Machine Learning Lab - Exercise Sheet 2

Author: Sebastian Pineda Arango
ID: 246098
Universität Hildesheim - Data Analytics Master

Exercise 1

We want to analyze in tihs lab a data set which content information about gas stations and gas prices.

Firstly, we want to load the data and take a look to it.

In [45]:
```
#Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#Reading data
data = pd.read_csv("GasPrices.csv")

#Printing top dataset
data.head()
```

Out[45]:

	Unnamed: 0	ID	Name	Price	Pumps	Interior	Restaurant	CarWash	Highway	Intersect
0	1	1	Shell	1.79	4	Y	N	N	N	Y
1	2	2	Valero	1.83	4	Y	N	N	N	Y
2	3	3	7-Eleven	1.88	4	Y	N	N	N	Y
3	4	4	Texaco	1.88	4	Y	N	Y	N	Y
4	5	5	Shell	1.84	6	Y	N	N	N	Y

In [3]:
```
#eliminating the first column, since it is not important
data = data.drop('Unnamed: 0',axis=1) #eliminating column
```

Now we want to describe the daa torugh statistical information and boxplot graphs.

In [47]:
```
#summarising important statistical information
data.describe()
```

Out[47]:

	Unnamed: 0	ID	Price	Pumps	Gasolines	Zipcode	
count	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	101.000
mean	51.000000	51.000000	1.864257	6.950495	3.465347	78730.782178	56727.2
std	29.300171	29.300171	0.081515	3.925242	0.557931	22.054298	25868.3
min	1.000000	1.000000	1.730000	2.000000	1.000000	78701.000000	12786.0
25%	26.000000	26.000000	1.790000	4.000000	3.000000	78704.000000	37690.0
50%	51.000000	51.000000	1.850000	6.000000	3.000000	78731.000000	52306.0
75%	76.000000	76.000000	1.920000	8.000000	4.000000	78752.000000	70095.0
max	101.000000	101.000000	2.090000	24.000000	4.000000	78759.000000	128556

In [48]:
```
#grouping by the name of the gas station
data_grouped = data.groupby('Name')
```

In [49]:
```
#finding the average price, average income and average number of pumps for each
 group
data_mean = data_grouped['Name', 'Income', 'Price', 'Pumps'].mean()
data_mean
```

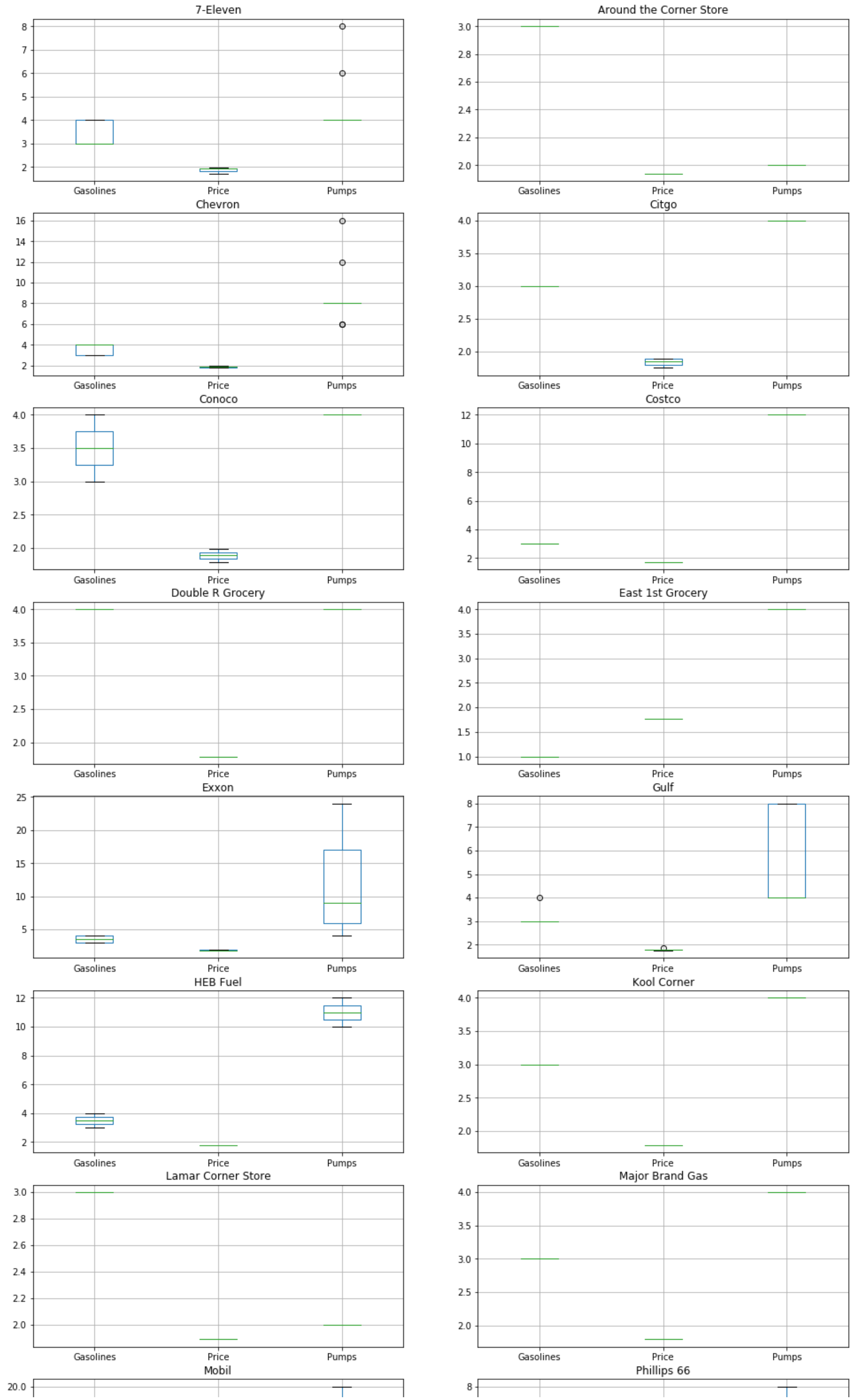
Out[49]:

