In [2]:
```
### Neural Network using sklearn ###
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
df=pd.read_csv('C:\\Users\\10526359\\Downloads\\concrete.csv', index_col=None)
```

Exploratory Data Analsyis

In [4]:
```
df.head()
```

Out[4]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength
0	141.3	212.0	0.0	203.5	0.0	971.8	748.5	28	29.89
1	168.9	42.2	124.3	158.3	10.8	1080.8	796.2	14	23.51
2	250.0	0.0	95.7	187.4	5.5	956.9	861.2	28	29.22
3	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28	45.85
4	154.8	183.4	0.0	193.3	9.1	1047.4	696.7	28	18.29

In [5]:
```
df.describe()
```

Out[5]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	10
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	3

In [6]:
```
df.isnull().sum() # if wish to drop nulls then df.dropna(inplace=True)
```

Out[6]:
```
cement          0
slag            0
ash             0
water           0
superplastic    0
coarseagg       0
fineagg         0
age             0
strength        0
dtype: int64
```

In [7]:
```
df.corr()
```

Out[7]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age
cement	1.000000	-0.275216	-0.397467	-0.081587	0.092386	-0.109349	-0.222718	0.081946
slag	-0.275216	1.000000	-0.323580	0.107252	0.043270	-0.283999	-0.281603	-0.044246
ash	-0.397467	-0.323580	1.000000	-0.256984	0.377503	-0.009961	0.079108	-0.154371
water	-0.081587	0.107252	-0.256984	1.000000	-0.657533	-0.182294	-0.450661	0.277618
superplastic	0.092386	0.043270	0.377503	-0.657533	1.000000	-0.265999	0.222691	-0.192700
coarseagg	-0.109349	-0.283999	-0.009961	-0.182294	-0.265999	1.000000	-0.178481	-0.003016
fineagg	-0.222718	-0.281603	0.079108	-0.450661	0.222691	-0.178481	1.000000	-0.156095
age	0.081946	-0.044246	-0.154371	0.277618	-0.192700	-0.003016	-0.156095	1.000000
strength	0.497832	0.134829	-0.105755	-0.289633	0.366079	-0.164935	-0.167241	0.328873

In [ ]:
```
sns.pairplot(data=df)
```

Data Preprocessing

In [8]:
```
### remove missing values if there are any
df=df.dropna()
```

In [9]:
```
### create taining and testing data subsets
y = df[['strength']]
X = df[df.columns.drop(y)]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2)
```

In [10]:
```
### scale (normalize) the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)  # Don't cheat - fit only on training data
X_train = scaler.transform(X_train)
# apply same transformation to test data
X_test = scaler.transform(X_test)
```

Neural Network Modeling

In [ ]:
```
# sklearn has two NN functions: MLPClassifier for classification tasks and MLPRegresso
# examples at https://towardsdatascience.com/deep-neural-multilayer-perceptron-mlp-wit
# https://scikit-learn.org/stable/modules/neural_networks_supervised.html
# https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifi
# https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegresso
# since this problem has a continuous target variable, it is a regression problem, and
# syntax: MLPRegressor(hidden_layer_sizes, activation (relu is the default), max_iter,
```

In [11]:
```
from sklearn.neural_network import MLPRegressor
```

In [12]:
```
# simple model named m with 1 hidden layer of 20 nodes and defaults(relu and alpha)
m = MLPRegressor(hidden_layer_sizes = (20,), max_iter=10000)
m.fit(X_train, np.ravel(y_train)) #must use np.ravel() to flatten the array from shape
```

