Import

In [17]:

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn import linear_model
from sklearn.model_selection import cross_val_score, KFold, GridSearchCV, RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
from sklearn.pipeline import Pipeline
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from keras.optimizers import Adam
import warnings
warnings.filterwarnings('ignore')
```

Fuel Consupmtion Dataset - Regression

Data load and clean

In [18]:

```
fuel = pd.read_csv('https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/FuelConsumptionCo2.csv')
```

In [19]:

```
fuel.head()
```

Out[19]:

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISS
0	2014	ACURA	ILX	COMPACT	2.0	4	
1	2014	ACURA	ILX	COMPACT	2.4	4	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	

In [20]:

```
print(fuel.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1067 entries, 0 to 1066
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   MODELYEAR                 1067 non-null   int64
 1   MAKE                      1067 non-null   object
 2   MODEL                     1067 non-null   object
 3   VEHICLECLASS              1067 non-null   object
 4   ENGINESIZE                1067 non-null   float64
 5   CYLINDERS                 1067 non-null   int64
 6   TRANSMISSION              1067 non-null   object
 7   FUELTYPE                  1067 non-null   object
 8   FUELCONSUMPTION_CITY      1067 non-null   float64
 9   FUELCONSUMPTION_HWY       1067 non-null   float64
 10  FUELCONSUMPTION_COMB      1067 non-null   float64
 11  FUELCONSUMPTION_COMB_MPG  1067 non-null   int64
 12  CO2EMISSIONS              1067 non-null   int64
dtypes: float64(4), int64(4), object(5)
memory usage: 108.5+ KB
None
```

In [21]:

```
#fuel['CYLINDERS'] = fuel['CYLINDERS'].astype('object')
print(fuel.MODELYEAR.unique())
fuel.drop('MODELYEAR', axis = 1, inplace = True)
fuel.drop('MODEL', axis = 1, inplace = True)
fuel.drop('MAKE', axis = 1, inplace = True)
fuel.head()
```

```
[2014]
```

Out[21]:

	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMF
0	COMPACT	2.0	4	AS5	Z	
1	COMPACT	2.4	4	M6	Z	
2	COMPACT	1.5	4	AV7	Z	
3	SUV - SMALL	3.5	6	AS6	Z	
4	SUV - SMALL	3.5	6	AS6	Z	

In [22]:

```
fuel = pd.get_dummies(fuel)
fuel.head()

# OR

# for i in fuel.describe(include = np.object).columns:
#     fuel[i] = fuel[i].astype('category')
#     fuel[i] = fuel[i].cat.codes
#     fuel[i] = fuel[i].astype('category')
```

Out[22]:

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELC
0	2.0	4	9.9	6.7	
1	2.4	4	11.2	7.7	
2	1.5	4	6.0	5.8	
3	3.5	6	12.7	9.1	
4	3.5	6	12.1	8.7	

5 rows × 49 columns

In [23]:

```
fuel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1067 entries, 0 to 1066
Data columns (total 49 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   ENGINESIZE                            1067 non-null   float6
4
 1   CYLINDERS                             1067 non-null   int64  
 2   FUELCONSUMPTION_CITY                  1067 non-null   float6
4
 3   FUELCONSUMPTION_HWY                   1067 non-null   float6
4
 4   FUELCONSUMPTION_COMB                  1067 non-null   float6
4
 5   FUELCONSUMPTION_COMB_MPG              1067 non-null   int64  
 6   CO2EMISSIONS                          1067 non-null   int64  
 7   VEHICLECLASS_COMPACT                  1067 non-null   uint8  
 8   VEHICLECLASS_FULL-SIZE                1067 non-null   uint8  
 9   VEHICLECLASS_MID-SIZE                 1067 non-null   uint8  
 10  VEHICLECLASS_MINICOMPACT              1067 non-null   uint8  
 11  VEHICLECLASS_MINIVAN                  1067 non-null   uint8  
 12  VEHICLECLASS_PICKUP TRUCK - SMALL     1067 non-null   uint8  
 13  VEHICLECLASS_PICKUP TRUCK - STANDARD  1067 non-null   uint8  
 14  VEHICLECLASS_SPECIAL PURPOSE VEHICLE  1067 non-null   uint8  
 15  VEHICLECLASS_STATION WAGON - MID-SIZE 1067 non-null   uint8  
 16  VEHICLECLASS_STATION WAGON - SMALL    1067 non-null   uint8  
 17  VEHICLECLASS_SUBCOMPACT               1067 non-null   uint8  
 18  VEHICLECLASS_SUV - SMALL              1067 non-null   uint8  
 19  VEHICLECLASS_SUV - STANDARD           1067 non-null   uint8  
 20  VEHICLECLASS_TWO-SEATER               1067 non-null   uint8  
 21  VEHICLECLASS_VAN - CARGO              1067 non-null   uint8  
 22  VEHICLECLASS_VAN - PASSENGER          1067 non-null   uint8  
 23  TRANSMISSION_A4                       1067 non-null   uint8  
 24  TRANSMISSION_A5                       1067 non-null   uint8  
 25  TRANSMISSION_A6                       1067 non-null   uint8  
 26  TRANSMISSION_A7                       1067 non-null   uint8  
 27  TRANSMISSION_A8                       1067 non-null   uint8  
 28  TRANSMISSION_A9                       1067 non-null   uint8  
 29  TRANSMISSION_AM5                      1067 non-null   uint8  
 30  TRANSMISSION_AM6                      1067 non-null   uint8  
 31  TRANSMISSION_AM7                      1067 non-null   uint8  
 32  TRANSMISSION_AS4                      1067 non-null   uint8  
 33  TRANSMISSION_AS5                      1067 non-null   uint8  
 34  TRANSMISSION_AS6                      1067 non-null   uint8  
 35  TRANSMISSION_AS7                      1067 non-null   uint8  
 36  TRANSMISSION_AS8                      1067 non-null   uint8  
 37  TRANSMISSION_AS9                      1067 non-null   uint8  
 38  TRANSMISSION_AV                       1067 non-null   uint8  
 39  TRANSMISSION_AV6                      1067 non-null   uint8  
 40  TRANSMISSION_AV7                       1067 non-null   uint8
 41  TRANSMISSION_AV8                       1067 non-null   uint8
 42  TRANSMISSION_M5                        1067 non-null   uint8
 43  TRANSMISSION_M6                        1067 non-null   uint8
 44  TRANSMISSION_M7                        1067 non-null   uint8
 45  FUELTYPE_D                             1067 non-null   uint8
 46  FUELTYPE_E                             1067 non-null   uint8
 47  FUELTYPE_X                             1067 non-null   uint8
 48  FUELTYPE_Z                             1067 non-null   uint8
dtypes: float64(4), int64(3), uint8(42)
memory usage: 102.2 KB
```

In [24]:

