Voleon Solution

February 23, 2018

1 Voleon Interview Problem

1.0.1 Summary

This is a regression learning problem and the hypothesis set is assumed to be a first degree polynomials. We will estimate the coefficients of the model by minimizing the loss function. Three loss functions will be used: Ordinary Least Square (OLS), OLS with ${}_1$ penalty, and OLS with ${}_2$ penalty. We will then use K-Fold cross-validation to average the estimated weights and compare the results.

1.0.2 General learning problem

Data points (x_n, y_n) are generated from the joint distribution

$$P(x, y) = P(x)P(y|x).$$

$P(x)$ is the distribution of the independent variable x and

$$P(y|x) = f(x) + P(e|x).$$

$f(x)$ is the deterministic target function from input space $\mathcal{X}$ to output space $\mathcal{Y}$ ($f : \mathcal{X} \to \mathcal{Y}$), and e is the noise in the target function. The target function can be found by:

$$f(x) = \mathbb{E}[y|x].$$

Moreover, the noise in the target has the property

$$\mathbb{E}[e|x] = 0$$

As instructed in the statement of the problem, we assume a hypothesis set $\mathcal{H}$ with first degree polynomials:

$$\mathcal{H} = \{h \mid h(x) = w_0 + w_1 x \;\forall\; w_0, w_1 \in \mathbb{R}\}$$

The goal is to find the best hypothesis g from our hypothesis set $\mathcal{H}$ that minimizes the error between $g(x)$ and $f(x)$.

1.0.3 Model Formulation with linear regression algorithm

As mentioned before, we assume our model (hypothesis) function to be a linear combination of the independent variable x:

$$h(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \ , \ \mathbf{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}$$

Our learning algorithm is to find the weights $a = w_0$ and $b = w_1$ such that they minimize the error. In order to find the best values, we will use three loss functions as our error measures:

1. Mean Squared Error (MSE): Ordinary Least Square (OLS) linear regression
2. MSE + $_1$ regularizer: Lasso linear regression
3. MSE + $_2$ regularizer: Ridge linear regression

Note: General Method of Moments (GMM) will result in the same error measure as using Mean Squared Error (MSE).

1.0.4 Optimization problem

OLS linear regression:

$$Loss = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - y_n)^2$$

MSE loss function is beneficial when we would like to accomodate effect of every data point. It is a very simple model and minimization of the loss function has a closed form solution:

$$\mathbf{w} = \mathbf{X}^{\dagger}\mathbf{y}$$

where:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_N^{\mathrm{T}} \end{bmatrix} \ , \ \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

$$\mathbf{X}^{\dagger} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}$$

Lasso linear regression:

$$Loss = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - y_n)^2 + \alpha\|\mathbf{w}\|$$

Lasso is more robust to the effect of outliers. I will use the coordinate descent optimization method (the default method for Lasso regression in python) to minimize the loss function and find weight parameters a and b.

Ridge linear regression:

$$Loss = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - y_n)^2 + \alpha\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

Ridge linear regression has a smooth loss function but it is less robust to the effect of outliers. Conjugate gradient method will be used to minimize the loss function and find the weight parameters a and b.

1.0.5 1. Importing the libraries

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.gridspec as gridspec
        import seaborn as sns
        from sklearn.preprocessing import scale
        from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV
        from sklearn.model_selection import KFold
        from sklearn.metrics import mean_squared_error
        from IPython.display import display

        sns.set()
        sns.set_style("whitegrid")
        # sns.set(style="ticks")
        %matplotlib inline
```

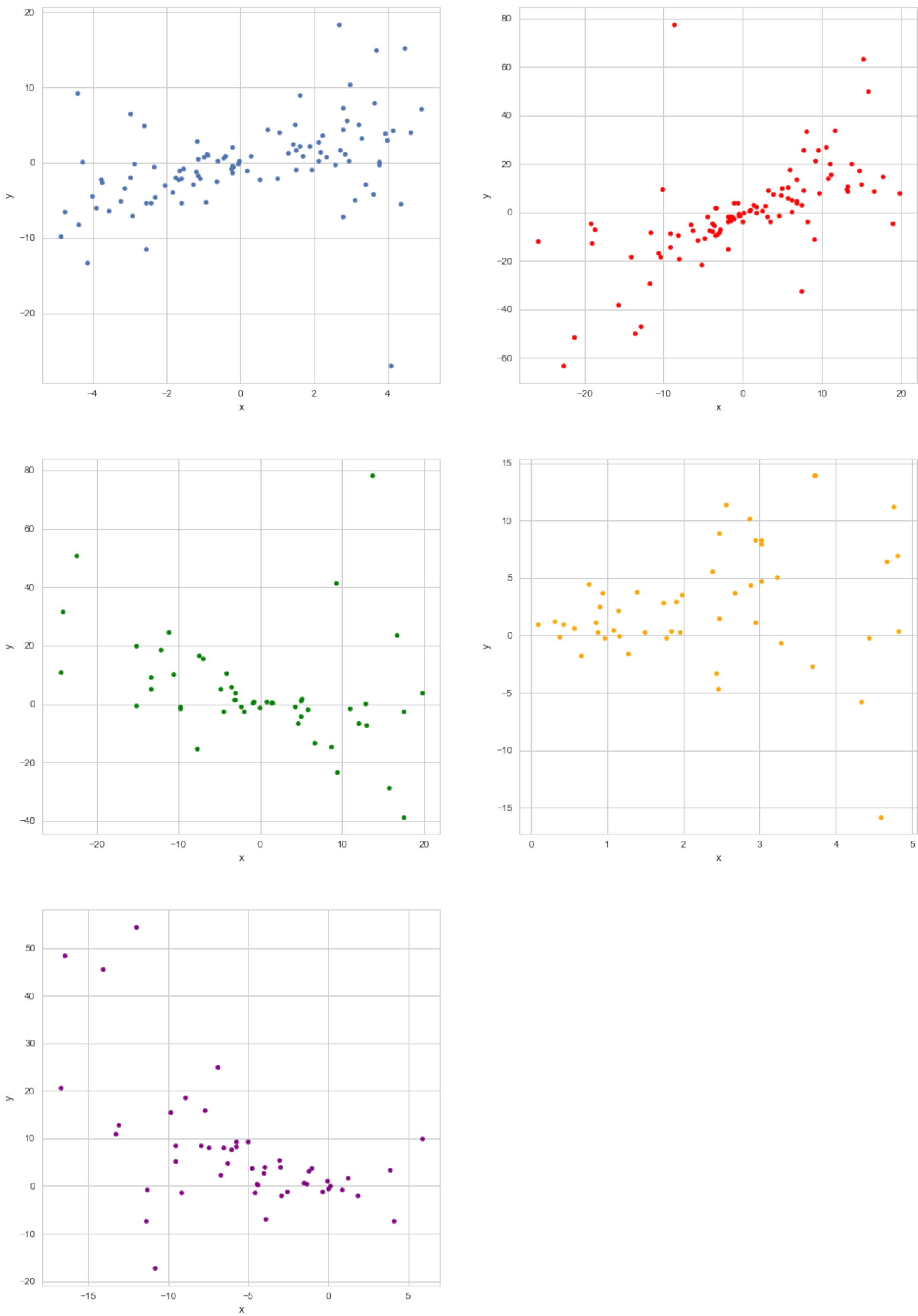
1.0.6 2. Loading data:

```
In [2]: data = {}
        data[1] = pd.read_csv('data_1_1.csv')
        data[2] = pd.read_csv('data_1_2.csv')
        data[3] = pd.read_csv('data_1_3.csv')
        data[4] = pd.read_csv('data_1_4.csv')
        data[5] = pd.read_csv('data_1_5.csv')
```

1.0.7 3. Initial exploratory data visualization

```
In [3]: fig = plt.figure(figsize=[16,24])
        ax1 = fig.add_subplot(321)
        data[1].plot.scatter('x','y', ax=ax1);
        ax2 = fig.add_subplot(322)
        data[2].plot.scatter('x','y', ax=ax2, color='red');
        ax3 = fig.add_subplot(323)
        data[3].plot.scatter('x','y', ax=ax3, color='green');
        ax4 = fig.add_subplot(324)
        data[4].plot.scatter('x','y', ax=ax4, color='orange');
        ax5 = fig.add_subplot(325)
        data[5].plot.scatter('x','y', ax=ax5, color='purple');
```

```
# fig, axes = plt.subplots(3, 2, figsize=[16,24])
# data1.plot.scatter('x','y', ax=axes[0,0]);
# data2.plot.scatter('x','y', ax=axes[0,1], color='red');
# data3.plot.scatter('x','y', ax=axes[1,0], color='green');
# data4.plot.scatter('x','y', ax=axes[1,1], color='orange');
# data5.plot.scatter('x','y', ax=axes[2,0], color = 'black');
```


y
x
y
x
y
x
y
x
y
x

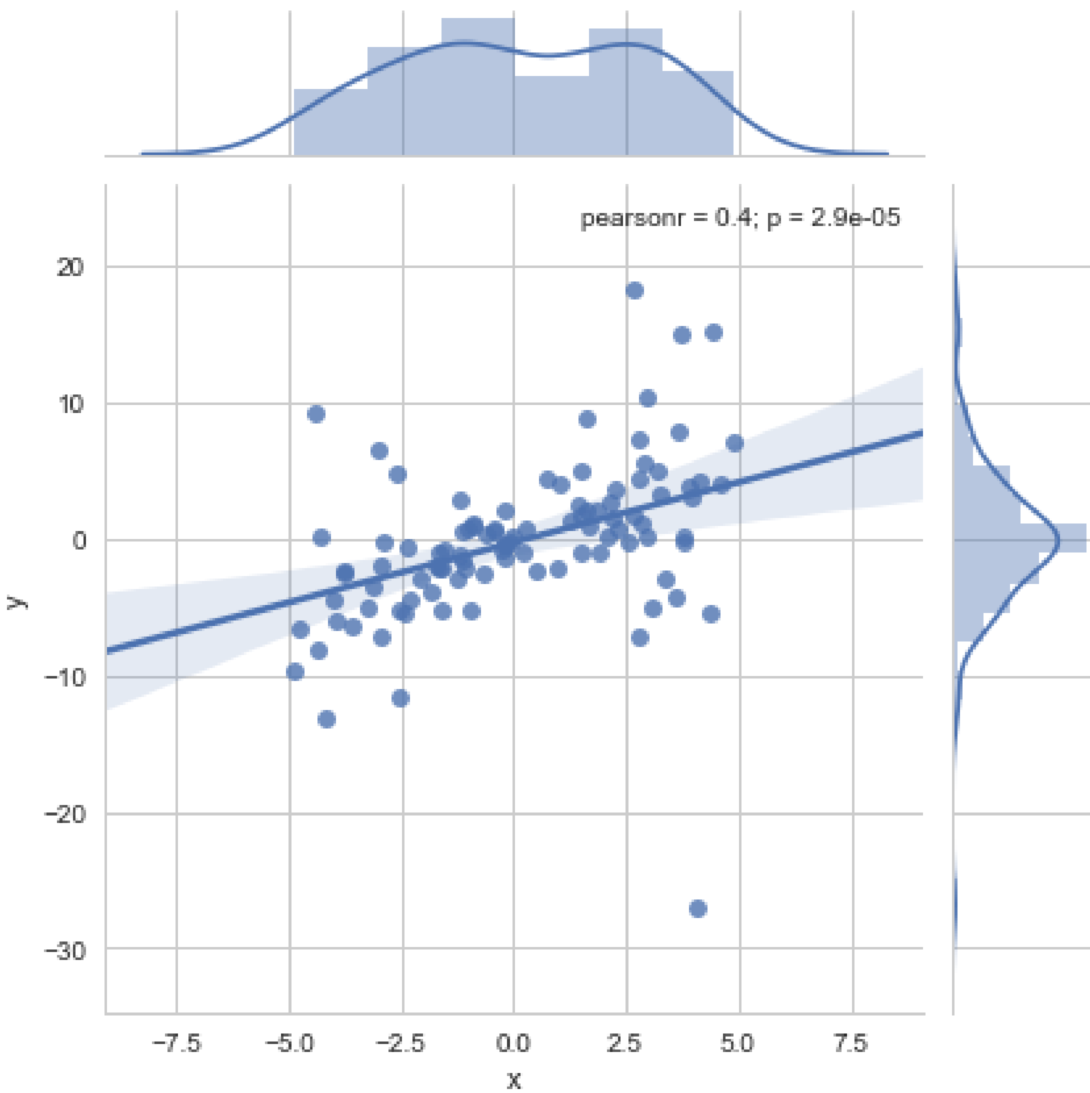
1.0.8 4. Modeling data

4.1 Getting data statistics

In [4]:
```
sns.jointplot(x='x', y='y', data=data[1], kind='reg');
data[1].describe()
```

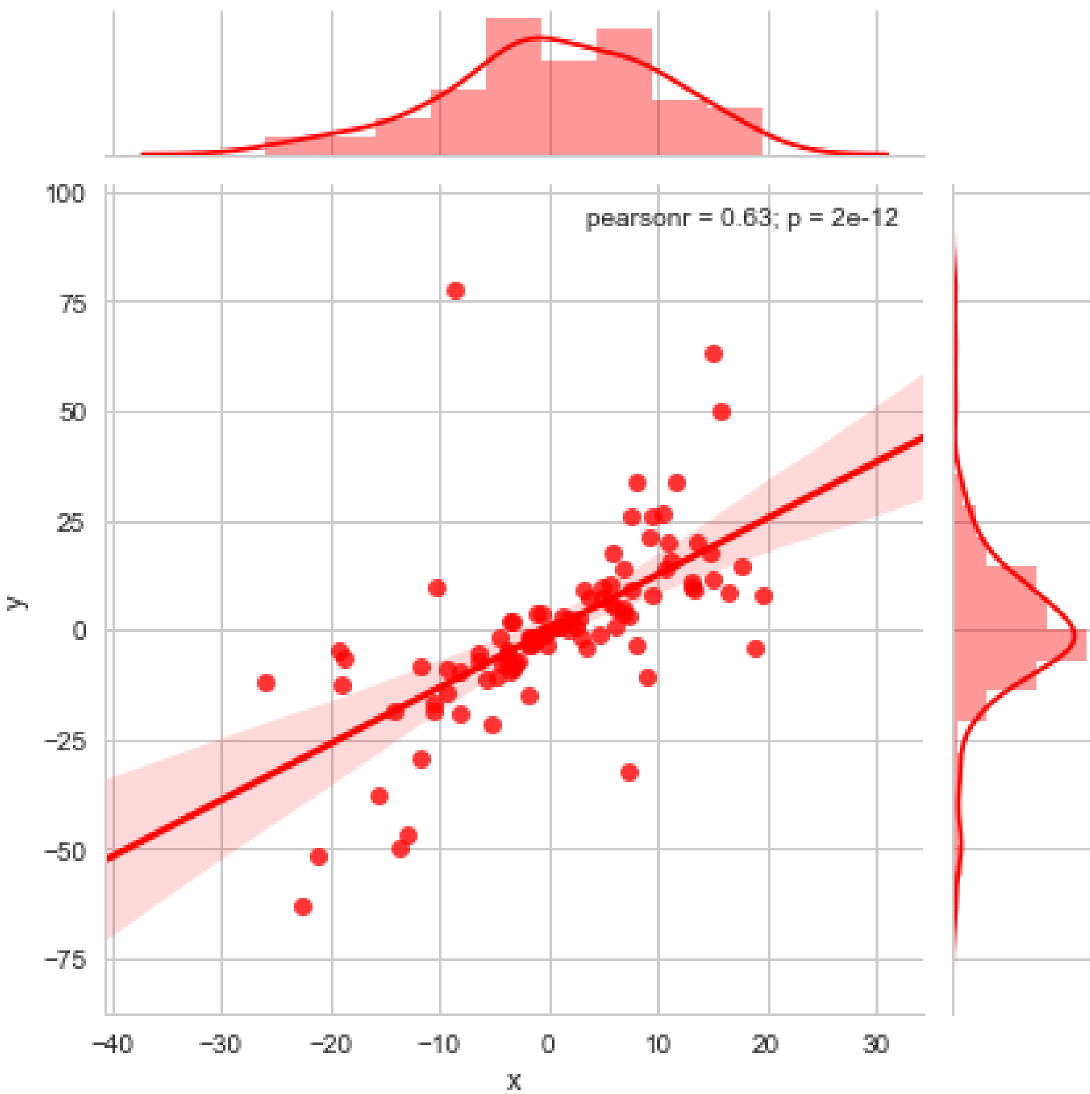
Out[4]:

	x	y
count	100.000000	100.000000
mean	0.178471	-0.112328
std	2.675848	5.820218
min	-4.866097	-27.008035
25%	-1.769180	-2.698194
50%	-0.121893	-0.179928
75%	2.671934	2.565632
max	4.919061	18.387174



```
In [5]: sns.jointplot(x='x', y='y', data=data[2], kind='reg', color='red');
        data[2].describe()
```

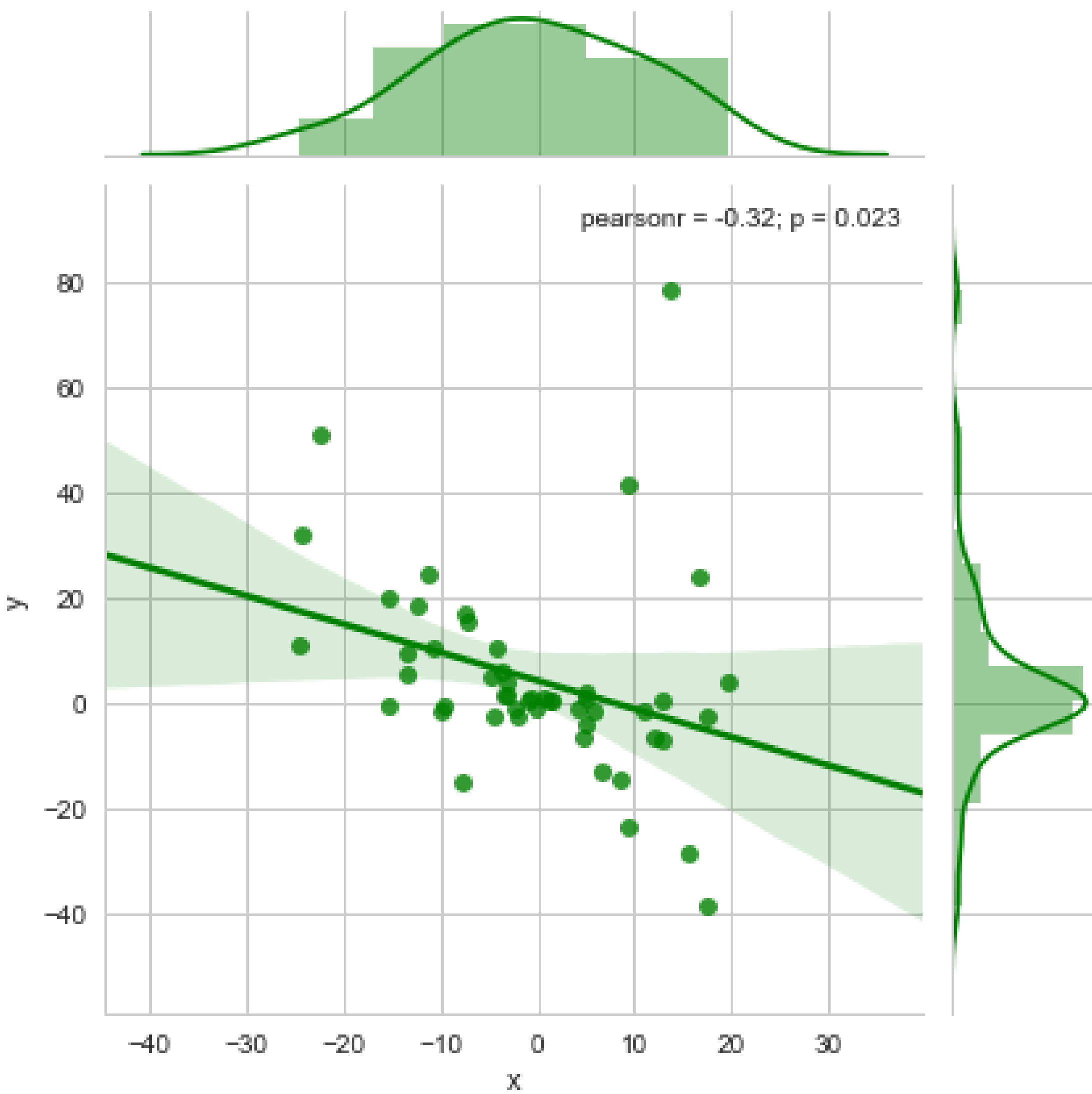
```
Out[5]:
                x           y
count  100.000000  100.000000
mean     0.473652    0.213075
std      9.805974   19.980543
min    -25.923277  -63.359915
25%     -4.615964   -7.803607
50%     -0.019552   -0.839115
75%      7.453385    8.893503
max     19.713374   77.642259
```



In [6]:
```
sns.jointplot(x='x', y='y', data=data[3], kind='reg', color='green');
data[3].describe()
```

Out[6]:

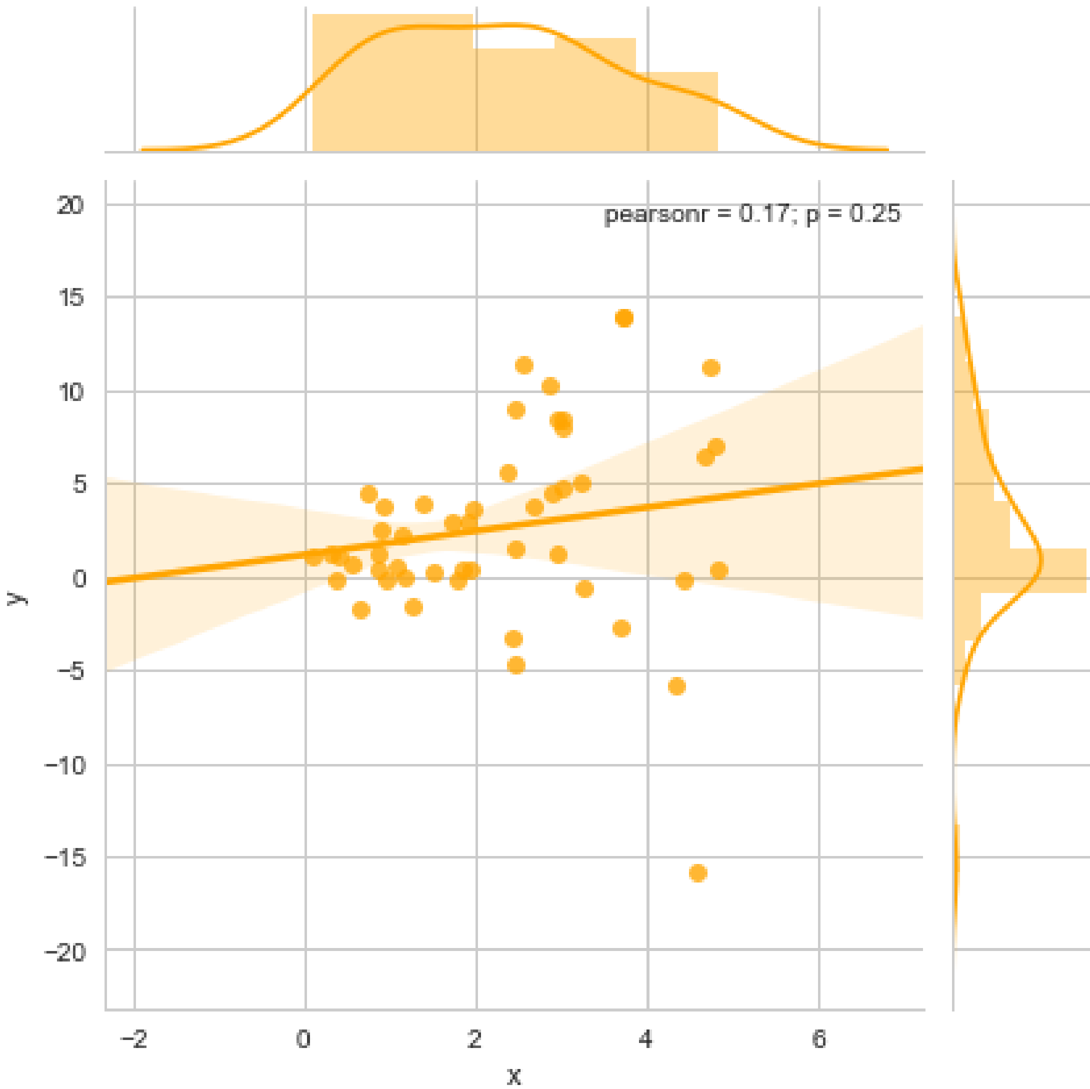
	x	y
count	50.000000	50.000000
mean	-0.421341	4.401773
std	11.160999	18.686335
min	-24.423113	-38.677853
25%	-7.731586	-2.249828
50%	-0.841930	0.618328
75%	8.115973	9.969540
max	19.715720	78.281136



In [7]:
```
sns.jointplot(x='x', y='y', data=data[4], kind='reg', color='orange');
data[4].describe()
```

Out[7]:

	x	y
count	50.000000	50.000000
mean	2.291112	2.599265
std	1.363023	5.196983
min	0.091918	-15.828510
25%	1.095533	0.062211
50%	2.403090	1.351297
75%	3.024151	4.982735
max	4.822850	13.952864

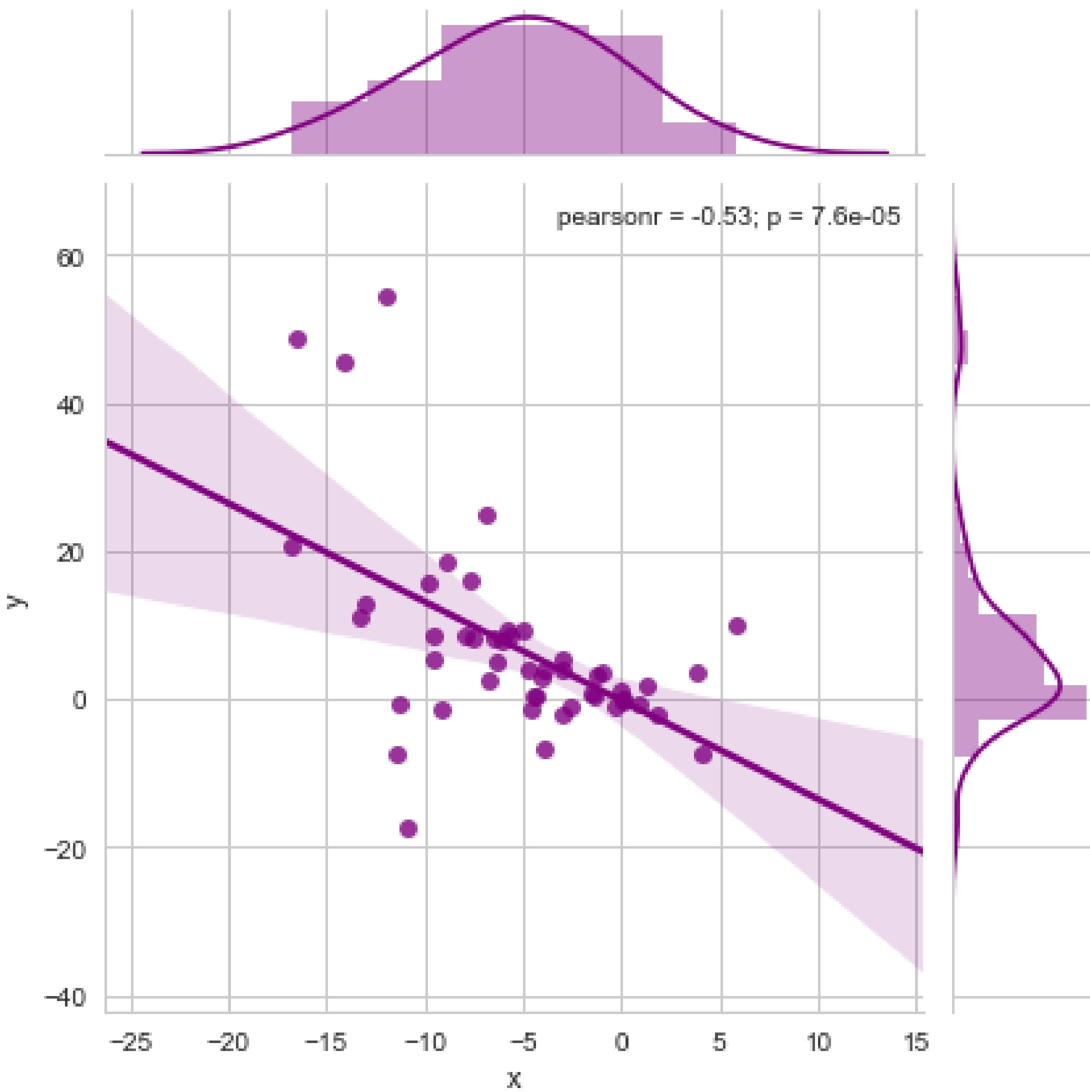


In [8]:
```
sns.jointplot(x='x', y='y', data=data[5], kind='reg', color='purple');
data[5].describe()
```

Out[8]:

	x	y
count	50.000000	50.000000

```
mean   -5.364524   6.909975
std     5.253889  13.222342
min   -16.740024 -17.285211
25%    -9.128192  -0.325283
50%    -4.890221   3.785614
75%    -1.398920   9.109861
max     5.861221  54.599125
```



4.2 Modeling data:

```
In [9]: for i in xrange(1, len(data)+1):
            print 'Modeling data in file data1_%d.csv' % i
            X = data[i].x.values.reshape(-1, 1)
```

```
y = data[i].y.values#.reshape(-1, 1)

print 'Linear regression model: y = ax + b'
ols = LinearRegression()
kf = KFold(5)
w_0 = np.array([])
w_1 = np.array([])
error = np.array([])
r2 = np.array([])
for k, (train, test) in enumerate(kf.split(X, y)):
    ols.fit(X[train], y[train])
    w_0 = np.append(w_0, ols.intercept_)
    w_1 = np.append(w_1, ols.coef_[0])
    error = np.append(error, mean_squared_error(y[test], ols.predict(X[
    r2 = np.append(r2, ols.score(X[test], y[test]))
a = w_1.mean()
b = w_0.mean()
e = error.mean()
print 'OLS   lin Reg: a = %.2f , b = %.2f , MSE = %.3f, R^2 = %.3f' % 
# clf = GridSearchCV(LinearRegression(), cv=5)
# clf.fit(data1.x, data1.y)


lso = LassoCV(cv=5)
kf = KFold(5)
w_0 = np.array([])
w_1 = np.array([])
error = np.array([])
r2 = np.array([])
for k, (train, test) in enumerate(kf.split(X, y)):
    lso.fit(X[train], y[train])
    w_0 = np.append(w_0, lso.intercept_)
    w_1 = np.append(w_1, lso.coef_[0])
    error = np.append(error, mean_squared_error(y[test], lso.predict(X[
    r2 = np.append(r2, lso.score(X[test], y[test]))
#     print lso.alpha_
a = w_1.mean()
b = w_0.mean()
e = error.mean()
print 'LASSO lin reg: a = %.2f , b = %.2f , MSE = %.3f, R^2 = %.3f' % 

alphas = np.logspace(-4, 4, 30)
rdg = RidgeCV(alphas=alphas, cv=5)
kf = KFold(5)
w_0 = np.array([])
w_1 = np.array([])
error = np.array([])
r2 = np.array([])
```

```
            for k, (train, test) in enumerate(kf.split(X, y)):
                rdg.fit(X[train], y[train])
                w_0 = np.append(w_0, rdg.intercept_)
                w_1 = np.append(w_1, rdg.coef_[0])
                error = np.append(error, mean_squared_error(y[test], rdg.predict(X
                r2 = np.append(r2, rdg.score(X[test], y[test]))
            #      print rdg.alpha_
            a = w_1.mean()
            b = w_0.mean()
            e = error.mean()
            print 'Ridge lin reg: a = %.2f , b = %.2f , MSE = %.3f, R^2 = %.3f' %
            print
```

```
Modeling data in file data1_1.csv
Linear regression model: y = ax + b
OLS   lin Reg: a = 0.88 , b = -0.27 , MSE = 28.560, R^2 = 0.178
LASSO lin reg: a = 0.85 , b = -0.27 , MSE = 28.557, R^2 = 0.177
Ridge lin reg: a = 0.86 , b = -0.27 , MSE = 28.492, R^2 = 0.179

Modeling data in file data1_2.csv
Linear regression model: y = ax + b
OLS   lin Reg: a = 1.28 , b = -0.42 , MSE = 248.041, R^2 = 0.375
LASSO lin reg: a = 1.26 , b = -0.42 , MSE = 248.644, R^2 = 0.374
Ridge lin reg: a = 1.27 , b = -0.41 , MSE = 247.239, R^2 = 0.376

Modeling data in file data1_3.csv
Linear regression model: y = ax + b
OLS   lin Reg: a = -0.54 , b = 4.15 , MSE = 324.986, R^2 = 0.110
LASSO lin reg: a = -0.52 , b = 4.17 , MSE = 322.565, R^2 = 0.115
Ridge lin reg: a = -0.52 , b = 4.17 , MSE = 320.336, R^2 = 0.116

Modeling data in file data1_4.csv
Linear regression model: y = ax + b
OLS   lin Reg: a = 0.63 , b = 1.18 , MSE = 27.047, R^2 = -0.053
LASSO lin reg: a = 0.35 , b = 1.83 , MSE = 27.806, R^2 = -0.102
Ridge lin reg: a = 0.30 , b = 1.93 , MSE = 27.665, R^2 = -0.097

Modeling data in file data1_5.csv
Linear regression model: y = ax + b
OLS   lin Reg: a = -1.33 , b = -0.21 , MSE = 129.884, R^2 = -0.174
LASSO lin reg: a = -1.31 , b = -0.14 , MSE = 129.658, R^2 = -0.155
Ridge lin reg: a = -0.76 , b = 2.85 , MSE = 142.588, R^2 = -0.153
```

1.0.9 Observation:

Based on the current datasets, and more specifically for datasets from files data1_4.csv and data1_5.csv, a first degree polynomial hypothesis set may not be the best choice to model our samples data generation process. ***This is a high bias problem!***