

```
mirror_mod = modifier_ob.  
set mirror object to mirror.  
mirror_mod.mirror_object  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True  
  
selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
  
print("please select exactly  
  
-- OPERATOR CLASSES ----  
  
types.Operator):  
on X mirror to the selected  
object.mirror_mirror_x"  
mirror X"  
  
context):  
context.active_object is not
```

# ML Applied Programming Basics

Defining and Running Basic Models and Evaluating Performance

Previously  
we  
learned...

What is machine learning?

What are the major categories of machine learning

(Supervised vs Unsupervised)?

What are some types of machine learning models

(Regression vs Classification, Logistic, Lasso, K-Means, etc.)

Basics of how we set up a model

(Data Prep – Training/Testing Split, Train the Model, Evaluate – Cross validation, etc.)

# Today's Topic - So what does this translate to for practical purposes?

---

We're going to break this into 5 Steps...



**Assess the data**



**Pick your model/  
framework**

Literature review



**Prepare your data**

Basic training/ testing split



**Train your model**



**Evaluate!**

# Today we will cover...



Today's talk will cover supervised learning



Will be in python



Not going to involve any theory

# Some general tips before we get started...

---

It helps to comment your code



Stack Overflow can be your best friend

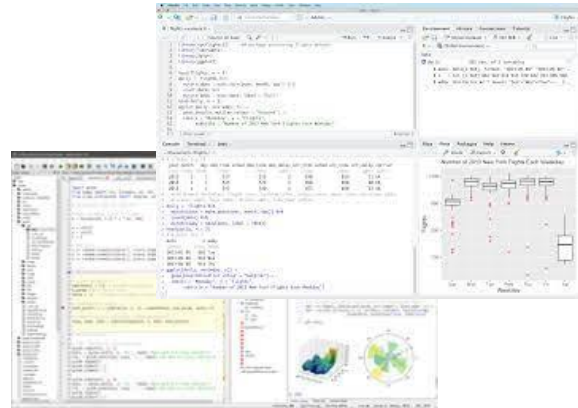


Functions make it easy to repeat code



When all else fails – google

# Getting Started



1. Pick Your Language

2. Choose How You Want to Code

3. Import Your Packages/Libraries

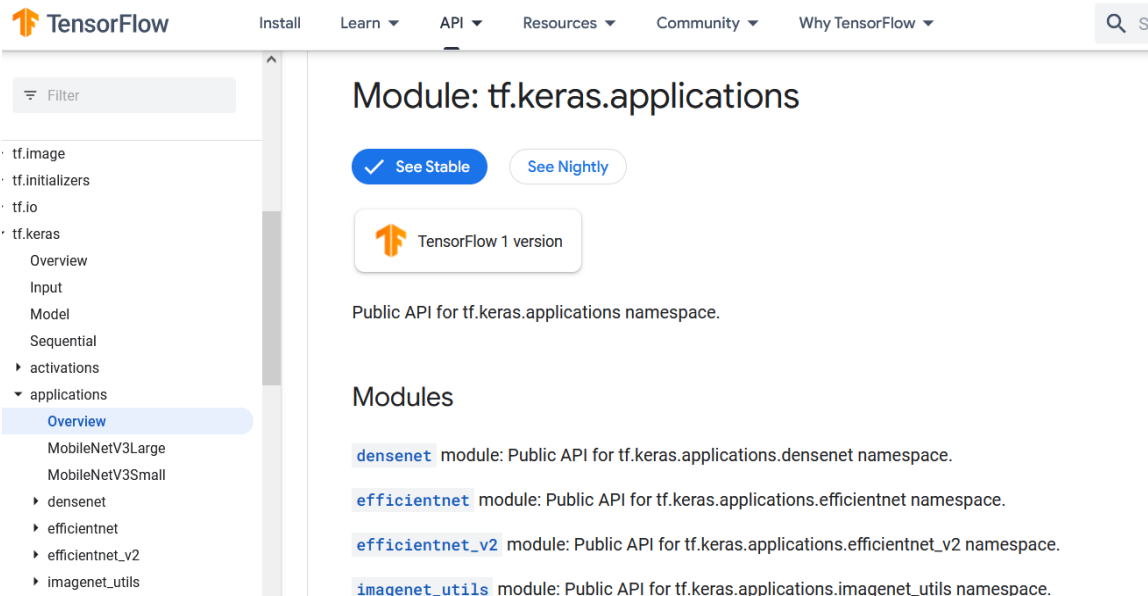
```
import numpy as np  
library(stats)
```

Go!