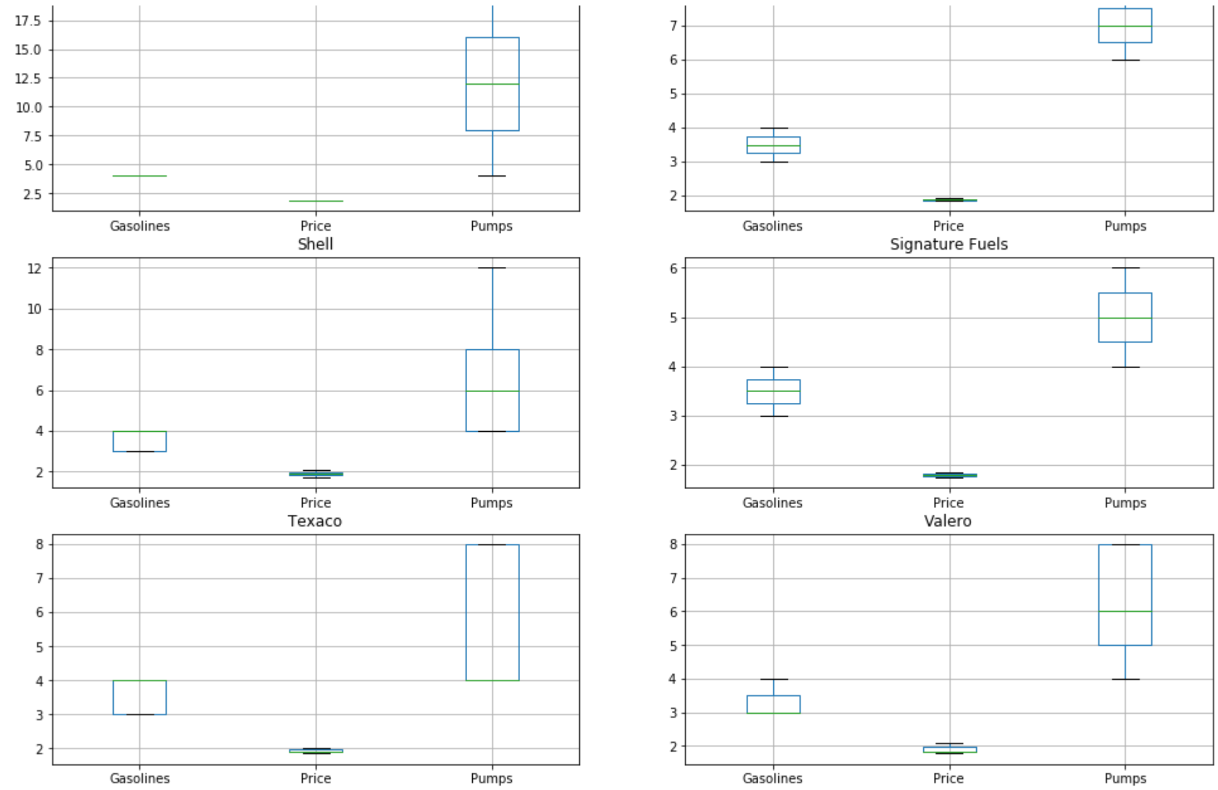
	Income	Price	Pumps
Name			
7-Eleven	53432.333333	1.887778	4.666667
Around the Corner Store	63750.000000	1.940000	2.000000
Chevron	61754.636364	1.871818	8.727273
Citgo	49387.000000	1.835000	4.000000
Conoco	43545.500000	1.890000	4.000000
Costco	70095.000000	1.730000	12.000000
Double R Grocery	37690.000000	1.790000	4.000000
East 1st Grocery	37690.000000	1.770000	4.000000
Exxon	52344.333333	1.855000	11.500000
Gulf	50084.142857	1.788571	5.714286
HEB Fuel	36903.500000	1.790000	11.000000
Kool Corner	42615.000000	1.790000	4.000000
Lamar Corner Store	37396.000000	1.890000	2.000000
Major Brand Gas	60856.000000	1.790000	4.000000
Mobil	47460.500000	1.865000	12.000000
Phillips 66	59796.500000	1.890000	7.000000
Shell	62972.793103	1.883793	6.482759
Signature Fuels	61200.500000	1.795000	5.000000
Texaco	75105.800000	1.912000	5.600000
Valero	49049.000000	1.891429	6.285714

In [50]:
```
%matplotlib inline
#Plotting the distribution of gasolines, price and pumps for all the station gr
oups
fig, ax = plt.subplots(10,2,figsize=(16, 40))
data[['Name','Gasolines', 'Price', 'Pumps']].groupby('Name').boxplot(ax=ax)
```

```
C:\Users\User\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:286
2: UserWarning: When passing multiple axes, sharex and sharey are ignored. Thes
e settings must be specified when creating axes
  exec(code_obj, self.user_global_ns, self.user_ns)
```

```
Out[50]: 7-Eleven                        AxesSubplot(0.1,0.836441;0.363636x0.0635593)
         Around the Corner Store    AxesSubplot(0.536364,0.836441;0.363636x0.0635593)
         Chevron                         AxesSubplot(0.1,0.760169;0.363636x0.0635593)
         Citgo                      AxesSubplot(0.536364,0.760169;0.363636x0.0635593)
         Conoco                          AxesSubplot(0.1,0.683898;0.363636x0.0635593)
         Costco                     AxesSubplot(0.536364,0.683898;0.363636x0.0635593)
         Double R Grocery                AxesSubplot(0.1,0.607627;0.363636x0.0635593)
         East 1st Grocery           AxesSubplot(0.536364,0.607627;0.363636x0.0635593)
         Exxon                           AxesSubplot(0.1,0.531356;0.363636x0.0635593)
         Gulf                       AxesSubplot(0.536364,0.531356;0.363636x0.0635593)
         HEB Fuel                        AxesSubplot(0.1,0.455085;0.363636x0.0635593)
         Kool Corner                AxesSubplot(0.536364,0.455085;0.363636x0.0635593)
         Lamar Corner Store              AxesSubplot(0.1,0.378814;0.363636x0.0635593)
         Major Brand Gas            AxesSubplot(0.536364,0.378814;0.363636x0.0635593)
         Mobil                           AxesSubplot(0.1,0.302542;0.363636x0.0635593)
         Phillips 66                AxesSubplot(0.536364,0.302542;0.363636x0.0635593)
         Shell                           AxesSubplot(0.1,0.226271;0.363636x0.0635593)
         Signature Fuels            AxesSubplot(0.536364,0.226271;0.363636x0.0635593)
         Texaco                              AxesSubplot(0.1,0.15;0.363636x0.0635593)
         Valero                         AxesSubplot(0.536364,0.15;0.363636x0.0635593)
         dtype: object
```


7-Eleven
Gasolines Price Pumps
Around the Corner Store
Gasolines Price Pumps
Chevron
Gasolines Price Pumps
Citgo
Gasolines Price Pumps
Conoco
Gasolines Price Pumps
Costco
Gasolines Price Pumps
Double R Grocery
Gasolines Price Pumps
East 1st Grocery
Gasolines Price Pumps
Exxon
Gasolines Price Pumps
Gulf
Gasolines Price Pumps
HEB Fuel
Gasolines Price Pumps
Kool Corner
Gasolines Price Pumps
Lamar Corner Store
Gasolines Price Pumps
Major Brand Gas
Gasolines Price Pumps
Mobil
Phillips 66



After the boxplot, we can have some insights as for example:

- The price for the same gas stations grouped by named remains very concentrated.
- The number of pumps for each gas station has, in general, considerable variability.
- The variable "gasolines" which represent how many types of gasolines are offered has relatively low variability.

