Multiple Linear Regression

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
```

In [2]:
```
# 1. Import data
df = pd.read_csv("baseball.csv")
df.columns = ['x1','x2','x3','x4','x5','x6']
print(df.describe())
```

```
              x1         x2         x3         x4         x5         x6
count  45.000000  45.000000  45.000000  45.000000  45.000000  45.000000
mean    0.280467   0.150889   0.046356   0.011289   0.024267   0.104333
std     0.044002   0.042009   0.010452   0.006960   0.022260   0.063057
min     0.188000   0.064000   0.025000   0.001000   0.000000   0.000000
25%     0.240000   0.119000   0.039000   0.007000   0.009000   0.062000
50%     0.290000   0.150000   0.045000   0.009000   0.013000   0.095000
75%     0.308000   0.189000   0.053000   0.016000   0.039000   0.138000
max     0.367000   0.259000   0.068000   0.030000   0.085000   0.264000
```

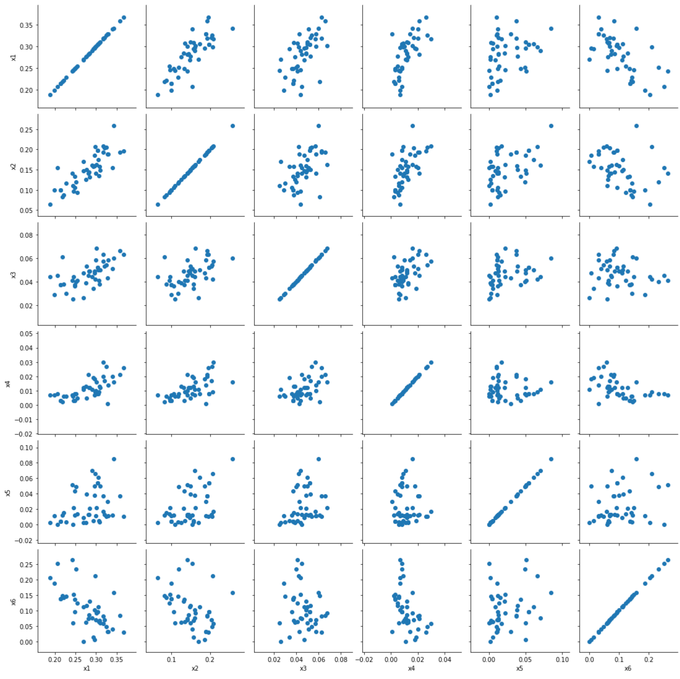
In [5]:
```
df.shape
```

Out[5]:
```
(45, 6)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45 entries, 0 to 44
Data columns (total 6 columns):
x1    45 non-null float64
x2    45 non-null float64
x3    45 non-null float64
x4    45 non-null float64
x5    45 non-null float64
x6    45 non-null float64
dtypes: float64(6)
memory usage: 2.2 KB
```

In [3]:
```
# 2.Observe the correlation of dependent varibles to the independent varible and
# independent variables to each other to avoid multi-collinearity
cor = sns.PairGrid(df)
cor.map(plt.scatter);
plt.show()
```



In [4]:
```
#3. Prepare data
# Remove the dependent variable and build an input dataset
X = df.iloc[:,df.columns != 'x1']
#Create a df of dependent variable by itself
Y = df.iloc[:, 0]
#Split the data into train and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, \
                                                    random_state= 0)
```

In [7]:
```
#4. Create the model
# Call the model function
model = linear_model.LinearRegression()

# Fit the data to model
model.fit(X_train, Y_train)


# Build the dataframe with coeffcients
coeff_df = pd.DataFrame(model.coef_, X.columns, columns=['Coef'])
print(coeff_df)
```

```
        Coef
x2  0.518154
x3  1.027110
x4  0.518150
x5  0.175801
x6 -0.248293
```

In [ ]:
```
x1 = .51 * x2 + 1.02 * x3 + .5 * x4 + .17 * x5 + -0.24 * x6
```

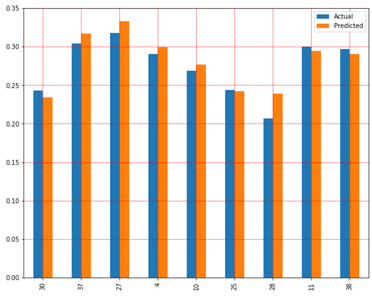
In [8]:
```
model.coef_
```

Out[8]:
```
array([ 0.51815391,  1.0271097 ,  0.51814958,  0.17580108, -0.24829316])
```

In [9]:
```
#5. Test the model
y_pred = model.predict(X_test)
df = pd.DataFrame({'Actual': Y_test, 'Predicted': y_pred})
print(df.head(10))
```

```
    Actual  Predicted
30   0.243   0.233866
37   0.304   0.317188
27   0.318   0.333119
4    0.290   0.299402
10   0.269   0.276333
25   0.244   0.242289
28   0.207   0.239242
11   0.300   0.294018
38   0.297   0.290114
```

In [10]:
```
#6. Observe the performance of the model
df.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
#plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



In [11]:
```
#6. Observe the performance of the model
# Root Mean Squared Error or Deviation

# Lower the rmse(rmsd) is, the better the fit
rmsd = np.sqrt(mean_squared_error(Y_test, y_pred))

# The closer towards 1, the better the fit
r2_value = r2_score(Y_test, y_pred)

print("Y-Intercept: \n", model.intercept_)
print("Root Mean Square Error(rmsd) \n", rmsd)
print("R^2 Value: \n", r2_value)
```

```
Y-Intercept:
 0.17165100673312098
Root Mean Square Error(rmsd)
 0.013959349401614382
R^2 Value:
 0.8356045255486346
```

In [ ]:
```
#7 communicate the results
```