tensorflow.org/xla) is an optimizing compiler
 |              for machine learning.
 |              `jit_compile` is not enabled for by default.
 |              This option cannot be enabled with `run_eagerly=True`.
 |              Note that `jit_compile=True`
```

```
 |          may not necessarily work for all models.
 |          For more information on supported operations please refer to the
 |          [XLA documentation](https://www.tensorflow.org/xla).
 |          Also refer to
 |          [known XLA issues](https://www.tensorflow.org/xla/known_issues)
 |          for more details.
 |        **kwargs: Arguments supported for backwards compatibility only.
 |
 |  compute_loss(self, x=None, y=None, y_pred=None, sample_weight=None)
 |      Compute the total loss, validate it, and return it.
 |
 |      Subclasses can optionally override this method to provide custom loss
 |      computation logic.
 |
 |      Example:
 |      ```python
 |      class MyModel(tf.keras.Model):
 |
 |        def __init__(self, *args, **kwargs):
 |          super(MyModel, self).__init__(*args, **kwargs)
 |          self.loss_tracker = tf.keras.metrics.Mean(name='loss')
 |
 |        def compute_loss(self, x, y, y_pred, sample_weight):
 |          loss = tf.reduce_mean(tf.math.squared_difference(y_pred, y))
 |          loss += tf.add_n(self.losses)
 |          self.loss_tracker.update_state(loss)
 |          return loss
 |
 |        def reset_metrics(self):
 |          self.loss_tracker.reset_states()
 |
 |        @property
 |        def metrics(self):
 |          return [self.loss_tracker]
 |
 |      tensors = tf.random.uniform((10, 10)), tf.random.uniform((10,))
 |      dataset = tf.data.Dataset.from_tensor_slices(tensors).repeat().batch(1)
 |
 |      inputs = tf.keras.layers.Input(shape=(10,), name='my_input')
 |      outputs = tf.keras.layers.Dense(10)(inputs)
 |      model = MyModel(inputs, outputs)
 |      model.add_loss(tf.reduce_sum(outputs))
 |
 |      optimizer = tf.keras.optimizers.SGD()
 |      model.compile(optimizer, loss='mse', steps_per_execution=10)
 |      model.fit(dataset, epochs=2, steps_per_epoch=10)
 |      print('My custom loss: ', model.loss_tracker.result().numpy())
 |      ```
 |
 |      Args:
 |        x: Input data.
 |        y: Target data.
 |        y_pred: Predictions returned by the model (output of `model(x)`)
 |        sample_weight: Sample weights for weighting the loss function.
 |
 |      Returns:
 |        The total loss as a `tf.Tensor`, or `None` if no loss results (which
 |        is the case when called by `Model.test_step`).
 |
 |  compute_metrics(self, x, y, y_pred, sample_weight)
```

```
 |      Update metric states and collect all metrics to be returned.
 |
 |      Subclasses can optionally override this method to provide custom metric
 |      updating and collection logic.
 |
 |      Example:
 |      ```python
 |      class MyModel(tf.keras.Sequential):
 |
 |        def compute_metrics(self, x, y, y_pred, sample_weight):
 |
 |          # This super call updates `self.compiled_metrics` and returns
 |          # results for all metrics listed in `self.metrics`.
 |          metric_results = super(MyModel, self).compute_metrics(
 |              x, y, y_pred, sample_weight)
 |
 |          # Note that `self.custom_metric` is not listed in `self.metrics`.
 |          self.custom_metric.update_state(x, y, y_pred, sample_weight)
 |          metric_results['custom_metric_name'] = self.custom_metric.result()
 |          return metric_results
 |      ```
 |
 |      Args:
 |        x: Input data.
 |        y: Target data.
 |        y_pred: Predictions returned by the model (output of `model.call(x)`)
 |        sample_weight: Sample weights for weighting the loss function.
 |
 |      Returns:
 |        A `dict` containing values that will be passed to
 |        `tf.keras.callbacks.CallbackList.on_train_batch_end()`. Typically, the
 |        values of the metrics listed in `self.metrics` are returned. Example:
 |        `{'loss': 0.2, 'accuracy': 0.7}`.
 |
 |  evaluate(self, x=None, y=None, batch_size=None, verbose='auto', sample_weight
=None, steps=None, callbacks=None, max_queue_size=10, workers=1, use_multiprocess
ing=False, return_dict=False, **kwargs)
 |      Returns the loss value & metrics values for the model in test mode.
 |
 |      Computation is done in batches (see the `batch_size` arg.)
 |
 |      Args:
 |          x: Input data. It could be:
 |            - A Numpy array (or array-like), or a list of arrays
 |              (in case the model has multiple inputs).
 |            - A TensorFlow tensor, or a list of tensors
 |              (in case the model has multiple inputs).
 |            - A dict mapping input names to the corresponding array/tensors,
 |              if the model has named inputs.
 |            - A `tf.data` dataset. Should return a tuple
 |              of either `(inputs, targets)` or
 |              `(inputs, targets, sample_weights)`.
 |            - A generator or `keras.utils.Sequence` returning `(inputs,
 |              targets)` or `(inputs, targets, sample_weights)`.
 |            A more detailed description of unpacking behavior for iterator
 |            types (Dataset, generator, Sequence) is given in the `Unpacking
 |            behavior for iterator-like inputs` section of `Model.fit`.
 |          y: Target data. Like the input data `x`, it could be either Numpy
 |            array(s) or TensorFlow tensor(s). It should be consistent with `x`
 |            (you cannot have Numpy inputs and tensor targets, or inversely).
```

```
 |          If `x` is a dataset, generator or `keras.utils.Sequence` instance,
 |          `y` should not be specified (since targets will be obtained from
 |          the iterator/dataset).
 |        batch_size: Integer or `None`. Number of samples per batch of
 |          computation. If unspecified, `batch_size` will default to 32. Do
 |          not specify the `batch_size` if your data is in the form of a
 |          dataset, generators, or `keras.utils.Sequence` instances (since
 |          they generate batches).
 |        verbose: `"auto"`, 0, 1, or 2. Verbosity mode.
 |            0 = silent, 1 = progress bar, 2 = single line.
 |            `"auto"` defaults to 1 for most cases, and to 2 when used with
 |            `ParameterServerStrategy`. Note that the progress bar is not
 |            particularly useful when logged to a file, so `verbose=2` is
 |            recommended when not running interactively (e.g. in a production
 |            environment).
 |        sample_weight: Optional Numpy array of weights for the test samples,
 |          used for weighting the loss function. You can either pass a flat
 |          (1D) Numpy array with the same length as the input samples
 |            (1:1 mapping between weights and samples), or in the case of
 |              temporal data, you can pass a 2D array with shape `(samples,
 |              sequence_length)`, to apply a different weight to every
 |              timestep of every sample. This argument is not supported when
 |              `x` is a dataset, instead pass sample weights as the third
 |              element of `x`.
 |        steps: Integer or `None`. Total number of steps (batches of samples)
 |          before declaring the evaluation round finished. Ignored with the
 |          default value of `None`. If x is a `tf.data` dataset and `steps`
 |          is None, 'evaluate' will run until the dataset is exhausted. This
 |          argument is not supported with array inputs.
 |        callbacks: List of `keras.callbacks.Callback` instances. List of
 |          callbacks to apply during evaluation. See
 |          [callbacks](/api_docs/python/tf/keras/callbacks).
 |        max_queue_size: Integer. Used for generator or
 |          `keras.utils.Sequence` input only. Maximum size for the generator
 |          queue. If unspecified, `max_queue_size` will default to 10.
 |        workers: Integer. Used for generator or `keras.utils.Sequence` input
 |          only. Maximum number of processes to spin up when using
 |          process-based threading. If unspecified, `workers` will default to
 |          1.
 |        use_multiprocessing: Boolean. Used for generator or
 |          `keras.utils.Sequence` input only. If `True`, use process-based
 |          threading. If unspecified, `use_multiprocessing` will default to
 |          `False`. Note that because this implementation relies on
 |          multiprocessing, you should not pass non-picklable arguments to
 |          the generator as they can't be passed easily to children
 |          processes.
 |        return_dict: If `True`, loss and metric results are returned as a
 |          dict, with each key being the name of the metric. If `False`, they
 |          are returned as a list.
 |        **kwargs: Unused at this time.
 |
 |    See the discussion of `Unpacking behavior for iterator-like inputs` for
 |    `Model.fit`.
 |
 |    Returns:
 |        Scalar test loss (if the model has a single output and no metrics)
 |        or list of scalars (if the model has multiple outputs
 |        and/or metrics). The attribute `model.metrics_names` will give you
 |        the display labels for the scalar outputs.
 |
```

```
 |      Raises:
 |          RuntimeError: If `model.evaluate` is wrapped in a `tf.function`.
 |
 |  evaluate_generator(self, generator, steps=None, callbacks=None, max_queue_siz
e=10, workers=1, use_multiprocessing=False, verbose=0)
 |      Evaluates the model on a data generator.
 |
 |      DEPRECATED:
 |        `Model.evaluate` now supports generators, so there is no longer any
 |        need to use this endpoint.
 |
 |  fit(self, x=None, y=None, batch_size=None, epochs=1, verbose='auto', callback
s=None, validation_split=0.0, validation_data=None, shuffle=True, class_weight=No
ne, sample_weight=None, initial_epoch=0, steps_per_epoch=None, validation_steps=N
one, validation_batch_size=None, validation_freq=1, max_queue_size=10, workers=1,
use_multiprocessing=False)
 |      Trains the model for a fixed number of epochs (iterations on a dataset).
 |
 |      Args:
 |          x: Input data. It could be:
 |            - A Numpy array (or array-like), or a list of arrays
 |              (in case the model has multiple inputs).
 |            - A TensorFlow tensor, or a list of tensors
 |              (in case the model has multiple inputs).
 |            - A dict mapping input names to the corresponding array/tensors,
 |              if the model has named inputs.
 |            - A `tf.data` dataset. Should return a tuple
 |              of either `(inputs, targets)` or
 |              `(inputs, targets, sample_weights)`.
 |            - A generator or `keras.utils.Sequence` returning `(inputs,
 |              targets)` or `(inputs, targets, sample_weights)`.
 |            - A `tf.keras.utils.experimental.DatasetCreator`, which wraps a
 |              callable that takes a single argument of type
 |              `tf.distribute.InputContext`, and returns a `tf.data.Dataset`.
 |              `DatasetCreator` should be used when users prefer to specify the
 |              per-replica batching and sharding logic for the `Dataset`.
 |              See `tf.keras.utils.experimental.DatasetCreator` doc for more
 |              information.
 |            A more detailed description of unpacking behavior for iterator
 |            types (Dataset, generator, Sequence) is given below. If these
 |            include `sample_weights` as a third component, note that sample
 |            weighting applies to the `weighted_metrics` argument but not the
 |            `metrics` argument in `compile()`. If using
 |            `tf.distribute.experimental.ParameterServerStrategy`, only
 |            `DatasetCreator` type is supported for `x`.
 |          y: Target data. Like the input data `x`,
 |            it could be either Numpy array(s) or TensorFlow tensor(s).
 |            It should be consistent with `x` (you cannot have Numpy inputs and
 |            tensor targets, or inversely). If `x` is a dataset, generator,
 |            or `keras.utils.Sequence` instance, `y` should
 |            not be specified (since targets will be obtained from `x`).
 |          batch_size: Integer or `None`.
 |              Number of samples per gradient update.
 |              If unspecified, `batch_size` will default to 32.
 |              Do not specify the `batch_size` if your data is in the
 |              form of datasets, generators, or `keras.utils.Sequence`
 |              instances (since they generate batches).
 |          epochs: Integer. Number of epochs to train the model.
 |              An epoch is an iteration over the entire `x` and `y`
 |              data provided
```

```
|                   (unless the `steps_per_epoch` flag is set to
|                   something other than None).
|                   Note that in conjunction with `initial_epoch`,
|                   `epochs` is to be understood as "final epoch".
|                   The model is not trained for a number of iterations
|                   given by `epochs`, but merely until the epoch
|                   of index `epochs` is reached.
|               verbose: 'auto', 0, 1, or 2. Verbosity mode.
|                   0 = silent, 1 = progress bar, 2 = one line per epoch.
|                   'auto' defaults to 1 for most cases, but 2 when used with
|                   `ParameterServerStrategy`. Note that the progress bar is not
|                   particularly useful when logged to a file, so verbose=2 is
|                   recommended when not running interactively (eg, in a production
|                   environment).
|               callbacks: List of `keras.callbacks.Callback` instances.
|                   List of callbacks to apply during training.
|                   See `tf.keras.callbacks`. Note
|                   `tf.keras.callbacks.ProgbarLogger` and
|                   `tf.keras.callbacks.History` callbacks are created automatically
|                   and need not be passed into `model.fit`.
|                   `tf.keras.callbacks.ProgbarLogger` is created or not based on
|                   `verbose` argument to `model.fit`.
|                   Callbacks with batch-level calls are currently unsupported with
|                   `tf.distribute.experimental.ParameterServerStrategy`, and users
|                   are advised to implement epoch-level calls instead with an
|                   appropriate `steps_per_epoch` value.
|               validation_split: Float between 0 and 1.
|                   Fraction of the training data to be used as validation data.
|                   The model will set apart this fraction of the training data,
|                   will not train on it, and will evaluate
|                   the loss and any model metrics
|                   on this data at the end of each epoch.
|                   The validation data is selected from the last samples
|                   in the `x` and `y` data provided, before shuffling. This
|                   argument is not supported when `x` is a dataset, generator or
|                   `keras.utils.Sequence` instance.
|                   If both `validation_data` and `validation_split` are provided,
|                   `validation_data` will override `validation_split`.
|                   `validation_split` is not yet supported with
|                   `tf.distribute.experimental.ParameterServerStrategy`.
|               validation_data: Data on which to evaluate
|                   the loss and any model metrics at the end of each epoch.
|                   The model will not be trained on this data. Thus, note the fact
|                   that the validation loss of data provided using
|                   `validation_split` or `validation_data` is not affected by
|                   regularization layers like noise and dropout.
|                   `validation_data` will override `validation_split`.
|                   `validation_data` could be:
|                     - A tuple `(x_val, y_val)` of Numpy arrays or tensors.
|                     - A tuple `(x_val, y_val, val_sample_weights)` of NumPy
|                       arrays.
|                     - A `tf.data.Dataset`.
|                     - A Python generator or `keras.utils.Sequence` returning
|                     `(inputs, targets)` or `(inputs, targets, sample_weights)`.
|                   `validation_data` is not yet supported with
|                   `tf.distribute.experimental.ParameterServerStrategy`.
|               shuffle: Boolean (whether to shuffle the training data
|                   before each epoch) or str (for 'batch'). This argument is
|                   ignored when `x` is a generator or an object of tf.data.Dataset.
|                   'batch' is a special option for dealing
```