# Main Packages Used Today (Python)

A

## Image Model: Keras Tensorflow



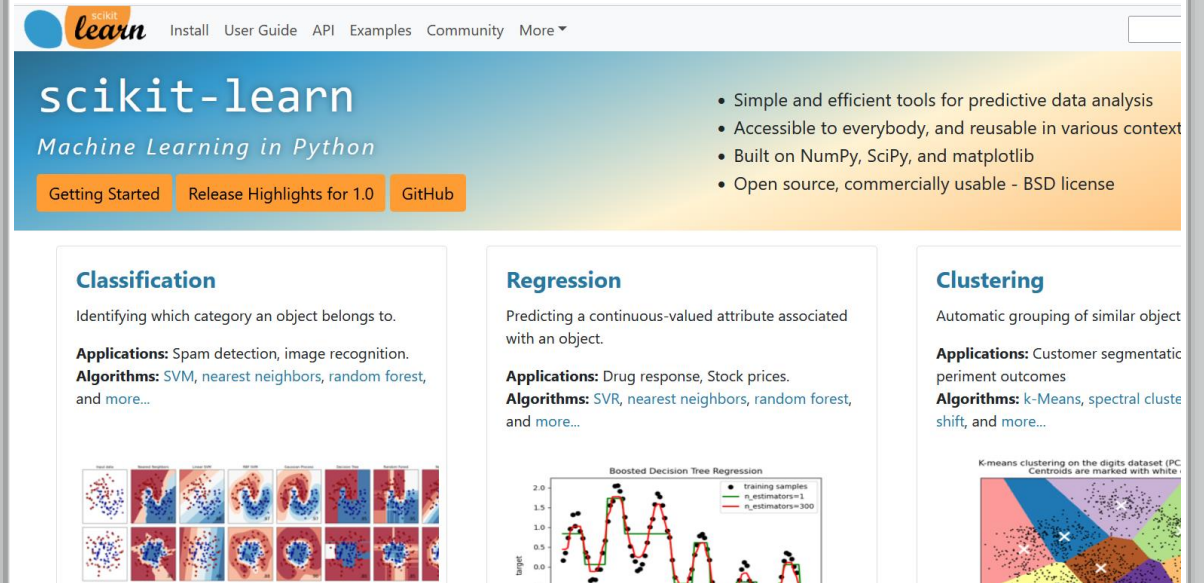
The screenshot shows the TensorFlow website's navigation for the Keras applications module. The main heading is "Module: tf.keras.applications". Below it, there are buttons for "See Stable" (checked) and "See Nightly". A "TensorFlow 1 version" button is also present. The page describes the "Public API for tf.keras.applications namespace." and lists several modules with their descriptions:

- densenet** module: Public API for tf.keras.applications.densenet namespace.
- efficientnet** module: Public API for tf.keras.applications.efficientnet namespace.
- efficientnet\_v2** module: Public API for tf.keras.applications.efficientnet\_v2 namespace.
- imagenet\_utils** module: Public API for tf.keras.applications.imagenet\_utils namespace.

A sidebar on the left lists navigation options: tf.image, tf.initializers, tf.io, tf.keras (with sub-items: Overview, Input, Model, Sequential, activations, applications), and applications (with sub-items: Overview, MobileNetV3Large, MobileNetV3Small, densenet, efficientnet, efficientnet\_v2, imagenet\_utils).

B

## Data Model: Scikit-Learn



The screenshot shows the Scikit-Learn website homepage. The main heading is "scikit-learn" with the tagline "Machine Learning in Python". Navigation links include "Install", "User Guide", "API", "Examples", "Community", and "More". A search bar is visible in the top right.

Key features listed on the right side:

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various context
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

The page is divided into three main sections:

- Classification**: Identifying which category an object belongs to. Applications: Spam detection, image recognition. Algorithms: SVM, nearest neighbors, random forest, and more...
- Regression**: Predicting a continuous-valued attribute associated with an object. Applications: Drug response, Stock prices. Algorithms: SVR, nearest neighbors, random forest, and more...
- Clustering**: Automatic grouping of similar object. Applications: Customer segmentatic periment outcomes. Algorithms: k-Means, spectral cluste shift, and more...

Each section includes a small illustrative figure: Classification shows handwritten digits; Regression shows a line plot of target vs. training samples; Clustering shows a scatter plot of digits with centroids marked.

# Follow Along

- A. For those who want to try to make an imaging model, use the sample eye images
- B. For those who want to try a data based model, use the diabetic retinopathy data





# All Packages Used

```
import cv2
import pydicom as dicom
import PIL as Image
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn import model_selection
from sklearn.model_selection import cross_val_score
```



**Assess the  
data**



**Pick your  
model/  
framework**



**Prepare  
your data**



**Train your  
model**



**Evaluate!**

# 1. Assess the Data

---

- This step doesn't have to be done with code
- If you have an image, display it.
- Dataset or Array? Look at the contents
- Count the number of rows and the number of images, don't lose data later!



# 1. Assess the Data – Image Model



A

#Sample Code to display the image using CV2, this code loads the package, loads in the file, then displays the image

```
import cv2
image = cv2.imread(yourdirectoryhere)
plt.imshow(image)
```

#Sample Code to display DICOM this code loads the package, loads the file, converts it appropriately, then displays the file

```
import pydicom as dicom
import PIL as Image
ds = dicom.dcmread(yourdirectoryhere)
                shape = ds.pixel_array
img = Image.fromarray(image_2d_scaled)
```

**Example Code – You  
Need to Input Your Own  
Directory!**

# 1. Assess the Data – Data Model



B

#Sample Code that looks at the number of rows in a dataset, then the first 5 columns of the dataset

```
import pandas as pd
Dataset = pd.read_csv(yourdirectoryhere)
len(dataset)
pd.dataset.columns
```

#Sample Code to look at the total number of rows and columns in an array and then save them to a variable

```
import numpy as np
nparray = ([], [], yourarraygoeshere)
rows, columns = nparray.shape
```



Assess the  
data



**Pick your  
model/  
framework**



Prepare  
your data



Train your  
model



Evaluate!

## 2. Pick Your Model/ Framework

---



**A** For Images, we'll use a CNN

```
import tensorflow as tf
from tensorflow import keras
```

**B** For data model – a Logistic regression

```
from sklearn.linear_model import
LogisticRegression
```

*The nice thing about scikit-learn and tensorflow is they both have plenty of pre-packaged models and great documentation!*



**Assess the  
data**



**Pick your  
model/  
framework**



**Prepare  
your data**



**Train your  
model**



**Evaluate!**



# 3. Prepare Your Data

---

- Think through any issues with the data:
  - Do any values need to be standardized? How are we going to handle missing values?
  - For example data with missing outcome variables likely can't be used for training/ testing purposes
- Next let's make a training/ testing split



# 3. Prepare Your Data – Image Model

A

```
directory = yourdirectoryhere

img_height = 256
img_width = 256
batch_size = 1

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    directory,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    directory,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width) ,
    batch_size=batch_size)
```

**There are lots of ways to load your data – I like this one because you can specify batch\_size =**



# 3. Prepare Your Data – Data Model

B

```
dataset = pd.read_csv(yourdirectoryhere)
from sklearn.model_selection import train_test_split
```

```
X = dataset[["Pregnancies", "Glucose", "BloodPressure",
"SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction",
"Age"]]
```

```
y = dataset[['Outcome']]
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42)
```





**Assess the  
data**



**Pick your  
model/  
framework**



**Prepare  
your data**



**Train your  
model**



**Evaluate!**

# 4. Train Your Model

---

- Usually for simpler models (like Logistic Regression) we'd have a feature selection step here
- We're going to ignore this for today and focus on just defining the model then training it
- Nice thing about deep learning – does this for you!



# Training – Image Model, Define the Model First

A

```
#Start the model
num_classes = 2

#Make the layers
Model = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.Rescaling(1./255),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(num_classes)
])

#Compile the model
model.compile(
    optimizer='adam',
    loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'])
```



# Training – Image Model, Training is Easy!

A

```
#Train the model
model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=3
)
```



# Training – Data Model (Much Simpler)

B

```
logreg = LogisticRegression()  
logreg.fit(X_train, y_train)
```







Assess the  
data



Pick your  
model/  
framework



Prepare  
your data



Train your  
model



**Evaluate!**



## 5. Evaluate: Image Model

A

- Run model on testing sample

```
accuracy = model.evaluate( val_ds) [1]
```

- Cross-Validation

```
from sklearn.model_selection import KFold
```

```
sklearn.model_selection.KFold(n_splits=5, *,  
shuffle=False, random_state=None)
```

# Cross Validation for the image model is a bit complicated, requires a loop

A

```
j = 0
```

```
for train_idx, val_idx in list(kfold.split(train_x, train_y)):  
    x_train_df = df.iloc[train_idx]  
    x_valid_df = df.iloc[val_idx]  
    j+=1  
  
    train_ds = tf.keras.preprocessing.image_dataset_from_directory(  
        directory,  
        validation_split=0.2,  
        subset="training",  
        ...  
    )  
    ###Same code as your model goes here then evaluate for each kfold  
    accuracy = model.evaluate( val_ds)[1]  
    print("Kfold = " + str(j) + str(accuracy))
```





# 5. Evaluate: Data Model

B

- **Run model on testing sample**

```
y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on test set: {:.2f}'
      .format(logreg.score(X_test, y_test)))
```

- **Cross-Validation**

```
from sklearn import model_selection
from sklearn.model_selection import cross_val_score
kfold = model_selection.KFold(n_splits=10)
modelCV = LogisticRegression()
scoring = 'accuracy'
results = model_selection.cross_val_score(modelCV, X_train, y_train,
cv=kfold, scoring=scoring)
print("10-fold cross validation average accuracy: {:.3f}" % (results.mean()))
```

Putting it all together...

A

# Full Image Model

```
import tensorflow as tf

from tensorflow import keras

directory = yourdirectoryhere

batch_size = 1
img_height = 256
img_width = 256

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    directory,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    directory,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

#Train the model

model.fit(

    train_ds,

    validation_data=val_ds,

    epochs=3

)
```

```
#Start the model
num_classes = 2

#Make the layers
Model = tf.keras.Sequential([

tf.keras.layers.experimental.preprocessing.Rescaling(1./255)
,
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(num_classes)
])

#Compile the model
model.compile(
    optimizer='adam',

loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'])

accuracy = model.evaluate( val_ds)[1]
```

B

# Full Data Model

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split
dataset = pd.read_csv(r"C:\Users\desti\Desktop\Diabetic_Retinopathy_Data.csv")

X = dataset[["Pregnancies",
             "Glucose",
             "BloodPressure",
             "SkinThickness",
             "Insulin",
             "BMI",
             "DiabetesPedigreeFunction",
             "Age"]]

y = dataset[["Outcome"]]
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42)

logreg = LogisticRegression()
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on test set: {:.2f}'
      .format(logreg.score(X_test, y_test)))

from sklearn import model_selection
from sklearn.model_selection import cross_val_score
kfold = model_selection.KFold(n_splits=10)
modelCV = LogisticRegression()
scoring = 'accuracy'
results = model_selection.cross_val_score(modelCV, X_train, y_train,
                                          cv=kfold, scoring=scoring)
print("10-fold cross validation average accuracy: {:.3f}" % (results.mean()))
```

# Common Pitfalls When You're Just Getting Started (For Python)

- Pathways to a directory can change depending on if you're on a Mac or Windows
- Packages can be hard to install or require dependencies – sometimes need to reinstall things
- Need to indent code
- Forget to save your variable to a value
- Forget a parenthesis

***When all else fails – Stack Overflow and Google!!!***