Now we calculate the parameters for a linear regression using the normal equations. We plot the results of the predicted line vs. true data. Then, we make the same plot normalizing the data using the following equation:

$$x_{norm} = \frac{(x - x_{min})}{(x_{max} - x_{min})}$$

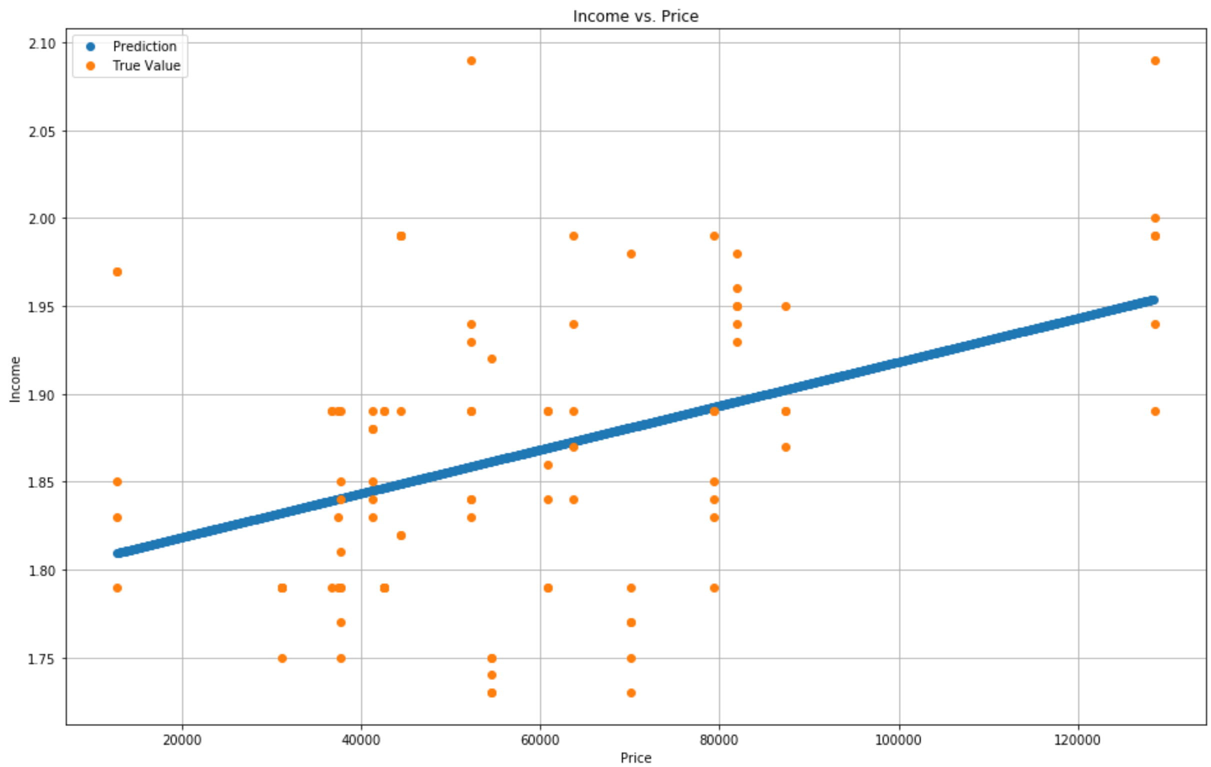
Where x_{min} and x_{max} are the minimum and maximum value of the x vector.

In the linear regression we fit, we use the *Income* as our features (*X*) and *Price* as our target (*y*).

In [81]:
```
##LEARN-LINREG-NORMEQ
##Defining important functions to train a linear regression and make prediction
s
def learn_linreg_normeq(x,y):

    '''This function takes a two columns matrix A and usse the first column as
 predictor and the second one as target
    to fit a basi linear regression model. The output es then the parameter vec
tor (beta) which better fits the regression.'''

    #Separating columns

    #Adding column of ones
    x = np.hstack((x, np.ones(np.shape(x))))

    #Converting to matrix data type, so that it is easy to operate
    x = np.matrix(x)
    y = np.matrix(y)

    #applying the mathematical solution
    beta = (np.linalg.inv(x.T*x))*x.T*y

    return beta

def predict_simple_linreg(beta,x):

    '''This function recieves to parameters: beta and x, to calculate the predi
ctions of a basic linear regression model.'''

    #Organizing data to be of size = NX1
    x = np.reshape(x, (-1,1))

    #Adding new column
    x = np.hstack((x, np.ones(np.shape(x))))

    #Casting data
    x = np.matrix(x)
    beta = np.matrix(beta)

    #Applying matrix multiplication
    y_pred = x*beta

    return y_pred

#getting our features or predictive variable
x = np.array(data[['Income']])

#getting our target
y = np.array(data[['Price']])

#calculating parameters through normal equations
beta = learn_linreg_normeq(x, y)

#creating test data to draw the prediction line
x_test = np.arange(min(x), max(x), (max(x)-min(x))/1000)

#making predictions over test data
y_test = predict_simple_linreg(beta, x_test)
```

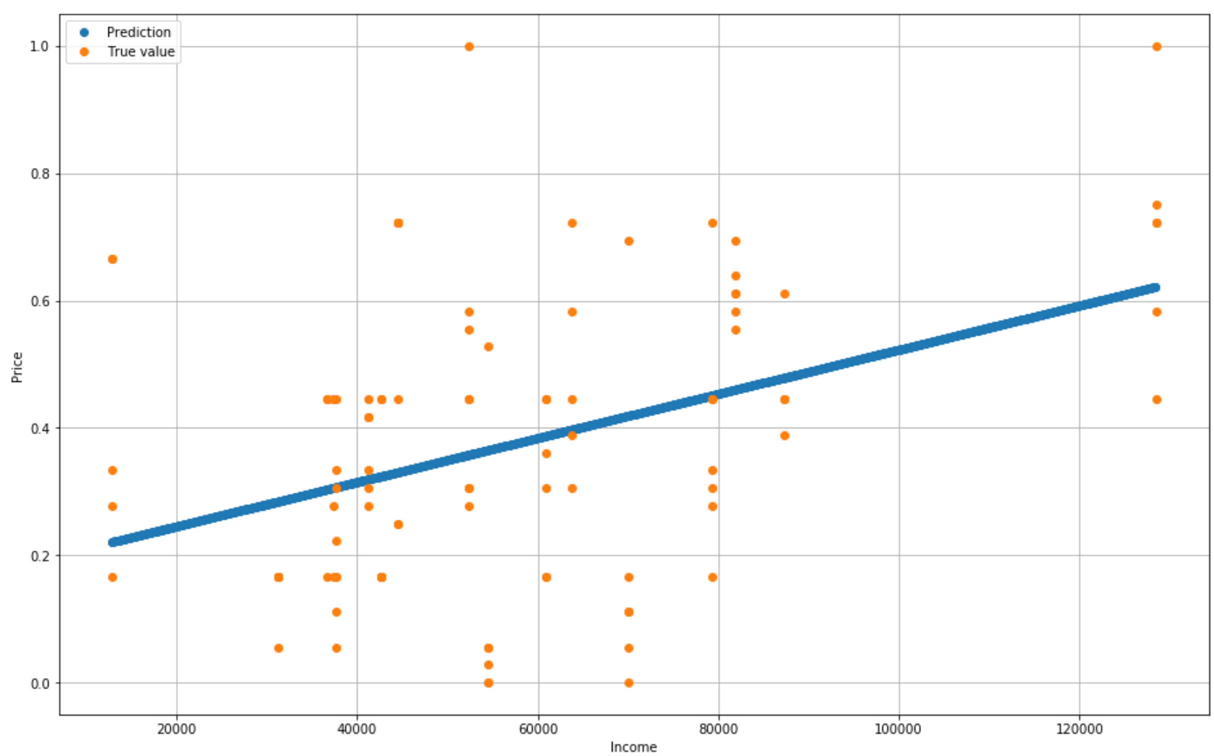
```
#creating the figure which compares predictions with true values
fig2, ax2 = plt.subplots(figsize=(16, 10))

ax2.plot(x_test, y_test, 'o')
ax2.plot(x,y,'o')
ax2.grid()
ax2.legend(("Prediction", "True Value"))
plt.xlabel('Price')
plt.ylabel('Income')
plt.title("Income vs. Price")
```