```
|               with the limitations of HDF5 data; it shuffles in batch-sized
|               chunks. Has no effect when `steps_per_epoch` is not `None`.
|           class_weight: Optional dictionary mapping class indices (integers)
|               to a weight (float) value, used for weighting the loss function
|               (during training only).
|               This can be useful to tell the model to
|               "pay more attention" to samples from
|               an under-represented class.
|           sample_weight: Optional Numpy array of weights for
|               the training samples, used for weighting the loss function
|               (during training only). You can either pass a flat (1D)
|               Numpy array with the same length as the input samples
|               (1:1 mapping between weights and samples),
|               or in the case of temporal data,
|               you can pass a 2D array with shape
|               `(samples, sequence_length)`,
|               to apply a different weight to every timestep of every sample.
|               This argument is not supported when `x` is a dataset, generator,
|               or `keras.utils.Sequence` instance, instead provide the
|               sample_weights as the third element of `x`.
|               Note that sample weighting does not apply to metrics specified
|               via the `metrics` argument in `compile()`. To apply sample
|               weighting to your metrics, you can specify them via the
|               `weighted_metrics` in `compile()` instead.
|           initial_epoch: Integer.
|               Epoch at which to start training
|               (useful for resuming a previous training run).
|           steps_per_epoch: Integer or `None`.
|               Total number of steps (batches of samples)
|               before declaring one epoch finished and starting the
|               next epoch. When training with input tensors such as
|               TensorFlow data tensors, the default `None` is equal to
|               the number of samples in your dataset divided by
|               the batch size, or 1 if that cannot be determined. If x is a
|               `tf.data` dataset, and 'steps_per_epoch'
|               is None, the epoch will run until the input dataset is
|               exhausted.  When passing an infinitely repeating dataset, you
|               must specify the `steps_per_epoch` argument. If
|               `steps_per_epoch=-1` the training will run indefinitely with an
|               infinitely repeating dataset.  This argument is not supported
|               with array inputs.
|               When using `tf.distribute.experimental.ParameterServerStrategy`:
|                 * `steps_per_epoch=None` is not supported.
|           validation_steps: Only relevant if `validation_data` is provided and
|               is a `tf.data` dataset. Total number of steps (batches of
|               samples) to draw before stopping when performing validation
|               at the end of every epoch. If 'validation_steps' is None,
|               validation will run until the `validation_data` dataset is
|               exhausted. In the case of an infinitely repeated dataset, it
|               will run into an infinite loop. If 'validation_steps' is
|               specified and only part of the dataset will be consumed, the
|               evaluation will start from the beginning of the dataset at each
|               epoch. This ensures that the same validation samples are used
|               every time.
|           validation_batch_size: Integer or `None`.
|               Number of samples per validation batch.
|               If unspecified, will default to `batch_size`.
|               Do not specify the `validation_batch_size` if your data is in
|               the form of datasets, generators, or `keras.utils.Sequence`
|               instances (since they generate batches).
```

```
 |          validation_freq: Only relevant if validation data is provided.
 |            Integer or `collections.abc.Container` instance (e.g. list, tuple,
 |            etc.).  If an integer, specifies how many training epochs to run
 |            before a new validation run is performed, e.g. `validation_freq=2`
 |            runs validation every 2 epochs. If a Container, specifies the
 |            epochs on which to run validation, e.g.
 |            `validation_freq=[1, 2, 10]` runs validation at the end of the
 |            1st, 2nd, and 10th epochs.
 |          max_queue_size: Integer. Used for generator or
 |            `keras.utils.Sequence` input only. Maximum size for the generator
 |            queue.  If unspecified, `max_queue_size` will default to 10.
 |          workers: Integer. Used for generator or `keras.utils.Sequence` input
 |              only. Maximum number of processes to spin up
 |              when using process-based threading. If unspecified, `workers`
 |              will default to 1.
 |          use_multiprocessing: Boolean. Used for generator or
 |              `keras.utils.Sequence` input only. If `True`, use process-based
 |              threading. If unspecified, `use_multiprocessing` will default to
 |              `False`. Note that because this implementation relies on
 |              multiprocessing, you should not pass non-picklable arguments to
 |              the generator as they can't be passed easily to children
 |              processes.
 |
 |      Unpacking behavior for iterator-like inputs:
 |          A common pattern is to pass a tf.data.Dataset, generator, or
 |        tf.keras.utils.Sequence to the `x` argument of fit, which will in fact
 |        yield not only features (x) but optionally targets (y) and sample
 |        weights.  Keras requires that the output of such iterator-likes be
 |        unambiguous. The iterator should return a tuple of length 1, 2, or 3,
 |        where the optional second and third elements will be used for y and
 |        sample_weight respectively. Any other type provided will be wrapped in
 |        a length one tuple, effectively treating everything as 'x'. When
 |        yielding dicts, they should still adhere to the top-level tuple
 |        structure.
 |        e.g. `({"x0": x0, "x1": x1}, y)`. Keras will not attempt to separate
 |        features, targets, and weights from the keys of a single dict.
 |          A notable unsupported data type is the namedtuple. The reason is
 |        that it behaves like both an ordered datatype (tuple) and a mapping
 |        datatype (dict). So given a namedtuple of the form:
 |            `namedtuple("example_tuple", ["y", "x"])`
 |        it is ambiguous whether to reverse the order of the elements when
 |        interpreting the value. Even worse is a tuple of the form:
 |            `namedtuple("other_tuple", ["x", "y", "z"])`
 |        where it is unclear if the tuple was intended to be unpacked into x,
 |        y, and sample_weight or passed through as a single element to `x`. As
 |        a result the data processing code will simply raise a ValueError if it
 |        encounters a namedtuple. (Along with instructions to remedy the
 |        issue.)
 |
 |      Returns:
 |          A `History` object. Its `History.history` attribute is
 |          a record of training loss values and metrics values
 |          at successive epochs, as well as validation loss values
 |          and validation metrics values (if applicable).
 |
 |      Raises:
 |          RuntimeError: 1. If the model was never compiled or,
 |          2. If `model.fit` is  wrapped in `tf.function`.
 |
 |          ValueError: In case of mismatch between the provided input data
```

```
 |              and what the model expects or when the input data is empty.
 |
 |  fit_generator(self, generator, steps_per_epoch=None, epochs=1, verbose=1, cal
lbacks=None, validation_data=None, validation_steps=None, validation_freq=1, clas
s_weight=None, max_queue_size=10, workers=1, use_multiprocessing=False, shuffle=T
rue, initial_epoch=0)
 |      Fits the model on data yielded batch-by-batch by a Python generator.
 |
 |      DEPRECATED:
 |        `Model.fit` now supports generators, so there is no longer any need to
 |        use this endpoint.
 |
 |  get_layer(self, name=None, index=None)
 |      Retrieves a layer based on either its name (unique) or index.
 |
 |      If `name` and `index` are both provided, `index` will take precedence.
 |      Indices are based on order of horizontal graph traversal (bottom-up).
 |
 |      Args:
 |          name: String, name of layer.
 |          index: Integer, index of layer.
 |
 |      Returns:
 |          A layer instance.
 |
 |  get_weights(self)
 |      Retrieves the weights of the model.
 |
 |      Returns:
 |          A flat list of Numpy arrays.
 |
 |  load_weights(self, filepath, by_name=False, skip_mismatch=False, options=Non
e)
 |      Loads all layer weights, either from a TensorFlow or an HDF5 weight file.
 |
 |      If `by_name` is False weights are loaded based on the network's
 |      topology. This means the architecture should be the same as when the
 |      weights were saved.  Note that layers that don't have weights are not
 |      taken into account in the topological ordering, so adding or removing
 |      layers is fine as long as they don't have weights.
 |
 |      If `by_name` is True, weights are loaded into layers only if they share
 |      the same name. This is useful for fine-tuning or transfer-learning
 |      models where some of the layers have changed.
 |
 |      Only topological loading (`by_name=False`) is supported when loading
 |      weights from the TensorFlow format. Note that topological loading
 |      differs slightly between TensorFlow and HDF5 formats for user-defined
 |      classes inheriting from `tf.keras.Model`: HDF5 loads based on a
 |      flattened list of weights, while the TensorFlow format loads based on
 |      the object-local names of attributes to which layers are assigned in the
 |      `Model`'s constructor.
 |
 |      Args:
 |          filepath: String, path to the weights file to load. For weight files
 |              in TensorFlow format, this is the file prefix (the same as was
 |              passed to `save_weights`). This can also be a path to a
 |              SavedModel saved from `model.save`.
 |          by_name: Boolean, whether to load weights by name or by topological
 |              order. Only topological loading is supported for weight files in
```

```
|               TensorFlow format.
|           skip_mismatch: Boolean, whether to skip loading of layers where
|               there is a mismatch in the number of weights, or a mismatch in
|               the shape of the weight (only valid when `by_name=True`).
|           options: Optional `tf.train.CheckpointOptions` object that specifies
|               options for loading weights.
|
|       Returns:
|           When loading a weight file in TensorFlow format, returns the same
|           status object as `tf.train.Checkpoint.restore`. When graph building,
|           restore ops are run automatically as soon as the network is built
|           (on first call for user-defined classes inheriting from `Model`,
|           immediately if it is already built).
|
|           When loading weights in HDF5 format, returns `None`.
|
|       Raises:
|           ImportError: If `h5py` is not available and the weight file is in
|             HDF5 format.
|           ValueError: If `skip_mismatch` is set to `True` when `by_name` is
|             `False`.
|
|   make_predict_function(self, force=False)
|       Creates a function that executes one step of inference.
|
|       This method can be overridden to support custom inference logic.
|       This method is called by `Model.predict` and `Model.predict_on_batch`.
|
|       Typically, this method directly controls `tf.function` and
|       `tf.distribute.Strategy` settings, and delegates the actual evaluation
|       logic to `Model.predict_step`.
|
|       This function is cached the first time `Model.predict` or
|       `Model.predict_on_batch` is called. The cache is cleared whenever
|       `Model.compile` is called. You can skip the cache and generate again the
|       function with `force=True`.
|
|       Args:
|         force: Whether to regenerate the predict function and skip the cached
|           function if available.
|
|       Returns:
|         Function. The function created by this method should accept a
|         `tf.data.Iterator`, and return the outputs of the `Model`.
|
|   make_test_function(self, force=False)
|       Creates a function that executes one step of evaluation.
|
|       This method can be overridden to support custom evaluation logic.
|       This method is called by `Model.evaluate` and `Model.test_on_batch`.
|
|       Typically, this method directly controls `tf.function` and
|       `tf.distribute.Strategy` settings, and delegates the actual evaluation
|       logic to `Model.test_step`.
|
|       This function is cached the first time `Model.evaluate` or
|       `Model.test_on_batch` is called. The cache is cleared whenever
|       `Model.compile` is called. You can skip the cache and generate again the
|       function with `force=True`.
|
```

```
 |      Args:
 |        force: Whether to regenerate the test function and skip the cached
 |          function if available.
 |
 |      Returns:
 |        Function. The function created by this method should accept a
 |        `tf.data.Iterator`, and return a `dict` containing values that will
 |        be passed to `tf.keras.Callbacks.on_test_batch_end`.
 |
 |  make_train_function(self, force=False)
 |      Creates a function that executes one step of training.
 |
 |      This method can be overridden to support custom training logic.
 |      This method is called by `Model.fit` and `Model.train_on_batch`.
 |
 |      Typically, this method directly controls `tf.function` and
 |      `tf.distribute.Strategy` settings, and delegates the actual training
 |      logic to `Model.train_step`.
 |
 |      This function is cached the first time `Model.fit` or
 |      `Model.train_on_batch` is called. The cache is cleared whenever
 |      `Model.compile` is called. You can skip the cache and generate again the
 |      function with `force=True`.
 |
 |      Args:
 |        force: Whether to regenerate the train function and skip the cached
 |          function if available.
 |
 |      Returns:
 |        Function. The function created by this method should accept a
 |        `tf.data.Iterator`, and return a `dict` containing values that will
 |        be passed to `tf.keras.Callbacks.on_train_batch_end`, such as
 |        `{'loss': 0.2, 'accuracy': 0.7}`.
 |
 |  predict(self, x, batch_size=None, verbose='auto', steps=None, callbacks=None,
max_queue_size=10, workers=1, use_multiprocessing=False)
 |      Generates output predictions for the input samples.
 |
 |      Computation is done in batches. This method is designed for batch
 |      processing of large numbers of inputs. It is not intended for use inside
 |      of loops that iterate over your data and process small numbers of inputs
 |      at a time.
 |
 |      For small numbers of inputs that fit in one batch,
 |      directly use `__call__()` for faster execution, e.g.,
 |      `model(x)`, or `model(x, training=False)` if you have layers such as
 |      `tf.keras.layers.BatchNormalization` that behave differently during
 |      inference. You may pair the individual model call with a `tf.function`
 |      for additional performance inside your inner loop.
 |      If you need access to numpy array values instead of tensors after your
 |      model call, you can use `tensor.numpy()` to get the numpy array value of
 |      an eager tensor.
 |
 |      Also, note the fact that test loss is not affected by
 |      regularization layers like noise and dropout.
 |
 |      Note: See [this FAQ entry](
 |      https://keras.io/getting_started/faq/#whats-the-difference-between-model-
methods-predict-and-call)
 |      for more details about the difference between `Model` methods
```