```
fuel.head()
```

Out[24]:

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELC
0	2.0	4	9.9	6.7	
1	2.4	4	11.2	7.7	
2	1.5	4	6.0	5.8	
3	3.5	6	12.7	9.1	
4	3.5	6	12.1	8.7	

5 rows × 49 columns

Regression

In [25]:

```
x = fuel.drop('CO2EMISSIONS', axis = 1)
y = fuel['CO2EMISSIONS']

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.6)
```

In [26]:

```
linear = linear_model.LinearRegression()
ridge = linear_model.Ridge()
lasso= linear_model.Lasso()
elastic = linear_model.ElasticNet()
lasso_lars = linear_model.LassoLars()
bayes_ridge = linear_model.BayesianRidge()
logistics = linear_model.LogisticRegression()
sgd = linear_model.SGDClassifier()
passagg = linear_model.PassiveAggressiveClassifier()
ridgecv = linear_model.RidgeClassifierCV()
ridgeclass = linear_model.RidgeClassifier()

models_churn = [linear, ridge, lasso, elastic, lasso_lars, bayes_ridge, logist
ics, sgd, passagg, ridgecv, ridgeclass]
```

In [27]:

```
def get_cv_scores(model):
    scores = cross_val_score(model, x_train, y_train, cv=5, scoring='neg_root_
mean_squared_error')
    print('CV Mean: ', np.mean(scores))
    print('STD: ', np.std(scores))
    print('\n')
```

In [28]:

```
for i in models_churn:
    print(i)
    get_cv_scores(i)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, nor
malize=False)
CV Mean:  -5.568129523825883
STD:  0.736552577166622



Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001
)
CV Mean:  -5.609066774025289
STD:  0.8041810228548972



Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_st
ate=None,
      selection='cyclic', tol=0.0001, warm_start=False)
CV Mean:  -8.91487029464611
STD:  1.5701638550563086
```

```
ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.
5,
           max_iter=1000, normalize=False, positive=False, precomp
ute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm
_start=False)
CV Mean:  -20.372459824583984
STD:  2.761703817318036


LassoLars(alpha=1.0, copy_X=True, eps=2.220446049250313e-16, fit_i
ntercept=True,
          fit_path=True, max_iter=500, normalize=True, positive=Fa
lse,
          precompute='auto', verbose=False)
CV Mean:  -32.217937653789605
STD:  3.7430626640724554


BayesianRidge(alpha_1=1e-06, alpha_2=1e-06, alpha_init=None,
              compute_score=False, copy_X=True, fit_intercept=True
,
              lambda_1=1e-06, lambda_2=1e-06, lambda_init=None, n_
iter=300,
              normalize=False, tol=0.001, verbose=False)
CV Mean:  -5.540264290478805
STD:  0.7369043882596081


LogisticRegression(C=1.0, class_weight=None, dual=False, fit_inter
cept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10
0,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                   warm_start=False)
CV Mean:  -20.31535769981978
STD:  0.8534805377822611


SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_int
ercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge'
,
              max_iter=1000, n_iter_no_change=5, n_jobs=None, pena
lty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.
001,
              validation_fraction=0.1, verbose=0, warm_start=False
)
CV Mean:  -56.025583409900534
```

```
STD:  3.6584154466486605


PassiveAggressiveClassifier(C=1.0, average=False, class_weight=Non
e,
                            early_stopping=False, fit_intercept=Tr
ue,
                            loss='hinge', max_iter=1000, n_iter_no
_change=5,
                            n_jobs=None, random_state=None, shuffl
e=True,
                            tol=0.001, validation_fraction=0.1, ve
rbose=0,
                            warm_start=False)
CV Mean:  -45.81636097300676
STD:  6.679853455813455


RidgeClassifierCV(alphas=array([ 0.1,  1. , 10. ]), class_weight=N
one, cv=None,
                  fit_intercept=True, normalize=False, scoring=Non
e,
                  store_cv_values=False)
CV Mean:  -40.06360226773207
STD:  2.5586785325580603


RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_int
ercept=True,
                max_iter=None, normalize=False, random_state=None,
                solver='auto', tol=0.001)
CV Mean:  -39.33511788091995
STD:  4.115527158716331
```

In [29]:

```
alpha = [0.5, 1, 1.5, 0.01, 2.5, 0.0001, 10, 100, 0.35]
#solver = ['auto', 'svd']

param_grid = dict(alpha = alpha)#, solver = solver)
```

In [30]:

```
grids = GridSearchCV(estimator = lasso_lars, param_grid = param_grid, scoring
= 'r2', cv = 10)
grid_result = grids.fit(x_train, y_train)
```

In [31]:

```
rnds = RandomizedSearchCV(estimator = ridge, param_distributions = param_grid,
scoring='r2', cv = 10)
rnds_result = rnds.fit(x_train, y_train)
```

In [32]:

```
print(grid_result.best_params_)
print(rnds_result.best_params_)
```

```
{'alpha': 0.0001}
{'alpha': 0.35}
```

Best Model

In [34]:

```
print(grid_result.best_score_)
print(rnds_result.best_score_)
```

```
0.9916810501375295
0.9917607214483969
```

Iris Dataset - Classification

In [35]:

```
from sklearn.datasets import load_iris
iris = load_iris()
data1 = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['target'])
data1
```

Out[35]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2.0
146	6.3	2.5	5.0	1.9	2.0
147	6.5	3.0	5.2	2.0	2.0
148	6.2	3.4	5.4	2.3	2.0
149	5.9	3.0	5.1	1.8	2.0

150 rows × 5 columns

In [36]:

```
x_iris = data1.drop('target', axis = 1)
y_iris = data1['target']
x1_train, x1_test, y1_train, y1_test = train_test_split(x_iris, y_iris, train_
size = 0.7)
print(x1_train.shape)
print(x1_test.shape)
print(y1_train.shape)
print(y1_test.shape)
```

```
(105, 4)
(45, 4)
(105,)
(45,)
```

Model Building

In [37]:

```
def get_cv_scores1(model):
    scores = cross_val_score(model, x1_train, y1_train, cv=5, scoring='accurac
y')
    print('CV Mean Score: ', np.mean(scores))
    print('STD: ', np.std(scores))
    print('\n')

models_log = [logistics, sgd, passagg, ridgecv, ridgeclass]
```

In [38]:

```
for i in models_log:
    print(i)
    get_cv_scores1(i)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_inter
cept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10
0,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                   warm_start=False)
CV Mean Score:  0.9428571428571428
STD:  0.05553287518900288


SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_int
ercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge'
,
              max_iter=1000, n_iter_no_change=5, n_jobs=None, pena
lty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.
001,
              validation_fraction=0.1, verbose=0, warm_start=False
)
CV Mean Score:  0.780952380952381
STD:  0.06459361888690732


PassiveAggressiveClassifier(C=1.0, average=False, class_weight=Non
e,
                            early_stopping=False, fit_intercept=Tr
ue,
                            loss='hinge', max_iter=1000, n_iter_no
_change=5,
                            n_jobs=None, random_state=None, shuffl
e=True,
                            tol=0.001, validation_fraction=0.1, ve
rbose=0,
                            warm_start=False)
CV Mean Score:  0.9142857142857144
STD:  0.06317380553057904


RidgeClassifierCV(alphas=array([ 0.1,  1. , 10. ]), class_weight=N
one, cv=None,
                  fit_intercept=True, normalize=False, scoring=Non
e,
                  store_cv_values=False)
CV Mean Score:  0.838095238095238
STD:  0.07126966450997983


RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_int
ercept=True,
                max_iter=None, normalize=False, random_state=None,
                solver='auto', tol=0.001)
CV Mean Score:  0.838095238095238
STD:  0.07126966450997983
```

In [39]:

```
grid_list = []
random_list = []
for i in models_log:
    grids = GridSearchCV(estimator = lasso_lars, param_grid = param_grid, scor
ing = 'r2', cv = 10)
    grid_result = grids.fit(x_train, y_train)
    grid_list.append(grid_result.best_score_)
    rnds = RandomizedSearchCV(estimator = ridge, param_distributions = param_g
rid, scoring='r2', cv = 10)
    rnds_result = rnds.fit(x_train, y_train)
    random_list.append(rnds_result.best_score_)
```

Best model

In [40]:

```
print(max(grid_list))
print(max(random_list))
```

```
0.9916810501375295
0.9917607214483969
```

In [ ]:

```

```