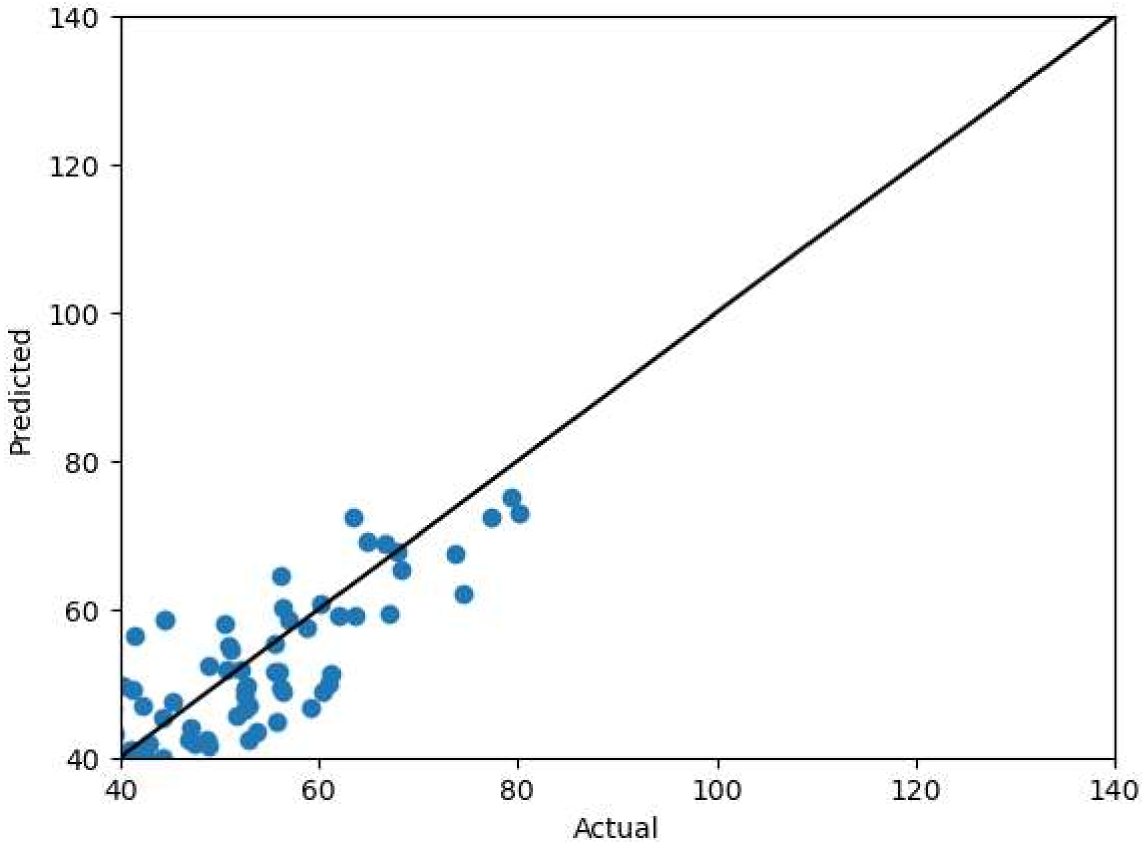
Out[12]:
```
MLPRegressor(hidden_layer_sizes=(20,), max_iter=10000)
```

In [13]:
```
# predict on the testing data
y_pred = m.predict(X_test)
```

In [14]:
```
# validating model results (for a estimation model - continuous target var)
from sklearn import metrics
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = mse**0.5
r2 = metrics.r2_score(y_test, y_pred)
print(f"""
MAE: \t{mae:.2f}
RMSE: \t{rmse:.2f}
r2: \t{r2:.2f}
""")
```

```
MAE:    4.36
RMSE:   5.75
r2:     0.88
```

In [15]:
```
# we can also plot this actual vs predictied to visualize model performance
plt.scatter(y_test, y_pred)
plt.xlim(40, 140)
plt.ylim(40, 140)
plt.ylabel('Predicted')
plt.xlabel('Actual')
plt.plot([40,140], [40,140], 'black') #1 to 1 line
plt.show()
```



In [16]:
```
# a more complex model m2 with 3 hidden layers of many nodes
m2 = MLPRegressor(hidden_layer_sizes = (256, 128, 64), max_iter=10000)
m2.fit(X_train, np.ravel(y_train))
```

Out[16]:
```
MLPRegressor(hidden_layer_sizes=(256, 128, 64), max_iter=10000)
```

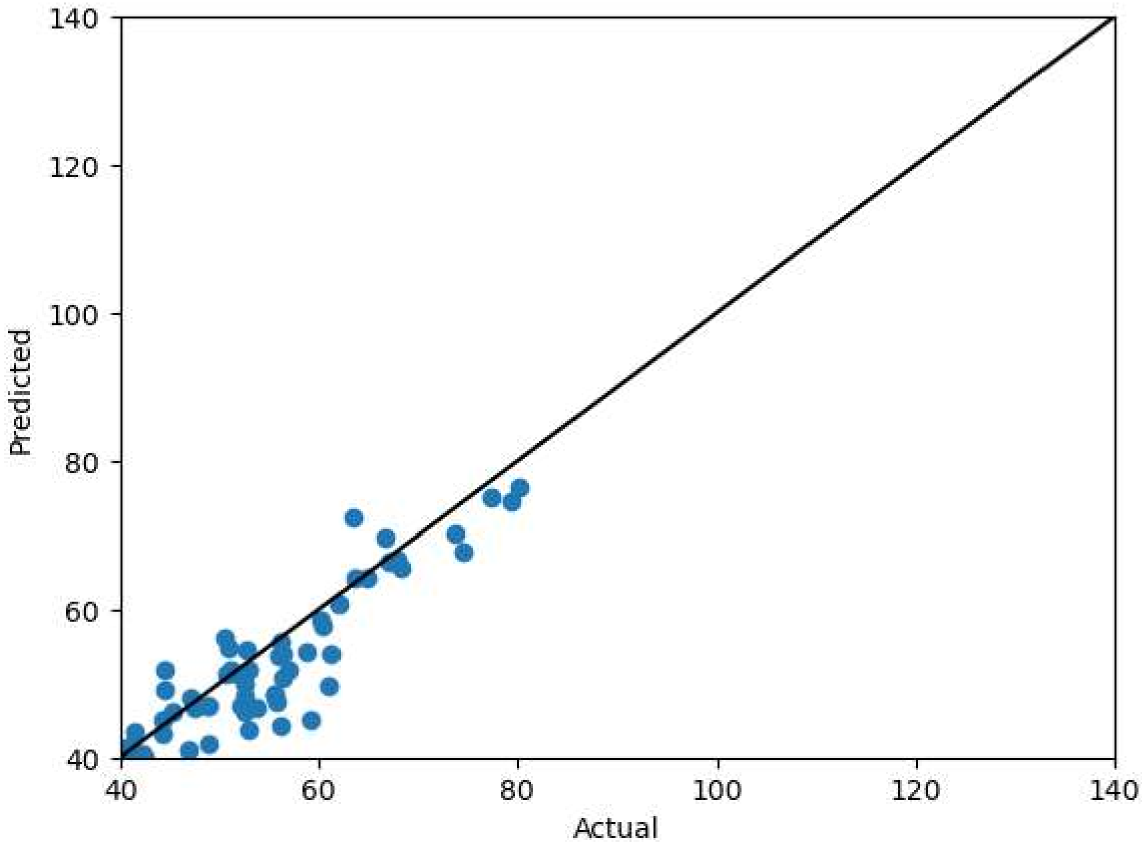
In [17]:
```
# predict on the testing data
y_pred2 = m2.predict(X_test)
```

In [18]:
```
# validating model results (for a estimation model - continuous target var)
from sklearn import metrics
mae = metrics.mean_absolute_error(y_test, y_pred2)
mse = metrics.mean_squared_error(y_test, y_pred2)
rmse = mse**0.5
r2 = metrics.r2_score(y_test, y_pred2)
print(f"""
MAE: \t{mae:.2f}
RMSE: \t{rmse:.2f}
r2: \t{r2:.2f}
""")
```

```
MAE:    3.29
RMSE:   4.48
r2:     0.92
```

In [19]:
```
# we can also plot this actual vs predictied to visualize model performance
plt.scatter(y_test, y_pred2)
plt.xlim(40, 140)
plt.ylim(40, 140)
plt.ylabel('Predicted')
```

```
plt.xlabel('Actual')
plt.plot([40,140], [40,140], 'black') #1 to 1 line
plt.show()
```



In [20]:

```
# optionally get info on the model
print("Loss: ", m2.loss_)
print("Number of Coefs : ", len(m2.coefs_))
print("Number of Intercepts : ", len(m2.intercepts_))
print("Number of Iterations for Which Estimator Ran : ", m2.n_iter_)
print("Name of Output Layer Activation Function : ", m2.out_activation_)
```

```
Loss:  6.155926879053878
Number of Coefs :  4
Number of Intercepts :  4
Number of Iterations for Which Estimator Ran :  317
Name of Output Layer Activation Function :  identity
```

In [21]:

```
%%time
from sklearn.model_selection import GridSearchCV
import itertools

params = {'activation': ['relu', 'tanh', 'logistic', 'identity'],
          'hidden_layer_sizes': [(3,), (3,5,), (3,5,3,)],
          'solver': ['adam', 'lbfgs'],
          'learning_rate' : ['constant', 'adaptive', 'invscaling']
         }
# GridSearchCV https://towardsdatascience.com/gridsearchcv-for-beginners-db48a90114ee
m2_grid = GridSearchCV(m2, param_grid=params, n_jobs=-1, cv=5, verbose=5)
m2_grid.fit(X_train,np.ravel(y_train))

print('Train R^2 Score : %.3f'% m2_grid.best_estimator_.score(X_train, y_train))
print('Test R^2 Score : %.3f'% m2_grid.best_estimator_.score(X_test, y_test))
```

```
print('Best R^2 Score Through Grid Search : %.3f'% m2_grid.best_score_)
print('Best Parameters : ', m2_grid.best_params_)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Train R^2 Score : 0.873
Test R^2 Score : 0.881
Best R^2 Score Through Grid Search : 0.841
Best Parameters :  {'activation': 'logistic', 'hidden_layer_sizes': (3,), 'learning_r
ate': 'invscaling', 'solver': 'lbfgs'}
Wall time: 10min 29s
```

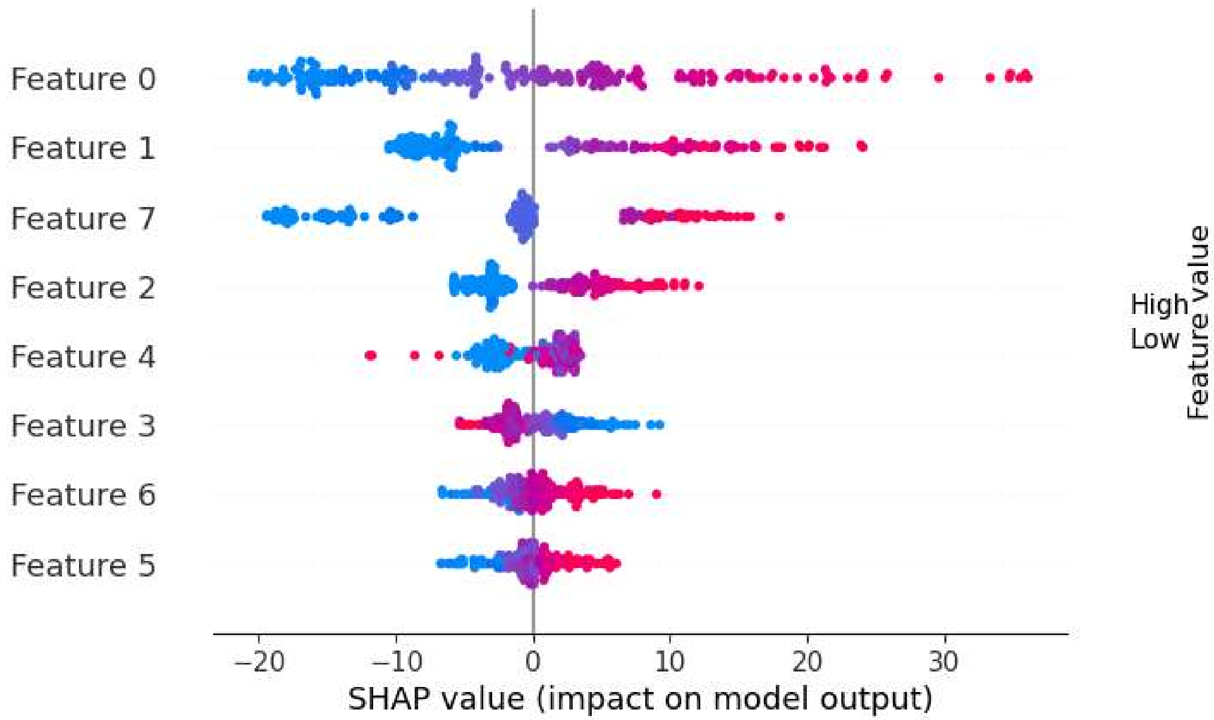
In [22]:

```
# Interesting explainer named SHAP  https://shap.readthedocs.io/en/latest/index.html#
import shap
shap.initjs()

# rather than use the whole training set to estimate expected values, we summarize wit
# a set of weighted kmeans, each weighted by the number of points they represent.
X_train_summary = shap.kmeans(X_train, 100)

shap_explainer = shap.KernelExplainer(m.predict,X_train_summary)
# shap_explainer = shap.KernelExplainer(m.predict,X_train) to run against entire datas
shap_values = shap_explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

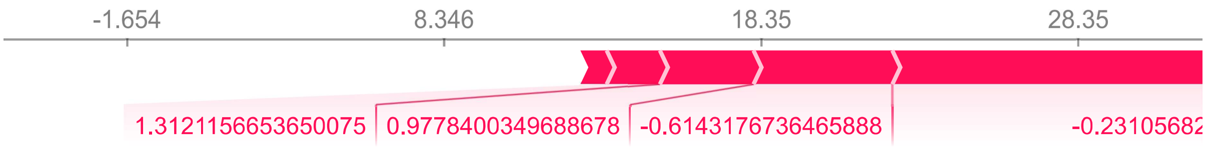
```
  0%|          | 0/206 [00:00<?, ?it/s]
```



In [23]:

```
# explainer graph for a single row (row 5)
shap.force_plot(shap_explainer.expected_value, shap_values[1, :], X_test[5,:])
```

Out[23]:



Classification NN Problem code (no data)

In [ ]:

```
# if this had been a classification problem we would have used MLPClassifier and a cor
from sklearn.nerual_network import MLPClassifier

read data

nnclass = MLPClassifier()
nnclass.fit(X_train,np.ravel(y_train))

y_predclass = nnclass.predict(X_test)
print('Test Accuracy : %.3f'%mlp_classifier.score(X_test, y_test)) # Score method defa

# plot fancy confusion matrix
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(Y_test, y_predclass):
    conf_mat = confusion_matrix(Y_test, y_predclass)
    #print(conf_mat)
    fig = plt.figure(figsize=(6,6))
    plt.matshow(conf_mat, cmap=plt.cm.Blues, fignum=1)
    plt.yticks(range(10), range(10))
    plt.xticks(range(10), range(10))
    plt.colorbar();
    for i in range(10):
        for j in range(10):
            plt.text(i-0.2,j+0.1, str(conf_mat[j, i]), color='tab:red')

plot_confusion_matrix(y_test, nnclass.predict(X_test))

from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_predclass)
np.mean(y_predclass == y_test) #accuracy

#Grid Search for Classification model
%%time

from sklearn.model_selection import GridSearchCV

params = {'activation': ['relu', 'tanh', 'logistic', 'identity'],
          'hidden_layer_sizes': [(100,), (50,100,), (50,75,100,)],
          'solver': ['adam', 'sgd', 'lbfgs'],
          'learning_rate' : ['constant', 'adaptive', 'invscaling']
         }

mlp_classif_grid = GridSearchCV(MLPClassifier(random_state=123), param_grid=params, n_
```

```
mlp_classif_grid.fit(X_train,Y_train)

print('Train Accuracy : %.3f'%mlp_classif_grid.best_estimator_.score(X_train, Y_train)
print('Test Accuracy : %.3f'%mlp_classif_grid.best_estimator_.score(X_test, Y_test))
print('Best Accuracy Through Grid Search : %.3f'%mlp_classif_grid.best_score_)
print('Best Parameters : ',mlp_classif_grid.best_params_)
```

In [ ]:

```
def print_accuracy(f):
    print("Root mean squared test error = {0}".format(np.sqrt(np.mean((f(X_test) - y_t
    time.sleep(0.5) # to let the print get out before any progress bars

print_accuracy(m.predict)
```

In [ ]:

```
# nice reference: https://coderzcolumn.com/tutorials/machine-learning/scikit-learn-skl
# another reference: https://towardsdatascience.com/deep-neural-multilayer-perceptron-
```