```
 |      `predict()` and `__call__()`.
 |
 |      Args:
 |          x: Input samples. It could be:
 |            - A Numpy array (or array-like), or a list of arrays
 |              (in case the model has multiple inputs).
 |            - A TensorFlow tensor, or a list of tensors
 |              (in case the model has multiple inputs).
 |            - A `tf.data` dataset.
 |            - A generator or `keras.utils.Sequence` instance.
 |            A more detailed description of unpacking behavior for iterator
 |            types (Dataset, generator, Sequence) is given in the `Unpacking
 |            behavior for iterator-like inputs` section of `Model.fit`.
 |          batch_size: Integer or `None`.
 |              Number of samples per batch.
 |              If unspecified, `batch_size` will default to 32.
 |              Do not specify the `batch_size` if your data is in the
 |              form of dataset, generators, or `keras.utils.Sequence` instances
 |              (since they generate batches).
 |          verbose: `"auto"`, 0, 1, or 2. Verbosity mode.
 |              0 = silent, 1 = progress bar, 2 = single line.
 |              `"auto"` defaults to 1 for most cases, and to 2 when used with
 |              `ParameterServerStrategy`. Note that the progress bar is not
 |              particularly useful when logged to a file, so `verbose=2` is
 |              recommended when not running interactively (e.g. in a production
 |              environment).
 |          steps: Total number of steps (batches of samples)
 |              before declaring the prediction round finished.
 |              Ignored with the default value of `None`. If x is a `tf.data`
 |              dataset and `steps` is None, `predict()` will
 |              run until the input dataset is exhausted.
 |          callbacks: List of `keras.callbacks.Callback` instances.
 |              List of callbacks to apply during prediction.
 |              See [callbacks](/api_docs/python/tf/keras/callbacks).
 |          max_queue_size: Integer. Used for generator or
 |              `keras.utils.Sequence` input only. Maximum size for the
 |              generator queue. If unspecified, `max_queue_size` will default
 |              to 10.
 |          workers: Integer. Used for generator or `keras.utils.Sequence` input
 |              only. Maximum number of processes to spin up when using
 |              process-based threading. If unspecified, `workers` will default
 |              to 1.
 |          use_multiprocessing: Boolean. Used for generator or
 |              `keras.utils.Sequence` input only. If `True`, use process-based
 |              threading. If unspecified, `use_multiprocessing` will default to
 |              `False`. Note that because this implementation relies on
 |              multiprocessing, you should not pass non-picklable arguments to
 |              the generator as they can't be passed easily to children
 |              processes.
 |
 |      See the discussion of `Unpacking behavior for iterator-like inputs` for
 |      `Model.fit`. Note that Model.predict uses the same interpretation rules
 |      as `Model.fit` and `Model.evaluate`, so inputs must be unambiguous for
 |      all three methods.
 |
 |      Returns:
 |          Numpy array(s) of predictions.
 |
 |      Raises:
 |          RuntimeError: If `model.predict` is wrapped in a `tf.function`.
```

```
 |          ValueError: In case of mismatch between the provided
 |              input data and the model's expectations,
 |              or in case a stateful model receives a number of samples
 |              that is not a multiple of the batch size.
 |
 |  predict_generator(self, generator, steps=None, callbacks=None, max_queue_size
=10, workers=1, use_multiprocessing=False, verbose=0)
 |      Generates predictions for the input samples from a data generator.
 |
 |      DEPRECATED:
 |        `Model.predict` now supports generators, so there is no longer any
 |        need to use this endpoint.
 |
 |  predict_on_batch(self, x)
 |      Returns predictions for a single batch of samples.
 |
 |      Args:
 |          x: Input data. It could be:
 |            - A Numpy array (or array-like), or a list of arrays (in case the
 |                model has multiple inputs).
 |            - A TensorFlow tensor, or a list of tensors (in case the model has
 |                multiple inputs).
 |
 |      Returns:
 |          Numpy array(s) of predictions.
 |
 |      Raises:
 |          RuntimeError: If `model.predict_on_batch` is wrapped in a
 |            `tf.function`.
 |
 |  predict_step(self, data)
 |      The logic for one inference step.
 |
 |      This method can be overridden to support custom inference logic.
 |      This method is called by `Model.make_predict_function`.
 |
 |      This method should contain the mathematical logic for one step of
 |      inference.  This typically includes the forward pass.
 |
 |      Configuration details for *how* this logic is run (e.g. `tf.function`
 |      and `tf.distribute.Strategy` settings), should be left to
 |      `Model.make_predict_function`, which can also be overridden.
 |
 |      Args:
 |        data: A nested structure of `Tensor`s.
 |
 |      Returns:
 |        The result of one inference step, typically the output of calling the
 |        `Model` on data.
 |
 |  reset_metrics(self)
 |      Resets the state of all the metrics in the model.
 |
 |      Examples:
 |
 |      >>> inputs = tf.keras.layers.Input(shape=(3,))
 |      >>> outputs = tf.keras.layers.Dense(2)(inputs)
 |      >>> model = tf.keras.models.Model(inputs=inputs, outputs=outputs)
 |      >>> model.compile(optimizer="Adam", loss="mse", metrics=["mae"])
 |
```

```
 |      >>> x = np.random.random((2, 3))
 |      >>> y = np.random.randint(0, 2, (2, 2))
 |      >>> _ = model.fit(x, y, verbose=0)
 |      >>> assert all(float(m.result()) for m in model.metrics)
 |
 |      >>> model.reset_metrics()
 |      >>> assert all(float(m.result()) == 0 for m in model.metrics)
 |
 |  reset_states(self)
 |
 |  save(self, filepath, overwrite=True, include_optimizer=True, save_format=Non
e, signatures=None, options=None, save_traces=True)
 |      Saves the model to Tensorflow SavedModel or a single HDF5 file.
 |
 |      Please see `tf.keras.models.save_model` or the
 |      [Serialization and Saving guide](
 |      https://keras.io/guides/serialization_and_saving/)
 |      for details.
 |
 |      Args:
 |          filepath: String, PathLike, path to SavedModel or H5 file to save
 |              the model.
 |          overwrite: Whether to silently overwrite any existing file at the
 |              target location, or provide the user with a manual prompt.
 |          include_optimizer: If True, save optimizer's state together.
 |          save_format: Either `'tf'` or `'h5'`, indicating whether to save the
 |              model to Tensorflow SavedModel or HDF5. Defaults to 'tf' in TF
 |              2.X, and 'h5' in TF 1.X.
 |          signatures: Signatures to save with the SavedModel. Applicable to
 |              the 'tf' format only. Please see the `signatures` argument in
 |              `tf.saved_model.save` for details.
 |          options: (only applies to SavedModel format)
 |              `tf.saved_model.SaveOptions` object that specifies options for
 |              saving to SavedModel.
 |          save_traces: (only applies to SavedModel format) When enabled, the
 |              SavedModel will store the function traces for each layer. This
 |              can be disabled, so that only the configs of each layer are
 |              stored.  Defaults to `True`. Disabling this will decrease
 |              serialization time and reduce file size, but it requires that
 |              all custom layers/models implement a `get_config()` method.
 |
 |      Example:
 |
 |      ```python
 |      from keras.models import load_model
 |
 |      model.save('my_model.h5')  # creates a HDF5 file 'my_model.h5'
 |      del model  # deletes the existing model
 |
 |      # returns a compiled model
 |      # identical to the previous one
 |      model = load_model('my_model.h5')
 |      ```
 |
 |  save_spec(self, dynamic_batch=True)
 |      Returns the `tf.TensorSpec` of call inputs as a tuple `(args, kwargs)`.
 |
 |      This value is automatically defined after calling the model for the
 |      first time. Afterwards, you can use it when exporting the model for
 |      serving:
```

```
 |
 |      ```python
 |      model = tf.keras.Model(...)
 |
 |      @tf.function
 |      def serve(*args, **kwargs):
 |        outputs = model(*args, **kwargs)
 |        # Apply postprocessing steps, or add additional outputs.
 |        ...
 |        return outputs
 |
 |      # arg_specs is `[tf.TensorSpec(...), ...]`. kwarg_specs, in this
 |      # example, is an empty dict since functional models do not use keyword
 |      # arguments.
 |      arg_specs, kwarg_specs = model.save_spec()
 |
 |      model.save(path, signatures={
 |        'serving_default': serve.get_concrete_function(*arg_specs,
 |                                                        **kwarg_specs)
 |      })
 |      ```
 |
 |      Args:
 |        dynamic_batch: Whether to set the batch sizes of all the returned
 |          `tf.TensorSpec` to `None`. (Note that when defining functional or
 |          Sequential models with `tf.keras.Input([...], batch_size=X)`, the
 |          batch size will always be preserved). Defaults to `True`.
 |      Returns:
 |        If the model inputs are defined, returns a tuple `(args, kwargs)`. All
 |        elements in `args` and `kwargs` are `tf.TensorSpec`.
 |        If the model inputs are not defined, returns `None`.
 |        The model inputs are automatically set when calling the model,
 |        `model.fit`, `model.evaluate` or `model.predict`.
 |
 |  save_weights(self, filepath, overwrite=True, save_format=None, options=None)
 |      Saves all layer weights.
 |
 |      Either saves in HDF5 or in TensorFlow format based on the `save_format`
 |      argument.
 |
 |      When saving in HDF5 format, the weight file has:
 |        - `layer_names` (attribute), a list of strings
 |            (ordered names of model layers).
 |        - For every layer, a `group` named `layer.name`
 |            - For every such layer group, a group attribute `weight_names`,
 |                a list of strings
 |                (ordered names of weights tensor of the layer).
 |            - For every weight in the layer, a dataset
 |                storing the weight value, named after the weight tensor.
 |
 |      When saving in TensorFlow format, all objects referenced by the network
 |      are saved in the same format as `tf.train.Checkpoint`, including any
 |      `Layer` instances or `Optimizer` instances assigned to object
 |      attributes. For networks constructed from inputs and outputs using
 |      `tf.keras.Model(inputs, outputs)`, `Layer` instances used by the network
 |      are tracked/saved automatically. For user-defined classes which inherit
 |      from `tf.keras.Model`, `Layer` instances must be assigned to object
 |      attributes, typically in the constructor. See the documentation of
 |      `tf.train.Checkpoint` and `tf.keras.Model` for details.
 |
```

```
 |      While the formats are the same, do not mix `save_weights` and
 |      `tf.train.Checkpoint`. Checkpoints saved by `Model.save_weights` should
 |      be loaded using `Model.load_weights`. Checkpoints saved using
 |      `tf.train.Checkpoint.save` should be restored using the corresponding
 |      `tf.train.Checkpoint.restore`. Prefer `tf.train.Checkpoint` over
 |      `save_weights` for training checkpoints.
 |
 |      The TensorFlow format matches objects and variables by starting at a
 |      root object, `self` for `save_weights`, and greedily matching attribute
 |      names. For `Model.save` this is the `Model`, and for `Checkpoint.save`
 |      this is the `Checkpoint` even if the `Checkpoint` has a model attached.
 |      This means saving a `tf.keras.Model` using `save_weights` and loading
 |      into a `tf.train.Checkpoint` with a `Model` attached (or vice versa)
 |      will not match the `Model`'s variables. See the
 |      [guide to training checkpoints](
 |      https://www.tensorflow.org/guide/checkpoint) for details on
 |      the TensorFlow format.
 |
 |      Args:
 |          filepath: String or PathLike, path to the file to save the weights
 |              to. When saving in TensorFlow format, this is the prefix used
 |              for checkpoint files (multiple files are generated). Note that
 |              the '.h5' suffix causes weights to be saved in HDF5 format.
 |          overwrite: Whether to silently overwrite any existing file at the
 |              target location, or provide the user with a manual prompt.
 |          save_format: Either 'tf' or 'h5'. A `filepath` ending in '.h5' or
 |              '.keras' will default to HDF5 if `save_format` is `None`.
 |              Otherwise `None` defaults to 'tf'.
 |          options: Optional `tf.train.CheckpointOptions` object that specifies
 |              options for saving weights.
 |
 |      Raises:
 |          ImportError: If `h5py` is not available when attempting to save in
 |              HDF5 format.
 |
 |  summary(self, line_length=None, positions=None, print_fn=None, expand_nested=
False, show_trainable=False, layer_range=None)
 |      Prints a string summary of the network.
 |
 |      Args:
 |          line_length: Total length of printed lines
 |              (e.g. set this to adapt the display to different
 |              terminal window sizes).
 |          positions: Relative or absolute positions of log elements
 |              in each line. If not provided,
 |              defaults to `[.33, .55, .67, 1.]`.
 |          print_fn: Print function to use. Defaults to `print`.
 |              It will be called on each line of the summary.
 |              You can set it to a custom function
 |              in order to capture the string summary.
 |          expand_nested: Whether to expand the nested models.
 |              If not provided, defaults to `False`.
 |          show_trainable: Whether to show if a layer is trainable.
 |              If not provided, defaults to `False`.
 |          layer_range: a list or tuple of 2 strings,
 |              which is the starting layer name and ending layer name
 |              (both inclusive) indicating the range of layers to be printed
 |              in summary. It also accepts regex patterns instead of exact
 |              name. In such case, start predicate will be the first element
 |              it matches to `layer_range[0]` and the end predicate will be
```