Out[81]: Text(0.5,1,'Income vs. Price')



In [82]:
```
#defining function to normalize
def normalize (x):
    x = (x-min(x))/(max(x)-min(x))
    return x

#Normalizing income
y_normalized = normalize(y)

#calculating parameters using the normalized target
beta = learn_linreg_normeq(x, y_normalized)

#generating test data and predicting over this to plot
#a prediction line
x_test = np.arange(min(x), max(x), (max(x)-min(x))/1000)
y_test = predict_simple_linreg(beta, x_test)

#Plotting
fig2, ax2 = plt.subplots(figsize=(16, 10))

ax2.plot(x_test, y_test, 'o')
ax2.plot(x,y_normalized,'o')
ax2.grid()
ax2.legend(("Prediction", "True value"))
plt.xlabel('Income')
plt.ylabel('Price')
```

Out[82]: `Text(0,0.5,'Price')`



After normalizing the input, we see that the prediction line and is reescaled, so that it is easier to compare the true and the predicted value. Before, we had very great numbers which made this taks very difficult.

Exercise 2

We explore the meaning of the variables. After a search on the web, we find in [1] out that the meaning of the variables is the following:

The variables in the data set are as follows:

- ID: Order in which gas stations were visited
- Name: Name of gas station
- Price: Price of regular unleaded gasoline, gathered on Sunday, April 3rd, 2016
- Pumps: How many pumps does the gas station have?
- Interior: Does the gas station have an interior convenience store?
- Restaurant: Is there a restaurant inside the gas station?
- CarWash: Does the gas station have a car wash attached?
- Highway: Is the gas station accessible from either a highway or a highway access road?
- Intersection: Is the gas station located at an intersection?
- Stoplight: Is there a stoplight in front of the gas station?
- IntersectionStoplight: three-way variable for if the gas station was at an intersection and/or a stoplight (None, Intersection (only), or Both).
- Gasolines: How many types of gasoline are offered? (Regular, midgrade, etc.)
- Competitors: Are there any gas stations in sight?
- Zipcode: Zip code in which gas station is located
- Address: Physical location of gas station
- Income: Median Household Income of the ZIP code where the gas station is located based on 2014 data from the U.S. Census Bureau
- Brand: is the gas station branded by one of the major oil companies (ExxonMobil, ChevronTexas, Shell) or not (Other)?

Now we read the data and take a look at the type of variables they have.

In [10]:
```
X_data = pd.read_csv("GasPrices.csv")

X_data = X_data.drop('Unnamed: 0',axis=1) #eliminating column

X_data.head()
```

Out[10]:

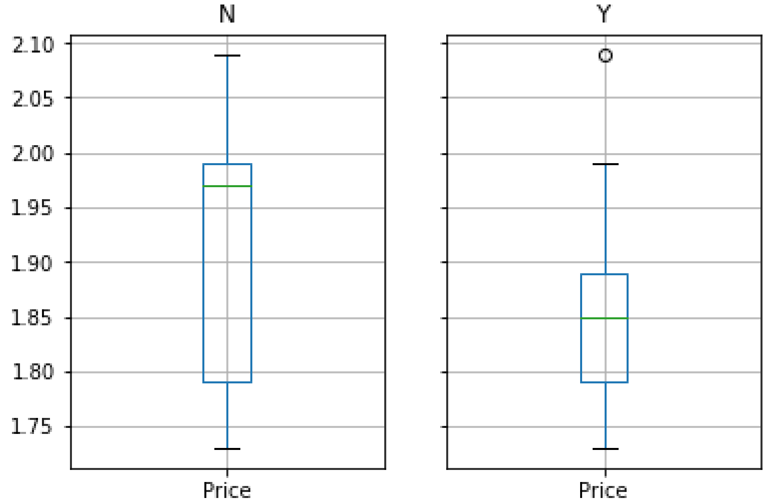
	ID	Name	Price	Pumps	Interior	Restaurant	CarWash	Highway	Intersection	Stoplig
0	1	Shell	1.79	4	Y	N	N	N	Y	N
1	2	Valero	1.83	4	Y	N	N	N	Y	N
2	3	7-Eleven	1.88	4	Y	N	N	N	Y	Y
3	4	Texaco	1.88	4	Y	N	Y	N	Y	Y
4	5	Shell	1.84	6	Y	N	N	N	Y	Y

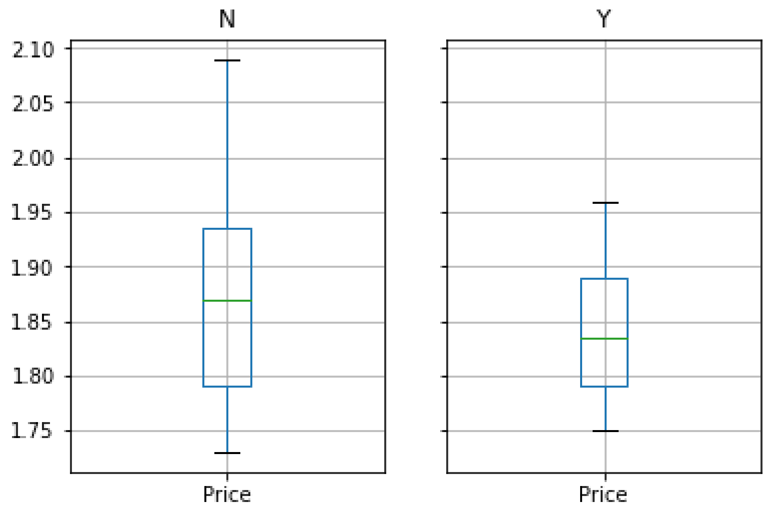
After looking at the data, we can notice the following:

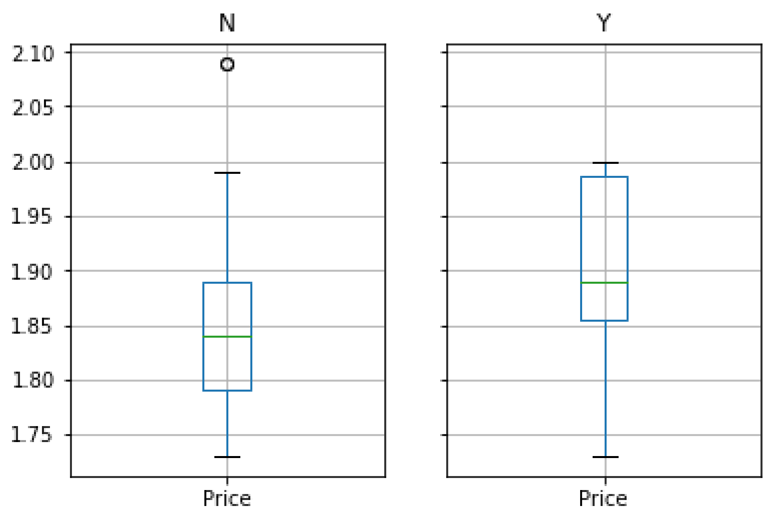
- ID is not a important variable since it is merely an index.
- Name of the gas station could be highly important but is a categorical variable with multiple categories and that would increase our preprocessing. However, we also noticed before, that other features like price and pumps are very correlated with the name of the gas station.
- Pumps, Gasolines, Income are numerical variables that we will use since their meaning could be related to the price of gas station (we hipothetize that more pumps, gasolines and income could mean a higher price, since it means a better service).
- There are binary variables (Interior, Restaurant, CarWash, Highway, Intersection, Spotlight, Competitors). We use boxplot to analyze the significance of the target distribution for each binary class.
- IntersectionSpotlight and Brand are multi-categorical variables that we discard to ease the preprocessing.
- Zipcode and Addreess are also discarded since their geographical meaning implies a further preprocessing step.

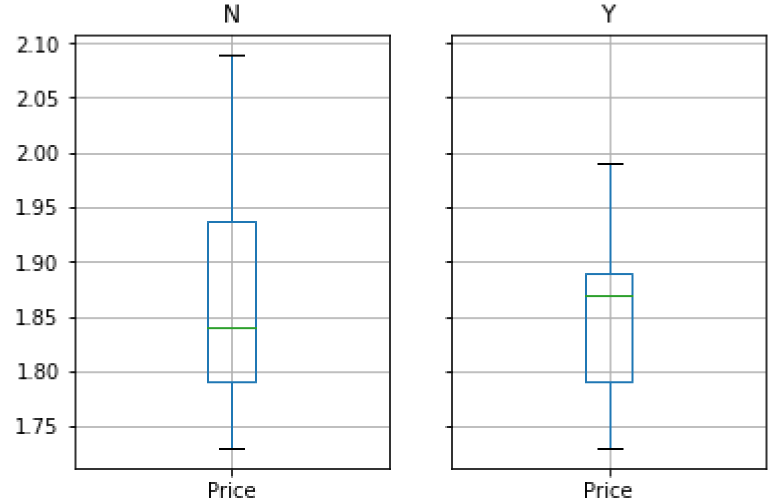
Now we make boxplot for the binary categorical variables to see which ones may be significative.

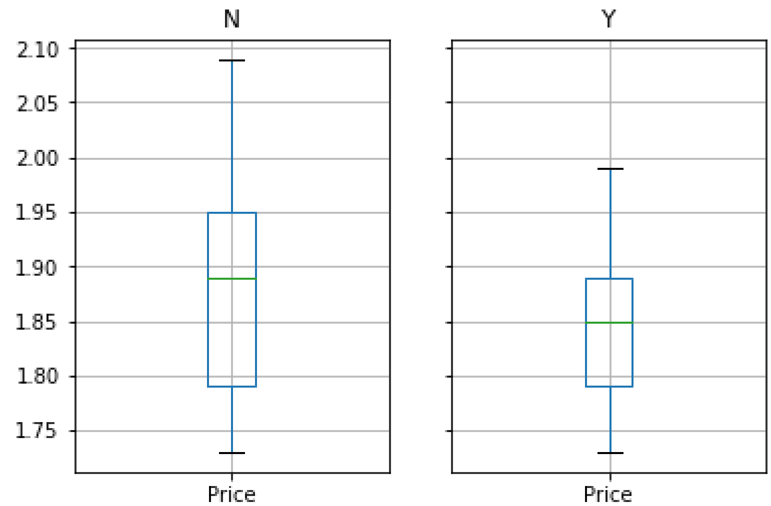
In [84]:
```
data[['Price','Interior']].groupby('Interior').boxplot()
data[['Price','CarWash']].groupby('CarWash').boxplot()
data[['Price','Highway']].groupby('Highway').boxplot()
data[['Price','Stoplight']].groupby('Stoplight').boxplot()
data[['Price','Competitors']].groupby('Competitors').boxplot()

features = ['Income', 'Pumps', 'Gasolines', 'Interior', 'Highway']
target = ['Price']
```


N
Y
2.10
2.05
2.00
1.95
1.90
1.85
1.80
1.75
Price
Price


N
Y
2.10
2.05
2.00
1.95
1.90
1.85
1.80
1.75
Price
Price


N
Y
2.10
2.05
2.00
1.95
1.90
1.85
1.80
1.75
Price
Price





After looking at it, we see that *Interior* and *Highway* have their means for both categorial values realtively well separated. Therefore, we consider that these two variables could be valuable to build the model.

Now we separate the data in train and test set, using 80% of the samples for training and the rest for testing.

In [103]:
```
#normalizing target
data[target] = normalize(np.array(data[target]))

#getting the total number of training samples
data_size = data.shape[0]
train_size = int(0.8*data_size)

#shuffling indexes to separate train and test randoming
idx = np.arange(0,101)
np.random.shuffle(idx)

#creating test indexes
train_idx = idx[:train_size]

#creating test indexes
test_idx = idx[train_size:]

#selecting train data (features)
X_train = data[features].iloc[train_idx,]

#selecting train data (target)
y_train = data[target].iloc[train_idx,]

#selecting test data (features)
X_test = data[features].iloc[test_idx,]

#selecting test data (target)
y_test = data[target].iloc[test_idx,]

def transformation(x):

    '''This function transformates the binary classes which
    have two possible classes (Y-N) to numeric.
    This classes are mapped to 1 and 0 respectively.'''

    x_tran = [1 if i == 'Y' else 0 for i in x]
    return x_tran

#Transforming variables 'Highway' and 'Interior' to numerical in train data
X_train_transf = X_train
X_train_transf[['Highway', 'Interior']] = X_train[['Highway', 'Interior']].appl
y(transformation)
X_train.head(10)

#Tansforming variables 'Highway' and 'Interior' to numerical in test data
X_test_transf = X_test
X_test_transf[['Highway', 'Interior']] = X_test[['Highway', 'Interior']].apply(
transformation)

#Generating X and y matrices for further processing
X = np.matrix(X_train_transf)
y = np.matrix(y_train)

#Setting a column of ones
X = np.hstack((np.ones((X.shape[0],1)), X))
```

We are going to use know a linear regression to predict the price (our target). For that we use the following equation:

$$Y = X\beta$$

And we also know that the values of beta can be found in a closed form using the norm equations:

$$X^T X\hat{\beta} = X^T Y$$

And we also can express this equation in the following way:

$$A\hat{\beta} = C$$

Where $A = X^T X$, $C = X^T Y$ and $\hat{\beta}$ is unknown.To solve this equations, we are going to use three different approaches:

- Gaussian elimination
- Cholesky decomposition
- QR decomposition

Gaussian Elimination

Gaussian elimination tries to convert A into a identity matrix applying row operations whereas the c vector is also affected by the rows operations.

The possible row operations are [2]:

- Swapping two rows
- Multiplying a row by a nonzero number
- Adding a multiple of one row to another row

With gaussian eliminations we first create a upper triangular matrix and then we use *backward substitution* [5] to get the identitity matrix [3].

Backwards substitution finds iteratively the unknown values once we have got the upper triangular matrix, applying the following equation.

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{i,j} x_j}{a_{i,i}} \text{ for } i = n, n-1, \ldots, 1$$

This algorithm is based on ideas taken from [2] and [3].

In [104]:
```
#Gaussian Elimination

def gaussian_elimination(a,c):

    '''This function performs forward gaussian elimination.
    It should be completed with a backward substitution step
    to complete the backward elimination'''

    n = a.shape[0]

    for k in range(0,n-1):
        for i in range(k+1, n):
            factor = a[i,k]/a[k,k]
            for j in range(k, n):
                a[i,j]= a[i,j]-factor*a[k,j]#updating values for A
            c[i] = c[i] - factor*c[k]#updating values for c


    return a,c

def backward_substitution (R,B):

    '''This algorithm performs backwards substitution to transform
    an upper triangular matrix in a diagonal matrix.'''

    n = R.shape[0]
    beta = np.zeros((n,1))
    for m in reversed(range(n)):
        s=0
        for i in reversed(range(m,n)):
            s=s+R[m,i]*beta[i,0]
        beta[m,0]= (B[m,0]-s)/R[m,m]

    return beta

def norm_eq_gauss(X,y):

    '''This functions solves a X*beta=y matrix equation
    using normal equations and applying gauss elimination.'''

    A = X.T*X
    c = X.T*y
    a, c = gaussian_elimination(A,c)
    beta = backward_substitution (a,c)

    return beta

beta_gauss = norm_eq_gauss(X,y)
```

QR Factorization

With QR factorization we can express the above mentioned A matrix as:

$$A = QR$$

Where Q is an orthogonal matrix and R is an upper triangular matrix. Since Q is an orthogonal matix, it holds that:

$$Q^{-1} = Q^T$$

Therefore, the equation $A\hat{\beta} = C$ converts to $QR\hat{\beta} = C$

Operating: $Q^T QR\hat{\beta} = Q^T C$
$Q^{-1} QR\hat{\beta} = Q^T C$
$R\hat{\beta} = Q^T C$

Since R is a upper triangular matrix, we could solve the last equation for $\hat{\beta}$ using backwars substition. The function _norm_eq_QR_ implement this idea. The ideas for the algorithmic implementation for QR decomposition where taken from [4].

For QR decomposition of the matrix A, we first calculate matrix. For that, we calculate n vector (where n is the number of rows of the matrix), applying this equation:

$$u_k = a_k - \sum_{j=1}^{k-1} proj_{u_j} a_k$$
$$e_k = \frac{u_k}{||u_k||}$$

The Q matrix is composed by the e_k vector as column vectors.

We get R as:

$$R = Q^T * A$$

In [105]:
```
###QR Factorization

def projection(u, a):

    '''Calculate the projetion of vector u over vector a'''

    return ((u.T*a)/(u.T*u))[0,0]*u


def QR_factorization (A):

    ''' Performs QR factorization over A matrix '''

    list_u = [A[:,0]/np.linalg.norm(A[:,0])]

    for col in range(1,A.shape[1]):

        a = A[:,col]
        proj= 0
        for i in range(len(list_u)):
            proj = proj + projection(list_u[i],a)

        u = a - proj
        norm_u = np.linalg.norm(u)
        list_u.append(u/norm_u)

    Q = np.hstack(list_u)
    R = (Q.T*A)

    return Q,R



def norm_eq_QR(X,y):

    '''This function uses QR decomposition to solve a matrix
    equation of the form X*beta=y, where beta is unknown.'''

    A = X.T*X
    c = X.T*y
    Q,R = QR_factorization(A)
    B=Q.T*c
    beta = backward_substitution(R,B)

    return beta

beta_QR = norm_eq_QR(X,y)
```

Cholesky decomposition

With Choleksy decomposition, we factorize a matrix A as:

$$A = LL^T$$

where L is a lower triangular matrix.

Since we want to solve a linear system of the following way:

$$A\hat{\beta} = C$$

We have then:

$$LL^T\hat{\beta} = C$$

If we set:

$y = L^T\hat{\beta}$, we get:

$Ly = C$. With y unknown.

