* Conda and Setup:
* **Create project Environment using Conda:**

Open ‘ Anaconda Prompt (Miniconda)’

Change directory to required location and then give command:

## **conda create --prefix ./env pandas numpy matplotlib scikit-learn jupyter notebook**

* **To activate:**

E.g.; **conda activate "D:\yash\Machine learning\mlconda\env"**

To install new library:  **conda install jupyter notebook**

* **To deactivate: conda deactivate**
* **To Share The environment**:

There a couple of ways to do this:

1. Share your entire project folder (including the environment folder containing all of your Conda packages).

2. Share a .yml (pronounced YAM-L) file of your Conda environment.

The benefit of 1 is it's a very simple setup, share the folder, activate the environment, run the code. However, an environment folder can be quite a large file to share.

That's where 2 comes in. A .yml is basically a text file with instructions to tell Conda how to set up an environment.

For example, to export the environment we created earlier at /Users/daniel/Desktop/project\_1/env as a YAML file called environment.yml we can use the command:

conda env export --prefix /Users/daniel/Desktop/project\_1/env > environment.yml

After running the export command, we can see our new .yml file stored as environment.yml.

Of course, your actual file will depend on the packages you've installed in your environment.

Creating replica environment for new project:

conda env export > environment.yml      #on previous environment with conda active

# create  new folder and paste environment.yml there.

conda env create --prefix ./env -f ./environment.yml

Installing  library within notebook:

1. import sys
2. !conda install --yes --prefix {sys.prefix} seaborn

Within conda activated enviroment:

conda install seaborn

To see version of sklearn:

1. import sklearn
2. sklearn.\_\_version\_\_

To  upgrade the sklearn:

conda activated environment :

1. pip install -U scikit-learn

* **PANDAS :**
* There are 2 main Data types in pandas : Series and DATA frames

Series in one Dimension i.e 1 colunm

It takes only lists , only 1 (2 as index).

series = pd.Series(a)

Data frame in 2 Dimensional i.e. ros and colunms

Takes only dictionary .

df = pd.DataFrame({"he":h,"name":series})

* **To create a data frame of a csv file with pandas:**

df = pandas.read\_csv("filename.csv")   or df = pandas.read\_csv("url.csv")

To export df  into a  csv file :

df.to\_csv("exportfilename.csv", index = False) # index=False is for, not to export the default index column created by pandas into csv file.

In pandas DataFrame , axis=0 is row and  axis=1 is column .

## For time& dateuse

df= pd.read\_csv("data/TrainandValid.csv",low\_memory=False, parse\_dates =["saledate"])

##Sort Dataframe by saledate

When working with time series data,it's good idea to sort it by date.

df.sort\_values(by=["saledate"],inplace=True,ascending=True)

### Make a copy of original DataFrame

df\_copy = df.copy()

* **Attributes and Functions:**

Attributes shows information of dataframe . they do not have () .

Some Attributes:

.dtypes    -- shows the types of data in columns

.columns  --  returns column names

.index    --   ranges of the index i.e. no of columns

.empty -- whether data is empty or not ,boolean value.

.size  -- total elements in df , row \*colmns .

Functions performs operations on the dataframe . they do have () .

Some  functions :

.describe()   -- only focuses on int data type columns and returns statistical operations values .

.info()   -- index combine with dtypes and no. of non null items and memory usage .

.mean()  --  mean of int columns .

.sum()  -- adds all the data in all colunms .

.memory\_usage()  -- memory usage of each columns .

len(df)  -- no. of rows .

* **Selecting and Viewing Data with Pandas:**

1.   .head() -- shows no. of rows from above

   ,  .tail() -- shows no. of rows from below.

2.  .loc[#no.]   -- shows rows of given no. with given  index

,    .iloc[#no.]   -- shows rows with given no. with original index i.e. 0 to n.

3.slicing . e.g; cs\_df[2:]

4.boolean filtering.  e.g ; cs\_df[cs\_df["Doors"] >= 4]

5. Crosstab:  pandas.crosstab(column1,column2)   #It creates a tally of given 2 columns in a table.

6.   .groupby() :  cs\_df.groupby(["Make","Colour"]).sum(numeric\_only=True)

         # It create a table of given columns with along with operating the given function on int datatype of dataframe.

1. #for importing matplotlib
2. %matplotlib inline
3. import matplotlib.pyplot as plt

7. Plotting : cs\_df["Odometer (KM)"].plot(kind="bar")

* String, Fillling ,Deleting Rows:
* DF["stringcolumn"].str       # **.str** enables the string function for the column .

It can be used for lowercase , uppercase,stc purposes.

DF["stringcolumn"].str.lower()       ,     DF["stringcolumn"].str.upper()

car\_sales["Price"] = car\_sales["Price"].str.replace("[\$\,\.]", "").astype(int)

* # To  assign any modifications on the Dataframe , rewrite with  = .

# to directly modify the dataframe enable inplace condition in function : **(inplace = True) .**

* Function to fill Nan in Dataframe :  DF.**fillna(**value to be filled with)

csmdf["Odometer"].fillna(csmdf["Odometer"].mean(),inplace = True) #  odomter Nan filled by odometer column's mean .

* To delete rows with having No value (Nan) use : **DF.dropna()**
* Creating and deleting Columns:

1. Creating New column with series :
   1. seats = pd.Series([5,5,5,5,5,5,5])
   2. csmdf["Seats"] = seats
2. Creating New column with List:
   1. fuel\_eco = [12,2,54,56,90,89,21,3,5,99] # the number of items in list must be equal to no. of already columns .
   2. csmdf["Fuel\_per 100km"]=fuel\_eco
3. Creating New column with existing columns:

csmdf["total fuel used"] = csmdf['Odometer']/100\*csmdf['Fuel\_per 100km']        #you can perform mathematical operatons directly on numerative columns.

1. Directly creating column of same value in all rows :

csmdf["heels"] = 4

1. Deleting a column :

csmdf.drop("Fuel\_eco",axis=1,inplace=True)

for column , axis = 1 for rows ,axis = 0.

* Shuffle , resert\_Index,Apply :
* Shuffling the Rows: use        DF.sample(frac=1)

# frac = 1 for 100 percent, 0.5 for half .

* Resesting Index of Rows:  css.reset\_index(drop = True,inplace=True)
* .apply() function is used to apply a mathematical, operating function on data. lambda is for assigning a unknown value.

css["mile ODometer"] = css["Odometer"].apply(lambda x :x /1.6)

* **NUMPY:**

NumPy is “Numerical Python”. NumPy is used for converting Data into Numbers .

* **Creating Ndarrays and their attributes:**

Create an array:

a1 = np.array([1,2,3])    # pass a list in np.array([])

To check type :

type(a1)

To get Shape of matrix/arrray:

a1.shape

To  get number of dimnsions :

a1.ndim

To get Total number of elements in array:

a1.size

Create a DataFrame from Numpy:

1. import pandas as pd
2. df = pd.DataFrame(a2)
4. # Data frame can be created of upto 2 Dimensions only .

Creating arrays of ones :

ones = numpy.ones((11,4),dtype = int)           #(shape , dtype)

creating arrays of zeros :

zeros = numpy.zeros((2,3))

Creating arrays  with range and specific interval:

range\_array = numpy.arange(1,9,2)   #  1 is starting , 9 is stoping  , 2 is interval.

ar = np.linspace(start,stop, no. of equal parts)

**Creating arrays  with Random :**

random\_array = numpy.random.randint(1,99,(2,3))  # (min.max ,(shape))

OR    random\_array = numpy.random.randint(10,size=(2,3))   #(max,Shape)

random\_array2 = numpy.random.random((4,5)) #GIves random numbers between 0 to 1 , It takes shape .

random\_array3 = numpy.random.rand(4,5) #Similar to random .

* **Random Seed:**

numpy.random.seed(value)

This seed value used further will give same pattern .

It sets pattern for that seed value.

* **Viewing arrays/Matrices:**

To view the unique elements in the Matrix:

numpy.unique(a4)

Slicing can be used to view Elements of matrix:

1. a6[1,0:,:3,:2] # Rows and columns are second last and last numbers.
2. a8[:,:,:,:1]
3. a8[1,1,1,4]

* **Reshape and Transpose:**

Reshape:

a2.reshape(2,3,1)     # It  adds 1's  into dimensions .

Transpose:

a2.T             # Make's transpose of given Matrix/array .

* **Dot product and Element wise multiplication:**

Element  wise Multiplication: Multiplies each element with its respective element.

a1 \* a2         or        numpy.multiply(a1,a2)

Dot Product Multiplication:  Gives the matrix multiplication of to matrices .

numpy.dot(a1,a2)       or      a1.dot(a2)

* **Sorting Ndarrays:**

To sort array and arrange them in ascending order in each row:

1. numpy.sort(random\_array5)
2. array([[ 2, 4, 10, 11, 12],
3. [ 2, 4, 12, 13, 14],
4. [ 1, 7, 7, 7, 9]])

To display index of sorted element in actual Matrix:

1. numpy.argsort(random\_array5)
2. array([[0, 1, 4, 2, 3],
3. [3, 4, 0, 1, 2],
4. [3, 0, 1, 4, 2]], dtype=int64)
5. numpy.argmax(random\_array5) #Highest element at which Index.
6. numpy.argmin(random\_array5) #smallest element at which Index.

* **Turn Images into Ndarrays:**

To Turn Images into Numpy arrays:

1. # import imread from matplotlib.
2. from matplotlib.image import imread
4. panda = imread("images/panda.png") # create ndarrays or images.