```
 |              the last element it matches to `layer_range[1]`.
 |              By default `None` which considers all layers of model.
 |
 |      Raises:
 |          ValueError: if `summary()` is called before the model is built.
 |
 |  test_on_batch(self, x, y=None, sample_weight=None, reset_metrics=True, return
_dict=False)
 |      Test the model on a single batch of samples.
 |
 |      Args:
 |          x: Input data. It could be:
 |            - A Numpy array (or array-like), or a list of arrays (in case the
 |                model has multiple inputs).
 |            - A TensorFlow tensor, or a list of tensors (in case the model has
 |                multiple inputs).
 |            - A dict mapping input names to the corresponding array/tensors,
 |                if the model has named inputs.
 |          y: Target data. Like the input data `x`, it could be either Numpy
 |            array(s) or TensorFlow tensor(s). It should be consistent with `x`
 |            (you cannot have Numpy inputs and tensor targets, or inversely).
 |          sample_weight: Optional array of the same length as x, containing
 |            weights to apply to the model's loss for each sample. In the case
 |            of temporal data, you can pass a 2D array with shape (samples,
 |            sequence_length), to apply a different weight to every timestep of
 |            every sample.
 |          reset_metrics: If `True`, the metrics returned will be only for this
 |            batch. If `False`, the metrics will be statefully accumulated
 |            across batches.
 |          return_dict: If `True`, loss and metric results are returned as a
 |            dict, with each key being the name of the metric. If `False`, they
 |            are returned as a list.
 |
 |      Returns:
 |          Scalar test loss (if the model has a single output and no metrics)
 |          or list of scalars (if the model has multiple outputs
 |          and/or metrics). The attribute `model.metrics_names` will give you
 |          the display labels for the scalar outputs.
 |
 |      Raises:
 |          RuntimeError: If `model.test_on_batch` is wrapped in a
 |            `tf.function`.
 |
 |  test_step(self, data)
 |      The logic for one evaluation step.
 |
 |      This method can be overridden to support custom evaluation logic.
 |      This method is called by `Model.make_test_function`.
 |
 |      This function should contain the mathematical logic for one step of
 |      evaluation.
 |      This typically includes the forward pass, loss calculation, and metrics
 |      updates.
 |
 |      Configuration details for *how* this logic is run (e.g. `tf.function`
 |      and `tf.distribute.Strategy` settings), should be left to
 |      `Model.make_test_function`, which can also be overridden.
 |
 |      Args:
 |        data: A nested structure of `Tensor`s.
```

```
 |
 |      Returns:
 |        A `dict` containing values that will be passed to
 |        `tf.keras.callbacks.CallbackList.on_train_batch_end`. Typically, the
 |        values of the `Model`'s metrics are returned.
 |
 |  to_json(self, **kwargs)
 |      Returns a JSON string containing the network configuration.
 |
 |      To load a network from a JSON save file, use
 |      `keras.models.model_from_json(json_string, custom_objects={})`.
 |
 |      Args:
 |          **kwargs: Additional keyword arguments to be passed to
 |              *`json.dumps()`.
 |
 |      Returns:
 |          A JSON string.
 |
 |  to_yaml(self, **kwargs)
 |      Returns a yaml string containing the network configuration.
 |
 |      Note: Since TF 2.6, this method is no longer supported and will raise a
 |      RuntimeError.
 |
 |      To load a network from a yaml save file, use
 |      `keras.models.model_from_yaml(yaml_string, custom_objects={})`.
 |
 |      `custom_objects` should be a dictionary mapping
 |      the names of custom losses / layers / etc to the corresponding
 |      functions / classes.
 |
 |      Args:
 |          **kwargs: Additional keyword arguments
 |              to be passed to `yaml.dump()`.
 |
 |      Returns:
 |          A YAML string.
 |
 |      Raises:
 |          RuntimeError: announces that the method poses a security risk
 |
 |  train_on_batch(self, x, y=None, sample_weight=None, class_weight=None, reset_
metrics=True, return_dict=False)
 |      Runs a single gradient update on a single batch of data.
 |
 |      Args:
 |          x: Input data. It could be:
 |            - A Numpy array (or array-like), or a list of arrays
 |                (in case the model has multiple inputs).
 |            - A TensorFlow tensor, or a list of tensors
 |                (in case the model has multiple inputs).
 |            - A dict mapping input names to the corresponding array/tensors,
 |                if the model has named inputs.
 |          y: Target data. Like the input data `x`, it could be either Numpy
 |            array(s) or TensorFlow tensor(s).
 |          sample_weight: Optional array of the same length as x, containing
 |            weights to apply to the model's loss for each sample. In the case
 |            of temporal data, you can pass a 2D array with shape (samples,
 |            sequence_length), to apply a different weight to every timestep of
```

```
 |          every sample.
 |        class_weight: Optional dictionary mapping class indices (integers)
 |          to a weight (float) to apply to the model's loss for the samples
 |          from this class during training. This can be useful to tell the
 |          model to "pay more attention" to samples from an under-represented
 |          class.
 |        reset_metrics: If `True`, the metrics returned will be only for this
 |          batch. If `False`, the metrics will be statefully accumulated
 |          across batches.
 |        return_dict: If `True`, loss and metric results are returned as a
 |          dict, with each key being the name of the metric. If `False`, they
 |          are returned as a list.
 |
 |    Returns:
 |        Scalar training loss
 |        (if the model has a single output and no metrics)
 |        or list of scalars (if the model has multiple outputs
 |        and/or metrics). The attribute `model.metrics_names` will give you
 |        the display labels for the scalar outputs.
 |
 |    Raises:
 |      RuntimeError: If `model.train_on_batch` is wrapped in a `tf.function`.
 |
 |  train_step(self, data)
 |      The logic for one training step.
 |
 |      This method can be overridden to support custom training logic.
 |      For concrete examples of how to override this method see
 |      [Customizing what happens in fit](
 |      https://www.tensorflow.org/guide/keras/customizing_what_happens_in_fit).
 |      This method is called by `Model.make_train_function`.
 |
 |      This method should contain the mathematical logic for one step of
 |      training.  This typically includes the forward pass, loss calculation,
 |      backpropagation, and metric updates.
 |
 |      Configuration details for *how* this logic is run (e.g. `tf.function`
 |      and `tf.distribute.Strategy` settings), should be left to
 |      `Model.make_train_function`, which can also be overridden.
 |
 |      Args:
 |        data: A nested structure of `Tensor`s.
 |
 |      Returns:
 |        A `dict` containing values that will be passed to
 |        `tf.keras.callbacks.CallbackList.on_train_batch_end`. Typically, the
 |        values of the `Model`'s metrics are returned. Example:
 |        `{'loss': 0.2, 'accuracy': 0.7}`.
 |
 |  ----------------------------------------------------------------------
 |  Static methods inherited from keras.engine.training.Model:
 |
 |  __new__(cls, *args, **kwargs)
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from keras.engine.training.Model:
 |
 |  distribute_strategy
 |      The `tf.distribute.Strategy` this model was created under.
```

```
 |
 |  metrics
 |      Returns the model's metrics added using `compile()`, `add_metric()` APIs.
 |
 |      Note: Metrics passed to `compile()` are available only after a
 |      `keras.Model` has been trained/evaluated on actual data.
 |
 |      Examples:
 |
 |      >>> inputs = tf.keras.layers.Input(shape=(3,))
 |      >>> outputs = tf.keras.layers.Dense(2)(inputs)
 |      >>> model = tf.keras.models.Model(inputs=inputs, outputs=outputs)
 |      >>> model.compile(optimizer="Adam", loss="mse", metrics=["mae"])
 |      >>> [m.name for m in model.metrics]
 |      []
 |
 |      >>> x = np.random.random((2, 3))
 |      >>> y = np.random.randint(0, 2, (2, 2))
 |      >>> model.fit(x, y)
 |      >>> [m.name for m in model.metrics]
 |      ['loss', 'mae']
 |
 |      >>> inputs = tf.keras.layers.Input(shape=(3,))
 |      >>> d = tf.keras.layers.Dense(2, name='out')
 |      >>> output_1 = d(inputs)
 |      >>> output_2 = d(inputs)
 |      >>> model = tf.keras.models.Model(
 |      ...    inputs=inputs, outputs=[output_1, output_2])
 |      >>> model.add_metric(
 |      ...    tf.reduce_sum(output_2), name='mean', aggregation='mean')
 |      >>> model.compile(optimizer="Adam", loss="mse", metrics=["mae", "acc"])
 |      >>> model.fit(x, (y, y))
 |      >>> [m.name for m in model.metrics]
 |      ['loss', 'out_loss', 'out_1_loss', 'out_mae', 'out_acc', 'out_1_mae',
 |      'out_1_acc', 'mean']
 |
 |  metrics_names
 |      Returns the model's display labels for all outputs.
 |
 |      Note: `metrics_names` are available only after a `keras.Model` has been
 |      trained/evaluated on actual data.
 |
 |      Examples:
 |
 |      >>> inputs = tf.keras.layers.Input(shape=(3,))
 |      >>> outputs = tf.keras.layers.Dense(2)(inputs)
 |      >>> model = tf.keras.models.Model(inputs=inputs, outputs=outputs)
 |      >>> model.compile(optimizer="Adam", loss="mse", metrics=["mae"])
 |      >>> model.metrics_names
 |      []
 |
 |      >>> x = np.random.random((2, 3))
 |      >>> y = np.random.randint(0, 2, (2, 2))
 |      >>> model.fit(x, y)
 |      >>> model.metrics_names
 |      ['loss', 'mae']
 |
 |      >>> inputs = tf.keras.layers.Input(shape=(3,))
 |      >>> d = tf.keras.layers.Dense(2, name='out')
 |      >>> output_1 = d(inputs)
```

```
 |      >>> output_2 = d(inputs)
 |      >>> model = tf.keras.models.Model(
 |      ...    inputs=inputs, outputs=[output_1, output_2])
 |      >>> model.compile(optimizer="Adam", loss="mse", metrics=["mae", "acc"])
 |      >>> model.fit(x, (y, y))
 |      >>> model.metrics_names
 |      ['loss', 'out_loss', 'out_1_loss', 'out_mae', 'out_acc', 'out_1_mae',
 |      'out_1_acc']
 |
 |  non_trainable_weights
 |      List of all non-trainable weights tracked by this layer.
 |
 |      Non-trainable weights are *not* updated during training. They are
 |      expected to be updated manually in `call()`.
 |
 |      Returns:
 |        A list of non-trainable variables.
 |
 |  state_updates
 |      Deprecated, do NOT use!
 |
 |      Returns the `updates` from all layers that are stateful.
 |
 |      This is useful for separating training updates and
 |      state updates, e.g. when we need to update a layer's internal state
 |      during prediction.
 |
 |      Returns:
 |          A list of update ops.
 |
 |  trainable_weights
 |      List of all trainable weights tracked by this layer.
 |
 |      Trainable weights are updated via gradient descent during training.
 |
 |      Returns:
 |        A list of trainable variables.
 |
 |  weights
 |      Returns the list of all layer variables/weights.
 |
 |      Note: This will not track the weights of nested `tf.Modules` that are
 |      not themselves Keras layers.
 |
 |      Returns:
 |        A list of variables.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from keras.engine.training.Model:
 |
 |  run_eagerly
 |      Settable attribute indicating whether the model should run eagerly.
 |
 |      Running eagerly means that your model will be run step by step,
 |      like Python code. Your model might run slower, but it should become
 |      easier for you to debug it by stepping into individual layer calls.
 |
 |      By default, we will attempt to compile your model to a static graph to
 |      deliver the best execution performance.
 |
```

```
 |      Returns:
 |        Boolean, whether the model should run eagerly.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from keras.engine.base_layer.Layer:
 |
 |  __delattr__(self, name)
 |      Implement delattr(self, name).
 |
 |  __getstate__(self)
 |
 |  __setstate__(self, state)
 |
 |  add_loss(self, losses, **kwargs)
 |      Add loss tensor(s), potentially dependent on layer inputs.
 |
 |      Some losses (for instance, activity regularization losses) may be
 |      dependent on the inputs passed when calling a layer. Hence, when reusing
 |      the same layer on different inputs `a` and `b`, some entries in
 |      `layer.losses` may be dependent on `a` and some on `b`. This method
 |      automatically keeps track of dependencies.
 |
 |      This method can be used inside a subclassed layer or model's `call`
 |      function, in which case `losses` should be a Tensor or list of Tensors.
 |
 |      Example:
 |
 |      ```python
 |      class MyLayer(tf.keras.layers.Layer):
 |        def call(self, inputs):
 |          self.add_loss(tf.abs(tf.reduce_mean(inputs)))
 |          return inputs
 |      ```
 |
 |      This method can also be called directly on a Functional Model during
 |      construction. In this case, any loss Tensors passed to this Model must
 |      be symbolic and be able to be traced back to the model's `Input`s. These
 |      losses become part of the model's topology and are tracked in
 |      `get_config`.
 |
 |      Example:
 |
 |      ```python
 |      inputs = tf.keras.Input(shape=(10,))
 |      x = tf.keras.layers.Dense(10)(inputs)
 |      outputs = tf.keras.layers.Dense(1)(x)
 |      model = tf.keras.Model(inputs, outputs)
 |      # Activity regularization.
 |      model.add_loss(tf.abs(tf.reduce_mean(x)))
 |      ```
 |
 |      If this is not the case for your loss (if, for example, your loss
 |      references a `Variable` of one of the model's layers), you can wrap your
 |      loss in a zero-argument lambda. These losses are not tracked as part of
 |      the model's topology since they can't be serialized.
 |
 |      Example:
 |
 |      ```python
 |      inputs = tf.keras.Input(shape=(10,))
```

