In [1]:
```
import itertools
import numpy as np
import pandas as pd
from pprint import pprint
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold, cross_validate, train_test_split
from sklearn.metrics import silhouette_score, mean_absolute_error, mean_squarec

from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LogisticRegression, Lasso, BayesianRidge, Line

from sklearn.cluster import KMeans
```

In [2]:
```
ds_1 = pd.read_csv('ds1.csv', index_col = 0)
ds_2 = pd.read_csv('ds2.csv', index_col = 0)
print(ds_1.isnull().values.any())
print(ds_2.isnull().values.any())
```

```
False
False
```

In [3]:
```
ds_1.head()
```

Out[3]:

	x1	x2	x3	x5	x6	ya	yb	yc
1	2.642583	-1.715220	1.909334	0.027139	-3.447187	13.630850	1.828765	0.008386
2	4.588761	-2.507543	4.239107	1.704150	-2.782809	7.834582	2.162110	0.000008
3	7.919796	-5.108415	3.039451	0.992815	5.551587	-5.107041	2.797083	-0.000005
4	2.616757	-2.124040	2.855570	0.990079	1.694697	19.015046	1.953887	0.038017
5	3.300856	-5.159684	0.764544	0.143581	3.277496	-9.818862	1.922446	0.001178

In [4]:
```
ds_2.head()
```

Out[4]:

	X1	X2	X3	X4	X5	X6	X7	X8
1	23.778224	13.319974	15.565124	-3.713626	7.296793	-19.371013	-0.894130	-6.110282
2	16.602950	23.311281	21.099052	-0.304154	-3.218990	2.357643	12.027277	7.070349
3	12.084683	19.710443	9.837102	-1.081918	-1.201942	9.738019	16.125920	19.119391
4	13.044534	10.749040	5.884407	-11.703525	-4.134358	-22.344666	-1.263349	0.493711
5	8.314115	6.748794	5.388535	-0.000290	-4.724787	-16.346812	3.293600	-10.848273

Question 1

a. Describe the data set in a few sentences.
E.g. What are the distributions of each feature?
Summary statistics?

b. Try to come up with a predictive model, e.g. y = f(x_1 , ... , x_n) for each y sequence.
Describe your models and how you came up with them.
What (if any) are the predictive variables?
How good would you say each of your models is?

a)
The data set has 5 features (the x columns) and 3 targets (the y columns).
On a side note, not sure if the missing `x4` is a typo or intentional, but thats why I decided to create the 2 functions `get_targets_list` and `get_features_list` to get the `x` and `y` columns from the dataset.

Class `Describe` does the EDA for the data. Each feature column contains float values. None of the columns contain any null values, if they did, I would use the mean for each column to fill them, using `df.fillna(df.mean())`

I plot the histogram and the kernel density estimation (kde) plots for each feature.

For summary stats, I list out the count, mean, standard deviation, min, max, 25% quantile, 50% quantile, 75% quantile, median, skew, average (same as mean), variance, sum of all values and standard error of the mean (sem) for each of the features.

Every column contains 10000 values. The values for the summary stas are printed when the `runner` function in `Describe` class is called.

Based on the histogram plots and kde, heres what I can conclude about the features in terms of distribution:

1. **x1** has a Uniform Distribution
2. **x2** has a Cauchy Distribution (could also be Normal Distribution)
3. **x3** has a Normal Distribution
4. **x5** has a F Distribution
5. **x6** has a Double Exponential Distribution

I also made 15 additional plots, one for each individual feature vs each individual target. This can show which features are important for each target, also how the data is scattered, where the data is centered and if there are any outliers.

b) Looking at the data, we can see that its a Regression problem.
I used the basic ML training methodology.

First, I broke the data into a train test split, where 20% of data is for testing.
Next, I used K Fold cross validation (5 splits) to ensure the accuracy of the models, and to

make sure there isnt over fitting.

Then to see what features are useful, and what features are not, I used a permutation combination to get all possible combinations of the features.
I ran each combination through k fold, and recorded the accuracy of the models. See self.scoring array to see all the different metrics I use to measure the performance of the models.

Since this is a regression problem, I tried several different models from SkLearn. The list of models are shown in the first cell up top. The results were very similar, so ultimately I choose `LinearRegression` model.

The `evaluate` function in `Regression_Evaluation` class evaluates the different combinations. And then produces a table of R2 scores for each combination. I only show the top 3 combination sorted by score.

For each target, I then take the top combination of variables, and train a model, and then test it using a train test split of 0.9.
I also use all the features and do the same for each target.
The results are shown below, as well as the exact equation, which is a combination of coefficients and intercept.
I also plot the predicted y values vs the actual y values to show how well the model predicts values, and how close the predicted values are to the true values.

For ya, I get an R2 score of 67%, which shows the model is ok, but needs improvement. This shows that there is a correlation between the features and the target, but it is not strong.
For ya, the important features are [x1, x2, x3].

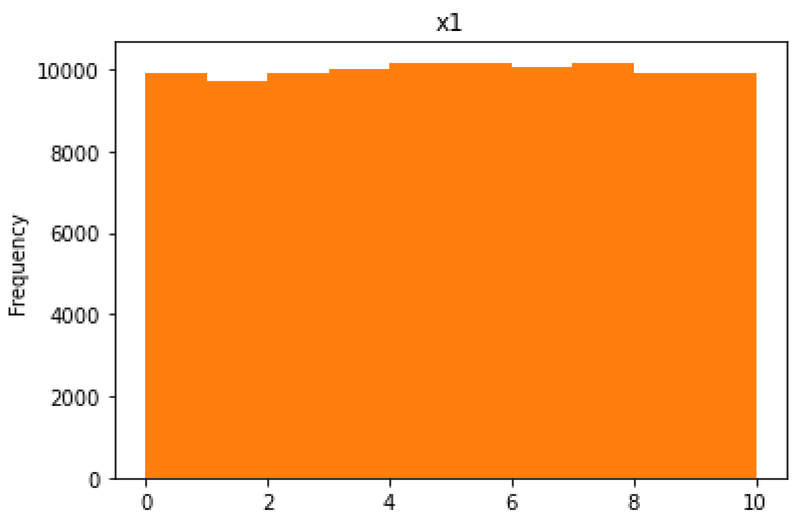
For yb, I get an R3 score of 89%, which shows the model is very good. This shows that there is a strong correlation between the features and the target. For yb, the important features are [x1, x3].