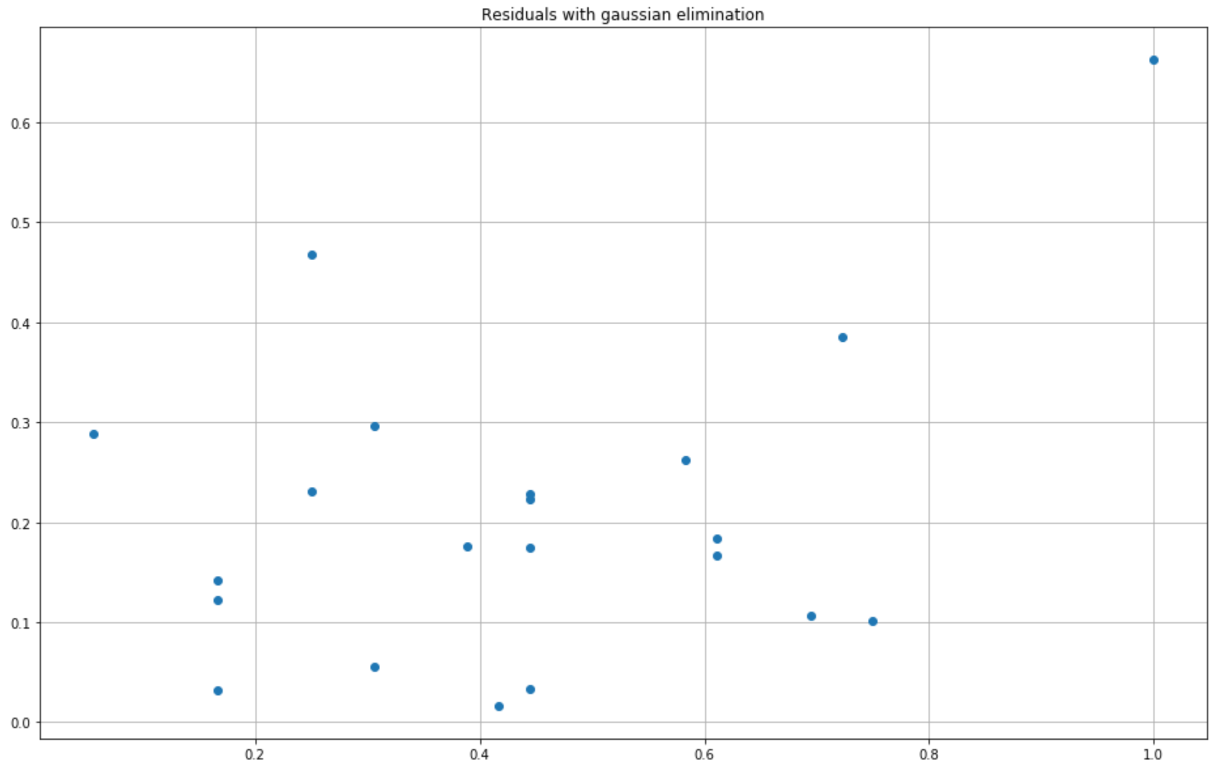
As L is a lower triangular matrix, we can solve it using forward substitution. Once we get y, we can solve $y = L^T\hat{\beta}$ using backward substitution.

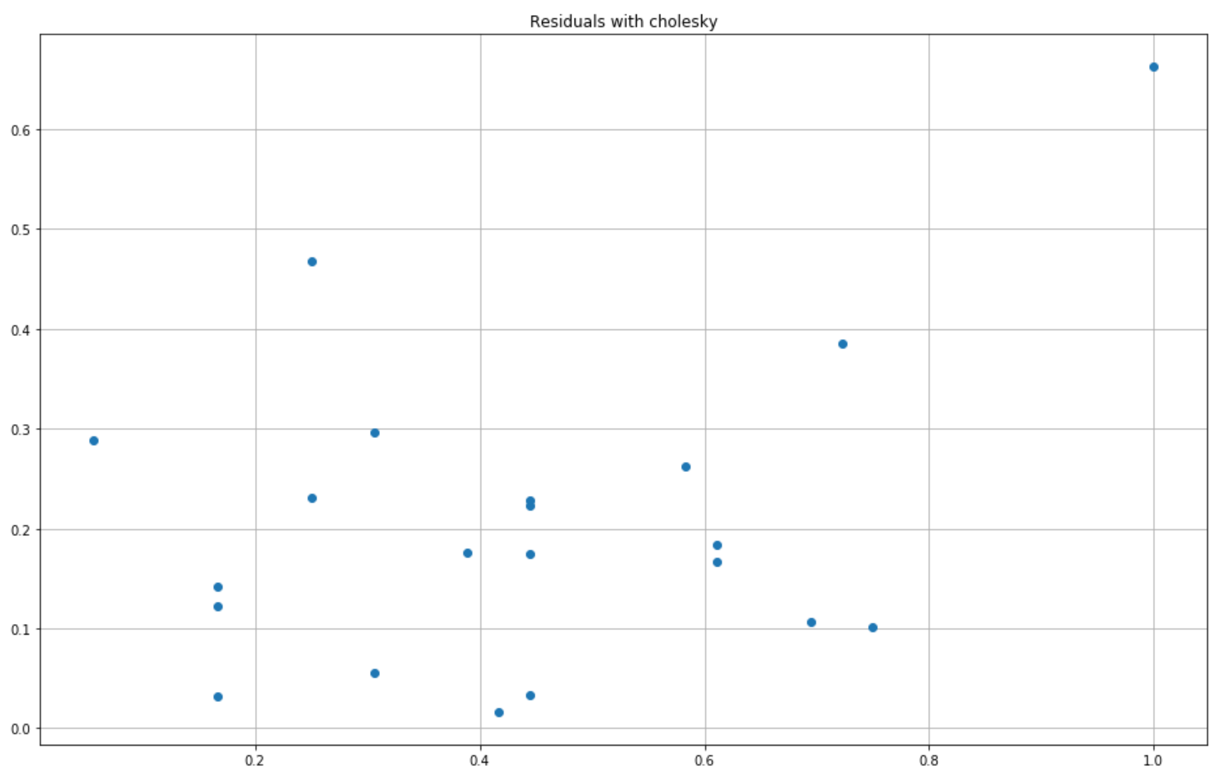
More information about how forward and backward substitution work can be found in [6]. The function norm_eq_cholesky defines this algorithms and used the backward susbtitution algorithm defined before.

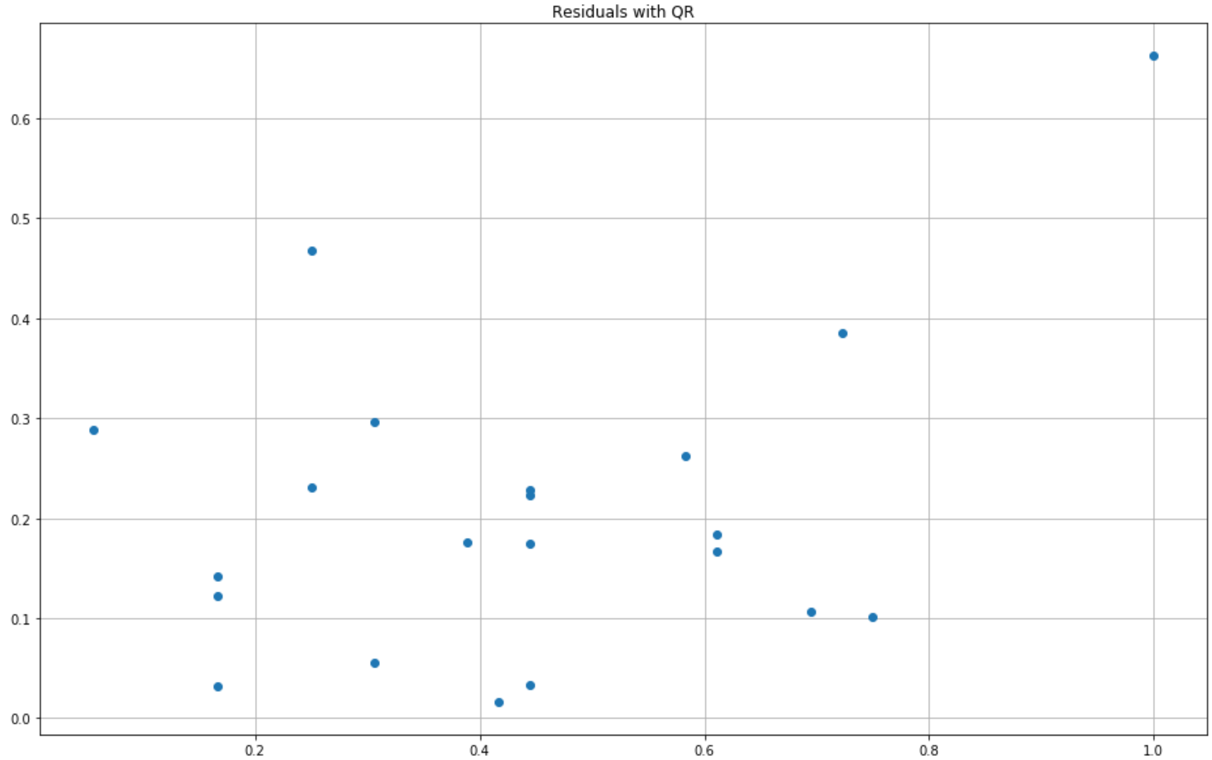
In [106]:
```
####Cholesky decomposition

def cholesky_decomposition(A):

    '''This function performs cholesky decomposition of matrix A'''

    n = A.shape[0]
    L = np.zeros((n,n))

    for i in range(0,n):
        for k in range(0,i+1):
            if(k==i):
                s=0
                for j in range(0,k):
                    s = s + (L[k,j])**2
                L[k,k] = np.sqrt((A[i,k])-s)
            else:
                s=0
                for j in range(0, k):
                    s = s + L[i,j]*L[k,j]
                L[i,k] = (1/L[k,k])*(A[i,k]-s)
    return L

def forward_substitution(a,b):

    '''This algorithm perform forward substitution to solve
    a linear system which involves a lower triangular matrix.'''
    n = a.shape[0]
    beta = np.zeros((n,1))
    for i in range(n):
        s=0
        for j in range(i):
            s=s+a[i,j]*beta[j]
        beta[i] = (b[i]-s)/a[i,i]
    return beta


def norm_eq_cholesky(X,y):

    '''This function uses cholesky decomposition to solve a
    matrix equation of the form X*beta=y, where beta is unknown.'''

    A = X.T*X
    c = X.T*y
    L = cholesky_decomposition(A)
    beta_aux = forward_substitution(L,c)
    beta = backward_substitution(L.T, beta_aux)

    return beta
```

We fit the linear model using the above mentioned methods and then we plot the residuals. Since the objective is the same, but it only differs the method, the results should also be the same. It indeed happens, as we can see on the plots. Moreover, we calculate the RMSE for each method (the results should be the same.

In [107]:
```
#fitting beta using gaussian elimination
beta_gauss = norm_eq_gauss(X,y)

#fitting beta using cholesky
beta_cholesky = norm_eq_cholesky(X,y)

#fitting beta using QR
beta_QR = norm_eq_QR(X,y)

#giving format to the matrix
X_test = np.matrix(X_test)
X_test = np.hstack((np.ones((X_test.shape[0],1)), X_test))
y_test = np.matrix(y_test)

#making predictions using the fitted values
pred_test_gauss = X_test*beta_gauss
pred_test_cholesky = X_test*beta_cholesky
pred_test_QR = X_test*beta_cholesky

#calculating residuals for fitted values
residuals_gauss = np.array(np.abs(pred_test_gauss-y_test))
residuals_cholesky = np.array(np.abs(pred_test_cholesky-y_test))
residuals_QR = np.array(np.abs(pred_test_QR-y_test))

#plotting residuals
fig, ax = plt.subplots(figsize=(16, 10))
ax.plot(y_test, residuals_gauss, 'o')
plt.title("Residuals with gaussian elimination")
plt.grid()

#plotting residuals
fig, ax = plt.subplots(figsize=(16, 10))
ax.plot(y_test, residuals_cholesky, 'o')
plt.title("Residuals with cholesky")
plt.grid()

#plotting residuals
fig, ax = plt.subplots(figsize=(16, 10))
ax.plot(y_test, residuals_QR, 'o')
plt.title("Residuals with QR")
plt.grid()
```


Residuals with gaussian elimination
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.2
0.4
0.6
0.8
1.0


Residuals with cholesky
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.2
0.4
0.6
0.8
1.0



In [108]:

```
#Finding average residuals
avg_residuals_gauss = np.mean(residuals_gauss)
avg_residuals_QR = np.mean(residuals_QR)
avg_residuals_cholesky = np.mean(residuals_cholesky)

print("Average residuals for gaussian elimination: %.4f"%avg_residuals_gauss)
print("Average residuals for QR decomposition: %.4f"%avg_residuals_QR)
print("Average residuals for cholesky decomposition: %.4f"%avg_residuals_choles
ky)

rmse_gauss = np.sqrt(np.mean(residuals_gauss**2))
rmse_QR = np.sqrt(np.mean(residuals_QR**2))
rmse_cholesky = np.sqrt(np.mean(residuals_cholesky**2))

print("RMSE for gaussian elimination: %.4f"%rmse_gauss)
print("RMSE for QR decomposition: %.4f"%rmse_QR)
print("RMSE for cholesky decomposition: %.4f"%rmse_cholesky)
```

```
Average residuals for gaussian elimination: 0.2073
Average residuals for QR decomposition: 0.2073
Average residuals for cholesky decomposition: 0.2073
RMSE for gaussian elimination: 0.2568
RMSE for QR decomposition: 0.2568
RMSE for cholesky decomposition: 0.2568
```

References

- [1] Data source in Github: https://github.com/jgscott/learnR/blob/master/cases/gasprices/gasprices.md (https://github.com/jgscott/learnR/blob/master/cases/gasprices/gasprices.md)
- [2] Gaussian elimination: https://en.wikipedia.org/wiki/Gaussian_elimination (https://en.wikipedia.org/wiki/Gaussian_elimination)
- [3] Gaussian elimination algorithm: https://learnche.org/3E4/Assignment_2_-_2010_-_Solution/Bonus_question (https://learnche.org/3E4/Assignment_2_-_2010_-_Solution/Bonus_question)
- [4] QR decomposition: https://en.wikipedia.org/wiki/QR_decomposition (https://en.wikipedia.org/wiki/QR_decomposition)
- [5] Cholesky decomposition: https://rosettacode.org/wiki/Cholesky_decomposition#Python (https://rosettacode.org/wiki/Cholesky_decomposition#Python)
- [6] Forward and backward substitution: http://mathfaculty.fullerton.edu/mathews/n2003/backsubstitutionmod.html (http://mathfaculty.fullerton.edu/mathews/n2003/backsubstitutionmod.html)