```
 |      d = tf.keras.layers.Dense(10)
 |      x = d(inputs)
 |      outputs = tf.keras.layers.Dense(1)(x)
 |      model = tf.keras.Model(inputs, outputs)
 |      # Weight regularization.
 |      model.add_loss(lambda: tf.reduce_mean(d.kernel))
 |      ```
 |
 |      Args:
 |        losses: Loss tensor, or list/tuple of tensors. Rather than tensors,
 |          losses may also be zero-argument callables which create a loss
 |          tensor.
 |        **kwargs: Used for backwards compatibility only.
 |
 |  add_metric(self, value, name=None, **kwargs)
 |      Adds metric tensor to the layer.
 |
 |      This method can be used inside the `call()` method of a subclassed layer
 |      or model.
 |
 |      ```python
 |      class MyMetricLayer(tf.keras.layers.Layer):
 |        def __init__(self):
 |          super(MyMetricLayer, self).__init__(name='my_metric_layer')
 |          self.mean = tf.keras.metrics.Mean(name='metric_1')
 |
 |        def call(self, inputs):
 |          self.add_metric(self.mean(inputs))
 |          self.add_metric(tf.reduce_sum(inputs), name='metric_2')
 |          return inputs
 |      ```
 |
 |      This method can also be called directly on a Functional Model during
 |      construction. In this case, any tensor passed to this Model must
 |      be symbolic and be able to be traced back to the model's `Input`s. These
 |      metrics become part of the model's topology and are tracked when you
 |      save the model via `save()`.
 |
 |      ```python
 |      inputs = tf.keras.Input(shape=(10,))
 |      x = tf.keras.layers.Dense(10)(inputs)
 |      outputs = tf.keras.layers.Dense(1)(x)
 |      model = tf.keras.Model(inputs, outputs)
 |      model.add_metric(math_ops.reduce_sum(x), name='metric_1')
 |      ```
 |
 |      Note: Calling `add_metric()` with the result of a metric object on a
 |      Functional Model, as shown in the example below, is not supported. This
 |      is because we cannot trace the metric result tensor back to the model's
 |      inputs.
 |
 |      ```python
 |      inputs = tf.keras.Input(shape=(10,))
 |      x = tf.keras.layers.Dense(10)(inputs)
 |      outputs = tf.keras.layers.Dense(1)(x)
 |      model = tf.keras.Model(inputs, outputs)
 |      model.add_metric(tf.keras.metrics.Mean()(x), name='metric_1')
 |      ```
 |
 |      Args:
```

```
 |          value: Metric tensor.
 |          name: String metric name.
 |          **kwargs: Additional keyword arguments for backward compatibility.
 |            Accepted values:
 |            `aggregation` - When the `value` tensor provided is not the result
 |            of calling a `keras.Metric` instance, it will be aggregated by
 |            default using a `keras.Metric.Mean`.
 |
 |  add_update(self, updates)
 |      Add update op(s), potentially dependent on layer inputs.
 |
 |      Weight updates (for instance, the updates of the moving mean and
 |      variance in a BatchNormalization layer) may be dependent on the inputs
 |      passed when calling a layer. Hence, when reusing the same layer on
 |      different inputs `a` and `b`, some entries in `layer.updates` may be
 |      dependent on `a` and some on `b`. This method automatically keeps track
 |      of dependencies.
 |
 |      This call is ignored when eager execution is enabled (in that case,
 |      variable updates are run on the fly and thus do not need to be tracked
 |      for later execution).
 |
 |      Args:
 |        updates: Update op, or list/tuple of update ops, or zero-arg callable
 |          that returns an update op. A zero-arg callable should be passed in
 |          order to disable running the updates by setting `trainable=False`
 |          on this Layer, when executing in Eager mode.
 |
 |  add_variable(self, *args, **kwargs)
 |      Deprecated, do NOT use! Alias for `add_weight`.
 |
 |  add_weight(self, name=None, shape=None, dtype=None, initializer=None, regular
izer=None, trainable=None, constraint=None, use_resource=None, synchronization=<V
ariableSynchronization.AUTO: 0>, aggregation=<VariableAggregationV2.NONE: 0>, **k
wargs)
 |      Adds a new variable to the layer.
 |
 |      Args:
 |        name: Variable name.
 |        shape: Variable shape. Defaults to scalar if unspecified.
 |        dtype: The type of the variable. Defaults to `self.dtype`.
 |        initializer: Initializer instance (callable).
 |        regularizer: Regularizer instance (callable).
 |        trainable: Boolean, whether the variable should be part of the layer's
 |          "trainable_variables" (e.g. variables, biases)
 |          or "non_trainable_variables" (e.g. BatchNorm mean and variance).
 |          Note that `trainable` cannot be `True` if `synchronization`
 |          is set to `ON_READ`.
 |        constraint: Constraint instance (callable).
 |        use_resource: Whether to use a `ResourceVariable` or not.
 |          See [this guide](
 |          https://www.tensorflow.org/guide/migrate/tf1_vs_tf2#resourcevariables
_instead_of_referencevariables)
 |          for more information.
 |        synchronization: Indicates when a distributed a variable will be
 |          aggregated. Accepted values are constants defined in the class
 |          `tf.VariableSynchronization`. By default the synchronization is set
 |          to `AUTO` and the current `DistributionStrategy` chooses when to
 |          synchronize. If `synchronization` is set to `ON_READ`, `trainable`
 |          must not be set to `True`.
```

```
 |        aggregation: Indicates how a distributed variable will be aggregated.
 |          Accepted values are constants defined in the class
 |          `tf.VariableAggregation`.
 |        **kwargs: Additional keyword arguments. Accepted values are `getter`,
 |          `collections`, `experimental_autocast` and `caching_device`.
 |
 |      Returns:
 |        The variable created.
 |
 |      Raises:
 |        ValueError: When giving unsupported dtype and no initializer or when
 |          trainable has been set to True with synchronization set as
 |          `ON_READ`.
 |
 |  compute_output_signature(self, input_signature)
 |      Compute the output tensor signature of the layer based on the inputs.
 |
 |      Unlike a TensorShape object, a TensorSpec object contains both shape
 |      and dtype information for a tensor. This method allows layers to provide
 |      output dtype information if it is different from the input dtype.
 |      For any layer that doesn't implement this function,
 |      the framework will fall back to use `compute_output_shape`, and will
 |      assume that the output dtype matches the input dtype.
 |
 |      Args:
 |        input_signature: Single TensorSpec or nested structure of TensorSpec
 |          objects, describing a candidate input for the layer.
 |
 |      Returns:
 |        Single TensorSpec or nested structure of TensorSpec objects,
 |          describing how the layer would transform the provided input.
 |
 |      Raises:
 |        TypeError: If input_signature contains a non-TensorSpec object.
 |
 |  count_params(self)
 |      Count the total number of scalars composing the weights.
 |
 |      Returns:
 |          An integer count.
 |
 |      Raises:
 |          ValueError: if the layer isn't yet built
 |            (in which case its weights aren't yet defined).
 |
 |  finalize_state(self)
 |      Finalizes the layers state after updating layer weights.
 |
 |      This function can be subclassed in a layer and will be called after
 |      updating a layer weights. It can be overridden to finalize any
 |      additional layer state after a weight update.
 |
 |      This function will be called after weights of a layer have been restored
 |      from a loaded model.
 |
 |  get_input_at(self, node_index)
 |      Retrieves the input tensor(s) of a layer at a given node.
 |
 |      Args:
 |          node_index: Integer, index of the node
```

```
 |              from which to retrieve the attribute.
 |              E.g. `node_index=0` will correspond to the
 |              first input node of the layer.
 |
 |      Returns:
 |          A tensor (or list of tensors if the layer has multiple inputs).
 |
 |      Raises:
 |        RuntimeError: If called in Eager mode.
 |
 |  get_input_mask_at(self, node_index)
 |      Retrieves the input mask tensor(s) of a layer at a given node.
 |
 |      Args:
 |          node_index: Integer, index of the node
 |              from which to retrieve the attribute.
 |              E.g. `node_index=0` will correspond to the
 |              first time the layer was called.
 |
 |      Returns:
 |          A mask tensor
 |          (or list of tensors if the layer has multiple inputs).
 |
 |  get_input_shape_at(self, node_index)
 |      Retrieves the input shape(s) of a layer at a given node.
 |
 |      Args:
 |          node_index: Integer, index of the node
 |              from which to retrieve the attribute.
 |              E.g. `node_index=0` will correspond to the
 |              first time the layer was called.
 |
 |      Returns:
 |          A shape tuple
 |          (or list of shape tuples if the layer has multiple inputs).
 |
 |      Raises:
 |        RuntimeError: If called in Eager mode.
 |
 |  get_output_at(self, node_index)
 |      Retrieves the output tensor(s) of a layer at a given node.
 |
 |      Args:
 |          node_index: Integer, index of the node
 |              from which to retrieve the attribute.
 |              E.g. `node_index=0` will correspond to the
 |              first output node of the layer.
 |
 |      Returns:
 |          A tensor (or list of tensors if the layer has multiple outputs).
 |
 |      Raises:
 |        RuntimeError: If called in Eager mode.
 |
 |  get_output_mask_at(self, node_index)
 |      Retrieves the output mask tensor(s) of a layer at a given node.
 |
 |      Args:
 |          node_index: Integer, index of the node
 |              from which to retrieve the attribute.
```

```
 |              E.g. `node_index=0` will correspond to the
 |              first time the layer was called.
 |
 |      Returns:
 |          A mask tensor
 |          (or list of tensors if the layer has multiple outputs).
 |
 |  get_output_shape_at(self, node_index)
 |      Retrieves the output shape(s) of a layer at a given node.
 |
 |      Args:
 |          node_index: Integer, index of the node
 |              from which to retrieve the attribute.
 |              E.g. `node_index=0` will correspond to the
 |              first time the layer was called.
 |
 |      Returns:
 |          A shape tuple
 |          (or list of shape tuples if the layer has multiple outputs).
 |
 |      Raises:
 |        RuntimeError: If called in Eager mode.
 |
 |  set_weights(self, weights)
 |      Sets the weights of the layer, from NumPy arrays.
 |
 |      The weights of a layer represent the state of the layer. This function
 |      sets the weight values from numpy arrays. The weight values should be
 |      passed in the order they are created by the layer. Note that the layer's
 |      weights must be instantiated before calling this function, by calling
 |      the layer.
 |
 |      For example, a `Dense` layer returns a list of two values: the kernel
 |      matrix and the bias vector. These can be used to set the weights of
 |      another `Dense` layer:
 |
 |      >>> layer_a = tf.keras.layers.Dense(1,
 |      ...   kernel_initializer=tf.constant_initializer(1.))
 |      >>> a_out = layer_a(tf.convert_to_tensor([[1., 2., 3.]]))
 |      >>> layer_a.get_weights()
 |      [array([[1.],
 |             [1.],
 |             [1.]], dtype=float32), array([0.], dtype=float32)]
 |      >>> layer_b = tf.keras.layers.Dense(1,
 |      ...   kernel_initializer=tf.constant_initializer(2.))
 |      >>> b_out = layer_b(tf.convert_to_tensor([[10., 20., 30.]]))
 |      >>> layer_b.get_weights()
 |      [array([[2.],
 |             [2.],
 |             [2.]], dtype=float32), array([0.], dtype=float32)]
 |      >>> layer_b.set_weights(layer_a.get_weights())
 |      >>> layer_b.get_weights()
 |      [array([[1.],
 |             [1.],
 |             [1.]], dtype=float32), array([0.], dtype=float32)]
 |
 |      Args:
 |        weights: a list of NumPy arrays. The number
 |          of arrays and their shape must match
 |          number of the dimensions of the weights
```

```
 |          of the layer (i.e. it should match the
 |          output of `get_weights`).
 |
 |      Raises:
 |        ValueError: If the provided weights list does not match the
 |          layer's specifications.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from keras.engine.base_layer.Layer:
 |
 |  compute_dtype
 |      The dtype of the layer's computations.
 |
 |      This is equivalent to `Layer.dtype_policy.compute_dtype`. Unless
 |      mixed precision is used, this is the same as `Layer.dtype`, the dtype of
 |      the weights.
 |
 |      Layers automatically cast their inputs to the compute dtype, which
 |      causes computations and the output to be in the compute dtype as well.
 |      This is done by the base Layer class in `Layer.__call__`, so you do not
 |      have to insert these casts if implementing your own layer.
 |
 |      Layers often perform certain internal computations in higher precision
 |      when `compute_dtype` is float16 or bfloat16 for numeric stability. The
 |      output will still typically be float16 or bfloat16 in such cases.
 |
 |      Returns:
 |        The layer's compute dtype.
 |
 |  dtype
 |      The dtype of the layer weights.
 |
 |      This is equivalent to `Layer.dtype_policy.variable_dtype`. Unless
 |      mixed precision is used, this is the same as `Layer.compute_dtype`, the
 |      dtype of the layer's computations.
 |
 |  dtype_policy
 |      The dtype policy associated with this layer.
 |
 |      This is an instance of a `tf.keras.mixed_precision.Policy`.
 |
 |  dynamic
 |      Whether the layer is dynamic (eager-only); set in the constructor.
 |
 |  inbound_nodes
 |      Return Functional API nodes upstream of this layer.
 |
 |  input_mask
 |      Retrieves the input mask tensor(s) of a layer.
 |
 |      Only applicable if the layer has exactly one inbound node,
 |      i.e. if it is connected to one incoming layer.
 |
 |      Returns:
 |          Input mask tensor (potentially None) or list of input
 |          mask tensors.
 |
 |      Raises:
 |          AttributeError: if the layer is connected to
 |          more than one incoming layers.
```

```
 |
 |  losses
 |      List of losses added using the `add_loss()` API.
 |
 |      Variable regularization tensors are created when this property is
 |      accessed, so it is eager safe: accessing `losses` under a
 |      `tf.GradientTape` will propagate gradients back to the corresponding
 |      variables.
 |
 |      Examples:
 |
 |      >>> class MyLayer(tf.keras.layers.Layer):
 |      ...   def call(self, inputs):
 |      ...     self.add_loss(tf.abs(tf.reduce_mean(inputs)))
 |      ...     return inputs
 |      >>> l = MyLayer()
 |      >>> l(np.ones((10, 1)))
 |      >>> l.losses
 |      [1.0]
 |
 |      >>> inputs = tf.keras.Input(shape=(10,))
 |      >>> x = tf.keras.layers.Dense(10)(inputs)
 |      >>> outputs = tf.keras.layers.Dense(1)(x)
 |      >>> model = tf.keras.Model(inputs, outputs)
 |      >>> # Activity regularization.
 |      >>> len(model.losses)
 |      0
 |      >>> model.add_loss(tf.abs(tf.reduce_mean(x)))
 |      >>> len(model.losses)
 |      1
 |
 |      >>> inputs = tf.keras.Input(shape=(10,))
 |      >>> d = tf.keras.layers.Dense(10, kernel_initializer='ones')
 |      >>> x = d(inputs)
 |      >>> outputs = tf.keras.layers.Dense(1)(x)
 |      >>> model = tf.keras.Model(inputs, outputs)
 |      >>> # Weight regularization.
 |      >>> model.add_loss(lambda: tf.reduce_mean(d.kernel))
 |      >>> model.losses
 |      [<tf.Tensor: shape=(), dtype=float32, numpy=1.0>]
 |
 |      Returns:
 |        A list of tensors.
 |
 |  name
 |      Name of the layer (string), set in the constructor.
 |
 |  non_trainable_variables
 |      Sequence of non-trainable variables owned by this module and its submodul
es.
 |
 |      Note: this method uses reflection to find variables on the current instan
ce
 |      and submodules. For performance reasons you may wish to cache the result
 |      of calling this method if you don't expect the return value to change.
 |
 |      Returns:
 |        A sequence of variables for the current module (sorted by attribute
 |        name) followed by variables from all submodules recursively (breadth
 |        first).
```

```
 |
 |  outbound_nodes
 |      Return Functional API nodes downstream of this layer.
 |
 |  output_mask
 |      Retrieves the output mask tensor(s) of a layer.
 |
 |      Only applicable if the layer has exactly one inbound node,
 |      i.e. if it is connected to one incoming layer.
 |
 |      Returns:
 |          Output mask tensor (potentially None) or list of output
 |          mask tensors.
 |
 |      Raises:
 |          AttributeError: if the layer is connected to
 |          more than one incoming layers.
 |
 |  trainable_variables
 |      Sequence of trainable variables owned by this module and its submodules.
 |
 |      Note: this method uses reflection to find variables on the current instan
ce
 |      and submodules. For performance reasons you may wish to cache the result
 |      of calling this method if you don't expect the return value to change.
 |
 |      Returns:
 |        A sequence of variables for the current module (sorted by attribute
 |        name) followed by variables from all submodules recursively (breadth
 |        first).
 |
 |  updates
 |
 |  variable_dtype
 |      Alias of `Layer.dtype`, the dtype of the weights.
 |
 |  variables
 |      Returns the list of all layer variables/weights.
 |
 |      Alias of `self.weights`.
 |
 |      Note: This will not track the weights of nested `tf.Modules` that are
 |      not themselves Keras layers.
 |
 |      Returns:
 |        A list of variables.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from keras.engine.base_layer.Layer:
 |
 |  activity_regularizer
 |      Optional regularizer function for the output of this layer.
 |
 |  stateful
 |
 |  supports_masking
 |      Whether this layer supports computing a mask using `compute_mask`.
 |
 |  trainable
 |
```

```
 |  ----------------------------------------------------------------------
 |  Class methods inherited from tensorflow.python.module.module.Module:
 |  
 |  with_name_scope(method) from builtins.type
 |      Decorator to automatically enter the module name scope.
 |      
 |      >>> class MyModule(tf.Module):
 |      ...   @tf.Module.with_name_scope
 |      ...   def __call__(self, x):
 |      ...     if not hasattr(self, 'w'):
 |      ...       self.w = tf.Variable(tf.random.normal([x.shape[1], 3]))
 |      ...     return tf.matmul(x, self.w)
 |      
 |      Using the above module would produce `tf.Variable`s and `tf.Tensor`s whos
e
 |      names included the module name:
 |      
 |      >>> mod = MyModule()
 |      >>> mod(tf.ones([1, 2]))
 |      <tf.Tensor: shape=(1, 3), dtype=float32, numpy=..., dtype=float32)>
 |      >>> mod.w
 |      <tf.Variable 'my_module/Variable:0' shape=(2, 3) dtype=float32,
 |      numpy=..., dtype=float32)>
 |      
 |      Args:
 |        method: The method to wrap.
 |      
 |      Returns:
 |        The original method wrapped such that it enters the module's name scop
e.
 |  
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from tensorflow.python.module.module.Module:
 |  
 |  name_scope
 |      Returns a `tf.name_scope` instance for this class.
 |  
 |  submodules
 |      Sequence of all sub-modules.
 |      
 |      Submodules are modules which are properties of this module, or found as
 |      properties of modules which are properties of this module (and so on).
 |      
 |      >>> a = tf.Module()
 |      >>> b = tf.Module()
 |      >>> c = tf.Module()
 |      >>> a.b = b
 |      >>> b.c = c
 |      >>> list(a.submodules) == [b, c]
 |      True
 |      >>> list(b.submodules) == [c]
 |      True
 |      >>> list(c.submodules) == []
 |      True
 |      
 |      Returns:
 |        A sequence of all submodules.
 |  
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from tensorflow.python.trackable.base.Trackable:
```

```
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

In [44]:
```
model = Sequential([Dense(4, activation='relu'), Dense(4, activation='relu'), De
```

