SKLearn using python

Jiarui Groves

4/8/2023

CS 4375: Machine Learning

```
from google.colab import files

uploaded·=·files.upload()
```

Browse... Auto.csv
Auto.csv(application/vnd.ms-excel) - 17859 bytes, last modified: n/a - 100% done

```
Saving Auto.csv to Auto (1).csv
```

```
import io
import pandas as pd
df = pd.read_csv(io.BytesIO(uploaded['Auto.csv']))
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	r
0	18.0	8	307.0	130	3504	12.0	70.0	1	chev che ma
1	15.0	8	350.0	165	3693	11.5	70.0	1	b sky
2	18.0	8	318.0	150	3436	11.0	70.0	1	plym sat

```
#df['Length'].value_counts
print(len(df))
print('\nDimensions of data frame:', df.shape)
```

```
392

Dimensions of data frame: (392, 9)
```

Descriibe on auto data shows

✓ 4s completed at 2:25 PM

3. Range of year is from 70 to 82 years old

```
df.describe()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	
count	392.000000	392.000000	392.000000	392.000000	392.000000	391.000000	39
mean	23.445918	5.471939	194.411990	104.469388	2977.584184	15.554220	7
std	7.805007	1.705783	104.644004	38.491160	849.402560	2.750548	
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	7
25%	17.000000	4.000000	105.000000	75.000000	2225.250000	13.800000	7
50%	22.750000	4.000000	151.000000	93.500000	2803.500000	15.500000	7
75%	29.000000	8.000000	275.750000	126.000000	3614.750000	17.050000	7
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	8

```
df.dtypes
```

```
mpg              float64
cylinders          int64
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin             int64
name              object
dtype: object
```

Type of data contained are summarized above.

```
# Change cylinder to categorial type
df.cylinders=df.cylinders.astype('category').cat.codes
print("\n Cylinders after category conversion is now \n",df.cylinders)
df.origin = df.origin.astype('category')
print("\n Origin after category conversion is now \n", df.origin)
# Now the new dtype on data
print(" Here are the data after conversion\n:", df.dtypes)
print(" Converted data: \n",df.head)
```

```
 Cylinders after category conversion is now
 0      4
1      4
2      4
3      4
4      4
      ..
387    1
388    1
389    1
390    1
391    1
Name: cylinders, Length: 392, dtype: int8

 Origin after category conversion is now
 0      1
1      1
2      1
3      1
4      1
      ..
387    1
388    2
389    1
390    1
391    1
Name: origin, Length: 392, dtype: category
Categories (3, int64): [1, 2, 3]
 Here are the data after conversion
: mpg             float64
cylinders          int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
 Converted data:
 <bound method NDFrame.head of       mpg  cylinders  displacement  horsepower  weight  \
0    18.0          4         307.0         130    3504          12.0  70.0
1    15.0          4         350.0         165    3693          11.5  70.0
2    18.0          4         318.0         150    3436          11.0  70.0
3    16.0          4         304.0         150    3433          12.0  70.0
4    17.0          4         302.0         140    3449           NaN  70.0
..    ...        ...           ...         ...     ...           ...   ...
387  27.0          1         140.0          86    2790          15.6  82.0
388  44.0          1          97.0          52    2130          24.6  82.0
389  32.0          1         135.0          84    2295          11.6  82.0
390  28.0          1         120.0          79    2625          18.6  82.0
391  31.0          1         119.0          82    2720          19.4  82.0

    origin                       name
0        1  chevrolet chevelle malibu
```

```
1        1          buick skylark 320
2        1         plymouth satellite
```

```
df.isnull().sum()
# Original data oconsists 3 N/As
# Drop NAs in dataset
df = df.dropna()
df.shape
```

```
(389, 9)
```

New dimension of data decrearsed from 392 to 389 after the removal of 3 NAs

```
#mpg_high = df[['mpg']]
#mpg_high
mean = df["mpg"].mean()
mean    #find mean of column mpg
```

```
23.445918367346938
```

Printed entire column of mpg_high after making it.

```
df.loc[df['mpg'] >mean, 'mpg_high'] = '1'
df.loc[df['mpg']<=mean,'mpg_high'] = '0'
#print(df.loc[:'mpg'])
print(df.mpg_high)
print(df[['mpg_high']].to_string(index=False))

#print(df.mpg)
```

```
0      0
1      0
2      0
3      0
6      0
      ..
387    1
388    1
389    1
390    1
391    1
Name: mpg_high, Length: 389, dtype: object
mpg_high
       0
       0
       0
       0
       0
```

```
0
0
0
0
0
0
0
1
0
0
1
1
1
1
1
1
0
0
0
0
0
1
1
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

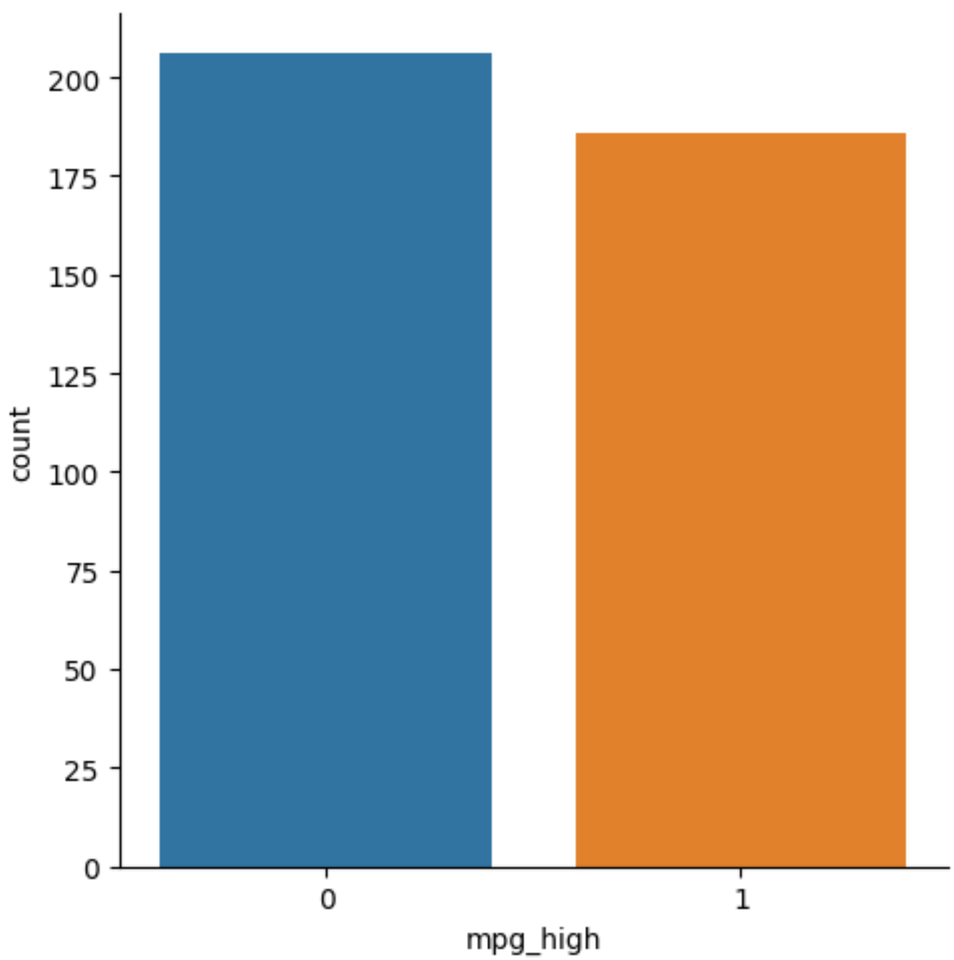
```
#Data after dropping original mpg is printed out
df.drop('mpg',inplace = True, axis = 1)
print(df.head)
```

```
<bound method NDFrame.head of      cylinders  displacement  horsepower  weight  accel
0            4         307.0         130    3504         12.0  70.0      1
1            4         350.0         165    3693         11.5  70.0      1
2            4         318.0         150    3436         11.0  70.0      1
3            4         304.0         150    3433         12.0  70.0      1
4            4         302.0         140    3449          NaN  70.0      1
..         ...           ...         ...     ...          ...   ...    ...
387          1         140.0          86    2790         15.6  82.0      1
```

```
388          1         97.0         52    2130          24.6  82.0       2
389          1        135.0         84    2295          11.6  82.0       1
390          1        120.0         79    2625          18.6  82.0       1
391          1        119.0         82    2720          19.4  82.0       1

                          name mpg_high
0    chevrolet chevelle malibu        0
1            buick skylark 320        0
2           plymouth satellite        0
3                amc rebel sst        0
4                  ford torino        0
..                         ...      ...
387            ford mustang gl        1
388                  vw pickup        1
389              dodge rampage        1
390                ford ranger        1
391                 chevy s-10        1

[392 rows x 9 columns]>
```

```
import seaborn as sb
from sklearn import datasets
sb.catplot(x="mpg_high", kind="count", data=df)
```

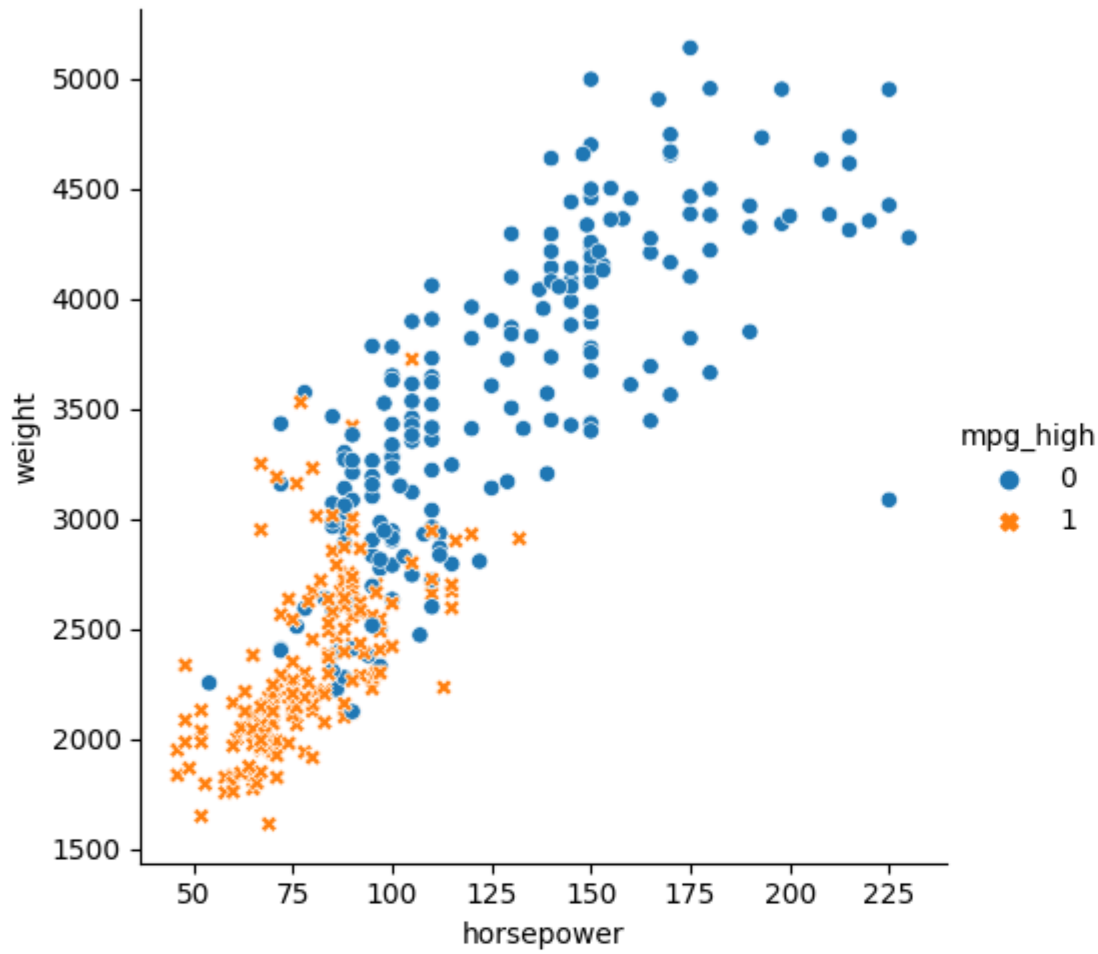
```
<seaborn.axisgrid.FacetGrid at 0x7f99bf399940>
```



```
# seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue or styl
```

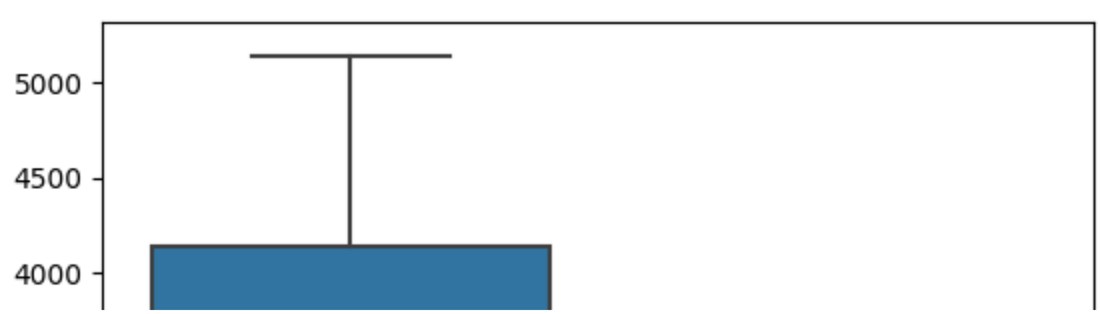
```
#Relaional plot shoes that there are a esignificant trend between weight and horsepower da

sb.relplot(x="horsepower", y="weight", data=df, hue=df.mpg_high, style=df.mpg_high)
```

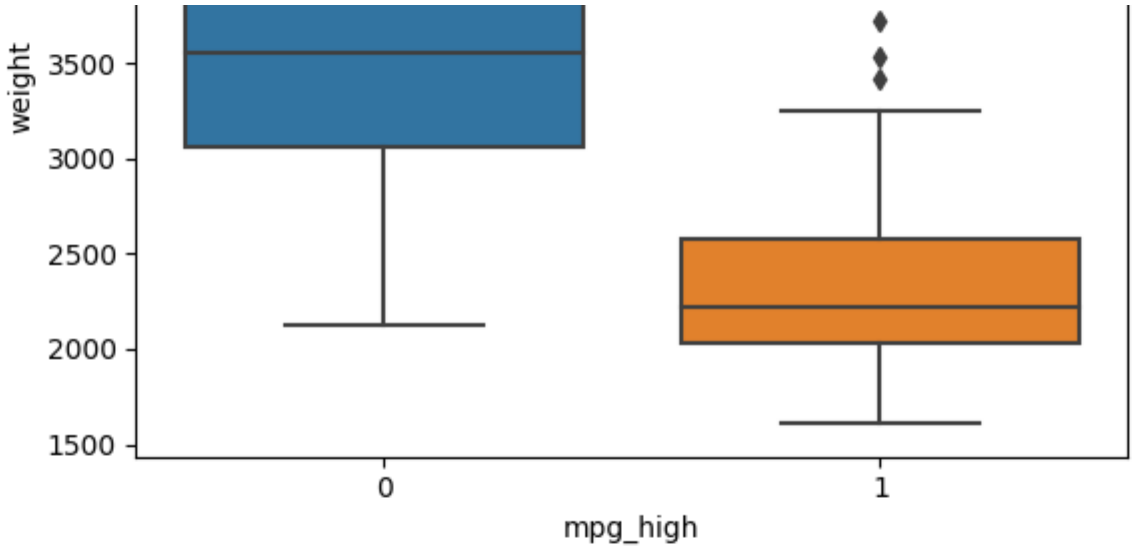
<seaborn.axisgrid.FacetGrid at 0x7f99ba93fb80>




```
#seaborn boxplot with mpg_high on the x axis and weight on the y axis
sb.boxplot(x = "mpg_high", y="weight", data=df)
```

<Axes: xlabel='mpg_high', ylabel='weight'>





Data tells us that amoung higher MPG/ more efficient vehicles, there are more higher weight ones while in vehicle with lower mpg/less efficient ones, there are no significant trends in weight

Created train and test of 80/20 percent. Obtained of train size eof 313 rows by 8 columns, and test size of 79 rows by 8 columns

```
from sklearn.model_selection import train_test_split
X = df.loc[:, ['cylinders','displacement','horsepower','weight','acceleration','year','orig
y = df.mpg_high

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
train size: (311, 7)
test size: (78, 7)
```

Logistic regression train a logistic regression model using solver lbfgs b. test and evaluate c. print metrics using the classification report

```
from sklearn.linear_model import LogisticRegression
#convert into catogrial data
clf = LogisticRegression()
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
0.9035369774919614
```

Logistic regressionshows around 90% accuracy

Copied df to a new dadaset newdf to perfrom decisiobn tree classfification without disturbing original dataset Using cylinders, year and horsepower as predictors

```
newdf = df.copy()
print(newdf.dtypes)
newdf.cylinders = newdf.cylinders.astype('category').cat.codes
newdf.year = newdf.year.astype('category').cat.codes
newdf.horsepower = newdf.horsepower.astype('category').cat.codes
print(newdf.dtypes)
```

```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
mpg_high         object
dtype: object
mpg             float64
cylinders          int8
displacement    float64
horsepower         int8
weight            int64
acceleration    float64
year               int8
origin            int64
name             object
mpg_high         object
dtype: object
```

```
#Decision tree using thge same train and test dataset
X = newdf.loc[:, ['cylinders','displacement','horsepower','weight','acceleration','year','
y = newdf.mpg_high

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

print('train size:', X_train.shape)
```

```
print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
train size: (311, 7)
test size: (78, 7)
```

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
# make predictions

pred = clf.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('accuracy score: ', accuracy_score(y_test, pred))
#print('precision score: ', precision_score(y_test, pred))
#print('recall score: ', recall_score(y_test, pred))
#print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.8974358974358975
```

```
from sklearn.metrics import confusion_matrix
# COnfusion matrix of decision tree gives 4 false positive, and 4 false negative
confusion_matrix(y_test, pred)
```

```
array([[39,  5],
       [ 3, 31]])
```

```
# Report shows result is slightly worse then the Logistic regtression model. with 3 false r

from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.89      0.91        44
```

```
           0       0.93      0.89      0.91        44
           1       0.86      0.91      0.89        34

    accuracy                           0.90        78
   macro avg       0.89      0.90      0.90        78
weighted avg       0.90      0.90      0.90        78
```

```
from sklearn import tree
tree.plot_tree(clf)
```

```
[Text(0.71328125, 0.9444444444444444, 'x[1] <= 190.5\ngini = 0.5\nsamples =
311\nvalue = [159, 152]'),
 Text(0.5015625, 0.8333333333333334, 'x[2] <= 96.5\ngini = 0.292\nsamples =
180\nvalue = [32, 148]'),
 Text(0.228125, 0.7222222222222222, 'x[1] <= 113.5\ngini = 0.188\nsamples =
152\nvalue = [16, 136]'),
 Text(0.05, 0.6111111111111112, 'x[0] <= 3.5\ngini = 0.041\nsamples = 96\nvalue =
[2, 94]'),
 Text(0.025, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.075, 0.5, 'x[2] <= 87.5\ngini = 0.021\nsamples = 95\nvalue = [1, 94]'),
 Text(0.05, 0.3888888888888889, 'gini = 0.0\nsamples = 82\nvalue = [0, 82]'),
 Text(0.1, 0.3888888888888889, 'x[4] <= 18.8\ngini = 0.142\nsamples = 13\nvalue =
[1, 12]'),
 Text(0.075, 0.2777777777777778, 'gini = 0.0\nsamples = 11\nvalue = [0, 11]'),
 Text(0.125, 0.2777777777777778, 'x[4] <= 19.3\ngini = 0.5\nsamples = 2\nvalue = [1,
1]'),
 Text(0.1, 0.16666666666666666, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.15, 0.16666666666666666, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.40625, 0.6111111111111112, 'x[5] <= 75.5\ngini = 0.375\nsamples = 56\nvalue
= [14, 42]'),
 Text(0.275, 0.5, 'x[4] <= 16.75\ngini = 0.475\nsamples = 18\nvalue = [11, 7]'),
 Text(0.2, 0.3888888888888889, 'x[1] <= 115.5\ngini = 0.444\nsamples = 9\nvalue =
[3, 6]'),
 Text(0.175, 0.2777777777777778, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.225, 0.2777777777777778, 'x[4] <= 16.0\ngini = 0.245\nsamples = 7\nvalue =
[1, 6]'),
 Text(0.2, 0.16666666666666666, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.25, 0.16666666666666666, 'x[3] <= 2338.5\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.225, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.275, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.35, 0.3888888888888889, 'x[4] <= 17.5\ngini = 0.198\nsamples = 9\nvalue =
[8, 1]'),
 Text(0.325, 0.2777777777777778, 'x[2] <= 79.0\ngini = 0.444\nsamples = 3\nvalue =
[2, 1]'),
 Text(0.3, 0.16666666666666666, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.35, 0.16666666666666666, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.375, 0.2777777777777778, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.5375, 0.5, 'x[4] <= 21.85\ngini = 0.145\nsamples = 38\nvalue = [3, 35]'),
 Text(0.475, 0.3888888888888889, 'x[2] <= 93.5\ngini = 0.105\nsamples = 36\nvalue =
[2, 34]'),
 Text(0.425, 0.2777777777777778, 'x[3] <= 2880.0\ngini = 0.059\nsamples = 33\nvalue
= [1, 32]'),
```

```
 Text(0.4, 0.16666666666666666, 'gini = 0.0\nsamples = 26\nvalue = [0, 26]'),
 Text(0.45, 0.16666666666666666, 'x[3] <= 2920.0\ngini = 0.245\nsamples = 7\nvalue =
[1, 6]'),
 Text(0.425, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.475, 0.05555555555555555, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(0.525, 0.2777777777777778, 'x[4] <= 14.5\ngini = 0.444\nsamples = 3\nvalue =
[1, 2]'),
 Text(0.5, 0.16666666666666666, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.55, 0.16666666666666666, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.6, 0.3888888888888889, 'x[5] <= 77.5\ngini = 0.5\nsamples = 2\nvalue = [1,
1]'),
 Text(0.575, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.625, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.775, 0.7222222222222222, 'x[5] <= 78.5\ngini = 0.49\nsamples = 28\nvalue =
[16, 12]'),
 Text(0.75, 0.6111111111111112, 'x[3] <= 2702.5\ngini = 0.266\nsamples = 19\nvalue =
[16, 3]'),
 Text(0.725, 0.5, 'x[5] <= 77.5\ngini = 0.5\nsamples = 6\nvalue = [3, 3]'),
 Text(0.7, 0.3888888888888889, 'x[3] <= 2630.0\ngini = 0.375\nsamples = 4\nvalue =
[3, 1]'),
 Text(0.675, 0.2777777777777778, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.725, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.75, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.775, 0.5, 'gini = 0.0\nsamples = 13\nvalue = [13, 0]'),
 Text(0.8, 0.6111111111111112, 'gini = 0.0\nsamples = 9\nvalue = [0, 9]'),
 Text(0.925, 0.8333333333333334, 'x[4] <= 21.6\ngini = 0.059\nsamples = 131\nvalue =
[127, 4]'),
 Text(0.9, 0.7222222222222222, 'x[5] <= 80.5\ngini = 0.045\nsamples = 130\nvalue =
[127, 3]'),
 Text(0.85, 0.6111111111111112, 'x[2] <= 83.0\ngini = 0.016\nsamples = 124\nvalue =
[123, 1]'),
 Text(0.825, 0.5, 'x[2] <= 79.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.8, 0.3888888888888889, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.85, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.875, 0.5, 'gini = 0.0\nsamples = 120\nvalue = [120, 0]'),
 Text(0.95, 0.6111111111111112, 'x[1] <= 247.0\ngini = 0.444\nsamples = 6\nvalue =
[4, 2]'),
 Text(0.925, 0.5, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.975, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.95, 0.7222222222222222, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]')]
```

Performing Linear Regression using Neural Network

```
## train the algorithm
from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
linreg.fit(X_train, y_train)

# make predictions
y_pred = linreg.predict(X_test)
```

```
# evaluation
from sklearn.metrics import mean_squared_error, r2_score
print('mse=', mean_squared_error(y_test, y_pred))
print('correlation=', r2_score(y_test, y_pred))
```

```
mse= 0.08450675765478677
correlation= 0.6563241219440358
```

MSE came out to be 0.08 that is very low with correlation being around 0.66

```
# scale the data using Python and pandas functionality

mean = X_train.mean(axis=0)
X_train -= mean
std = X_train.std(axis=0)
X_train /= std

X_test -= mean
X_test /= std
```

```
# scale the data using sklearn functionality
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
# train the algorithm using 6 then three layers with iteration 500
from sklearn.neural_network import MLPRegressor

regr = MLPRegressor(hidden_layer_sizes=(6, 3), max_iter=500, random_state=1234)
regr.fit(X_train, y_train)
```

```
▾ MLPRegressor
```

```
MLPRegressor(hidden_layer_sizes=(6, 3), max_iter=500, random_state=1234)
```

```
# Predicting using neural network
y_pred = regr.predict(X_test)
```

```
# evaluation
from sklearn.metrics import mean_squared_error, r2_score
print('mse=', mean_squared_error(y_test, y_pred))
print('correlation=', r2_score(y_test, y_pred))
```

```
mse= 0.08174039197427181
correlation= 0.6675745021581087
```

Newral network shows slight inmprovement after using the scaled data.

Now try using classfication

```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# train
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, random_state=
clf.fit(X_train_scaled, y_train)
```

```
▾ MLPClassifier
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

```
# make predictions

pred = clf.predict(X_test_scaled)
```

```
# output results

print('accuracy = ', accuracy_score(y_test, pred))

confusion_matrix(y_test, pred)
```

```
accuracy =  0.8589743589743589
array([[39,  5],
       [ 6, 28]])
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.89      0.88        44
           1       0.85      0.82      0.84        34

    accuracy                           0.86        78
   macro avg       0.86      0.85      0.86        78
weighted avg       0.86      0.86      0.86        78
```

```
# Try a different setting on linear regression that has max iiteration of 2500, and hidden
# No significant improvement observed

regr = MLPRegressor(hidden_layer_sizes=(6, 3), solver='lbfgs', max_iter=2500, random_state=
regr.fit(X_train, y_train)
```

```
▾                          MLPRegressor
MLPRegressor(hidden_layer_sizes=(6, 3), max_iter=2500, random_state=1234,
             solver='lbfgs')
```

```
y_pred = regr.predict(X_test)

print('mse=', mean_squared_error(y_test, y_pred))
print('correlation=', r2_score(y_test, y_pred))
```

```
mse= 0.08364533298943039
correlation= 0.6598274024681188
```

Accoding to the neural network accuracy result, classfication has a slight better result on the same data wioth 0.86 comparing to using linear regression t5hat has correlation of 0.67. That might be becasuse of classffication that is most likelt becasue we have more qualatative data after catagozie columns and classification was better suited for this specific dataset.

Neural nework V.S Regular linear regression Used two setting on neural setwork regression method to train the Regressor. After comparing to the two model, there are no significant improvement after asjudt=sting mmax-iteration and different hidden layers. Standard linear regression give out mse= 0.08450675765478677 correlation= 0.6563241219440358 and after scaling with neural network. Newral network linear regression model has mse= 0.08174039197427181 correlation= 0.6675745021581087

In R the data there wasn't as precise of operaion on data set as SKLearn using python but there were some simillar features such as tranfroming data using " Catogorial data v.s factors in R. I personally liked using Oython better as there were many efficient algorithms importng in SK Learn. As well as much precise matrix multiplication and array method.;