* **MATPLOTLIB :**
* **Importing and using Matplotlib:**

1. %matplotlib inline #makes the plot visibble in the notebook.
2. import matplotlib.pyplot as plt
3. import pandas as pd
4. import numpy as np

Ploting by **pyplot interface :**Here we use API's of matplotlib.pyplot only.  Less Flexible.

1. plt.plot(x,y); # x and y are axes, list, having same no. of elements.
2. # ; removes text from output or can use plt.show() .

Ploting by **Axes interface :**Here we use API's of subplot,axes,figure. More Flexible.

1st Method:

1. fig =plt.figure() #creates figure
2. ax = fig.add\_subplot() # add axes
3. ax.plot(x,y) # add data
4. plt.show()

2nd Method :

1. fig = plt.figure() #create a figure
2. ax = fig.add\_axes([1,1,1,1]) #add axes ,compulsory takes 4 arguments.
3. ax.plot(x,y); #add data

3rd Method (Recommended) :

1. fig,ax = plt.subplots()
2. ax.plot(x,y);

* **Matplotlib workflow :**

1. # 0. import matlplotlib and get it ready.
2. %matplotlib inline
3. import matplotlib.pyplot as plt
5. # 1. Prepare data
6. x= [1,2,3,4]
7. y = [11,22,33,44]
9. # 2. Setup plot
10. fig , ax = plt.subplots(figsize = (10,10)) # width and height
12. #3. Plot data
13. ax.plot(x,y)
15. # 4. Customize plot
16. ax. set (title = "It's Plot",
17. xlabel = "x axis",
18. ylabel = "y axis")
20. # 5. Save and show
21. fig.savefig("Images/Output\_fig")

* **Making Different Types of Plots:**

1. Lineplot:
   1. import numpy as np
   2. #plot line the data
   3. fig,ax = plt.subplots()
   4. ax.plot(x,x\*\*2) ;

Line plot is the Default plot of matplotlib.

1. Scattter plot:
   1. #Scatter plot
   2. fig,ax= plt.subplots()
   3. ax.scatter(x,np.exp(x))
   4. #another scatter
   5. fig,ax = plt.subplots()
   6. ax.scatter(x,np.sin(x));
2. Bar plot:
   1. # make a BAR plot from dictionary
   2. nut\_butter = {"Almond":10,"Peanut":5,"Cashew":15}
   3. fig,ax = plt.subplots()
   4. ax.bar(nut\_butter.keys(), height= nut\_butter.values())
   5. ax.set(title="Nut\_Butter Store", xlabel='nuts',ylabel='prices');
   6. # Horizontal Bar
   7. fig,ax = plt.subplots()
   8. ax.barh(list(nut\_butter.keys()),list(nut\_butter.values()))
3. Histogram
   1. #Histogram
   2. x = np.random.randn(100)#creates standard normal
   3. fig,ax = plt.subplots()
   4. ax.hist(x,color="RED");

* **Subplots:**

There are two ways of plot many plots in a same figure:

1. # 1
2. fig , ((ax1,ax2),(ax3,ax4)) = plt.subplots(nrows=2,ncols=2,figsize = (10,5))
3. #ploting to each different axes
4. ax1.plot(x,x\*2);
5. ax2.bar(nut\_butter.keys(),nut\_butter.values());
6. ax3.scatter(np.random.random(19),np.random.random(19));
7. ax4.hist(np.random.random(90));

# 2

1. fig,ax = plt.subplots(nrows=2,ncols=2,figsize=(10,5))
2. #[rows,colunms]
3. ax[0,0].plot(x,x\*2);
4. ax[0,1].bar(nut\_butter.keys(),nut\_butter.values());
5. ax[1,0].scatter(np.random.random(19),np.random.random(19));
6. ax[1,1].hist(np.random.random(90));

Both have the Same Plot.

* **A visualization of given data :**

1. #Create figure with subplots having 2 axes.
2. fig,(ax0,ax1) =plt.subplots(nrows=2, ncols=1, figsize=(10,10),
3. sharex=True,) #fig shares same x axis.
5. #add data to ax0
6. ax0.scatter(x=over50['age'],
7. y=over50["chol"],
8. c=over50['target']) # c is for color to be given according to given column.
10. #customize ax0
11. ax0.set(title="Heart Disease and Cholestrol level",
12. ylabel ="CHOl")
14. #add legend
15. ax0.legend(\*scatter.legend\_elements(),title="Target") # create a scale for axes.
17. #ADDs hozrizontal mean line for yaxis
18. ax0.axhline(over50["chol"].mean(),linestyle=":")
20. #add data to ax1
21. ax1.scatter(x= over50["age"],y=over50["thalach"],c=over50["target"])
22. ax1.set(title="Heart Diesase and Maximum heart rate",
23. xlabel = "Age",
24. ylabel=" Maximum heart rate")
25. ax1.legend(\*scatter.legend\_elements(),title="target")
26. ax1.axhline(over50["thalach"].mean(),linestyle=":")
28. #Add title to the figure
29. fig.suptitle("Heart Disease Analysis",fontsize = 16,fontweight = "bold");

* **Styles :**

Seeing Different Styles:

plt.style.available

Using Style:

plt.style.use("seaborn-v0\_8-whitegrid")

Changing color scheme in given style:

cmap="winter" # use in plotting

Setting limits:

1. ax0.set\_xlim([50,80])
2. ax0.set\_ylim([60,200])

* **Saving Image:**

fig.savefig("name.png")

* **SCIKIT-LEARN :**
* **Scikit learn Model WorkFlow:**

#1Getting data ready.

1. import pandas as pd
2. import numpy as np
3. heart\_disease = pd.read\_csv("heart-disease.csv")
4. # create x(feature matrix,)
5. x= heart\_disease.drop("target",axis= 1)
6. #create y (decision label) y = labels , output, answers for x
7. y = heart\_disease["target"]

#2Choose the right model and hyperparameter

1. from sklearn.ensemble import RandomForestClassifier # Random forest is a Classification machine learning Model.
2. clf = RandomForestClassifier()

#3Fit the model to training data

1. from sklearn.model\_selection import train\_test\_split
2. x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size = 0.2) # splitting the given data into train and test data to be used.
4. clf.fit(x\_train,y\_train); # finds patterns in training data , trains itself.
6. y\_pred = clf.predict(x\_test) # makes predictions of y(answers) on x\_test data according to traing done above by train data.
7. y\_pred # can only make predictions on similar kind of data.

#4 Evaluate the model

1. clf.score (x\_test,y\_test) # Gives mean accuracy on given test data and labels(y)
2. clf.score(x\_train,y\_train) # 1.0 is 100% max , here it is because model is train on same data only.
3. from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score # fe more evalution metrics
4. print(classification\_report(y\_test,y\_pred)) # compares y\_test and y\_prediction
5. confusion\_matrix(y\_test,y\_pred)
6. accuracy\_score(y\_test,y\_pred)

#5 Improve Model

1. #### try different estimators
2. np.random.seed(1)
3. max = 0
4. for i in range(10,100,10):
5. print(f"trying model with {i} estimator")
6. clf = RandomForestClassifier(n\_estimators = i).fit(x\_train,y\_train)
7. a = clf.score(x\_test,y\_test)
8. aa.append(a)
9. estim.append(i)
10. print(f"Model accuracy on test :{a \* 100:.2f} % ")
11. print("")
12. if a > max:
13. esti = i
14. max = a
15. print(max)
16. print(esti)
17. print(f" Max Model accuracy on test is :{max \* 100:.2f} % ")
18. print("with estimator",esti)

#6Save and Load

1. import pickle
2. pickle.dump(clf,open("random\_forest\_model1.pkl","wb")) #saving the model with pickle , wb stands for write binary.

5. loaded\_model = pickle.load(open("random\_forest\_model1.pkl","rb")) # loading the model pickle.load ,rd stands for read binary.
6. loaded\_model.score(x\_test,y\_test)

model.score()  gives "coefficient of determination of the prediction(R^2)".

X = feature matrix, features, feature variables, data, Independent variables.

Y = Labels, target, target variables ,answers for X , dependent variables.

* **Turning categories into numbers.**

1. from sklearn.preprocessing import OneHotEncoder
2. from sklearn.compose import ColumnTransformer
4. categorical\_features = ["Make","Colour","Doors"] #features/Columns that need to encoded in numbers.
5. onehot = OneHotEncoder() #Encodes categories into number(1,0)
6. transformer = ColumnTransformer([("onehot",onehot,categorical\_features)],remainder = "passthrough") #ColumnTransformer takes values as name,transformer,Columns.
7. #It applies tranformer(Encoding) to colunms of Dataframe.
9. Transformed\_X = transformer.fit\_transform(X)

#Another way using pandas get\_dummies

1. dummies = pd.get\_dummies(car\_sales[["Make","Colour","Doors","Odometer (KM)"]],dtype=float)
2. dummies

# Building a model To predict numbers.

1. from sklearn.ensemble import RandomForestRegressor #RandomForestRegressor is used to predict numbers(e.g prices)
2. model=RandomForestRegressor()
4. np.random.seed(11)
5. X\_train,X\_test,Y\_train,Y\_test =train\_test\_split(Transformed\_X,Y,test\_size = 0.2)
6. model.fit(X\_train,Y\_train)

A screenshot of a computer

Description automatically generated

* **Filling the missing data:**

#Option1  Fill the missing data with pandas

1. #Fill the "Make" column
2. car\_sales\_missing["Make"].fillna("missing",inplace = True)
3. #fill the "Colour" column
4. car\_sales\_missing["Colour"].fillna("missing",inplace = True)
5. #Fill the "Odometer (KM)" column
6. car\_sales\_missing["Odometer (KM)"].fillna(car\_sales\_missing["Odometer (KM)"].mean(),inplace=True)
7. #Fill the "Doors" column
8. car\_sales\_missing["Doors"].fillna(4,inplace=True)
10. #Remove the missing values from label(price) column
11. car\_sales\_missing.dropna(inplace=True)

#Option 2  filling missing data with Scikit-learn

1. from sklearn.impute import SimpleImputer
2. from sklearn.compose import ColumnTransformer
4. #Create Sub imputers
5. cat\_imputer = SimpleImputer(strategy="constant",fill\_value = "missing")
6. door\_imputer = SimpleImputer(strategy="constant",fill\_value = 4)
7. num\_imputer = SimpleImputer(strategy = "mean")
9. #Define columns
10. cat\_features = ["Make","Colour"]
11. door\_features = ["Doors"]
12. num\_features = ["Odometer (KM)"]
14. #Create main Imputer with columntransformer
15. imputer = ColumnTransformer([("cat\_impute",cat\_imputer,cat\_features),
16. ("door\_impute",door\_imputer,door\_features),
17. ("num\_impute",num\_imputer,num\_features)])
19. #transform the data
20. filled\_x\_train = imputer.fit\_transform(x\_train) #fit\_transform imputes the missing values from the training set and fills them simultaneously.
22. filled\_x\_test = imputer.transform(x\_test) # tranform takes the imputing missing values from the training set and fills the test set with them.

* **Another way of preprocessing Data**:

Turning object into Category values:

1. # In pandas , category values are treated as number codes undeer the hood.
2. for label,content in df.items():
3. if pd.api.types.is\_object\_dtype(content):
4. df[label] = content.astype("category").cat.as\_ordered() # also arranging categories alphabetically.

## ascessing the numeric values of categories.

df["state"].cat.codes

filling the numeric values:

1. # Fill numeric rows with the median. Median is robust than mean .Since it is the most repeated value.
2. for label,content in df.items():
3. if pd.api.types.is\_numeric\_dtype(content):
4. if pd.isnull(content).sum():
5. # Add a binary column which tells us if the data was misssing for the row of that colunms
6. df[label+"\_is\_missing"] = pd.isnull(content)
7. # Fill the missing value with median
8. df[label]=content.fillna(content.median())

filling missing categorical values:

1. # Turn categorical variables into numbers and filling missing
2. for label,content in df.items():
3. if not pd.api.types.is\_numeric\_dtype(content):
4. # Add binary column to indicate whether sample had missing values.
5. df[label+"\_is\_missing"] = pd.isnull(content)
6. #turn categories into numbers and add +1 because after turning into numbers missing value is -1.
7. df[label]=pd.Categorical(content).codes+1

### Save preprocessed Data

df.to\_csv("data/train\_temp.csv",index=False) ##while saving the category dtype is changed into object dtype for csv. we can use different format such pickle to avoid this.

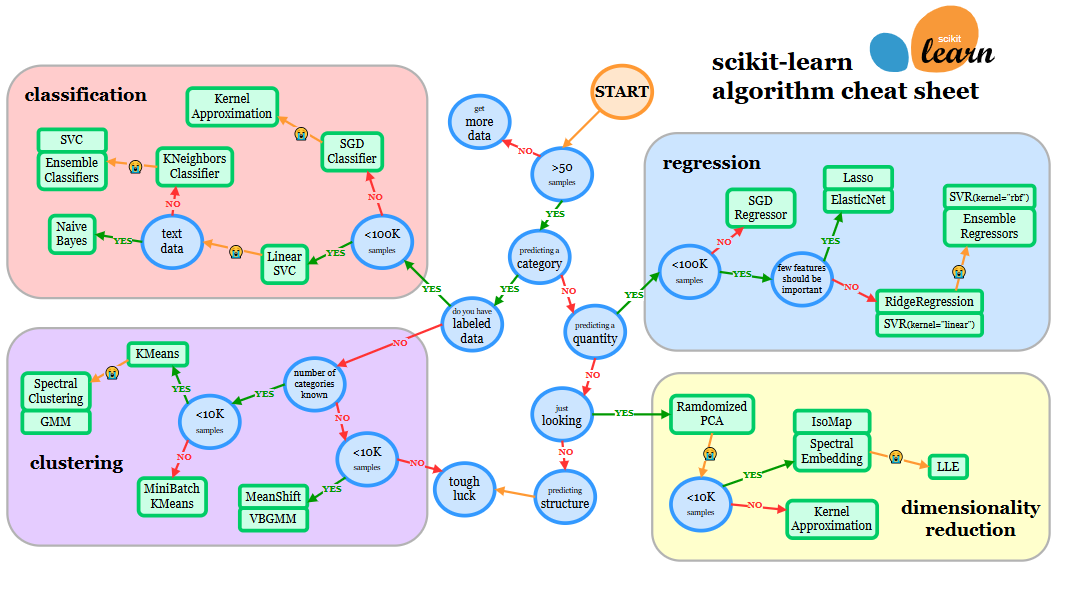
1. df.to\_pickle("tain\_temp.pkl")
2. dffff=pd.read\_pickle('tain\_temp.pkl')

# Preprocessing the Time Series Data:

1. def preprocess\_data(df):
2. """Performs transformation on DataFrame and gives Transformed Dataframe."""
3. #### Add datetime parameters for `saledate` column
4. df["saleyear"] = df.saledate.dt.year
5. df["salemonth"] = df.saledate.dt.month
6. df["saleday"] = df.saledate.dt.day
7. df["salemdayofyear"] = df.saledate.dt.dayofyear
8. df["saledayofweek"] = df.saledate.dt.dayofweek
9. #Removing saledate
10. df.drop("saledate",axis=1,inplace=True)

13. # Fill numeric rows with the median. Median is robust than mean .Since it is the most repeated value.
14. for label,content in df.items():
15. if pd.api.types.is\_numeric\_dtype(content):
16. if pd.isnull(content).sum():
17. # Add a binary column which tells us if the data was misssing for the row of that colunms
18. df[label+"\_is\_missing"] = pd.isnull(content)
19. # Fill the missing value with median
20. df[label]=content.fillna(content.median())
22. # Turn categorical variables into numbers and filling missing
23. for label,content in df.items():
24. if not pd.api.types.is\_numeric\_dtype(content):
25. # Add binary column to indicate whether sample had missing values.
26. df[label+"\_is\_missing"] = pd.isnull(content)
27. #turn categories into numbers and add +1 because after turning into numbers missing value is -1.
28. df[label]=pd.Categorical(content).codes+1
30. return df

* **Choosing The Right Model For Your Data:**



Choosing estimators using sklearn model map - https://scikit-learn.org/stable/tutorial/machine\_learning\_map/

Various Regression etimators:

    1. linear\_modela.LinearRegression

    2. linear\_model.Ridge

model.score()  gives "coefficient of determination of the prediction(R^2)".

we use random.seed  to  make reproducible results ,such that if we run again the cell the same output will be generated.

We can try ensemble model , Ensemble model is combination of smaller models to try and make better prediction then just a single model.

RandomForest is a ensemble model which is set of many Decision trees.It collects the votes from different decision trees to decide the final prediction.

It is one of most usefull estimator.

Tidbit:

    1. If you have structured data,use ensemble methods.

    2.If  you have unstructured data,use deep learning or transfer learning.

* **PREDICT VS PREDIC\_PROBA :**

2 ways to make predictions: 1. predict() 2. predict\_proba()

1. clf.predict(X\_test)
2. # it gives the y predicted results of x\_test.
3. clf.predict\_proba(X\_test[:10])
4. # it gives the probability of each sample of x regrading the possible outcome of y.
5. #can be only used for classification problems.

* **EVALUATING A MACHINE LEARNING MODEL:**

Cross validation : the model is trained on different versions of training data and then evaluated on different versions of training data .

1. from sklearn.model\_selection import cross\_val\_score
3. cross\_val\_score( clf, X, Y, cv = 5,scoring = None )
4. # if the scoring parameter is none ,it use default metric of estimator (i.e. here for classifer it's mean accuracy score).
5. # cv = cross validation folds , default cv is 5 .

A diagram of a test results

Description automatically generated with medium confidence

* **Evaluating A Classification Model :**

**1. Accuracy**

1. #no need of spilting for cross validation.
3. clf = RandomForestClassifier()
4. crossvalscore = cross\_val\_score(clf, X,Y) # by default its scoring it set as accuracy.
6. print(f"Heart Disease Classifier Cross-Validation Accuarcy is : {np.mean(crossvalscore)\*100:.2f}%")

**2. Area under the Receiver Operating Charrecteristic (ROC) Curve**

* Area under Curve(AUC)
* ROC curve

ROC Curve is comparison of model's True positive rate(tpr) and model's False positive rate(fpr) .

* True Positive = model predicts 1 when truth is 1.
* False Positive = model predicts 1 when truth is 0.
* True negative = model predicts 0 when truth is 0.
* False negative = model predicts 1 when truth is 1.

1. from sklearn.metrics import roc\_curve
2. X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2)
3. clf.fit(X\_train,Y\_train)
4. y\_probs = clf.predict\_proba(X\_test)
5. y\_probs\_positive = y\_probs[:,1]
7. # calculate fpr, tpr amd thershold
8. fpr, tpr , thershold = roc\_curve(Y\_test, y\_probs\_positive)
10. #roc\_curve takes Y\_test and probability estimates of the positive class.

ROC curves and AUC metrics are evaluation metrics for binary classification models (a model which predicts one thing or another, such as heart disease or not).

The ROC curve compares the true positive rate (tpr) versus the false positive rate (fpr) at different classification thresholds.

The AUC metric tells you how well your model is at choosing between classes (for example, how well it is at deciding whether someone has heart disease or not). A perfect model will get an AUC score of 1.

plotting ROC Curve:

1. def plot\_roc\_curve(fpr,tpr):
2. fig,ax = plt.subplots()
3. """
4. Plots a ROC curve given the false posiive rate (fpr) and
5. true positive rate(tpr) of a model .
6. """
7. #plot roc curve
8. ax.plot(fpr,tpr, color="orange",label="ROC")
10. #plot line with no predective power (baseline)
11. ax.plot([0,1],[1,0], color="darkblue",linestyle="--", label="Guessing")
13. #customize plot
14. ax.set(title = "Receiver Operating charecteristic (ROC) Curve" ,
15. xlabel = "False positive rate(fpr)" ,ylabel = "True positive rate(tpr)")
16. ax.legend()
17. plt.show()
19. plot\_roc\_curve(fpr,tpr)
20. from sklearn.metrics import roc\_auc\_score
21. roc\_auc\_score(Y\_test ,y\_probs\_positive) # ROC has perfectscore 1.

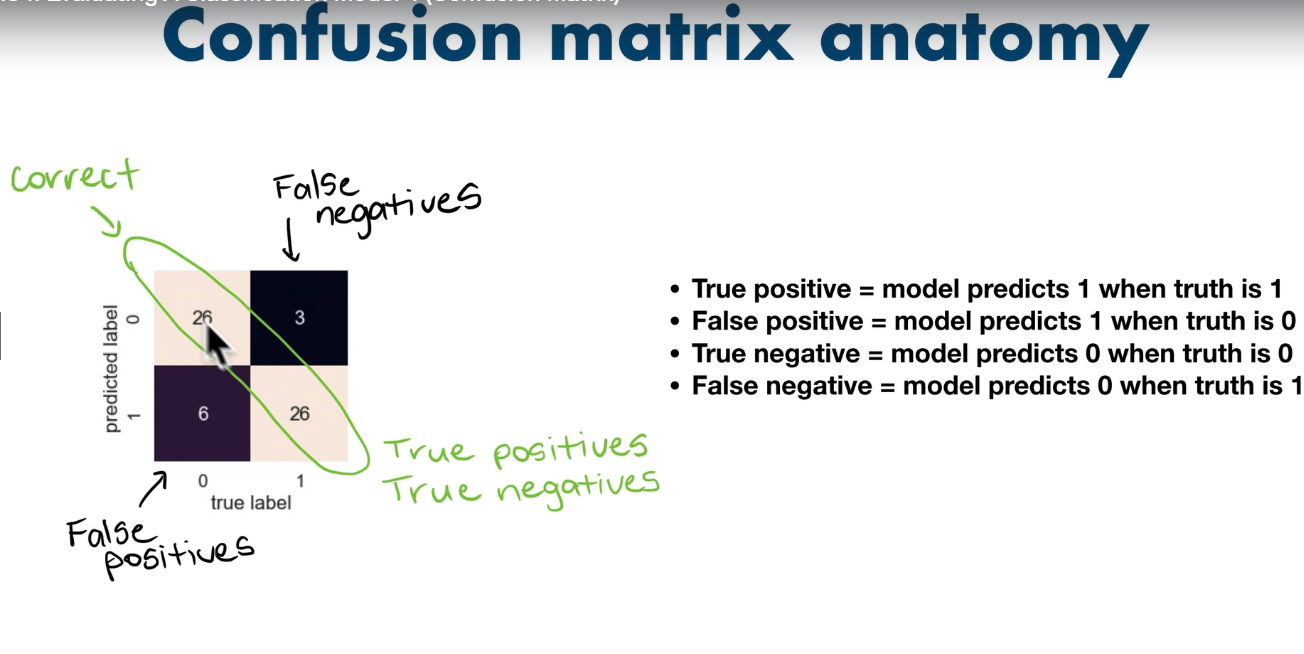
**3. Confusion Matrix**

A confusion matrix is a quick way to compare the labels a model predicts and the actual labels it was supposed to predict . In essence, giving you a idea of where the model is getting confused.

1. y\_pred = clf.predict(X\_test)
3. from sklearn.metrics import confusion\_matrix
4. confusion\_matrix(Y\_test, y\_pred)
5. # Visualising with pandas crosstab
6. pd.crosstab(Y\_test,y\_pred,rownames = ['Actual Labels'],colnames = ['predicted Labels'])

We want The Diagonal elements to be more as they are correct, True labels matches prediction labels (True positive , True negative).

1. # Visualising with pandas crosstab
2. pd.crosstab(Y\_test,y\_pred, rownames=['Actual Labels'], colnames = ['predicted Labels'])



Creating confusion matrix with sklearn:

from sklearn.metrics import ConfusionMatrixDisplay

1. #1st method by from\_estimator()
2. ConfusionMatrixDisplay.from\_estimator(estimator=clf, X=X,y=Y)
3. #Makes predictions by itself ,takes X and y and estimator.
4. #2nd method by from\_predictions()
5. ConfusionMatrixDisplay.from\_predictions(Y\_test, y\_pred)
6. #takes predictions.

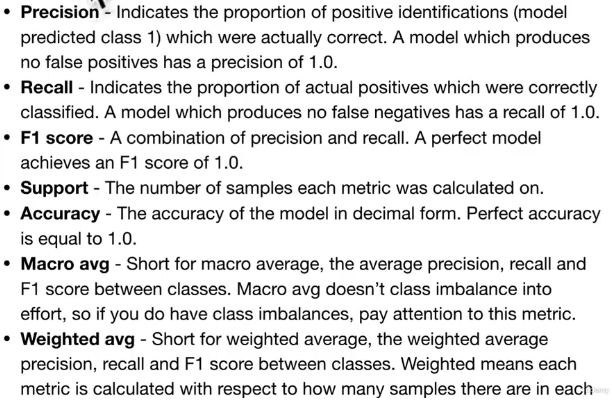
You can change the colour of graph by:

1. #cmap
2. ConfusionMatrixDisplay.from\_predictions(Y\_test, y\_pred,cmap="RdBu\_r")

**4. Classification\_report:**

1. from sklearn.metrics import classification\_report
2. print(classification\_report(Y\_test,y\_pred))

* **Accuracy** is a good measure to start with if all classes are balanced(e.g. same amount of samples ).
* **Precision** and **Recall** beacome more important when classes are imbalanced.
* If false positive predictions are worse than false negatives, aim for higher precision.
* If false negative predictions are worse than false positives,aim for higer recall.
* **F1-Score** is combonation of precision and recall.



# precision becomes 1 when ,model predicts 0 and truth is 1 i.e FalsePositives are None.

#recall becomes 1 when, model predicts 1 and truth is 0 i.e. FalseNegatives are None.

**5.Feature Importance:**

Feature Importance is asking,"which features contributed most to outcomes of the model and how did they contributed ?"

Feature Importance is different for each machine learning model.

Let's find features importance for our LogisticRegression model.

1. clf.coef\_ #It is Coefficient of the features w.r.t. the decision function in LogisticRegresion.
2. # +ve coef means feature is Directly related to decision feature.
3. #-ve coef means feature is inversly related to decision feature.
5. #Visualize feature importance
6. feature\_df = pd.DataFrame(feature\_dict,index=[0])
7. feature\_df.T.plot.bar(title = "Feature importance",legend=False);

* **Regression model evaluation metrics:**

1. Coefficient of determination (R^2):

Compares your models prediction to the mean of targets. Values can range from negative infinity(bad model) to 1(perfect model) . For e.g., if all your model does is predict the mean of the targets,it's R^2 value would be 0.And if your prefectly predicts a range of numbers it's R^2 value would be 1.

* 1. #if prediction is mean of Y\_test , it gives 0 for R^2 .
  2. # r2\_score is same as model.score for regression
  3. from sklearn.metrics import r2\_score
  5. r2\_score(y\_true=Y\_test,y\_pred= Y\_pred)

1. Mean absolute error(MAE):

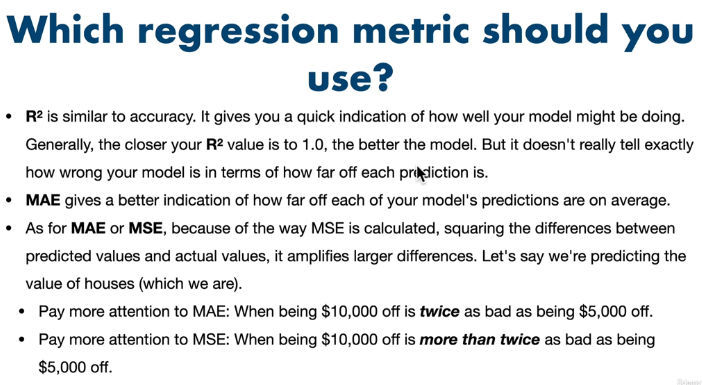
MAE is the average of absolute differences between predictions and actual values. It gives you an idea of how wrong your models predictions are .

* 1. from sklearn.metrics import mean\_absolute\_error
  3. mean\_absolute\_error(Y\_test,y\_pred)

1. Mean sqaured error (MSE):

MSE is the mean of the square of errors between actual valuesand predictions.

* 1. from sklearn.metrics import mean\_squared\_error
  2. mean\_squared\_error(Y\_test,y\_pred)



* **Finally using the scoring parameter:**

Classification:

1. from sklearn.model\_selection import cross\_val\_score
2. #cross validation
3. np.random.seed(9)
4. cross\_ac = cross\_val\_score(clf,X,y,cv=5,scoring=None) #scoring = None =default = accuracy for classifier
6. scoring can be any metric such as "precision","recall","f1"
8. np.random.seed(9)
9. cv\_precision = cross\_val\_score(clf, X, y,scoring="precision")

Regression:

1. np.random.seed(9)
2. cv\_r2 = cross\_val\_score(model,X,Y,cv=5) # for regression default scoring = R^2 coefficient of detewrmination.
4. np.random.seed(9)
5. #for MAE and MSE the scoring parameter gives negative values. Therefore value closer to 0 is better.
6. cv\_mae = cross\_val\_score(model,X,Y,cv=3,scoring="neg\_mean\_absolute\_error")
8. cv\_mse = cross\_val\_score(model,X,Y,cv=3,scoring="neg\_mean\_squared\_error")

Default scoring metric for classification is accuracy and for Regression is R^2 .

* **Evaluation function:**

1. from sklearn.metrics import accuracy\_score,precision\_score,recall\_score,f1\_score
2. def evaluate\_preds(ytrue,ypreds):
3. """Preforms evalutaion comparision on ytrue labels vs. y\_pred labels on classification."""
4. acc = accuracy\_score(ytrue,ypreds)
5. prec = precision\_score(ytrue,ypreds)
6. rec= recall\_score(ytrue,ypreds)
7. f1 = f1\_score(ytrue,ypreds)
8. metric\_dict ={"accuracy": round(acc,2),
9. "precision": round(prec,2),
10. "recall": round(rec,2),
11. "f1": round(f1,2)}
12. print(f"Accuracy: {acc\* 100:.2f}%")
13. print(f"precision: {prec\* 100:.2f}%")
14. print(f"recall: {rec\* 100:.2f}%")
15. print(f"f1\_score: {f1\* 100:.2f}%")
16. return metric\_dict

* **TUNING HYPERPARAMETERS:**
* **Shuffling and splitting data into 3 sets:**

#shuffling the data:

1. heart\_disease\_shuffled = heart\_disease.sample(frac=1)

#spiltting the data into train, validation & test sets:

1. train\_split = round(0.7 \* len(heart\_disease\_shuffled)) # 70 - 15- 15
2. valid\_split = round(train\_split + 0.15 \* len(heart\_disease\_shuffled))
3. X\_train ,Y\_train = X[:train\_split], Y[:train\_split]
4. X\_valid , Y\_valid = X[train\_split: valid\_split],Y[train\_split: valid\_split]
5. X\_test, Y\_test = X[valid\_split:] , Y[valid\_split:]

A diagram of a model of food

Description automatically generated

A diagram of a patient record

Description automatically generated

* **Hyperparameter tuning with RandomizedSearchCV**

1. from sklearn.model\_selection import RandomizedSearchCV
3. grid = {"n\_estimators":[10,100,200,500,1000,1200],
4. "max\_depth":[None,5,10,20,30],
5. "max\_features":["auto","sqrt"],
6. "min\_samples\_split":[2,4,6],
7. "min\_samples\_leaf":[1,2,4]}
9. np.random.seed(42)
11. #spilt into X and Y
12. X= heart\_disease\_shuffled.drop("target",axis=1)
13. Y= heart\_disease\_shuffled["target"]
15. #train and test data
16. X\_train,X\_test ,Y\_train,Y\_test = train\_test\_split(X,Y ,test\_size=0.2)
18. #Instantiate RandomforestClassifier
19. clf = RandomForestClassifier(n\_jobs=1) #n\_jobs : how much part of your computer processor will be dedicated for it.-1 is all of it.
21. #Setup Randomisedsearchcv
22. rs\_clf = RandomizedSearchCV(estimator=clf,
23. param\_distributions= grid,
24. n\_iter=10, #number of combinaions models to try
25. cv =5,
26. verbose=2)
28. #Fit the RandomizedSearchCV version of clf
29. rs\_clf.fit(X\_train,Y\_train)

Best parameters:

rs\_clf.best\_params\_

* **Hyperparameter tuning with GridSearchCV:**

1. from sklearn.model\_selection import GridSearchCV
3. grid2 = {"n\_estimators":[100,200,500],
4. "max\_depth":[None,10],
5. "max\_features":["sqrt"],
6. "min\_samples\_split":[6],
7. "min\_samples\_leaf":[1,2]}
9. np.random.seed(42)
11. #spilt into X and Y
12. X= heart\_disease\_shuffled.drop("target",axis=1)
13. Y= heart\_disease\_shuffled["target"]
15. #train and test data
16. X\_train,X\_test ,Y\_train,Y\_test = train\_test\_split(X,Y ,test\_size=0.2)
18. #Instantiate RandomforestClassifier
19. clf = RandomForestClassifier(n\_jobs=1) #n\_jobs : how much part of your computer processor will be dedicated for it.-1 is all of it.
21. #Setup GridSearchCV
22. gs\_clf = GridSearchCV(estimator=clf,
23. param\_grid= grid2, # It does not have no\_iters ,it peforms every combination.
24. cv =5,
25. verbose=2)
27. #Fit the GridSearchCV version of clf
28. gs\_clf.fit(X\_train,Y\_train)
30. gs\_eval =evaluate\_preds(Y\_test,gs\_y\_preds)

Testing our model on Subset(to tune the hyperparameters)

1. # Change max\_samples in model. It limits the number of samples .
2. model = RandomForestRegressor(n\_jobs=-1 ,random\_state=99, max\_samples=10000)

#compare metrices

1. compare\_matrics = pd.DataFrame({"Baseline":baseline\_metrics,
2. "ClfMetric2":clf\_2Metrics,
3. "clfmetric3": clf\_3Metrics,
4. "rs ":rs\_eval,
5. "gs":gs\_eval})
6. compare\_matrics.plot.bar()

* **Two ways to save and load machine learning models:**

1. With Python's pickle module:
   1. import pickle
   3. #save an extisting model to file
   4. pickle.dump(gs\_clf,open("gs\_random\_forest\_model.pkl","wb")jkl
   5. #Load a saved model
   6. loaded\_pickle\_model = pickle.load(open("gs\_random\_forest\_model.pkl","rb"))
   7. #Make some predictions
   8. pickle\_Ypreds = loaded\_pickle\_model.predict(X\_test)
   9. evaluate\_preds(Y\_test,pickle\_Ypreds)
2. With the joblib module:
   1. import joblib
   2. ##if model is large joblib can be more efficient to use.
   3. #saving model .
   4. joblib.dump(gs\_clf,filename="joblib\_random\_forest\_model.joblib")
   5. #load a saved model
   6. loaded\_joblib\_model = joblib.load(filename="joblib\_random\_forest\_model.joblib")
   7. #mkae and evaluate joblib predictions
   8. joblib\_Ypreds = loaded\_joblib\_model.predict(X\_test)
   9. evaluate\_preds(Y\_test, joblib\_Ypreds)

* **Putting It All Together with Pipeline:**

# Getting data ready

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder

from sklearn.pipeline import Pipeline

#Modelling

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import train\_test\_split, GridSearchCV

#Setting up random seed

np.random.seed(99)

#Import data and drop rows with mising labels

data= pd.read\_csv("car-sales-extended-missing-data.csv")

data.dropna(subset=["Price"],inplace=True)

#Import data and drop rows with mising labels

data= pd.read\_csv("car-sales-extended-missing-data.csv")

data.dropna(subset=["Price"],inplace=True)

# Define different features and transformer pipeline

categorical\_feastures = ["Make","Colour"]

categorical\_transformer = Pipeline(steps=[

("imputer",SimpleImputer(strategy="constant",fill\_value="missing")),

("onehot",OneHotEncoder(handle\_unknown="ignore"))

] )

door\_features = ["Doors"]

door\_tranformer = Pipeline(steps = [

("imputer",SimpleImputer(strategy="constant",fill\_value = 4))

])

numeric\_features = ["Odometer (KM)"]

numeric\_transformer = Pipeline(steps = [

("imputer",SimpleImputer(strategy="mean"))

])

#Setup prepocessing steps (fill ,missing values,then convert to numbers)

preprocessing = ColumnTransformer(

transformers=[

("cat",categorical\_transformer,categorical\_feastures),

("door",door\_tranformer,door\_features),

("num",numeric\_transformer,numeric\_features)

]

)

#Creating a preprocessing and modelling pipeline

model= Pipeline(steps=[("preprocessor",preprocessing),

("model",RandomForestRegressor())])

#Spilt the data

X = data.drop("Price",axis =1)

Y= data["Price"]

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2)

#Fit model

model.fit(X\_train,Y\_train)

model.score(X\_test,Y\_test)

# We can use GridSearchCV and also RandomisedSearchCV with Pipeline

# Using GridSearchCV with our regression Pipeline

pipe\_grid = {

"preprocessor\_\_num\_\_imputer\_\_strategy": ["mean","median"], # '\_\_' trace backs to name in pipeline

"model\_\_n\_estimators": [100,1000],

"model\_\_max\_depth": [None ,5],

"model\_\_max\_features": ["sqrt"],

"model\_\_min\_samples\_split": [2,4]

}

gs\_model = GridSearchCV(model,pipe\_grid,cv=5,verbose =2)

gs\_model.fit(X\_train,Y\_train)

gs\_model.score(X\_test,Y\_test)

* **DEEP LEARNING, NEURAL NETWORKS:**

We are going to use **TensorFlow** for this.

**Unstructured Data** is used for this.

**TensorFlow:**

**TensorFlow is Deep learning or numerical computing library.**

Why TensorFlow:

* Writing fast deep learning code in Python (able to run on a GPU).
* Able to access many pre-built deep learning models.
* Whole stack: preprocess ,models, deploy can be done.
* Originally designed and used inhouse by Google(now open-source).

What is GPU ?

GPU is Graphical Processing Unit .Its lot faster than CPU to do numerical computing. It increases the speed of finding patterns.

**Neural Networks:**

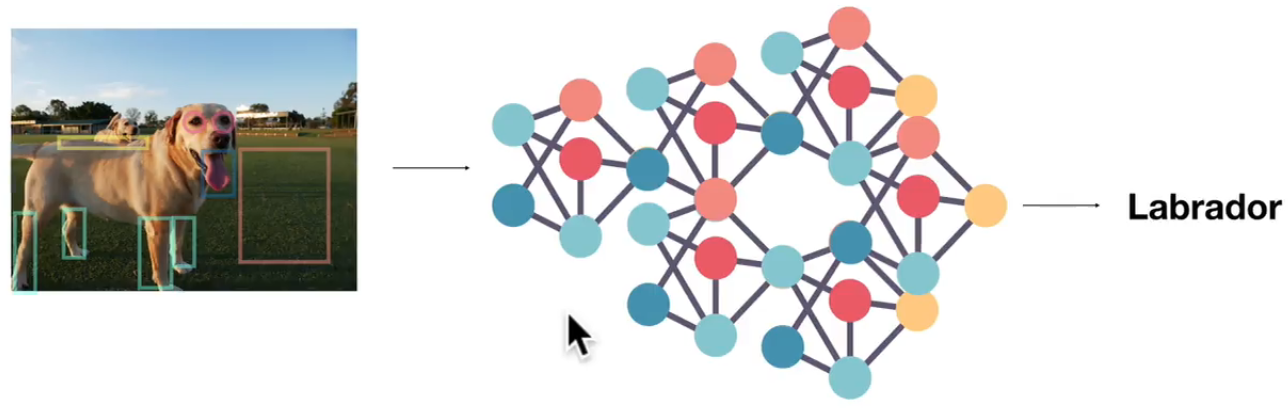
Algorithms of deep learning. A neural network is a method in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain.



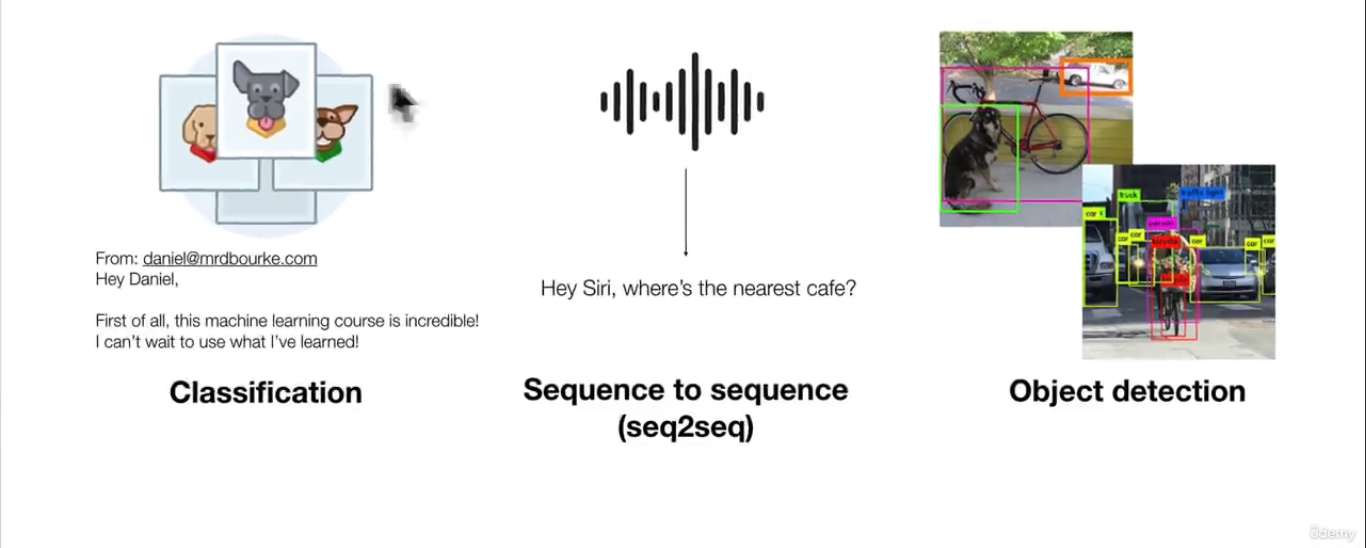
**Deep Learning:**

It is another form to machine learning.  Uses interconnected nodes or neurons in a layered structure that resembles the human brain.

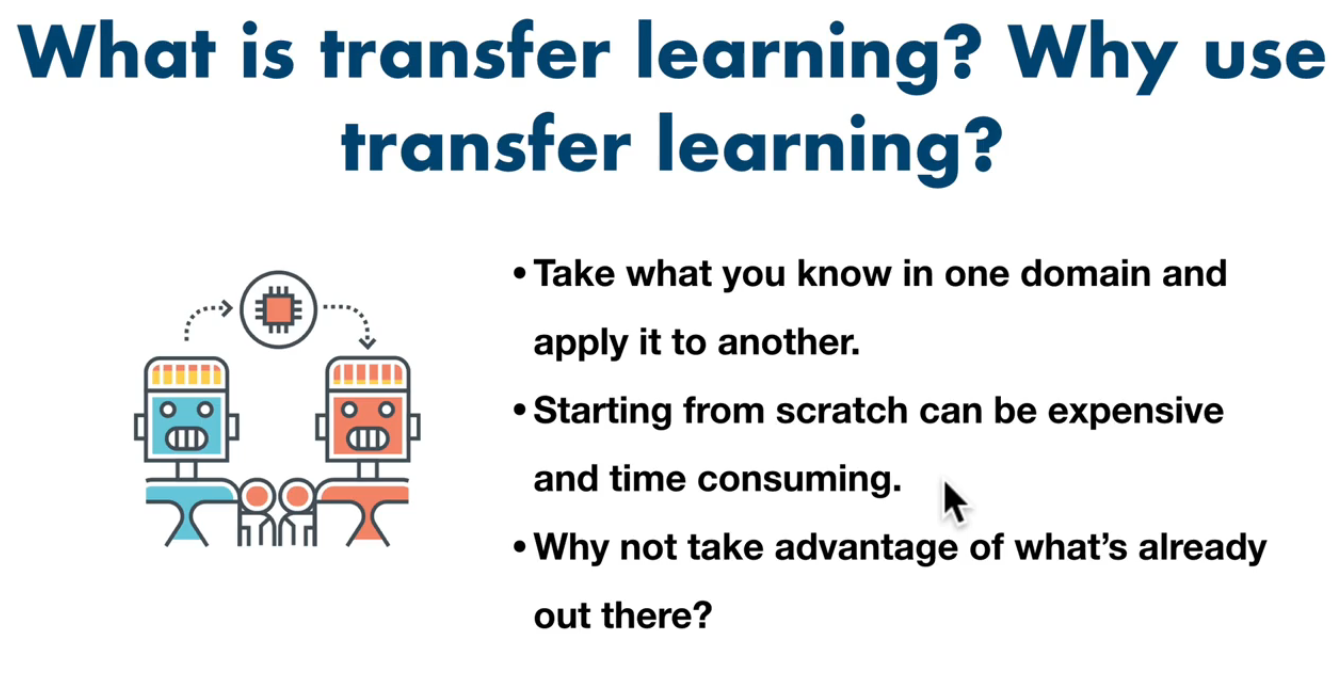
Nodes: Small Models.



**Kinds of Deep Learning Problems:**

****

**Transfer Learning:**

****

**Workflow: A diagram of a workflow

Description automatically generated**

#### **Getting our Google colab workspace ready:**

1. # Importing TensorFlow and Tensorflow\_hub
2. import tensorflow as tf
3. import tensorflow\_hub as hub
4. print("TensorFlow Version: ",tf.\_\_version\_\_)
5. print("tensorflow Hub version: "+hub.\_\_version\_\_)
7. # making sure we're using GPU
8. print("GPU"+"available(yess)" if tf.config.list\_physical\_devices("GPU") else "not available")
10. if GPU not available , go to runtime and change runtimr to GPU.