In [45]:
```
model.compile(loss='mse')
```

In [46]:
```
model.fit(x=X_train, y=y_train, epochs=250)
```

```
Epoch 1/250
22/22 [==============================] - 1s 3ms/step - loss: 256669.6875
Epoch 2/250
22/22 [==============================] - 0s 4ms/step - loss: 256581.6406
Epoch 3/250
22/22 [==============================] - 0s 4ms/step - loss: 256494.8594
Epoch 4/250
22/22 [==============================] - 0s 4ms/step - loss: 256400.7031
Epoch 5/250
22/22 [==============================] - 0s 4ms/step - loss: 256296.0938
Epoch 6/250
22/22 [==============================] - 0s 4ms/step - loss: 256178.1719
Epoch 7/250
22/22 [==============================] - 0s 4ms/step - loss: 256045.4844
Epoch 8/250
22/22 [==============================] - 0s 4ms/step - loss: 255892.7344
Epoch 9/250
22/22 [==============================] - 0s 4ms/step - loss: 255719.9375
Epoch 10/250
22/22 [==============================] - 0s 4ms/step - loss: 255525.1719
Epoch 11/250
22/22 [==============================] - 0s 4ms/step - loss: 255307.2031
Epoch 12/250
22/22 [==============================] - 0s 4ms/step - loss: 255066.0781
Epoch 13/250
22/22 [==============================] - 0s 4ms/step - loss: 254801.5469
Epoch 14/250
22/22 [==============================] - 0s 4ms/step - loss: 254512.7031
Epoch 15/250
22/22 [==============================] - 0s 4ms/step - loss: 254196.9844
Epoch 16/250
22/22 [==============================] - 0s 5ms/step - loss: 253856.0938
Epoch 17/250
22/22 [==============================] - 0s 4ms/step - loss: 253485.2344
Epoch 18/250
22/22 [==============================] - 0s 4ms/step - loss: 253083.0156
Epoch 19/250
22/22 [==============================] - 0s 4ms/step - loss: 252649.5156
Epoch 20/250
22/22 [==============================] - 0s 4ms/step - loss: 252184.2969
Epoch 21/250
22/22 [==============================] - 0s 3ms/step - loss: 251681.9844
Epoch 22/250
22/22 [==============================] - 0s 4ms/step - loss: 251143.2031
Epoch 23/250
22/22 [==============================] - 0s 3ms/step - loss: 250567.3438
Epoch 24/250
22/22 [==============================] - 0s 4ms/step - loss: 249948.1094
Epoch 25/250
22/22 [==============================] - 0s 4ms/step - loss: 249287.4219
Epoch 26/250
22/22 [==============================] - 0s 4ms/step - loss: 248581.7188
Epoch 27/250
22/22 [==============================] - 0s 4ms/step - loss: 247829.1719
Epoch 28/250
22/22 [==============================] - 0s 3ms/step - loss: 247030.4531
Epoch 29/250
22/22 [==============================] - 0s 6ms/step - loss: 246181.9844
Epoch 30/250
22/22 [==============================] - 0s 4ms/step - loss: 245276.7344
```

```
Epoch 31/250
22/22 [==============================] - 0s 3ms/step - loss: 244323.3125
Epoch 32/250
22/22 [==============================] - 0s 4ms/step - loss: 243310.9688
Epoch 33/250
22/22 [==============================] - 0s 4ms/step - loss: 242242.1719
Epoch 34/250
22/22 [==============================] - 0s 3ms/step - loss: 241113.0000
Epoch 35/250
22/22 [==============================] - 0s 3ms/step - loss: 239920.5469
Epoch 36/250
22/22 [==============================] - 0s 4ms/step - loss: 238674.9531
Epoch 37/250
22/22 [==============================] - 0s 4ms/step - loss: 237355.2031
Epoch 38/250
22/22 [==============================] - 0s 3ms/step - loss: 235961.9219
Epoch 39/250
22/22 [==============================] - 0s 3ms/step - loss: 234508.8906
Epoch 40/250
22/22 [==============================] - 0s 3ms/step - loss: 232985.7344
Epoch 41/250
22/22 [==============================] - 0s 3ms/step - loss: 231390.1094
Epoch 42/250
22/22 [==============================] - 0s 4ms/step - loss: 229721.6406
Epoch 43/250
22/22 [==============================] - 0s 4ms/step - loss: 227970.1094
Epoch 44/250
22/22 [==============================] - 0s 4ms/step - loss: 226135.7656
Epoch 45/250
22/22 [==============================] - 0s 4ms/step - loss: 224232.6094
Epoch 46/250
22/22 [==============================] - 0s 5ms/step - loss: 222253.7188
Epoch 47/250
22/22 [==============================] - 0s 5ms/step - loss: 220172.6094
Epoch 48/250
22/22 [==============================] - 0s 5ms/step - loss: 218014.5781
Epoch 49/250
22/22 [==============================] - 0s 5ms/step - loss: 215776.2031
Epoch 50/250
22/22 [==============================] - 0s 4ms/step - loss: 213457.5312
Epoch 51/250
22/22 [==============================] - 0s 5ms/step - loss: 211037.1406
Epoch 52/250
22/22 [==============================] - 0s 4ms/step - loss: 208524.8906
Epoch 53/250
22/22 [==============================] - 0s 4ms/step - loss: 205933.3906
Epoch 54/250
22/22 [==============================] - 0s 4ms/step - loss: 203253.5938
Epoch 55/250
22/22 [==============================] - 0s 4ms/step - loss: 200471.0000
Epoch 56/250
22/22 [==============================] - 0s 3ms/step - loss: 197599.3594
Epoch 57/250
22/22 [==============================] - 0s 5ms/step - loss: 194620.5938
Epoch 58/250
22/22 [==============================] - 0s 3ms/step - loss: 191569.8906
Epoch 59/250
22/22 [==============================] - 0s 3ms/step - loss: 188417.8281
Epoch 60/250
22/22 [==============================] - 0s 3ms/step - loss: 185167.3750
```

```
Epoch 61/250
22/22 [==============================] - 0s 3ms/step - loss: 181835.9062
Epoch 62/250
22/22 [==============================] - 0s 3ms/step - loss: 178401.3750
Epoch 63/250
22/22 [==============================] - 0s 4ms/step - loss: 174880.5625
Epoch 64/250
22/22 [==============================] - 0s 6ms/step - loss: 171265.0625
Epoch 65/250
22/22 [==============================] - 0s 4ms/step - loss: 167570.4062
Epoch 66/250
22/22 [==============================] - 0s 4ms/step - loss: 163775.4375
Epoch 67/250
22/22 [==============================] - 0s 3ms/step - loss: 159899.5938
Epoch 68/250
22/22 [==============================] - 0s 3ms/step - loss: 155925.2656
Epoch 69/250
22/22 [==============================] - 0s 3ms/step - loss: 151893.5781
Epoch 70/250
22/22 [==============================] - 0s 4ms/step - loss: 147772.7031
Epoch 71/250
22/22 [==============================] - 0s 3ms/step - loss: 143573.9531
Epoch 72/250
22/22 [==============================] - 0s 3ms/step - loss: 139290.9219
Epoch 73/250
22/22 [==============================] - 0s 4ms/step - loss: 134926.9062
Epoch 74/250
22/22 [==============================] - 0s 3ms/step - loss: 130508.2969
Epoch 75/250
22/22 [==============================] - 0s 4ms/step - loss: 126040.1484
Epoch 76/250
22/22 [==============================] - 0s 12ms/step - loss: 121489.2891
Epoch 77/250
22/22 [==============================] - 0s 7ms/step - loss: 116908.6406
Epoch 78/250
22/22 [==============================] - 0s 4ms/step - loss: 112268.8359
Epoch 79/250
22/22 [==============================] - 0s 4ms/step - loss: 107586.0078
Epoch 80/250
22/22 [==============================] - 0s 3ms/step - loss: 102905.5859
Epoch 81/250
22/22 [==============================] - 0s 3ms/step - loss: 98153.2812
Epoch 82/250
22/22 [==============================] - 0s 5ms/step - loss: 93384.5703
Epoch 83/250
22/22 [==============================] - 0s 4ms/step - loss: 88642.5469
Epoch 84/250
22/22 [==============================] - 0s 4ms/step - loss: 83875.5469
Epoch 85/250
22/22 [==============================] - 0s 3ms/step - loss: 79104.0781
Epoch 86/250
22/22 [==============================] - 0s 3ms/step - loss: 74388.1406
Epoch 87/250
22/22 [==============================] - 0s 4ms/step - loss: 69688.1641
Epoch 88/250
22/22 [==============================] - 0s 4ms/step - loss: 64994.6562
Epoch 89/250
22/22 [==============================] - 0s 3ms/step - loss: 60384.3477
Epoch 90/250
22/22 [==============================] - 0s 4ms/step - loss: 55838.3672
```

```
Epoch 91/250
22/22 [==============================] - 0s 6ms/step - loss: 51377.1953
Epoch 92/250
22/22 [==============================] - 0s 4ms/step - loss: 46998.4648
Epoch 93/250
22/22 [==============================] - 0s 3ms/step - loss: 42740.6445
Epoch 94/250
22/22 [==============================] - 0s 4ms/step - loss: 38587.2734
Epoch 95/250
22/22 [==============================] - 0s 3ms/step - loss: 34578.6484
Epoch 96/250
22/22 [==============================] - 0s 3ms/step - loss: 30717.8086
Epoch 97/250
22/22 [==============================] - 0s 3ms/step - loss: 27046.0137
Epoch 98/250
22/22 [==============================] - 0s 3ms/step - loss: 23593.9805
Epoch 99/250
22/22 [==============================] - 0s 3ms/step - loss: 20359.0918
Epoch 100/250
22/22 [==============================] - 0s 4ms/step - loss: 17321.1543
Epoch 101/250
22/22 [==============================] - 0s 4ms/step - loss: 14556.8359
Epoch 102/250
22/22 [==============================] - 0s 3ms/step - loss: 12057.3545
Epoch 103/250
22/22 [==============================] - 0s 3ms/step - loss: 9855.9365
Epoch 104/250
22/22 [==============================] - 0s 3ms/step - loss: 7980.1455
Epoch 105/250
22/22 [==============================] - 0s 4ms/step - loss: 6416.1924
Epoch 106/250
22/22 [==============================] - 0s 3ms/step - loss: 5202.8813
Epoch 107/250
22/22 [==============================] - 0s 3ms/step - loss: 4367.7812
Epoch 108/250
22/22 [==============================] - 0s 3ms/step - loss: 3844.0901
Epoch 109/250
22/22 [==============================] - 0s 3ms/step - loss: 3610.0786
Epoch 110/250
22/22 [==============================] - 0s 3ms/step - loss: 3531.3564
Epoch 111/250
22/22 [==============================] - 0s 3ms/step - loss: 3496.9631
Epoch 112/250
22/22 [==============================] - 0s 3ms/step - loss: 3468.5608
Epoch 113/250
22/22 [==============================] - 0s 3ms/step - loss: 3434.3894
Epoch 114/250
22/22 [==============================] - 0s 3ms/step - loss: 3399.5039
Epoch 115/250
22/22 [==============================] - 0s 3ms/step - loss: 3366.1819
Epoch 116/250
22/22 [==============================] - 0s 3ms/step - loss: 3335.4500
Epoch 117/250
22/22 [==============================] - 0s 3ms/step - loss: 3305.3606
Epoch 118/250
22/22 [==============================] - 0s 3ms/step - loss: 3273.0967
Epoch 119/250
22/22 [==============================] - 0s 3ms/step - loss: 3242.9424
Epoch 120/250
22/22 [==============================] - 0s 3ms/step - loss: 3206.9714
```

```
Epoch 121/250
22/22 [==============================] - 0s 3ms/step - loss: 3179.1804
Epoch 122/250
22/22 [==============================] - 0s 3ms/step - loss: 3146.8735
Epoch 123/250
22/22 [==============================] - 0s 3ms/step - loss: 3118.6758
Epoch 124/250
22/22 [==============================] - 0s 3ms/step - loss: 3084.9307
Epoch 125/250
22/22 [==============================] - 0s 3ms/step - loss: 3055.4043
Epoch 126/250
22/22 [==============================] - 0s 3ms/step - loss: 3022.8354
Epoch 127/250
22/22 [==============================] - 0s 3ms/step - loss: 2991.0105
Epoch 128/250
22/22 [==============================] - 0s 3ms/step - loss: 2958.6587
Epoch 129/250
22/22 [==============================] - 0s 3ms/step - loss: 2928.7686
Epoch 130/250
22/22 [==============================] - 0s 3ms/step - loss: 2894.1025
Epoch 131/250
22/22 [==============================] - 0s 6ms/step - loss: 2862.6543
Epoch 132/250
22/22 [==============================] - 0s 3ms/step - loss: 2827.6001
Epoch 133/250
22/22 [==============================] - 0s 3ms/step - loss: 2798.2551
Epoch 134/250
22/22 [==============================] - 0s 4ms/step - loss: 2768.4961
Epoch 135/250
22/22 [==============================] - 0s 5ms/step - loss: 2734.8162
Epoch 136/250
22/22 [==============================] - 0s 3ms/step - loss: 2711.1099
Epoch 137/250
22/22 [==============================] - 0s 3ms/step - loss: 2681.1011
Epoch 138/250
22/22 [==============================] - 0s 5ms/step - loss: 2644.6653
Epoch 139/250
22/22 [==============================] - 0s 4ms/step - loss: 2617.4651
Epoch 140/250
22/22 [==============================] - 0s 4ms/step - loss: 2589.4026
Epoch 141/250
22/22 [==============================] - 0s 4ms/step - loss: 2558.7117
Epoch 142/250
22/22 [==============================] - 0s 4ms/step - loss: 2531.3167
Epoch 143/250
22/22 [==============================] - 0s 4ms/step - loss: 2504.3027
Epoch 144/250
22/22 [==============================] - 0s 4ms/step - loss: 2472.5715
Epoch 145/250
22/22 [==============================] - 0s 3ms/step - loss: 2447.0303
Epoch 146/250
22/22 [==============================] - 0s 5ms/step - loss: 2417.0134
Epoch 147/250
22/22 [==============================] - 0s 5ms/step - loss: 2388.3555
Epoch 148/250
22/22 [==============================] - 0s 8ms/step - loss: 2357.0532
Epoch 149/250
22/22 [==============================] - 0s 4ms/step - loss: 2326.2117
Epoch 150/250
22/22 [==============================] - 0s 5ms/step - loss: 2294.7671
```

```
Epoch 151/250
22/22 [==============================] - 0s 8ms/step - loss: 2262.1885
Epoch 152/250
22/22 [==============================] - 0s 5ms/step - loss: 2238.1206
Epoch 153/250
22/22 [==============================] - 0s 6ms/step - loss: 2208.6887
Epoch 154/250
22/22 [==============================] - 0s 4ms/step - loss: 2179.6489
Epoch 155/250
22/22 [==============================] - 0s 4ms/step - loss: 2151.7029
Epoch 156/250
22/22 [==============================] - 0s 3ms/step - loss: 2128.5547
Epoch 157/250
22/22 [==============================] - 0s 3ms/step - loss: 2102.6523
Epoch 158/250
22/22 [==============================] - 0s 4ms/step - loss: 2073.4790
Epoch 159/250
22/22 [==============================] - 0s 2ms/step - loss: 2047.2493
Epoch 160/250
22/22 [==============================] - 0s 2ms/step - loss: 2021.5590
Epoch 161/250
22/22 [==============================] - 0s 2ms/step - loss: 1997.5389
Epoch 162/250
22/22 [==============================] - 0s 2ms/step - loss: 1966.7186
Epoch 163/250
22/22 [==============================] - 0s 3ms/step - loss: 1938.3456
Epoch 164/250
22/22 [==============================] - 0s 2ms/step - loss: 1911.6132
Epoch 165/250
22/22 [==============================] - 0s 2ms/step - loss: 1883.1777
Epoch 166/250
22/22 [==============================] - 0s 2ms/step - loss: 1855.9683
Epoch 167/250
22/22 [==============================] - 0s 2ms/step - loss: 1830.6466
Epoch 168/250
22/22 [==============================] - 0s 2ms/step - loss: 1803.8029
Epoch 169/250
22/22 [==============================] - 0s 2ms/step - loss: 1780.0208
Epoch 170/250
22/22 [==============================] - 0s 2ms/step - loss: 1753.9244
Epoch 171/250
22/22 [==============================] - 0s 2ms/step - loss: 1728.4247
Epoch 172/250
22/22 [==============================] - 0s 3ms/step - loss: 1701.5580
Epoch 173/250
22/22 [==============================] - 0s 2ms/step - loss: 1674.8873
Epoch 174/250
22/22 [==============================] - 0s 2ms/step - loss: 1647.7418
Epoch 175/250
22/22 [==============================] - 0s 2ms/step - loss: 1621.2034
Epoch 176/250
22/22 [==============================] - 0s 2ms/step - loss: 1595.6647
Epoch 177/250
22/22 [==============================] - 0s 2ms/step - loss: 1566.8955
Epoch 178/250
22/22 [==============================] - 0s 2ms/step - loss: 1542.1577
Epoch 179/250
22/22 [==============================] - 0s 2ms/step - loss: 1510.2678
Epoch 180/250
22/22 [==============================] - 0s 2ms/step - loss: 1487.2109
```

```
Epoch 181/250
22/22 [==============================] - 0s 2ms/step - loss: 1459.6084
Epoch 182/250
22/22 [==============================] - 0s 1ms/step - loss: 1432.4683
Epoch 183/250
22/22 [==============================] - 0s 2ms/step - loss: 1410.1958
Epoch 184/250
22/22 [==============================] - 0s 2ms/step - loss: 1382.9072
Epoch 185/250
22/22 [==============================] - 0s 2ms/step - loss: 1360.2345
Epoch 186/250
22/22 [==============================] - 0s 2ms/step - loss: 1335.9716
Epoch 187/250
22/22 [==============================] - 0s 2ms/step - loss: 1311.9059
Epoch 188/250
22/22 [==============================] - 0s 2ms/step - loss: 1285.9893
Epoch 189/250
22/22 [==============================] - 0s 2ms/step - loss: 1261.5288
Epoch 190/250
22/22 [==============================] - 0s 2ms/step - loss: 1236.7725
Epoch 191/250
22/22 [==============================] - 0s 2ms/step - loss: 1212.1093
Epoch 192/250
22/22 [==============================] - 0s 2ms/step - loss: 1186.6194
Epoch 193/250
22/22 [==============================] - 0s 2ms/step - loss: 1164.6515
Epoch 194/250
22/22 [==============================] - 0s 4ms/step - loss: 1138.6422
Epoch 195/250
22/22 [==============================] - 0s 2ms/step - loss: 1117.6681
Epoch 196/250
22/22 [==============================] - 0s 2ms/step - loss: 1095.6549
Epoch 197/250
22/22 [==============================] - 0s 2ms/step - loss: 1072.4618
Epoch 198/250
22/22 [==============================] - 0s 2ms/step - loss: 1051.4398
Epoch 199/250
22/22 [==============================] - 0s 2ms/step - loss: 1030.2701
Epoch 200/250
22/22 [==============================] - 0s 2ms/step - loss: 1009.5002
Epoch 201/250
22/22 [==============================] - 0s 2ms/step - loss: 988.6779
Epoch 202/250
22/22 [==============================] - 0s 2ms/step - loss: 965.1537
Epoch 203/250
22/22 [==============================] - 0s 2ms/step - loss: 942.3541
Epoch 204/250
22/22 [==============================] - 0s 2ms/step - loss: 918.0474
Epoch 205/250
22/22 [==============================] - 0s 2ms/step - loss: 894.1251
Epoch 206/250
22/22 [==============================] - 0s 2ms/step - loss: 871.5606
Epoch 207/250
22/22 [==============================] - 0s 2ms/step - loss: 846.8628
Epoch 208/250
22/22 [==============================] - 0s 2ms/step - loss: 827.7665
Epoch 209/250
22/22 [==============================] - 0s 2ms/step - loss: 805.4969
Epoch 210/250
22/22 [==============================] - 0s 2ms/step - loss: 784.8796
```

```
Epoch 211/250
22/22 [==============================] - 0s 2ms/step - loss: 765.1651
Epoch 212/250
22/22 [==============================] - 0s 2ms/step - loss: 744.1163
Epoch 213/250
22/22 [==============================] - 0s 2ms/step - loss: 724.2258
Epoch 214/250
22/22 [==============================] - 0s 4ms/step - loss: 703.8245
Epoch 215/250
22/22 [==============================] - 0s 2ms/step - loss: 682.2206
Epoch 216/250
22/22 [==============================] - 0s 2ms/step - loss: 663.8043
Epoch 217/250
22/22 [==============================] - 0s 2ms/step - loss: 644.1346
Epoch 218/250
22/22 [==============================] - 0s 3ms/step - loss: 624.1998
Epoch 219/250
22/22 [==============================] - 0s 2ms/step - loss: 602.5823
Epoch 220/250
22/22 [==============================] - 0s 2ms/step - loss: 582.3586
Epoch 221/250
22/22 [==============================] - 0s 2ms/step - loss: 563.2145
Epoch 222/250
22/22 [==============================] - 0s 2ms/step - loss: 544.9669
Epoch 223/250
22/22 [==============================] - 0s 2ms/step - loss: 527.1445
Epoch 224/250
22/22 [==============================] - 0s 2ms/step - loss: 508.8384
Epoch 225/250
22/22 [==============================] - 0s 2ms/step - loss: 489.9565
Epoch 226/250
22/22 [==============================] - 0s 2ms/step - loss: 469.9984
Epoch 227/250
22/22 [==============================] - 0s 2ms/step - loss: 457.8904
Epoch 228/250
22/22 [==============================] - 0s 2ms/step - loss: 439.4278
Epoch 229/250
22/22 [==============================] - 0s 2ms/step - loss: 423.0347
Epoch 230/250
22/22 [==============================] - 0s 2ms/step - loss: 404.9425
Epoch 231/250
22/22 [==============================] - 0s 2ms/step - loss: 388.3860
Epoch 232/250
22/22 [==============================] - 0s 4ms/step - loss: 371.7105
Epoch 233/250
22/22 [==============================] - 0s 2ms/step - loss: 356.8831
Epoch 234/250
22/22 [==============================] - 0s 2ms/step - loss: 341.6101
Epoch 235/250
22/22 [==============================] - 0s 2ms/step - loss: 327.6884
Epoch 236/250
22/22 [==============================] - 0s 2ms/step - loss: 312.1833
Epoch 237/250
22/22 [==============================] - 0s 2ms/step - loss: 298.5418
Epoch 238/250
22/22 [==============================] - 0s 2ms/step - loss: 284.2587
Epoch 239/250
22/22 [==============================] - 0s 2ms/step - loss: 270.4089
Epoch 240/250
22/22 [==============================] - 0s 2ms/step - loss: 258.5845
```

```
Epoch 241/250
22/22 [==============================] - 0s 2ms/step - loss: 245.6975
Epoch 242/250
22/22 [==============================] - 0s 2ms/step - loss: 233.6224
Epoch 243/250
22/22 [==============================] - 0s 2ms/step - loss: 221.3953
Epoch 244/250
22/22 [==============================] - 0s 2ms/step - loss: 210.0626
Epoch 245/250
22/22 [==============================] - 0s 2ms/step - loss: 197.9543
Epoch 246/250
22/22 [==============================] - 0s 2ms/step - loss: 185.6946
Epoch 247/250
22/22 [==============================] - 0s 2ms/step - loss: 174.8311
Epoch 248/250
22/22 [==============================] - 0s 2ms/step - loss: 164.2820
Epoch 249/250
22/22 [==============================] - 0s 4ms/step - loss: 154.3989
Epoch 250/250
22/22 [==============================] - 0s 2ms/step - loss: 144.5810
```

