In [1]:

```
import numpy as np
import pandas as pd
from numpy.random import RandomState
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
```

In [2]:

```
df = pd.read_csv (r'C:\\Users\\rezam\\Desktop\\New folder\\Datasets\\Q3\Communities_Crime.csv', he
ader=None)
```

The cleaning process of the dataset:

- The dataset had missing values in some columns. The values have been replaced the avrage of the values of the column.

In [3]:

```
df.shape
df.dtypes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1994 entries, 0 to 1993
Columns: 127 entries, 0 to 126
dtypes: float64(127)
memory usage: 1.9 MB
```

The cleaning process of the dataset 2:

- The columns (indexes 0 & 3) had int64 datatypes which have been modified to float.

- The column 29 had a missing value (object), and it has ben replaced the the avrage of related column.

- The loops below indicate the indexes of those values.

In [4]:

```
df.columns
for col in df.columns:
    if str(df.iloc[:,col].dtypes) == 'object':
        print(col)
# df.iloc[:,1]
```

In [5]:

```
df.columns
for col in df.columns:
    if str(df.iloc[:,col].dtypes) == 'int64':
        print(col)
# df.iloc[:,1]
```

Data preparation:

- The imported dataset has been devided in 80% train and 20% test.

- The train dataset will be used for cross validation and the test data set will remain untouche for the final validation.

In [6]:

```
from numpy.random import RandomState
rng = RandomState()
train = df.sample(frac=0.8, random_state=rng)
final_test = df.loc[~df.index.isin(train.index)]
```

In [7]:

```
train.shape
```

Out[7]:

```
(1595, 127)
```

In [8]:

```
final_test.shape
```

Out[8]:

```
(399, 127)
```

Observation:

- The 80% train dataset has been allocated for cross validation.

- The remaining 20% will be fitted to the best performed model.

- The target has been set to the last column.

In [9]:

```
X_first_train = train.iloc[:,0:126].to_numpy()
y_first_train = train.iloc[:,-1:].to_numpy()
```

In [10]:

```
X_first_train.shape
```

Out[10]:

```
(1595, 126)
```

In [11]:

```
y_first_train.shape
```

Out[11]:

```
(1595, 1)
```

- Data preparation:

In [12]:

```
X_train, X_test, y_train, y_test = train_test_split(X_first_train, y_first_train, test_size=0.2)
```

KFold cross validation (basic)

In [13]:

```
kf = KFold(n_splits=5, random_state=42, shuffle=True)
```

In [14]:

```
for train_index, test_index in kf.split(df):
#     print(train_index, test_index)
    print(kf)
```

```
KFold(n_splits=5, random_state=42, shuffle=True)
KFold(n_splits=5, random_state=42, shuffle=True)
KFold(n_splits=5, random_state=42, shuffle=True)
KFold(n_splits=5, random_state=42, shuffle=True)
KFold(n_splits=5, random_state=42, shuffle=True)
```

- Function for model (basic):

In [15]:

```
def get_score(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    return model.score(X_test, y_test)
```

- Linear regression

In [16]:

```
get_score(LinearRegression(), X_train, X_test, y_train, y_test)
```

Out[16]:

```
-42.161247037054444
```

- RidgeCV

In [17]:

```
get_score(RidgeCV(), X_train, X_test, y_train, y_test)
```

Out[17]:

```
0.6469230681199838
```

- Cross_val_score function:

- Linear regression

In [18]:

```
cross_val_score(LinearRegression(), X_train, y_train,cv=5)
```

Out[18]:

```
array([0.5743145 , 0.56750283, 0.51762012, 0.66243545, 0.56591542])
```

- RidgeCV

In [19]:

```
cross_val_score(RidgeCV(), X_train, y_train,cv=5)
```

Out[19]:

```
array([0.62580057, 0.66909557, 0.57411788, 0.68452015, 0.6000395 ])
```

- Cross_val_score function:

- Tunning by using k fold cross validation.

- Linear regression:

In [20]:

```
scores1 = cross_val_score(LinearRegression(),X_train, y_train, cv=5)
np.average(scores1)
```

Out[20]:

```
0.5775576647160919
```

In [21]:

```
scores1 = cross_val_score(LinearRegression(),X_train, y_train, cv=3)
np.average(scores1)
```

Out[21]:

```
0.5641532199736795
```

In [22]:

```
scores1 = cross_val_score(LinearRegression(),X_train, y_train, cv=10)
np.average(scores1)
```

Out[22]:

```
0.5848903306930472
```

In [23]:

```
scores1 = cross_val_score(LinearRegression(),X_train, y_train, cv=20)
np.average(scores1)
```

Out[23]:

```
0.5836173462574048
```

In [24]:

```
scores1 = cross_val_score(LinearRegression(),X_train, y_train, cv=15)
np.average(scores1)
```

Out[24]:

```
0.577691403493722
```

- RidgeCV:

In [25]:

```
scores1 = cross_val_score(RidgeCV(alphas=[1]),X_train, y_train, cv=3)
np.average(scores1)
```

Out[25]:

```
0.616747442820497
```

In [26]:

```
scores1 = cross_val_score(RidgeCV(alphas=[1e1]),X_train, y_train, cv=5)
np.average(scores1)
```

Out[26]:

```
0.6307147357232425
```

In [27]:

```
scores1 = cross_val_score(RidgeCV(alphas=[1e2]),X_train, y_train, cv=10)
np.average(scores1)
```

Out[27]:

```
0.616987916282623
```

In [28]:

```
scores1 = cross_val_score(RidgeCV(alphas=[1e3]),X_train, y_train, cv=15)
np.average(scores1)
```

Out[28]:

```
0.42361616936340235
```

In [29]:

```
scores1 = cross_val_score(RidgeCV(alphas=[1e4]),X_train, y_train, cv=20)
np.average(scores1)
```

Out[29]:

```
0.12688661409200375
```

- Best performed:

In [30]:

```
X_final_train = final_test.iloc[:,0:126].to_numpy()
y_final_train = final_test.iloc[:,-1:].to_numpy()
```

In [31]:

```
X_final_train.shape
```

Out[31]:

```
(399, 126)
```

In [32]:

```
y_final_train.shape
```

Out[32]:

```
(399, 1)
```

- Linear regression:

In [33]:

```
scores1 = cross_val_score(LinearRegression(),X_final_train, y_final_train, cv=10)
np.average(scores1)
```

Out[33]:

0.5035571506266585

- RidgeCV:

In [34]:

```
scores1 = cross_val_score(RidgeCV(alphas=[1e1]),X_final_train, y_final_train, cv=5)
np.average(scores1)
```

Out[34]:

0.678992489073349