#### **Viewing Images**

1. #Let's view an image
2. from IPython.display import Image
3. Image("drive/My Drive/Dog\_Breed\_Vision/train/001513dfcb2ffafc82cccf4d8bbaba97.jpg")
4. ### Getting images and their Labels
5. #Let's get a list of all images file pathnames
7. filenames = ["drive/My Drive/Dog\_Breed\_Vision/train/"+ fname for fname in labels\_csv["id"]+".jpg"]
8. filenames[:10]
9. ## Check whether the amount of filename matches the files present in train data.
10. import os
11. if  len(os.listdir("drive/My Drive/Dog\_Breed\_Vision/train/")) == len(filenames):
12. print("amount matched . PROCEED!!")
13. else:
14. print("Amount not matched.")
15. Image(filenames[9999])
16. labels\_csv["breed"][9999]

* **turning labels into numbers:**

1. ##labels into nummbers
3. labels = labels\_csv['breed'].to\_numpy()
5. ## Check if amount of labels matches the amount of filenames
6. if len(labels)== len(filenames):
7. print("Amount matcheed.PROCEED!")
8. else:
9. print("Amount mismatched.")
11. import numpy as np
12. # Unuie labels
13. unique\_breeds= np.unique(labels)
14. len(unique\_breeds)
15. boolean\_labels = [label == unique\_breeds for label in labels]
16. boolean\_labels[:2]
17. print(boolean\_labels[0].astype(int)) # 1 where label is occurs ,else 0.

#### **Creating our validation set:**

1. #Set up X and Y
2. X = filenames
3. Y = boolean\_labels

A slider parameter:

1. # Set number of images to use Experimenting with slider
2. NUM\_IMAGES = 1000 #@param{type:"slider",min:1000, max:10222}
3. ##Spilting train dataset into train and validation set
4. from sklearn.model\_selection import train\_test\_split
5. Xtrain,Xval,Ytrain,Yval = train\_test\_split(X[:NUM\_IMAGES],
6. Y[:NUM\_IMAGES],
7. test\_size=0.2)
8. print(len(Xtrain),len(Xval),len(Ytrain),len(Yval))

#### **Preprocessing data(Turning images into tensors)**

1. IMG\_SIZE = 224
3. # Create a function for preprocessing images
4. def process\_images(image\_path , img\_size=IMG\_SIZE):
5. """
6. Takes an image file path and turns the image into tensors.
7. """
8. # read in a image filepath
9. image = tf.io.read\_file(image\_path) # return a tensor dtype string.
11. # turns the jpeg image into numerical Tensor with # color channels(Red,Green,Blue)
12. image = tf.image.decode\_jpeg(image,channels=3) # takes argument as dtype string.
14. # convert the colour channels values from 0-255 to 0-1 values(normalization)
15. image = tf.image.convert\_image\_dtype(image, tf.float32)# can convert into types like uint32, uint64, int8, int16 ,float64, float32.
17. # resize the image to our desired value(224,224)
18. image = tf.image.resize(image, size=[img\_size,img\_size])
20. return image

#### **Turning our data into batches**

Why : Let's say you're trying to process 10000+ images in one go... they all might not fit into memeory. Tensorflow like data to be into batches .

So that's why we do about 32(this is the batch size) images at a time (you can adjust the batcfh size if needed.)

In order to use TensorFlow effectively, we need our data to be in the form of Tensor tuples which look like this: (image, label) .

1. # Create a simple function to return a tuple(image,label)
3. def get\_image\_label\_tuple(image\_path, label):
4. """
5. takes an image file path name and th assosicated label, processses the image and returns a tuple of(image,label).
6. """
7. image = process\_images(image\_path)
8. return image , tf.constant(label)

Now we've got a way to turn oour data into tuple of Tensors in the form: (image,label) , let's make a function to turn all of our data (x and y) into batches!

1. # Define the batch size,32 is good.
2. BATCH\_SIZE = 32
4. #Create a function to turn data into batches
5. def create\_data\_batches(X,Y=None , batch\_size=BATCH\_SIZE, valid\_data=False ,test\_data=False):
6. """
7. Creates batches of data out of image(X) and label(Y) pairs.
8. shuffles the data if it's training data but doesn't shuffle if it's validation data.
9. Also accepts test data as input(no labels).
10. """
11. # If the data is a test dataset, we probably don't have labels
12. if test\_data:
13. print("Creating test data batches..")
14. data = tf.data.Dataset.from\_tensor\_slices((tf.constant(X))) #only filepaths(no labels)
15. data\_batch = data.map(process\_images).batch(BATCH\_SIZE)
16. return data\_batch
17. #If the data s a valid dataset, we don't need to shuffle it
18. elif valid\_data:
19. print("Creating validation data batches...")
20. data = tf.data.Dataset.from\_tensor\_slices((tf.constant(X),tf.constant(Y)))
21. data\_batch = data.map(get\_image\_label\_tuple).batch(BATCH\_SIZE)
22. else :
23. print("Creating training data batches...")
24. # turn filepaths and labels into Tensors
25. data = tf.data.Dataset.from\_tensor\_slices((tf.constant(X),tf.constant(Y)))
26. #shuffling pathnames and labels before mapping image processor function is faster than shuffing processed images
27. data = data.shuffle(buffer\_size=len(X))
29. #Create (image,label) tuples(image to preprocessed image)
30. data = data.map(get\_image\_label\_tuple)
32. # turn the training data into batches
33. data\_batch = data.batch(BATCH\_SIZE)
34. return data\_batch

Visualizing Data batches

Our data is now in batches, however, these can be a little hard to understand/comprehend, let's visualize them!

1. import matplotlib.pyplot as plt
2. # create a function for viewing images in a data batch
4. def show\_25\_images(images,labels):
5. """
6. Displays a plot of 2 images and their labels from a data batch.
7. """
8. # setup the figure
9. plt.figure(figsize=(10,10))
10. #loop through 25 (for displaying 2 images)
11. for i in range(25):
12. #create subplots(5rows, 5columns)
13. ax = plt.subplot(5,5,i+1)
14. #Display the image
15. plt.imshow(images[i])
16. #add the images label as the title
17. plt.title(unique\_breeds[labels[i].argmax()])
18. #Turn the grid line off
19. plt.axis("off")
20. train\_images ,train\_labels = next(train\_data.as\_numpy\_iterator()) # unmaps the traindata as numpy arrays
21. show\_25\_images(train\_images, train\_labels)

* **Deep Learning Hubs:**

1. TensorFlow

2.Pytorch

3. Model Zoo

4. Papers with code

* **Building a model**

Before we build a model, there are a few things we need to define:

* The input shape(our images shape, in the form of tensors) to our model.
* The output shape(images labels,in the form of tensors) of our model.
* The URL of the model we want to use.

1. # Setup input shape to the model
2. INPUT\_SHAPE = [None, IMG\_SIZE, IMG\_SIZE,3] #batch , height, width,colour channels
4. # Setup output shape of our model
5. OUTPUT\_SHAPE = len(unique\_breeds)
7. #  Setup Model URL from Tensorflow hub
8. MODEL\_URL = "https://tfhub.dev/google/imagenet/mobilenet\_v2\_130\_224/classification/4"
9. # Create a function which builds a keras model.
10. def create\_model(input\_shape=INPUT\_SHAPE,output\_shape=OUTPUT\_SHAPE,model\_url = MODEL\_URL):
11. print("Building model with:", model\_url)
12. # Setup the model layers
13. model = tf.keras.Sequential([hub.KerasLayer(model\_url),# Layer 1 (input layer) # calls the ur from hub . Mobile net which finds pattern in every pixel and gives numbers.
14. tf.keras.layers.Dense(units=OUTPUT\_SHAPE,activation="softmax") # Layer 2 (output layer)  # it tells that we want pattern of size of OUTPUT i.e 120 here. And activation normalizes it in 0 to 1 numbers. # softmax is for multiclass classification and sigmoid is for binary classification.
15. ])
17. # Compile the model
18. model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),  # measures difference between models predictions and true values.
19. optimizer=tf.keras.optimizers.Adam(), # optimization enhance efficiency , Adam is popular choice.
20. metrics=["accuracy"])
22. model.build(input\_shape)
23. return model

* **Callbacks**

Callbacks are helper functions a model can use during training to do things like : save its progress,check its progress or stop training early if a model stops improving.

we'll create two callbacks,one for TensorBoard which track our models progress and another for early stopping model from training for too long.

TensorBoard Callback:

1. #Load TensorBoard notebook extension
2. %load\_ext tensorboard
3. import datetime
5. # create a function to build a TensorBoard callback
6. def create\_tensorboard\_callback():
7. # Create a log directory for storing TensorBoard logs
8. logdir = os.path.join("drive/MyDrive/Deep\_Learning/logs",# Make it so the logs get tracked whenever we run an experiment
9. datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
10. return tf.keras.callbacks.TensorBoard(logdir)

* **Training a model (On a subset)**

Our First model is only going to train on 1000 images,to make sure everything is working.

1. NUM\_EPOCHS = 100 #@param {type:"slider",min:10,max:100,step:10 }
2. # Build a function to train and return a trained model
3. def train\_model():
4. """
5. Trains a given model and returns the trained version.
6. """
7. # Create a model
8. model = create\_model()
10. #Create new tensorboard session everytime we train a model
11. tensorboard = create\_tensorboard\_callback()
13. #Fit the momdel to the data passing it the callbacks we created
14. model.fit(x=train\_data,
15. epochs=NUM\_EPOCHS,
16. validation\_data = val\_data,
17. validation\_freq=1,
18. callbacks=[tensorboard,early\_stopping])
20. #Return model
21. return model
22. # Fit the model to the data
23. model = train\_model()

* **Checking the TensorBoard Logs:**  
  %tensorboard --logdir drive/MyDrive/Deep\_Learning/logs
* **Making and evaluating prediction on trained model.**

1. predictions = model.predict(val\_data,verbose=1)
2. predictions.shape #200 validation images with 120 labels.
3. predictions[0] # It shows how much precent(from 0 to 1, thanks to softmax, also by softmax all sums near to 1 for each image) the image belongs to the labels.
4. print("Predicted label: ",unique\_breeds[np.argmax(predictions[index])])
5. # Turning this into function.
7. def get\_pred\_label(predictions):
8. """
9. Finding maximum predicted label for a prediction.Turns an array of prediction probabilities into a label.
10. """
11. return unique\_breeds[np.argmax(predictions)]
13. p = get\_pred\_label(predictions[99])
14. p

* **A function of unbatch validation data.**

1. def unbatchify(dataset):
2. """
3. Takes a batched dataset of (image,label)Tensors and returns the image and label seperately
4. """
5. images = []
6. labels = []
7. for i,l in dataset.unbatch().as\_numpy\_iterator():
8. images.append(i)
9. labels.append(l)
10. return images,labels

13. val\_images , val\_labels = unbatchify(val\_data)

* **Visualizing model's predictions:**  
  Creating a function to plot the image of validation image , tell the predicted label and probablity of it and also tell truth label.

1. def plot\_pred(images,labels,prediction\_probabilities,n):
2. """
3. View the image , it's true label and predicted label.
4. """
5. #get pred label
6. pred\_label = get\_pred\_label(prediction\_probabilities[n])
8. #get truth label
9. true\_label = get\_pred\_label(labels[n])
11. #change color on right or wrong
12. if pred\_label == true\_label:
13. color = "green"
14. else:
15. color = "red"
17. # plot image & remove ticks
18. image = images[n]
19. plt.imshow(image)
20. plt.xticks([])
21. plt.yticks([])
23. plt.title("{} {:2.0f}% {}".format(pred\_label,np.max(prediction\_probabilities[n])\*100,true\_label),color=color)
25. plot\_pred(val\_images,val\_labels,predictions,n=100)

**Plot the top 10 prediction probability values and labels, coloring the true label green.**

1. def plot\_pred\_conf(prediction\_probabilities, labels, n):
2. """
3. Plots the top 10 highest prediction confidences along with the truth label for a sample n.
4. """
5. pred\_prob= prediction\_probabilities[n]
6. pred\_label = get\_pred\_label(prediction\_probabilities[n])
7. true\_label = get\_pred\_label(labels[n])
9. top\_10\_pred\_indexes = pred\_prob.argsort()[-10:][::-1]
10. top\_10\_pred\_labels = unique\_breeds[top\_10\_pred\_indexes]
11. top\_10\_pred\_values = pred\_prob[top\_10\_pred\_indexes]
13. # Setup plot
14. top\_plot=plt.bar(np.arange(len(top\_10\_pred\_labels)),top\_10\_pred\_values,color="grey")
16. plt.xticks(np.arange(len(top\_10\_pred\_labels)),labels=top\_10\_pred\_labels,rotation="vertical")
18. if np.isin(true\_label,top\_10\_pred\_labels):
19. top\_plot[np.argmax(top\_10\_pred\_labels == true\_label)].set\_color("green")
20. else:
21. pass
22. plot\_pred\_conf(predictions,val\_labels,n=10)

**Let's visualize aour predictions and evaluate our model. Let's check few predictions**

1. # let's vosualize the evaluation
3. i\_multiplier = 20
4. num\_rows = 10
5. num\_columns = 2
6. num\_images = num\_rows \* num\_columns
7. plt.figure(figsize=(10\*num\_columns,5\*num\_rows))
9. for i in range(num\_images):
10. plt.subplot(num\_rows,2\*num\_columns,2\*i+1)
11. plot\_pred(prediction\_probabilities=predictions,
12. labels = val\_labels,
13. images = val\_images,
14. n = i+i\_multiplier)
15. plt.subplot(num\_rows,2\*num\_columns,2\*i+2)
16. plot\_pred\_conf(prediction\_probabilities=predictions,
17. labels = val\_labels,
18. n = i+i\_multiplier)
20. plt.tight\_layout(h\_pad=1.0)
21. plt.show()

* **Saving and Loading Model:**

**Saving a model:**

1. #Create a function to save a model.
2. def save\_model(model,suffix=None):
3. """
4. Saves a given model in a models directory and appends a suffix(string) .
5. """
6. #Create a model directory pathname with current time
7. modeldir = os.path.join("drive/MyDrive/Deep\_Learning/models",datetime.datetime.now().strftime("%d%m%Y-%H%M%S"))
8. model\_path = modeldir + "-" + suffix + ".h5" # save format of model
9. print(f"Saving model to:{model\_path}...")
10. model.save(model\_path)
11. return model\_path
13. save\_model(model,suffix="1000-images-mobilenetv2-Adam")

**Loading a model:**

1. #Create a function to lad a model.
2. def load\_a\_model(model\_path):
3. """Loads a saved model from a specified path."""
4. print(f"Loading saved model from:{model\_path}...")
5. Model=tf.keras.models.load\_model(model\_path,custom\_objects={"KerasLayer":hub.KerasLayer})
6. return model
7. #loading model
8. loadedmodel = load\_a\_model('path/15092024-073300-1000-images-mobilenetv2-Adam.h5')

#### **Training Model on Full Data:**

1. ## creating Batches for full data
2. full\_data = create\_data\_batches(X,Y)
4. # Create a model for full model
5. full\_model = create\_model()
7. # creating full model callbacks
8. full\_model\_tensorboard = create\_tensorboard\_callback()
10. #There's No validation set wen traing an all the data, so we can't monitor validation accuracy
11. full\_model\_early\_stopping = tf.keras.callbacks.EarlyStopping(monitor="accuracy",patience=3)
13. # Fit the model to full data
14. full\_model.fit(x = full\_data,
15. epochs = NUM\_EPOCHS,
16. callbacks= [full\_model\_tensorboard,full\_model\_early\_stopping])
18. save\_model(full\_model,suffix="full-data-MobileNetV2\_adam\_model")

21. loaded\_full\_model = load\_a\_model('drive/MyDrive/Deep\_Learning/models/17092024-173324-full-data-MobileNetV2\_adam\_model.h5')
22. loaded\_full\_model.summary()

* **Making predictions on the test dataset**

Since our model has been trained on images in the form of Tensor batches, to make predictions on the test data, we'll have to get it into the same format.

To make predictions on the test data, we'll:

* Get the test image filenames
* Convert the filenames into test data batches using create\_data\_batches and setting the test\_data parameter to True (since the test data doesn't have labels).
* Make a predictions array by passing the test batches to the predict() method calleed on our model.

1. #load test image filenames
2. test\_path = "drive/MyDrive/Dog\_Breed\_Vision/test/"
3. test\_filenames = [test\_path + fname for fname in os.listdir(test\_path)]

6. # Creatin test\_data batches
7. test\_data = create\_data\_batches(test\_filenames,test\_data=True)
9. #Make predictions on test data batch using the loaded model
10. test\_prediction = loaded\_full\_model.predict(test\_data,verbose=1)

* **Saving and Loading predictions:**

1. # save predictions NumPy array to csv file
2. np.savetxt("/drive/MyDrive/Dog\_Breed\_Vision/preds\_array.csv",test\_predictions,delimiter=",")
4. # Load predictions Numpy array from csv file
5. test\_predictions = np.loadtxt("/content/drive/MyDrive/Dog\_Breed\_Vision/preds\_array.csv",delimiter=",")

* **Preparing test dataset predictions for kaggle**

For competition : https://www.kaggle.com/competitions/dog-breed-identification Data should be :

* Create a pandas Dataframe with an ID column as well as a column for each dog breed.
* Add data to the ID column by extracting the test image ID's from their filepaths.
* Add data (the predictions probabilities) to each of the dog breed columns.
* Export the DataFrame as a csv to submit it to kaggle.

1. # Create a pandas DaaFrame with empty columns
2. preds\_df = pd.DataFrame(columns=["id"]+list(unique\_breeds))
3. test\_path = "drive/MyDrive/Dog\_Breed\_Vision/test/"
5. # Append test image ID's to predictions DataFrame
6. test\_ids = [os.path.splitext(path)[0] for path in os.listdir(test\_path)]
7. preds\_df["id"] = test\_ids
9. # Add prediction probabilities to columns.
10. preds\_df[list(unique\_breeds)] = test\_predictions
12. # Save our predictions dataaframe to CSV for submission on kaggle
13. preds\_df.to\_csv("drive/MyDrive/Dog\_Breed\_Vision/full\_model\_mobilenetv2\_predictions.csv",index=False)

* **Making predictions on customs images:**

1. # Path to directory
2. custom\_path = "drive/MyDrive/Dog\_Breed\_Vision/custom\_dog-photos/"
3. custom\_img\_path =[ custom\_path + img\_name for img\_name in os.listdir(custom\_path)]
5. # Turn custom images into Data batches
6. custom\_data = create\_data\_batches(custom\_img\_path,test\_data=True)
8. # Making predictions on Custom Data
9. custom\_preds = loaded\_full\_model.predict(custom\_data)
11. # Get the labels of custom predictions
12. custom\_pred\_labels = [get\_pred\_label(custom\_preds[i]) for i in range(len(custom\_preds))]
14. # Get custom images (Here `unbatchify` won't work as custom data don't have labels ).
15. custom\_images = []
16. # Loop through unbatched data
17. for img in custom\_data.unbatch().as\_numpy\_iterator():
18. custom\_images.append(img)
20. # Check Custom image predictions through plot
21. plt.figure(figsize=(10,10))
22. for i,img in enumerate(custom\_images):
23. plt.subplot(1,len(custom\_images), i+1)
24. plt.xticks([])
25. plt.yticks([])
26. plt.title(custom\_pred\_labels[i])
27. plt.imshow(img)

**THANK YOU!**