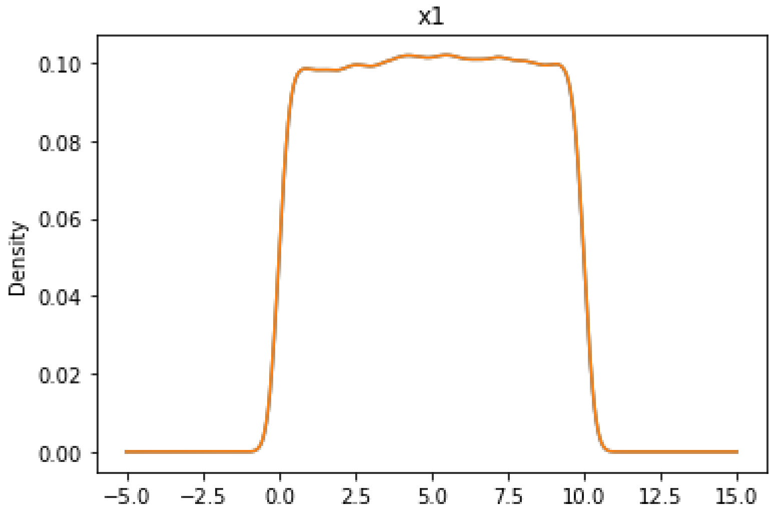
For yc, I get an R3 score of 0%, which shows the model is not at all correct. This shows that there is no correlation between the features and the target, regardless of the combination of features.

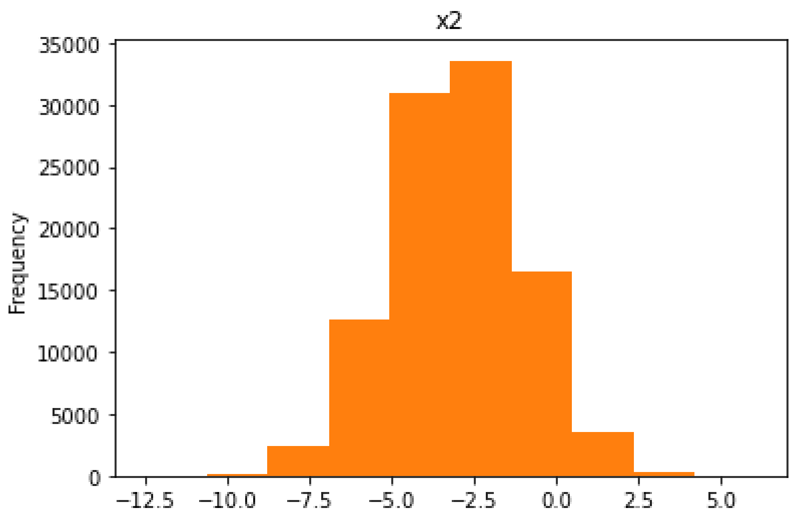
In [5]:
```
def get_targets_list(ds_1):
    target_list = []
    for col in ds_1.columns:
        if ('y' in col):
            target_list.append(col)
    return target_list

def get_features_list(ds_1):
    feature_list = []
    for col in ds_1.columns:
        if ('x' in col):
            feature_list.append(col)
    return feature_list
```

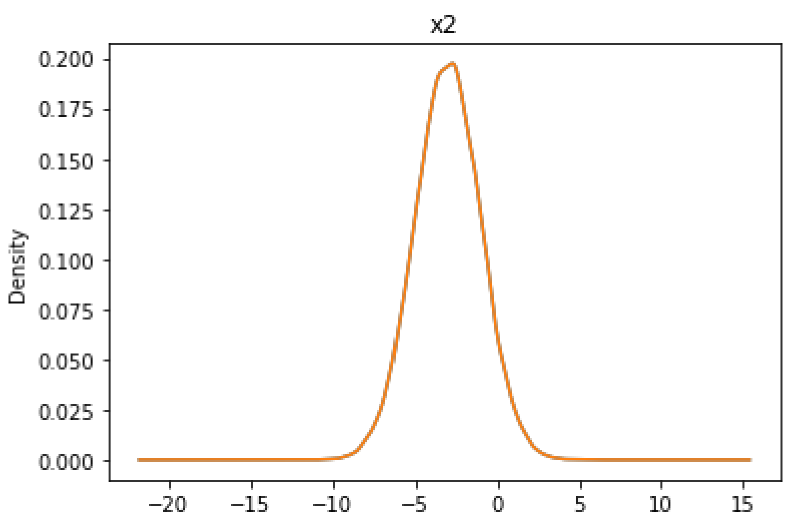
In [6]:
```
class Describe:
    def __init__(self, ds_1):
        self.ds_1 = ds_1
        self.kinds = ['hist', 'kde']
        self.summary_list = ['median', 'skew', 'average', 'var', 'sem', 'sum']
        self.features = get_features_list(self.ds_1)

    def distribution(self, feature, title = '', kind = 'hist'):
        feature.plot(kind = kind, title = title)
        feature.plot(kind = kind, title = title)
        plt.show()

    def summary(self, feature, title = ''):
        print(title, '\n')
        print(feature.describe(), '\n')
        print(feature.agg(self.summary_list), '\n')

    def runner(self):
        ds_1 = self.ds_1
        for feature in self.features:
            for kind in self.kinds:
                self.distribution(ds_1[feature], title = feature, kind = kind)

        for feature in self.features:
            self.summary(ds_1[feature], feature)
```

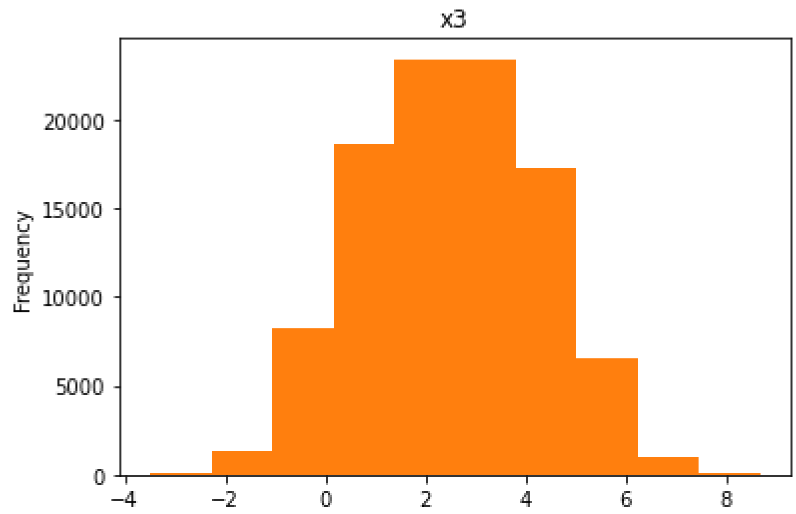
In [7]:
```
Describe(ds_1).runner()
```

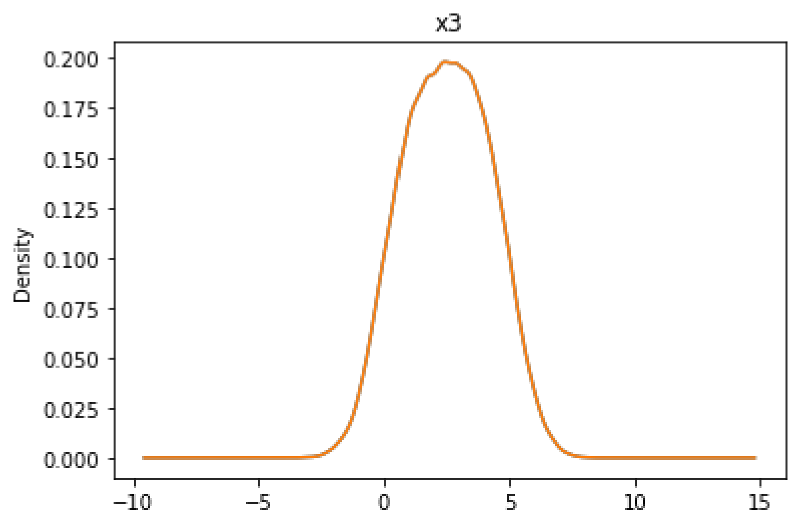


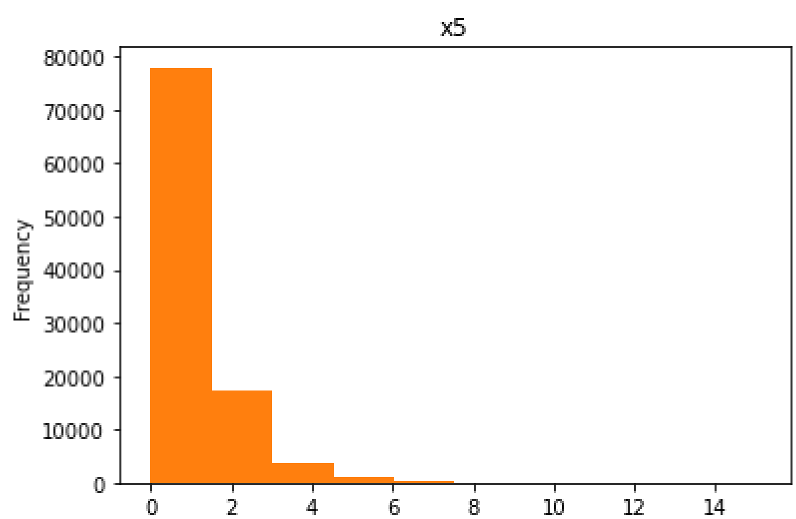

x1
Density
0.10
0.08
0.06
0.04
0.02
0.00
−5.0 −2.5 0.0 2.5 5.0 7.5 10.0 12.5 15.0

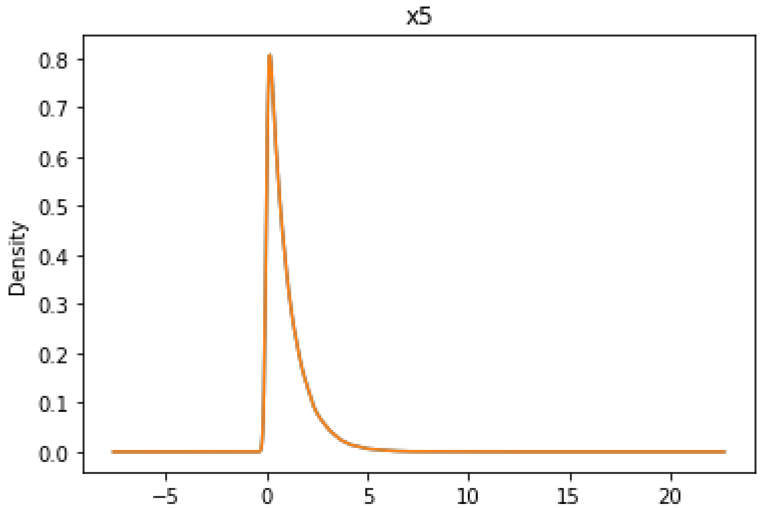

x2
Frequency
35000
30000
25000
20000
15000
10000
5000
0
−12.5 −10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0

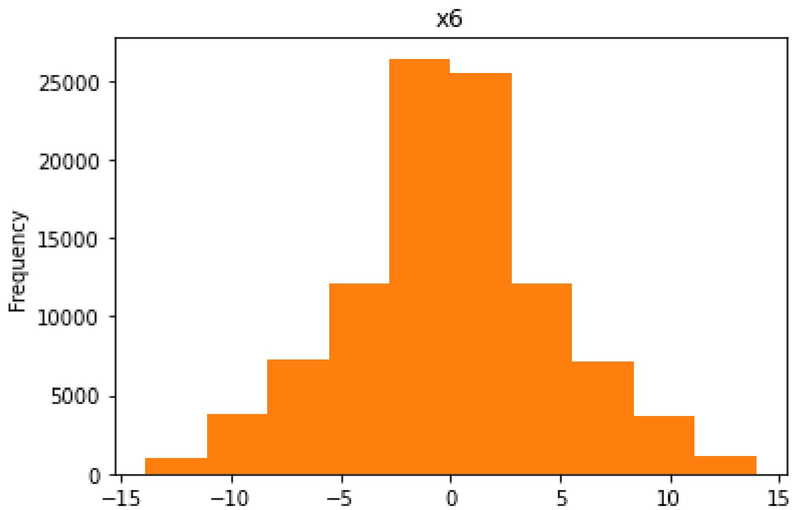

x2
Density
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−20 −15 −10 −5 0 5 10 15

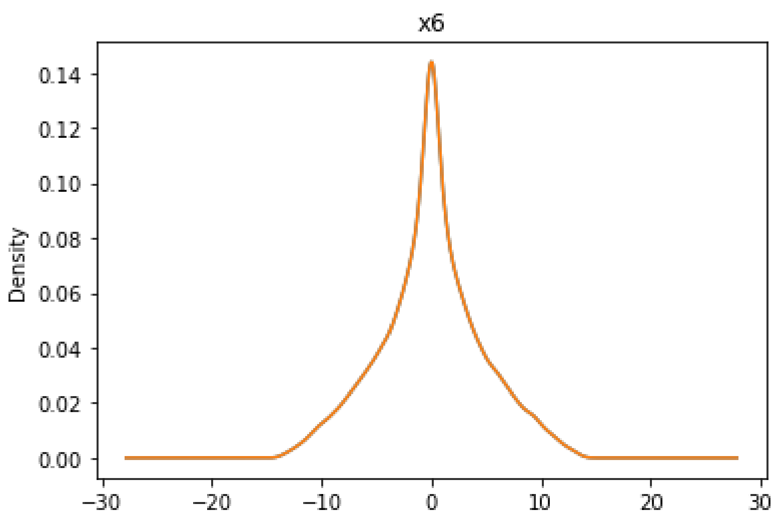

x3
Frequency
20000
15000
10000
5000
0
−4
−2
0
2
4
6
8


x3
Density
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−10
−5
0
5
10
15


x5
Frequency
80000
70000
60000
50000
40000
30000
20000
10000
0
0
2
4
6
8
10
12
14


x5
Density
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−5
0
5
10
15
20


x6
Frequency
25000
20000
15000
10000
5000
0
−15
−10
−5
0
5
10
15

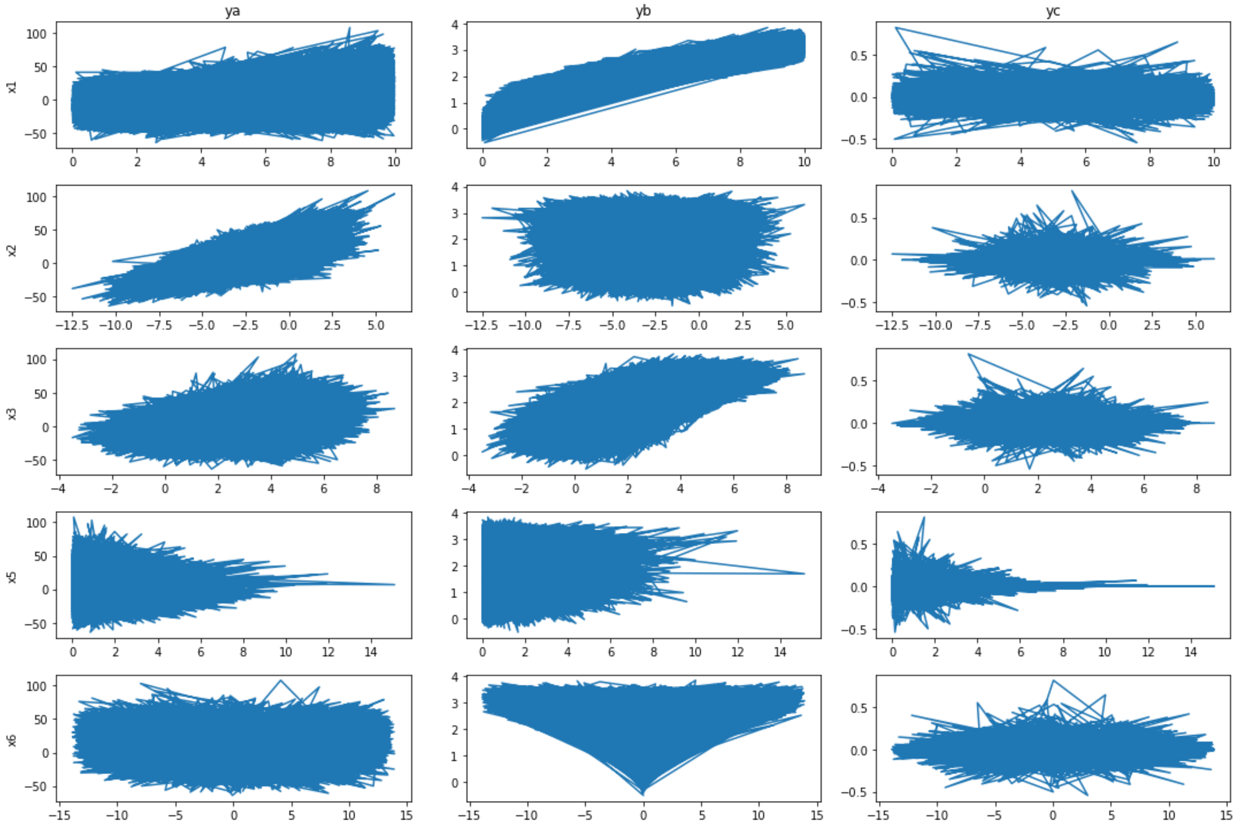

x6
Density
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−30
−20
−10
0
10
20
30

```
x1

count    100000.000000
mean          5.011059
std           2.873568
min           0.000015
25%           2.536309
50%           5.022191
75%           7.486275
max           9.999887
Name: x1, dtype: float64

median          5.022191
skew           -0.008450
average         5.011059
var             8.257391
sem             0.009087
sum        501105.931893
Name: x1, dtype: float64

x2

count    100000.000000
mean         -3.005565
std           2.000799
min         -12.498524
25%          -4.353844
50%          -3.002649
75%          -1.649283
max           6.089820
Name: x2, dtype: float64

median         -3.002649
skew           -0.000347
average        -3.005565
var             4.003195
sem             0.006327
sum       -300556.464753
Name: x2, dtype: float64

x3

count    100000.000000
mean          2.500593
std           1.752906
min          -3.489212
25%           1.189920
50%           2.503764
75%           3.802236
max           8.679097
Name: x3, dtype: float64

median          2.503764
skew           -0.003621
average         2.500593
var             3.072681
sem             0.005543
sum        250059.262904
Name: x3, dtype: float64
```

```
x5

count    100000.000000
mean          0.999136
std           1.002006
min           0.000003
25%           0.285629
50%           0.690903
75%           1.386862
max          15.102966
Name: x5, dtype: float64

median          0.690903
skew            2.039982
average         0.999136
var             1.004017
sem             0.003169
sum         99913.647275
Name: x5, dtype: float64

x6

count    100000.000000
mean          0.000647
std           4.663860
min         -13.885453
25%          -2.611943
50%          -0.000611
75%           2.621841
max          13.924740
Name: x6, dtype: float64

median     -0.000611
skew        0.003579
average     0.000647
var        21.751591
sem         0.014748
sum        64.684123
Name: x6, dtype: float64
```

In [8]:
```
def visualize(ds_1):
    targets_list = get_targets_list(ds_1)
    features_list = get_features_list(ds_1)
    fig, ax = plt.subplots(5, 3, figsize = (15, 10))
    fig.tight_layout()
    for i, feature in enumerate(features_list):
        for j, target in enumerate(targets_list):
            ax[i, j].plot(ds_1[feature], ds_1[target])
            ax[0, j].set_title(target)
            ax[i, 0].set(ylabel = feature)
    plt.show()
```

In [9]:
```
visualize(ds_1)
```



In [10]:
```
class Regression_Evaluation:
    def __init__(self, ds_1):
        self.ds_1 = ds_1

        self.model = LinearRegression()
        self.scoring = ['max_error', 'neg_mean_absolute_error',
               'neg_mean_squared_error', 'neg_root_mean_squared_error', 'r2']

        self.targets = get_targets_list(ds_1)
        self.features = get_features_list(ds_1)
        self.feature_permutations = self.get_feature_permutations(self.features

    def get_feature_permutations(self, features, subsets = []):
        for length in range(len(features) + 1):
            for subset in itertools.combinations(features, length):
                subsets.append(list(subset))
        return subsets[1:]

    def get_data(self, ds_1, features, target, test_size = 0.2):
        X = ds_1[features]
        y = ds_1[target]
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = t
        return (X_train, X_test, y_train, y_test)

    def print_score(self, score):
        for key, val in score.items():
            print(key, np.mean(val))

    def kfold(self, X_train, y_train, n_split = 5):
        kf = KFold(n_splits = n_split)
        scores = cross_validate(self.model, X_train, y_train,
                                scoring = self.scoring, cv = kf, return_train_s
        return scores
```

```
    def evaluate(self, to_print = False, eval_scores = []):
        feature_permutations = self.feature_permutations
        for target in self.targets:
            for feature_list in feature_permutations:
                X_train, x_test, y_train, y_test = self.get_data(self.ds_1, fea
                scores = self.kfold(X_train, y_train)
                if (to_print):
                    print('Target:', target)
                    print('Features:', feature_list)
                    self.print_score(scores, '\n')
                eval_scores.append({
                    'target': target,
                    'feature_list': feature_list,
                    'score': round(np.mean(scores['test_r2']), 7)
                })
        return pd.DataFrame(eval_scores)
```

In [11]:
```
scores_features_df = Regression_Evaluation(ds_1).evaluate(to_print = False)
scores_features_df.sort_values(['score'], ascending = False).groupby('target').
```

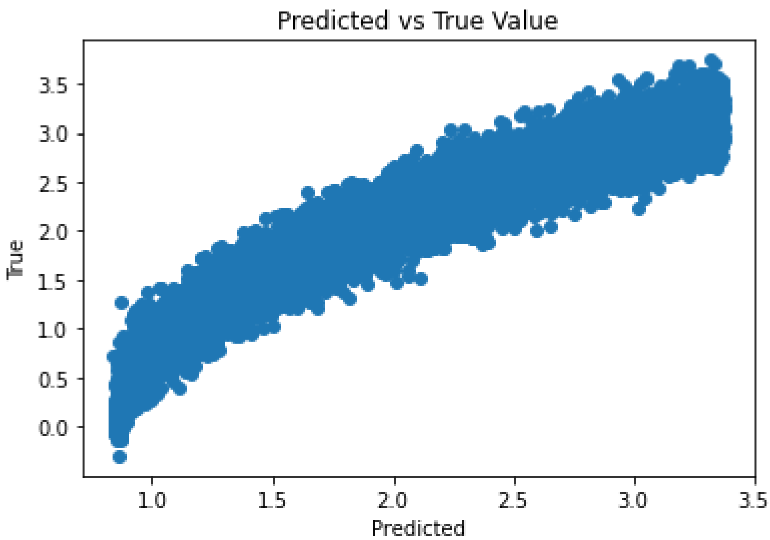
Out[11]:

	target	feature_list	score
37	yb	[x1, x3]	0.894749
31	yb	[x1]	0.894744
49	yb	[x1, x3, x5]	0.894744
15	ya	[x1, x2, x3]	0.676735
25	ya	[x1, x2, x3, x5]	0.676721
26	ya	[x1, x2, x3, x6]	0.676717
65	yc	[x5]	-0.000037
72	yc	[x2, x5]	-0.000046
63	yc	[x2]	-0.000047

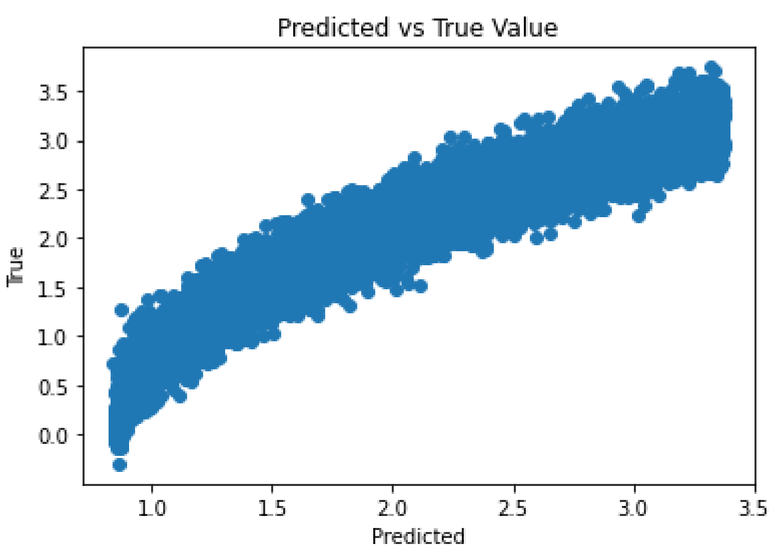
In [12]:
```
def print_equation(features, target, coef, intercept):
    temp = ''
    for i in range(len(features)):
        temp += str(round(coef[i], 4)) + '*' + features[i] + ' '
    return '{} = {} + {}'.format(target, temp[:-1], intercept)

def train(features, target, test_size = 0.1):
    X = ds_1[features]
    y = ds_1[target]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = test_
    model = LinearRegression()
    model.fit(X_train, y_train)
    print('Target: {}, Features: {}'.format(target, features))
    print('Model Score (R²):', np.round(model.score(X_test, y_test), 7))
    print('Model Coefficient:', model.coef_)
    print('Model Intercept:', model.intercept_)
    print('Equation:', print_equation(features, target, model.coef_, model.inte

    y_pred = model.predict(X_test)
    plt.scatter(y_pred, y_test)
```

```
    plt.title('Predicted vs True Value')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()
    print()

best_predictive_variables = scores_features_df.sort_values(['score'], ascending
for index, row in best_predictive_variables.iterrows():
    train(row['feature_list'], row['target'])
    train(get_features_list(ds_1), row['target'])
```

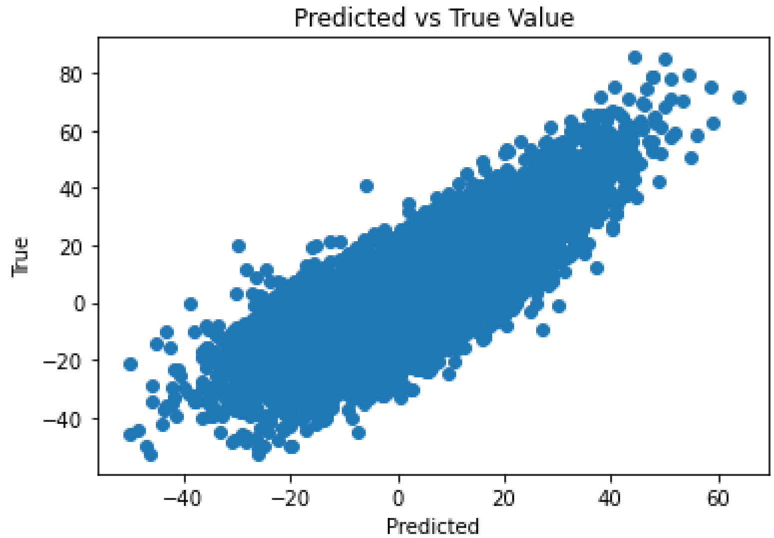
```
Target: yb, Features: ['x1', 'x3']
Model Score (R²): 0.8952979
Model Coefficient: [ 0.25431713 -0.00209127]
Model Intercept: 0.842373986968475
Equation: yb = 0.2543*x1 -0.0021*x3 + 0.842373986968475
```



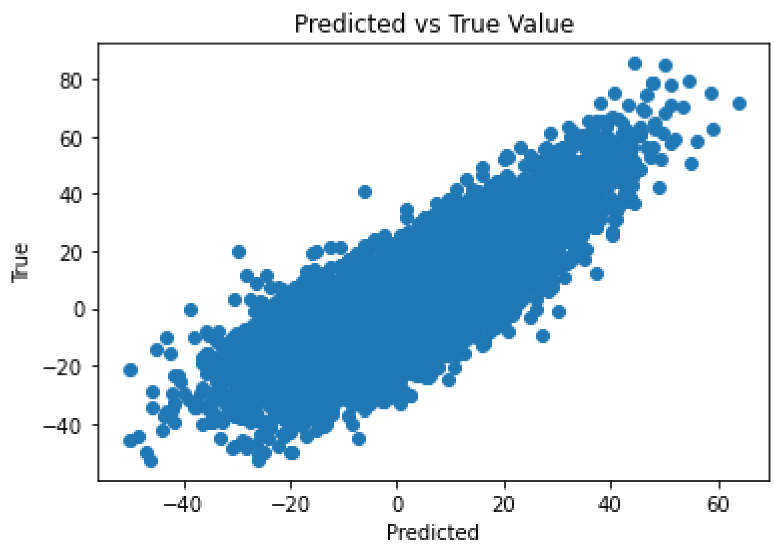

```
Target: yb, Features: ['x1', 'x2', 'x3', 'x5', 'x6']
Model Score (R²): 0.8952963
Model Coefficient: [ 2.54315379e-01  5.49181362e-05 -2.08919220e-03  4.1489970
9e-04
  8.24706281e-05]
Model Intercept: 0.8421275974510247
Equation: yb = 0.2543*x1 0.0001*x2 -0.0021*x3 0.0004*x5 0.0001*x6 + 0.84212759
74510247
```



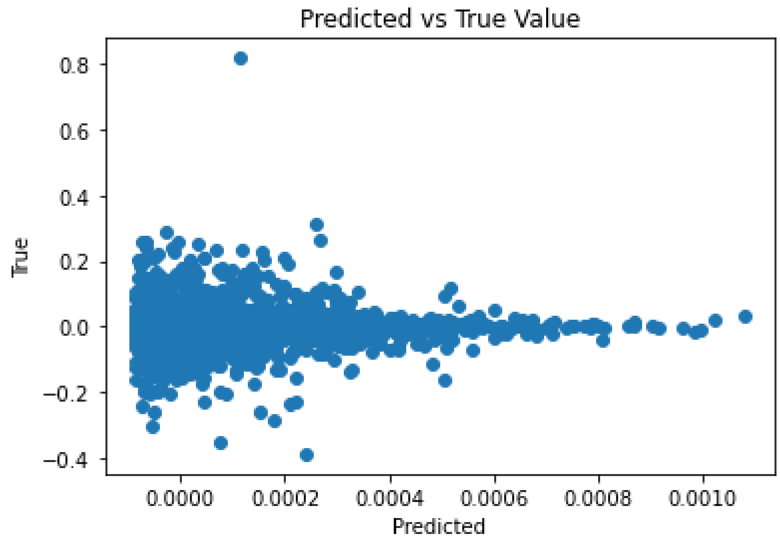
```
Target: ya, Features: ['x1', 'x2', 'x3']
Model Score (R²): 0.676793
Model Coefficient: [2.52373744 6.23396718 0.93839982]
Model Intercept: 7.555240400565005
Equation: ya = 2.5237*x1 6.234*x2 0.9384*x3 + 7.555240400565005
```



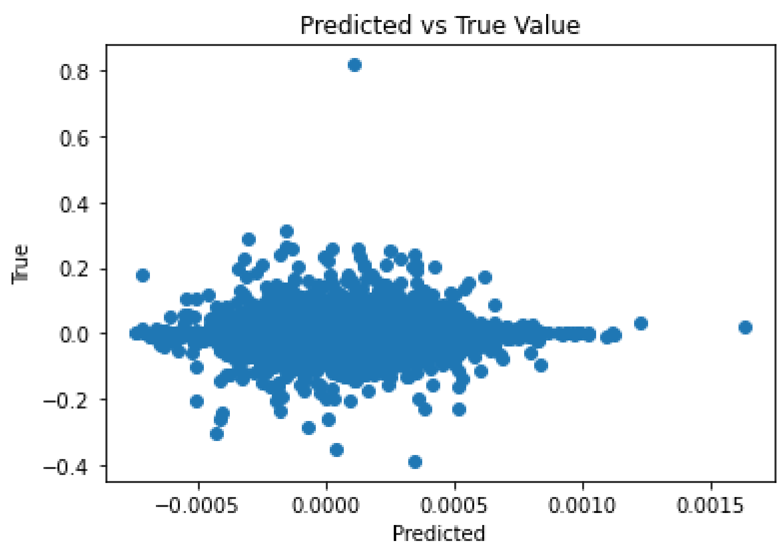

```
Target: ya, Features: ['x1', 'x2', 'x3', 'x5', 'x6']
Model Score (R²): 0.6767721
Model Coefficient: [2.52368961 6.23392286 0.93844665 0.00952342 0.00653299]
Model Intercept: 7.545714130783381
Equation: ya = 2.5237*x1 6.2339*x2 0.9384*x3 0.0095*x5 0.0065*x6 + 7.545714130
783381
```




```
Target: yc, Features: ['x5']
Model Score (R²): -0.0004263
Model Coefficient: [0.00013382]
Model Intercept: -8.692217632199213e-05
Equation: yc = 0.0001*x5 + -8.692217632199213e-05
```




```
Target: yc, Features: ['x1', 'x2', 'x3', 'x5', 'x6']
Model Score (R²): -0.0004205
Model Coefficient: [ 1.24223052e-04  1.09113732e-08 -1.95718882e-04  1.3257358
3e-04
 -8.50130073e-07]
Model Intercept: -0.00021882072471436637
Equation: yc = 0.0001*x1 0.0*x2 -0.0002*x3 0.0001*x5 -0.0*x6 + -0.000218820724
71436637
```



Question 2

a. Describe the data in a few sentences

b. How would you visualize this data set?

c. Can you identify the number of groups in the data and assign each row to its group?

d. Can you create a good visualization of your groupings?

a) I use similar techniques as the first part to describe the data. The results for each column are shown below.

b) This is a unsupervised dataset. There are 10 features. In order to visualize the data, we need to perform PCA on it to reduce the dimentionality to 2. Doing so gives us the graph that is shown below

c) Looking at the output from part b, we can see that there are 4 groups or clusters. But to verify it, I get the silhouette score of the data set. I normalize the data using standard scaler. Then I set a list of clusters, and plot the score of the data for each cluster. This indeed confirms that there are 4 clusters that the data can be grouped into.

Once I do that, I run KMeans Clustering on the data with the number of clusters set to 4, and then I print out the labels, which are the groups for each row.

d) The different coloured clusters and their centers are shown in the plot below

In [13]:
```
ds_2.agg(['median', 'skew', 'average', 'var', 'sem', 'sum'])
```

Out[13]:

	X1	X2	X3	X4	X5	X6
median	12.754335	11.896021	11.422441	-2.631318	2.484416	1.500838
skew	-0.810685	-0.106714	-0.590964	-0.004560	0.121531	-0.097249
average	8.677829	11.716801	9.252817	-2.679634	2.774942	0.077631
var	143.324648	44.293462	97.248049	112.894116	77.449728	236.405310
sem	0.267698	0.148818	0.220509	0.237586	0.196786	0.343806
sum	17355.657407	23433.602030	18505.634865	-5359.268003	5549.883133	155.261180

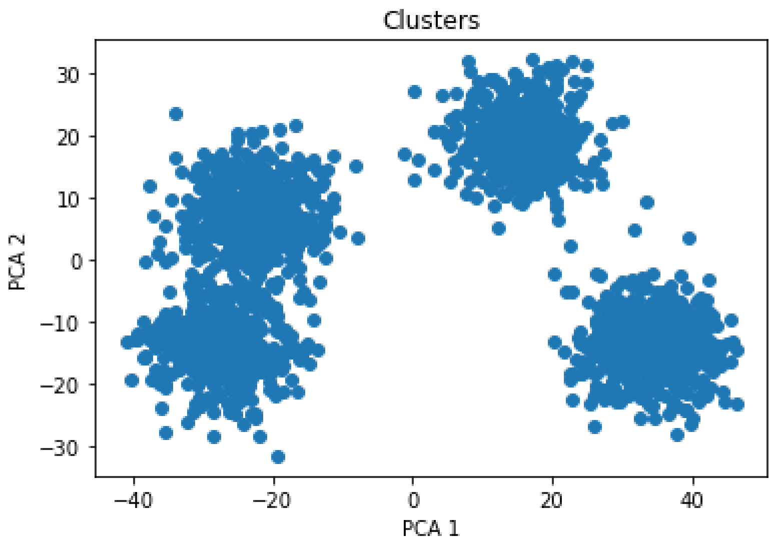
In [14]:
```
ds_2.describe()
```

Out[14]:

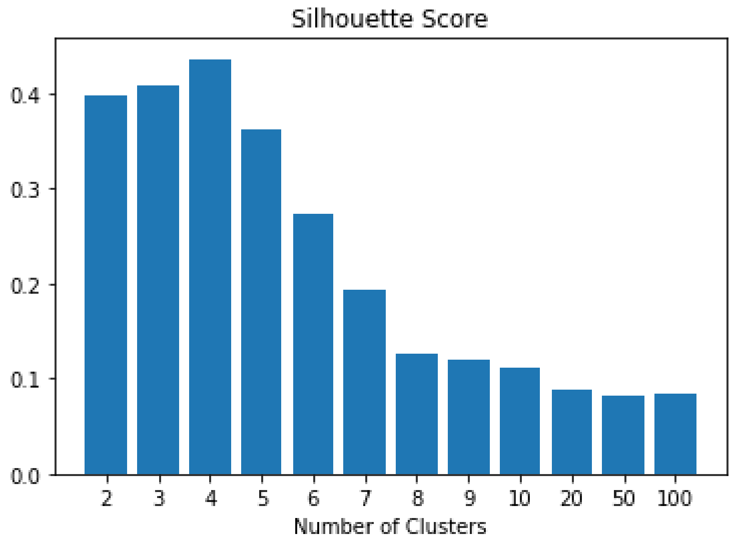
	X1	X2	X3	X4	X5	X6	
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	200
mean	8.677829	11.716801	9.252817	-2.679634	2.774942	0.077631	8
std	11.971827	6.655333	9.861443	10.625164	8.800553	15.375478	1
min	-25.824199	-8.497562	-23.666439	-29.429655	-22.033329	-35.264019	-2
25%	0.231327	7.161564	2.648845	-10.652694	-4.098043	-14.003670	-
50%	12.754335	11.896021	11.422441	-2.631318	2.484416	1.500838	8
75%	17.364337	16.279210	16.503676	5.340314	9.660898	14.050512	1
max	32.268570	32.909917	31.230550	26.422798	29.312010	31.727042	32

In [15]:
```
def visualize(ds_2, title = 'Clusters', n_components = 2):
    pca = PCA(n_components).fit_transform(ds_2)
    x = pca[:, 0]
    y = pca[:, 1]
    plt.scatter(x, y)
    plt.title(title)
    plt.xlabel('PCA 1')
    plt.ylabel('PCA 2')
```

```
    plt.show()
visualize(ds_2)
```



In [16]:
```
def kmean_silhouette(ds_2, to_print = False):
    best_score = -1
    scores = []
    clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 50, 100]

    ds_2_scaled = ds_2.copy()
    ds_2_scaled[ds_2_scaled.columns] = StandardScaler().fit_transform(ds_2_scal

    for cluster in clusters:
        kmeans = KMeans(n_clusters = cluster).fit(ds_2_scaled)
        score = silhouette_score(ds_2_scaled, kmeans.labels_)
        scores.append(score)

        if (to_print): print('Cluster: {}, Score: {}'.format(cluster, score))

        if (score > best_score):
            best_score = score
            best_cluster = cluster

    plt.bar(range(len(scores)), list(scores), align = 'center')
    plt.xticks(range(len(scores)), list(clusters))
    plt.title('Silhouette Score')
    plt.xlabel('Number of Clusters')
    plt.show()
    return best_cluster

num_clusters = kmean_silhouette(ds_2, to_print = False)
print('Number of groups:', num_clusters)
```

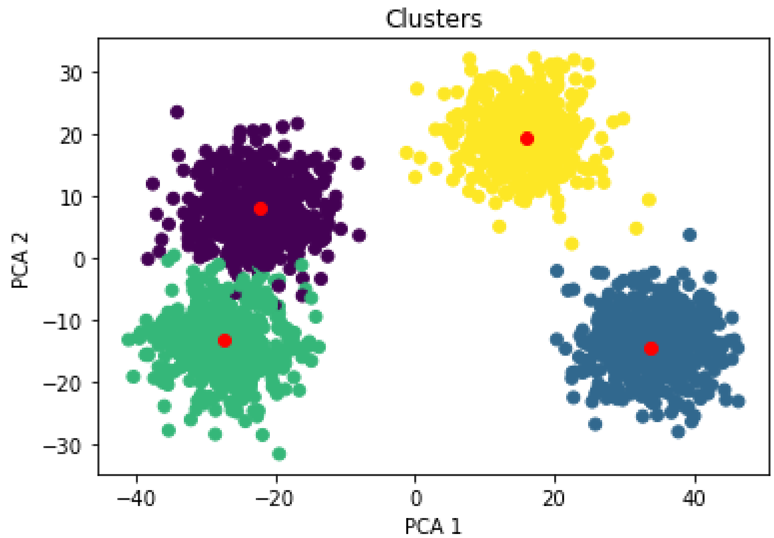



```
Number of groups: 4
```

In [17]:
```
def assign_kmeans(ds_2, num_clusters, n_components = 2):
    pca = PCA(n_components = n_components)
    kmeans = KMeans(n_clusters = num_clusters)
    kmeans.fit(ds_2)
    centers = kmeans.cluster_centers_
    centers_pca = pca.fit_transform(centers)
    return kmeans.labels_, centers_pca

labels, center_pca = assign_kmeans(ds_2, num_clusters)
print('Labels for each row:', labels)
```

```
Labels for each row: [0 3 3 ... 2 1 0]
```

In [18]:
```
def visualizing_groups(ds_2, labels, center_pca, n_components = 2):
    pca = PCA(n_components = n_components).fit_transform(ds_2)
    x = pca[:, 0]
    y = pca[:, 1]
    plt.scatter(x, y, c = labels)
    plt.title('Clusters')
    plt.xlabel('PCA 1')
    plt.ylabel('PCA 2')
    plt.scatter(center_pca[:, 0], center_pca[:, 1], marker = 'o', color = 'red'
    plt.show()

visualizing_groups(ds_2, labels, center_pca)
```



Question 3

Stack Overflow provides a tool at https://data.stackexchange.com/stackoverflow/query/new that allows SQL queries to be run against their data. After reviewing the database schema provided on their site, please answer the questions below by providing both your answer and the query used to derive it.

a. How many posts were created in 2017

In [19]:
```
sql = """SELECT COUNT(*) FROM Posts
         WHERE CreationDate BETWEEN '2017-01-01' AND '2017-12-31'
      """

# Note:
# usually you can use CreationDate::Date to get just the date value
# without the timestamp, but this server did not support it
```

Result: 5021226

b. What post/question received the most answers?

In [20]:
```
sql = """SELECT TOP(1) Id, AnswerCount FROM Posts
         ORDER BY AnswerCount DESC
      """

# OR

sql = """SELECT Id, AnswerCount FROM Posts
         ORDER BY AnswerCount DESC
         LIMIT 1
      """

# Note:
# LIMIT not supported in this server, only TOP
```

Result:
Id: 184618518

AnswerCount: 518

c. For posts created in 2020, what were the top 10 tags?

In [21]:
```
sql = """SELECT TOP(11) Tags, COUNT(*) FROM Posts
         WHERE CreationDate BETWEEN '2020-01-01' AND '2020-12-31'
         GROUP BY Tags
         ORDER BY COUNT(*) DESC
      """

# Note:
# I am showing TOP 11, becuse empty tag has the highest count, but not sure if
```

Result:

NO TAGS: 2442284

python: 16230

javascript: 12838

python, pandas: 9675

r: 8823

html, css: 7743

java: 7050

excel, vba: 6802

javascript, reactjs: 6717

c++: 6622

reactjs: 5463

d. *BONUS* For the questions created in 2017, what was the average time (in seconds) between when the question was created and when the accepted answer was provided?

In [22]:
```
sql = """
    SELECT
      AVG(
        CAST(
          DATEDIFF (SECOND, p.CreationDate, a.CreationDate)
        AS BIGINT)
       )
    FROM
      Posts p
    INNER JOIN
      Posts a
    ON
      p.AcceptedAnswerId = a.Id
    WHERE p.CreationDate BETWEEN '2017-01-01' AND '2017-12-31'
"""
```

Result: 838231

In [ ]: