```
# Machine Learning by Example: from Start to End

#===================Task 1: Checking Your Set Up=================================
import sys
assert sys.version_info >= (3, 7)

from packaging import version
import sklearn
assert version.parse(sklearn.__version__) >= version.parse("1.0.1")
```

Task 2: Getting Data

```
#============Task 2-1: Download the Data: Example Tabular Data======================
from pathlib import Path
import pandas as pd
import tarfile
import urllib.request

def load_housing_data():
    tarball_path = Path("datasets/housing.tgz")
    if not tarball_path.is_file():
        Path("datasets").mkdir(parents=True, exist_ok=True)
        url = "https://github.com/ageron/data/raw/main/housing.tgz"
        urllib.request.urlretrieve(url, tarball_path)
        with tarfile.open(tarball_path) as housing_tarball:
            housing_tarball.extractall(path="datasets")
    return pd.read_csv(Path("datasets/housing/housing.csv"))

housing = load_housing_data()

# Alternative if already downloaded
import pandas as pd
from pathlib import Path
housing = pd.read_csv(Path("datasets/housing/housing.csv"))

# Quick Look at housing data
housing.info()
housing["ocean_proximity"].value_counts()
housing.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
```

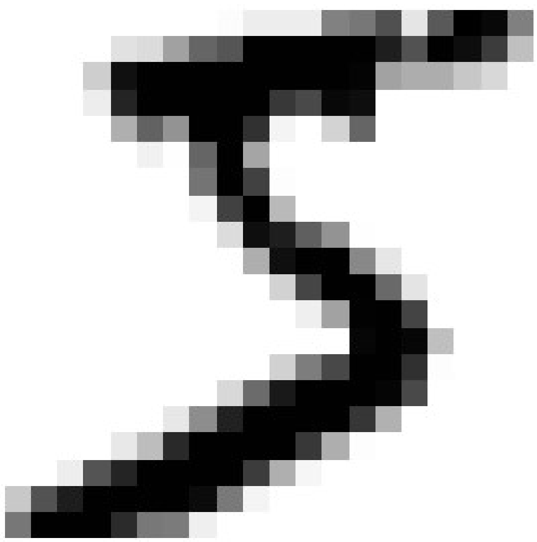
```
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	2
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	
std	2.003532	2.135952	12.585558	2181.615252	421.385070	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	3

```
#======================Task 2-2: Download the Data: Example Image Data====================
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', as_frame=False, parser='auto')

# Quick Look at MNIST
print(mnist.DESCR)
mnist.keys()

# Task 2-4: Identifying the Dimension of Images
images = mnist.data
categories = mnist.target
print(images.shape)
print(categories)

# Visualize a digit
import matplotlib.pyplot as plt

def plot_digit(image_data):
    image = image_data.reshape(28, 28)
    plt.imshow(image, cmap="binary")
    plt.axis("off")

some_digit = mnist.data[0]
plot_digit(some_digit)
```

```
plt.show()
```

```
    **Author**: Yann LeCun, Corinna Cortes, Christopher J.C. Burges
    **Source**: [MNIST Website](http://yann.lecun.com/exdb/mnist/) - Date unknown
    **Please cite**:

    The MNIST database of handwritten digits with 784 features, raw data available at: ht

    It is a subset of a larger set available from NIST. The digits have been size-normali

    With some classification methods (particularly template-based methods, such as SVM ar

    The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns f

    Downloaded from openml.org.
    (70000, 784)
    ['5' '0' '4' ... '4' '5' '6']
```



Task 3: Setting Aside the Test Data

```
#==================Task 3.1: Stratified Sample: Housing Data=======================
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)

# Stratified sampling
import numpy as np
housing["income_cat"] = pd.cut(housing["median_income"],
```

```
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])

strat_train_set, strat_test_set = train_test_split(
    housing, test_size=0.2, stratify=housing["income_cat"], random_state=42)

strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

income_cat	count
3	0.350533
2	0.318798
4	0.176357
5	0.114341
1	0.039971

dtype: float64

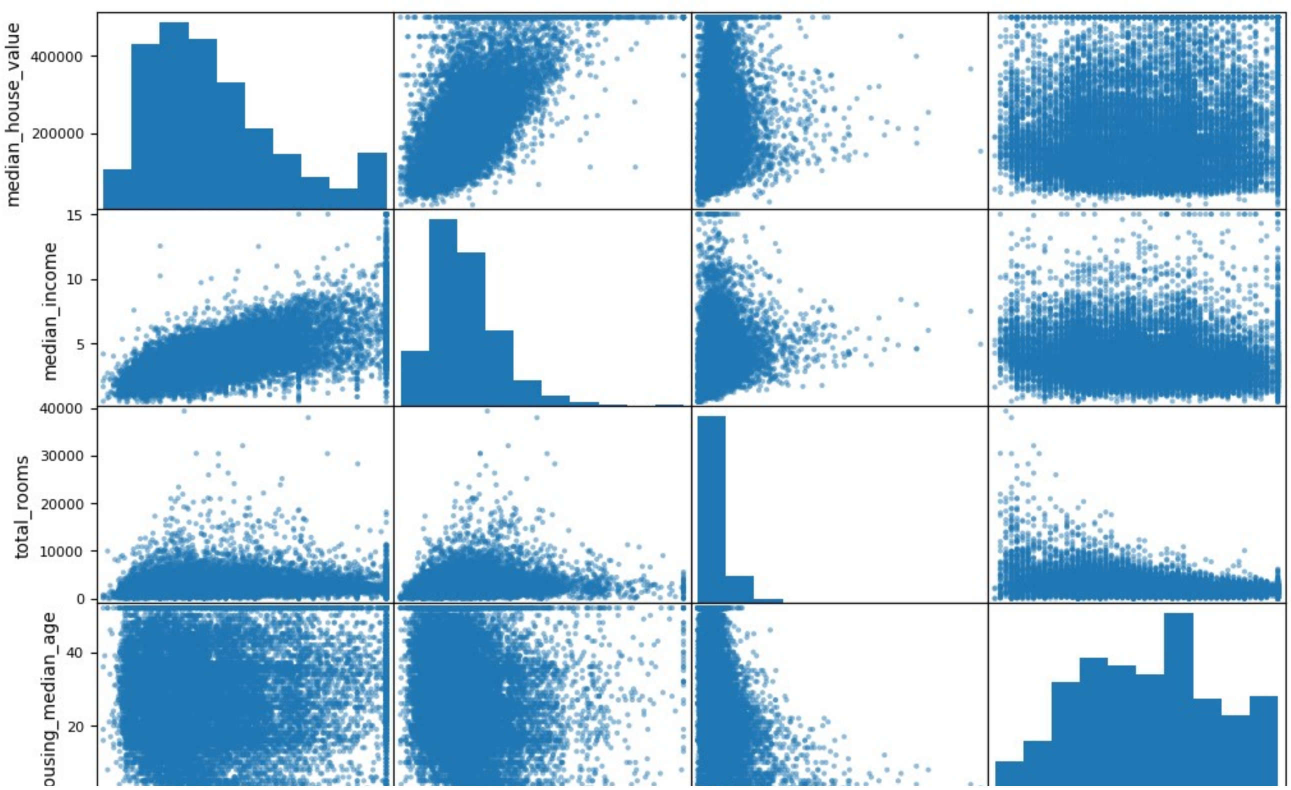
```
#=================Task 3.2: Setting Aside Test Set for Image Data========================
X_train = mnist.data[:60000]
y_train = mnist.target[:60000]
X_test = mnist.data[60000:]
y_test = mnist.target[60000:]
```

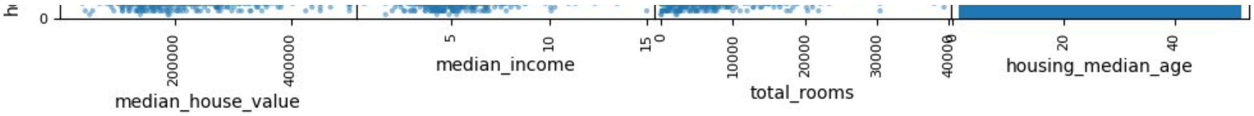
Task 4: Selecting and Training a Model

Double-click (or enter) to edit

```
#====================Task 4-1: Housing Data and Linear Regression========================
housing = strat_train_set.copy()

# Checking correlations
corr_matrix = housing.corr(numeric_only=True)
corr_matrix["median_house_value"].sort_values(ascending=False)

# Visualize correlations
from pandas.plotting import scatter_matrix
features = ["median_house_value", "median_income", "total_rooms", "housing_median_age"]
scatter_matrix(housing[features], figsize=(12, 8))
plt.show()

# Separate labels from data
```

```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()

# Handle missing values
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
housing_num = housing.select_dtypes(include=[np.number])
imputer.fit(housing_num)
housing_num[:] = imputer.transform(housing_num)

# Feature scaling
from sklearn.preprocessing import StandardScaler
std_scaler = StandardScaler()
housing_num_std_scaled = std_scaler.fit_transform(housing_num)

# Scale target labels
target_scaler = StandardScaler()
scaled_labels = target_scaler.fit_transform(housing_labels.to_frame())

# Train linear regression model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(housing_num, scaled_labels)

# Cross validation
from sklearn.model_selection import cross_val_score
rmses = -cross_val_score(model, housing_num, scaled_labels,
                         scoring="neg_root_mean_squared_error", cv=10)
pd.Series(rmses).describe()
```





	0
count	10.000000
mean	0.600606
std	0.007835
min	0.591452
25%	0.594204
50%	0.598207
75%	0.608375
max	0.611381

dtype: float64

```
## Task 4-2: Hand Written Digit Classification
import tensorflow as tf
mnist = tf.keras.datasets.mnist.load_data()

(X_train_full, y_train_full), (X_test, y_test) = mnist

# Scale pixel values
X_train_full = X_train_full / 255.
X_test = X_test / 255.

# Split into training and validation
X_train, X_valid = X_train_full[:-5000], X_train_full[-5000:]
y_train, y_valid = y_train_full[:-5000], y_train_full[-5000:]

# Add channel dimension
import numpy as np
```

```
import numpy as np
X_train = X_train[..., np.newaxis]
X_valid = X_valid[..., np.newaxis]
X_test = X_test[..., np.newaxis]

# Build CNN model
tf.keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, kernel_size=3, padding="same", activation="relu", kernel_in:
    tf.keras.layers.Conv2D(64, kernel_size=3, padding="same", activation="relu", kernel_in:
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Dense(128, activation="relu", kernel_initializer="he_normal"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation="softmax")
])

model.compile(loss="sparse_categorical_crossentropy", optimizer="nadam", metrics=["accuracy
model.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y_valid))

# Evaluate model
model.evaluate(X_test, y_test)

# Compare with SGD Classifier
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import cross_val_score

sgd_clf = SGDClassifier(random_state=42)
accuracy = cross_val_score(sgd_clf, images, categories, cv=10)
print(accuracy)
```

```
Epoch 1/10
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 253s 144ms/step - accuracy: 0.8847 - loss: 0.3857 - va
Epoch 2/10
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 258s 141ms/step - accuracy: 0.9735 - loss: 0.0887 - va
Epoch 3/10
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 254s 137ms/step - accuracy: 0.9797 - loss: 0.0646 - va
Epoch 4/10
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 233s 135ms/step - accuracy: 0.9847 - loss: 0.0506 - va
Epoch 5/10
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 270s 140ms/step - accuracy: 0.9876 - loss: 0.0406 - va
Epoch 6/10
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 261s 140ms/step - accuracy: 0.9883 - loss: 0.0377 - va
Epoch 7/10
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 262s 139ms/step - accuracy: 0.9898 - loss: 0.0304 - va
Epoch 8/10
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 240s 140ms/step - accuracy: 0.9902 - loss: 0.0301 - va
Epoch 9/10
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 257s 137ms/step - accuracy: 0.9914 - loss: 0.0252 - va
```

```
Epoch 10/10
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 237s 138ms/step - accuracy: 0.9923 - loss: 0.0219 - va
313/313 ━━━━━━━━━━━━━━━━━━━━ 10s 31ms/step - accuracy: 0.9906 - loss: 0.0339
[0.872      0.86071429 0.87614286 0.86442857 0.84942857 0.86271429
 0.87914286 0.858      0.892      0.889     ]
```

```
#=================Task 5-3 (Optional): Pre-trained Models==========================
from keras.applications.vgg19 import VGG19
model = VGG19()
print(model.summary())
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vg
574710816/574710816 ━━━━━━━━━━━━━━━━━━━━ 9s 0us/step
Model: "vgg19"
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808

block5_conv4 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102,764,544
fc2 (Dense)	(None, 4096)	16,781,312
predictions (Dense)	(None, 1000)	4,097,000

```
 Total params: 143,667,240 (548.05 MB)
 Trainable params: 143,667,240 (548.05 MB)
 Non-trainable params: 0 (0.00 B)
None
```