Out[46]:
```
<keras.callbacks.History at 0x27b3191a670>
```

In [50]:
```python
loss_df = pd.DataFrame(model.history.history)
```

In [54]:
```python
loss_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 0 entries
Empty DataFrame
```

In [41]:
```python
#Evaluation of model; how well model is perform on the data never seen before
```

In [55]:
```python
model.evaluate(X_test, y_test, verbose=0)
```

Out[55]:
```
135.16517639160156
```

In [56]:
```python
model.evaluate(X_train, y_train, verbose=0)
```

Out[56]:
```
139.91188049316406
```

In [57]:
```python
test_predictions = model.predict(X_test)
```

```
10/10 [==============================] - 0s 2ms/step
```

In [58]:
```python
test_predictions
```

```
Out[58]: array([[418.39847],
       [612.6908 ],
       [585.17664],
       [565.9918 ],
       [382.02237],
       [572.9095 ],
       [514.2187 ],
       [467.0733 ],
       [545.5685 ],
       [456.91183],
       [602.2751 ],
       [548.21515],
       [430.545  ],
       [421.13052],
       [639.3819 ],
       [447.4427 ],
       [511.84842],
       [643.42316],
       [646.7347 ],
       [561.8312 ],
       [354.13635],
       [453.4814 ],
       [397.39618],
       [394.8635 ],
       [561.8605 ],
       [600.14435],
       [531.2583 ],
       [438.675  ],
       [639.9403 ],
       [426.9883 ],
       [452.39566],
       [489.4803 ],
       [448.03287],
       [663.7151 ],
       [437.00095],
       [428.6107 ],
       [506.50925],
       [547.8075 ],
       [509.8763 ],
       [407.70612],
       [608.71405],
       [429.06094],
       [596.0527 ],
       [455.98355],
       [504.44662],
       [577.5778 ],
       [654.72705],
       [494.92734],
       [342.08957],
       [489.55356],
       [518.3941 ],
       [398.39413],
       [541.4695 ],
       [421.40778],
       [629.20514],
       [494.41794],
       [615.09357],
       [617.41705],
       [455.7144 ],
       [488.155  ],
```

```
       [494.10474],
       [480.56625],
       [665.77203],
       [415.96927],
       [681.8936 ],
       [579.096  ],
       [576.91376],
       [537.2094 ],
       [488.88852],
       [516.5376 ],
       [379.39578],
       [538.98236],
       [565.1118 ],
       [529.9635 ],
       [461.85483],
       [530.96765],
       [510.27188],
       [451.22086],
       [540.95636],
       [629.2926 ],
       [470.64835],
       [563.05914],
       [673.83984],
       [465.36325],
       [687.60034],
       [478.23367],
       [418.00098],
       [578.314  ],
       [447.9073 ],
       [493.599  ],
       [607.2725 ],
       [449.74042],
       [461.9061 ],
       [444.73264],
       [507.71973],
       [598.77264],
       [343.78043],
       [446.44943],
       [535.524  ],
       [518.44086],
       [595.007  ],
       [525.00946],
       [356.10358],
       [571.86847],
       [439.96378],
       [560.39636],
       [514.8572 ],
       [405.1473 ],
       [559.8668 ],
       [464.2247 ],
       [456.2629 ],
       [628.3588 ],
       [525.43787],
       [548.75323],
       [429.35208],
       [483.43463],
       [579.7542 ],
       [650.8828 ],
       [680.39874],
       [642.7178 ],
```

```
       [556.6105 ],
       [506.4194 ],
       [403.84946],
       [307.50143],
       [485.01193],
       [607.5755 ],
       [391.24716],
       [513.7102 ],
       [511.06427],
       [496.17917],
       [485.40298],
       [433.56375],
       [496.48868],
       [477.48938],
       [593.5266 ],
       [570.07336],
       [425.3471 ],
       [619.324  ],
       [473.44077],
       [560.78046],
       [418.6245 ],
       [531.1808 ],
       [567.2133 ],
       [374.80664],
       [546.5545 ],
       [594.6292 ],
       [399.2957 ],
       [539.5378 ],
       [558.53015],
       [461.73172],
       [620.5473 ],
       [387.56235],
       [478.8971 ],
       [527.1921 ],
       [389.6014 ],
       [467.23172],
       [445.03046],
       [501.1993 ],
       [366.8724 ],
       [410.26016],
       [596.3274 ],
       [509.0742 ],
       [474.50082],
       [494.9113 ],
       [535.5377 ],
       [363.9023 ],
       [516.0079 ],
       [281.4089 ],
       [506.40277],
       [537.6828 ],
       [492.96793],
       [478.91653],
       [407.23288],
       [429.84964],
       [546.37256],
       [480.0657 ],
       [572.9029 ],
       [492.37854],
       [590.9702 ],
       [545.38715],
```

```
       [540.6818 ],
       [503.37418],
       [632.12555],
       [556.1739 ],
       [573.93945],
       [454.4816 ],
       [428.63748],
       [430.34937],
       [565.1976 ],
       [599.1899 ],
       [448.15427],
       [492.45468],
       [579.6175 ],
       [524.42303],
       [375.34586],
       [631.5297 ],
       [526.87463],
       [357.8071 ],
       [496.06485],
       [423.91006],
       [597.92523],
       [367.28464],
       [523.80304],
       [418.46045],
       [285.68088],
       [521.1608 ],
       [362.98608],
       [378.54532],
       [572.0422 ],
       [429.75912],
       [547.4466 ],
       [520.89264],
       [509.64145],
       [347.1237 ],
       [415.82785],
       [592.862  ],
       [606.8047 ],
       [592.5543 ],
       [563.13446],
       [478.00607],
       [467.5994 ],
       [509.9711 ],
       [455.4181 ],
       [514.17737],
       [503.1556 ],
       [414.43945],
       [596.5465 ],
       [287.09836],
       [617.40094],
       [584.25006],
       [349.77612],
       [483.4514 ],
       [587.9591 ],
       [394.7569 ],
       [467.08432],
       [348.02017],
       [520.4397 ],
       [423.72888],
       [553.3029 ],
       [629.54816],
```

```
[535.23456],
[506.22397],
[623.9798 ],
[516.91815],
[530.92285],
[518.5015 ],
[465.1708 ],
[508.99954],
[468.5614 ],
[586.03955],
[471.9312 ],
[436.9844 ],
[538.67346],
[496.90576],
[660.0891 ],
[389.309  ],
[548.74335],
[572.2406 ],
[442.9645 ],
[540.42957],
[580.65283],
[574.46295],
[699.1895 ],
[441.23053],
[413.08182],
[337.7723 ],
[456.0394 ],
[404.7378 ],
[540.3757 ],
[522.61505],
[561.8919 ],
[456.5273 ],
[533.8483 ],
[397.80563],
[504.33624],
[623.902  ],
[501.2042 ],
[562.8967 ],
[475.38162],
[302.90704],
[517.3773 ],
[610.40784],
[369.5964 ],
[457.173  ],
[501.33182],
[540.87195],
[603.0571 ],
[401.57938],
[456.0519 ],
[487.51392],
[589.7326 ],
[500.72278],
[343.8138 ],
[551.70764],
[453.5452 ],
[527.69965],
[513.18054],
[600.54816],
[429.69885],
[422.62534]], dtype=float32)
```

In [59]:
```
test_predictions = pd.Series(test_predictions.reshape(300,))
```

In [60]:
```
test_predictions
```

Out[60]:
```
0      418.398468
1      612.690796
2      585.176636
3      565.991821
4      382.022369
          ...
295    527.699646
296    513.180542
297    600.548157
298    429.698853
299    422.625336
Length: 300, dtype: float32
```

In [61]:
```
pred_df = pd.DataFrame(y_test, columns=['Test True Y'])
```

In [62]:
```
pred_df = pd.concat([pred_df, test_predictions], axis=1)
```

In [63]:
```
pred_df.columns = ['Test True Y', 'Model Predictions']
```

In [64]:
```
pred_df
```

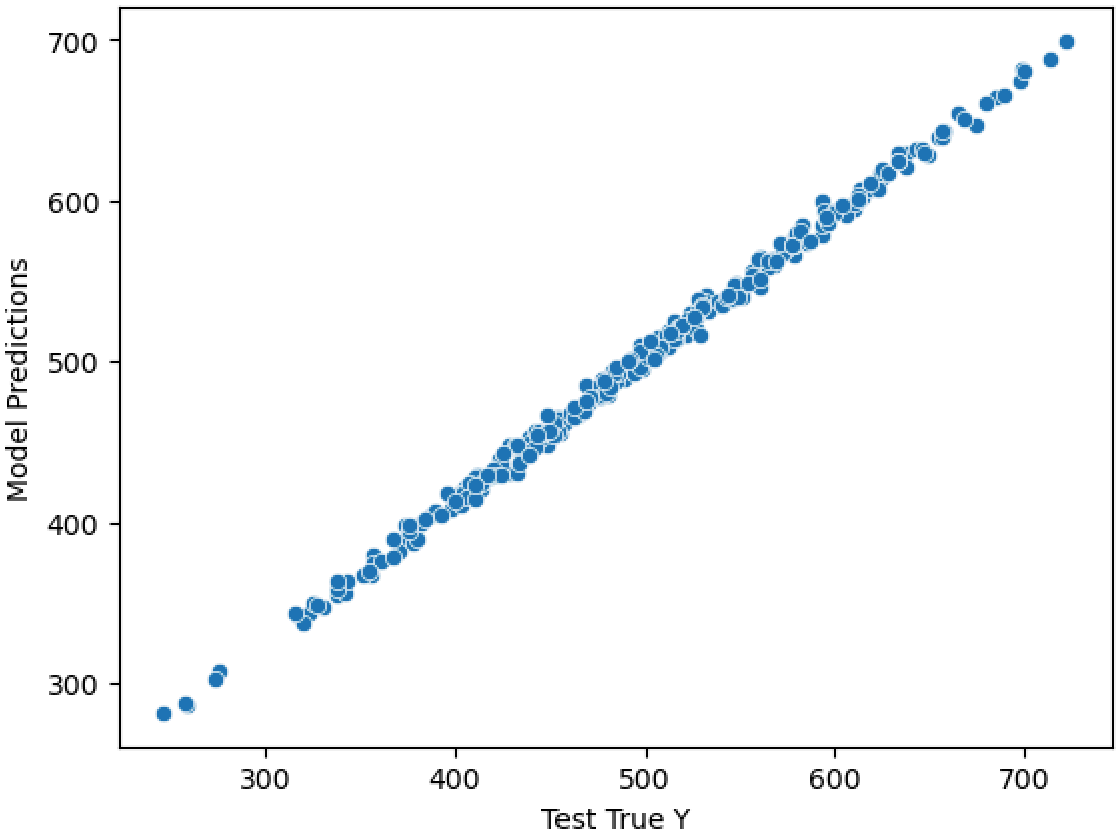
Out[64]:

	Test True Y	Model Predictions
0	402.296319	418.398468
1	624.156198	612.690796
2	582.455066	585.176636
3	578.588606	565.991821
4	371.224104	382.022369
...	...	...
295	525.704657	527.699646
296	502.909473	513.180542
297	612.727910	600.548157
298	417.569725	429.698853
299	410.538250	422.625336

300 rows × 2 columns

In [65]:
```
sns.scatterplot(x='Test True Y', y= 'Model Predictions', data=pred_df)
```

Out[65]:
```
<Axes: xlabel='Test True Y', ylabel='Model Predictions'>
```



In [66]:
```
#The above chart represents the model is working very well
```

In [67]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [68]:
```
mean_absolute_error(pred_df['Test True Y'], pred_df['Model Predictions'])
```

Out[68]: 9.319370338603512

In [69]:
```
df.describe()
```

Out[69]:

	price	feature1	feature2
count	1000.000000	1000.000000	1000.000000
mean	498.673029	1000.014171	999.979847
std	93.785431	0.974018	0.948330
min	223.346793	997.058347	996.995651
25%	433.025732	999.332068	999.316106
50%	502.382117	1000.009915	1000.002243
75%	564.921588	1000.637580	1000.645380
max	774.407854	1003.207934	1002.666308

In [72]:
```
#Root mean Square error
mean_squared_error(pred_df['Test True Y'], pred_df['Model Predictions'])**0.5
```

Out[72]: 11.626055334189301

In [73]:
```
new_gem = [[998, 1000]]
```

In [75]:
```
new_gem = scaler.transform(new_gem)
```

In [76]:
```
model.predict(new_gem)
```

```
1/1 [==============================] - 0s 33ms/step
```

Out[76]:
```
array([[429.40448]], dtype=float32)
```

In [77]:
```
from tensorflow.keras.models import load_model
```

In [78]:
```
model.save('my_gem_model.h5')
```

In [79]:
```
later_model = load_model('my_gem_model.h5')
```

In [80]:
```
later_model.predict(new_gem)
```

```
1/1 [==============================] - 0s 198ms/step
```

Out[80]:
```
array([[429.40448]], dtype=float32)
```

In [ ]: