# *featured selection engineering*

***the process of selecting, manipulating and transforming raw data into features that can be used in supervised learning. It's also necessary to design and train new machine learning features so it can tackle new tasks. A "feature" is any measurable input that can be used in a predictive model.***

In [11]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

In [12]:
```
df = pd.read_excel('D:\\Sales Report1.xls')
print(df)
```

```
            Product Customer    Qtr 1  Qtr 2   Qtr 3    Qtr 4
0      Alice Mutton    ANTON     0.00  702.0    0.00     0.00
1      Alice Mutton    BERGS   312.00    0.0    0.00     0.00
2      Alice Mutton    BOLID     0.00    0.0    0.00  1170.00
3      Alice Mutton    BOTTM  1170.00    0.0    0.00     0.00
4      Alice Mutton    ERNSH  1123.20    0.0    0.00  2607.15
..              ...      ...      ...    ...     ...      ...
272   Veggie-spread    FOLIG     0.00    0.0    0.00  1317.00
273   Veggie-spread    HUNGO   921.37    0.0    0.00     0.00
274   Veggie-spread    MORGK     0.00  263.4    0.00     0.00
275   Veggie-spread    PICCO     0.00    0.0    0.00   395.10
276   Veggie-spread    WHITC     0.00    0.0  842.88     0.00

[277 rows x 6 columns]
```

In [13]:
```
df.head(5)
```

Out[13]:

| | Product | Customer | Qtr 1 | Qtr 2 | Qtr 3 | Qtr 4 |
|---|---|---|---|---|---|---|
| **0** | Alice Mutton | ANTON | 0.0 | 702.0 | 0.0 | 0.00 |
| **1** | Alice Mutton | BERGS | 312.0 | 0.0 | 0.0 | 0.00 |
| **2** | Alice Mutton | BOLID | 0.0 | 0.0 | 0.0 | 1170.00 |
| **3** | Alice Mutton | BOTTM | 1170.0 | 0.0 | 0.0 | 0.00 |
| **4** | Alice Mutton | ERNSH | 1123.2 | 0.0 | 0.0 | 2607.15 |

In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 277 entries, 0 to 276
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Product   277 non-null    object 
 1   Customer  277 non-null    object 
 2   Qtr 1     277 non-null    float64
 3   Qtr 2     277 non-null    float64
 4   Qtr 3     277 non-null    float64
 5   Qtr 4     277 non-null    float64
dtypes: float64(4), object(2)
memory usage: 10.9+ KB
```

In [15]:
```
df.describe()
```

Out[15]:

| | Qtr 1 | Qtr 2 | Qtr 3 | Qtr 4 |
|---|---|---|---|---|
| **count** | 277.000000 | 277.000000 | 277.000000 | 277.000000 |
| **mean** | 88.855271 | 156.805199 | 150.327581 | 161.744621 |
| **std** | 254.991062 | 389.259507 | 433.856409 | 386.840078 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **50%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **75%** | 0.000000 | 96.500000 | 0.000000 | 110.400000 |
| **max** | 2281.500000 | 3159.000000 | 3900.000000 | 2700.000000 |

**convert into lower variance**

In [19]:
```python
import numpy as np
from sklearn.feature_selection import VarianceThreshold

# Just make a convenience function; this one wraps the VarianceThreshold
# transformer but you can pass it a pandas dataframe and get one in return

def get_low_variance_columns(dframe=None, columns=None,
                             skip_columns=None, thresh=0.0,
                             autoremove=False):
    """
    Wrapper for sklearn VarianceThreshold for use on pandas dataframes.
    """
    print("Finding low-variance features.")
    try:
        # get list of all the original df columns
        all_columns = dframe.columns

        # remove `skip_columns`
        remaining_columns = all_columns.drop(skip_columns)

        # get length of new index
        max_index = len(remaining_columns) - 1

        # get indices for `skip_columns`
        skipped_idx = [all_columns.get_loc(column)
                       for column
                       in skip_columns]

        # adjust insert location by the number of columns removed
        # (for non-zero insertion locations) to keep relative
        # locations intact
        for idx, item in enumerate(skipped_idx):
            if item > max_index:
                diff = item - max_index
                skipped_idx[idx] -= diff
            if item == max_index:
                diff = item - len(skip_columns)
                skipped_idx[idx] -= diff
            if idx == 0:
                skipped_idx[idx] = item

        # get values of `skip_columns`
        skipped_values = dframe.iloc[:, skipped_idx].values

        # get dataframe values
        X = dframe.loc[:, remaining_columns].values

        # instantiate VarianceThreshold object
        vt = VarianceThreshold(threshold=thresh)

        # fit vt to data
        vt.fit(X)

        # get the indices of the features that are being kept
        feature_indices = vt.get_support(indices=True)

        # remove low-variance columns from index
```

```
        feature_names = [remaining_columns[idx]
                         for idx, _
                         in enumerate(remaining_columns)
                         if idx
                         in feature_indices]

        # get the columns to be removed
        removed_features = list(np.setdiff1d(remaining_columns,
                                             feature_names))
        print("Found {0} low-variance columns."
              .format(len(removed_features)))

        # remove the columns
        if autoremove:
            print("Removing low-variance features.")
            # remove the low-variance columns
            X_removed = vt.transform(X)

            print("Reassembling the dataframe (with low-variance "
                  "features removed).")
            # re-assemble the dataframe
            dframe = pd.DataFrame(data=X_removed,
                                  columns=feature_names)

            # add back the `skip_columns`
            for idx, index in enumerate(skipped_idx):
                dframe.insert(loc=index,
                              column=skip_columns[idx],
                              value=skipped_values[:, idx])
            print("Succesfully removed low-variance columns.")

        # do not remove columns
        else:
            print("No changes have been made to the dataframe.")

    except Exception as e:
        print(e)
        print("Could not remove low-variance features. Something "
              "went wrong.")
        pass

    return dframe, removed_features
```

In [23]:
```python
from sklearn.feature_selection import VarianceThreshold
from itertools import compress

def fs_variance(df, threshold:float=0.1):
    """
    Return a list of selected variables based on the threshold.
    """

    # The list of columns in the data frame
    features = list(df.columns)

    # Initialize and fit the method
    vt = VarianceThreshold(threshold = threshold)
    _ = vt.fit(df)

    # Get which column names which pass the threshold
    feat_select = list(compress(features, vt.get_support()))

    return feat_select
```

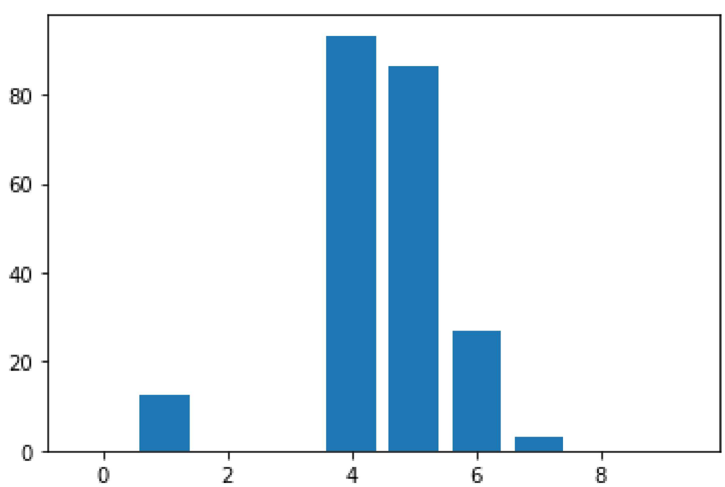
### feature importance attribute

In [25]:
```python
# test classification dataset
from sklearn.datasets import make_classification
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_re
# summarize the dataset
print(X.shape, y.shape)
```

```
(1000, 10) (1000,)
```

In [26]:
```python
# linear regression feature importance
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, random_s
# define the model
model = LinearRegression()
# fit the model
model.fit(X, y)
# get importance
importance = model.coef_
# summarize feature importance
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

```
Feature: 0, Score: -0.00000
Feature: 1, Score: 12.44483
Feature: 2, Score: 0.00000
Feature: 3, Score: -0.00000
Feature: 4, Score: 93.32225
Feature: 5, Score: 86.50811
Feature: 6, Score: 26.74607
Feature: 7, Score: 3.28535
Feature: 8, Score: 0.00000
Feature: 9, Score: 0.00000
```



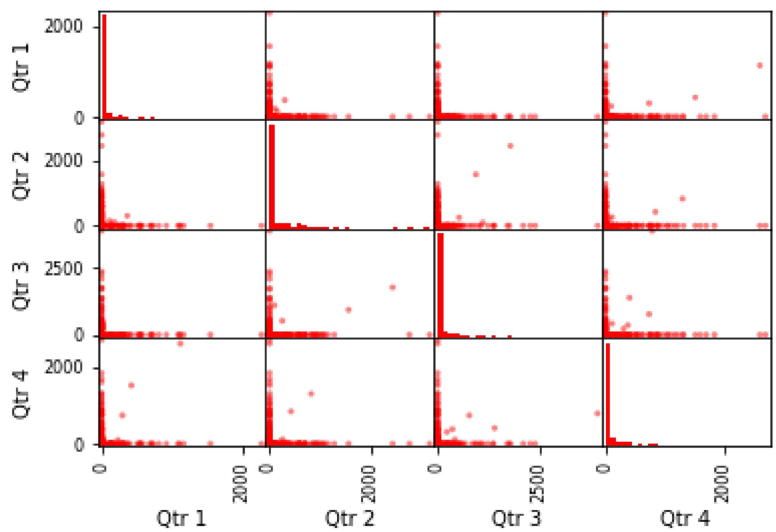

In [31]:
```python
Corr_Matrix = round(df.corr(),2)
print(Corr_Matrix)
```

```
       Qtr 1  Qtr 2  Qtr 3  Qtr 4
Qtr 1   1.00  -0.14  -0.12  -0.01
Qtr 2  -0.14   1.00  -0.01  -0.13
Qtr 3  -0.12  -0.01   1.00  -0.05
Qtr 4  -0.01  -0.13  -0.05   1.00
```

### feature selection

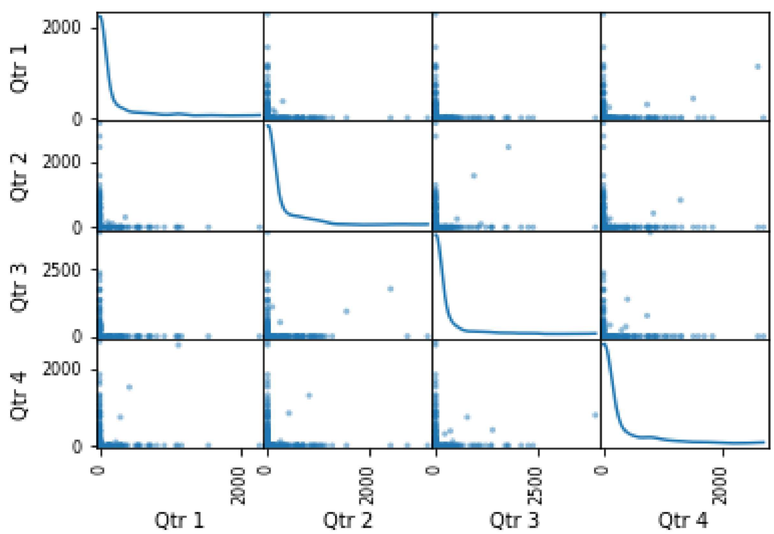
In [32]:
```
pd.plotting.scatter_matrix(df, color='red', hist_kwds={'bins':30, 'color':'red'
```

Out[32]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x057DD130>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x097E5700>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09804118>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09819AF0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0983A4F0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0984D730>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0984DEF8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0986E928>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x098A1CB8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x098C46B8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x098E30D0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x098F6AA8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x099174A8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0992CE80>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0994D8B0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0996D2C8>]],
      dtype=object)
```



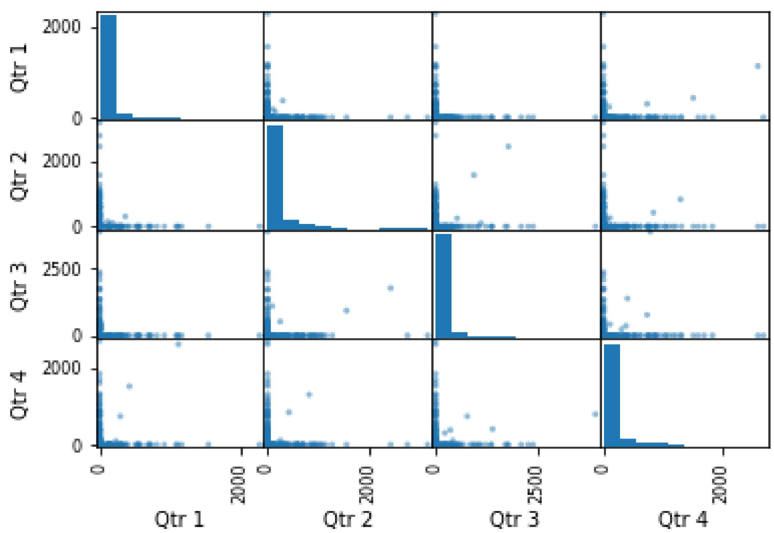
In [33]:
```
pd.plotting.scatter_matrix(df, diagonal='kde')
```

Out[33]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x09A7D658>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0983AEB0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0990A970>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09939DA8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x093343A0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x057D5C28>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x057D5088>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x056F7280>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x09B4A910>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09B6B310>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09B7DCE8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09B9F6E8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x09BC30E8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09BD5AC0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09BF74C0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09C09EC8>]],
      dtype=object)
```



In [35]:
```
pd.plotting.scatter_matrix(df)
```

Out[35]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x09CA76E8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09D02370>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09D15D48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09D35748>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x09D57148>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09D6A370>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09D6AB38>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09D8B568>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x09DBF8F8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09DDF310>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09DF3CE8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09E136E8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x09E350E8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09E4A850>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09E65BE0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x09E7FF70>]],
      dtype=object)
```




## Done by:

## K.K.Sreevalli