MLR_Assignment

May 18, 2021

1 LINEAR REGRESSION

Objective: To gets hands on experience on multiple linear regression. * Cost Function for regression * Gradient Descent, Learning rates * Feature Normalization * Iterative method vs Direct Method for regression * Using scikit-learn library for regression * How to work with categorical variables / one hot encoding

Problem: We will solve two set of problems 1. Predicting house price 2. Predicting weather station maintenence request counts

2 INTRODUCTION

Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope. Is used to predict values within a continuous range. (e.g. sales, price)

Simple Regression: Simple linear regression uses traditional slope-intercept form, where m and b are the variables our algorithm will try to "learn" to produce the most accurate predictions. x represents our input data and y represents our prediction.

$$\hat{y} = \theta_1 x + \theta_0$$

In order to compute the values of m and b we need to minimize a cost function. In the case of Linear Regression it is given by

$$J(\theta) = \frac{1}{2m}\sum_{i=i}^{m}(\hat{y}_i - y_i)^2$$

In multivariate regression we seek a set of parameters

$$\Theta = [\theta_0 \quad \theta_1 \quad \cdots \quad \theta_i \quad \cdots \quad \theta_m]^T$$

which minimizes the the cost function: Hypothesis $**H**=**X** Θ$

Cost function is

$$\mathbf{J(\Theta)} = \frac{1}{2m}(\mathbf{X}\Theta - \mathbf{Y})^T(\mathbf{X}\Theta - \mathbf{Y})$$

The advantages of Linear Regression are that they computationally efficient, simple and easy to interpret. However, the algorithm fails to capture non-linear behavior.

[2]:
```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import matplotlib.pyplot as plt
#from sklearn import datasets,linear_model
import pandas as pd
```

3 Loading the data from a csv file

The columns represent the surface area, number of rooms and the price of the apartment.

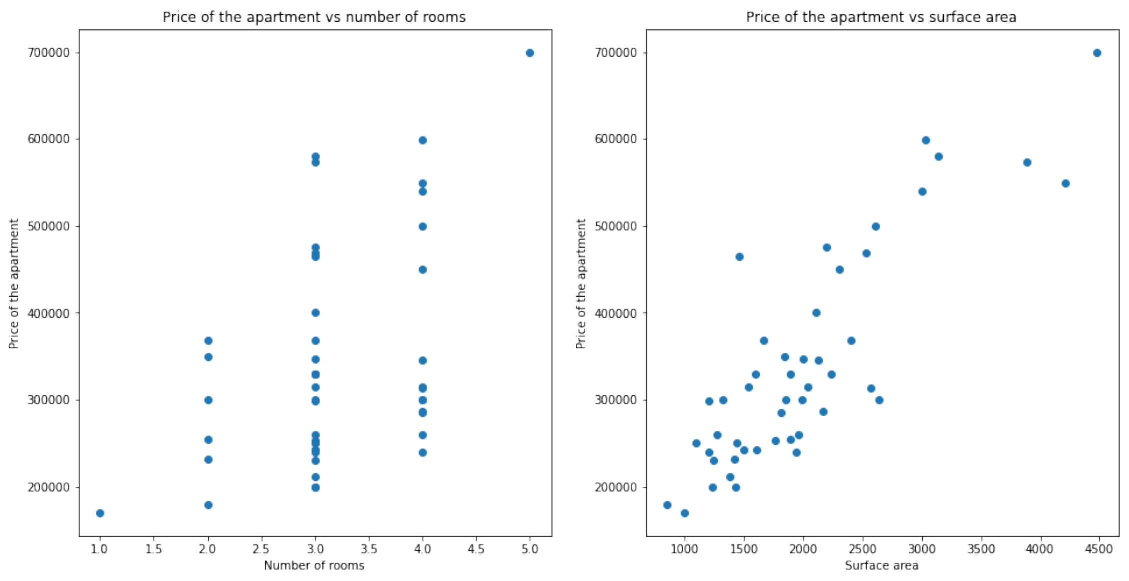
[3]:
```
data = np.loadtxt('../data/Housing/housing_data.csv', delimiter=",")
X = data[:,:2]
y = data[:,2]
m = len(y)
```

4 Data Exploration

Plot the data and have a look Note the large difference in the scale of "number of room variables" and "surface area". * "Number of room" has a range [1.0,5.0] * "Surface area" has a range [500,4500]

This causes problems during the cost minimization step (gradient descent).

[4]:
```
plt.figure(figsize=(16,8))
plt.subplot(121)
plt.scatter(X[:,1],y)
plt.xlabel("Number of rooms")
plt.ylabel("Price of the apartment")
plt.title("Price of the apartment vs number of rooms")
plt.subplot(122)
plt.scatter(X[:,0],y)
plt.xlabel("Surface area")
plt.ylabel("Price of the apartment")
plt.title("Price of the apartment vs surface area")
plt.show()
```



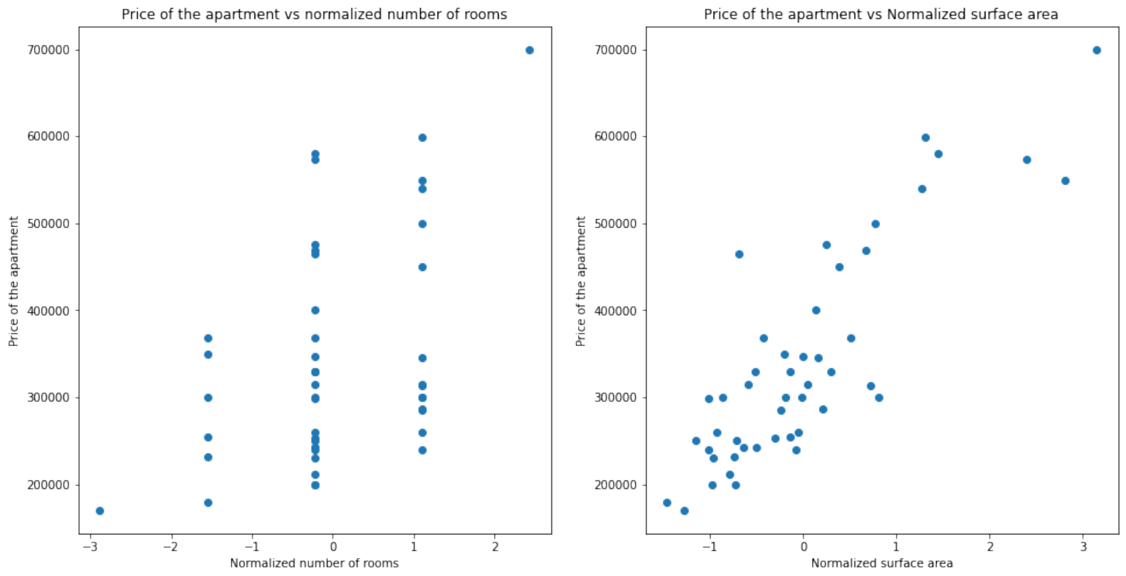
5 Feature normalization.

The input feature vectors are normalized so that they have comparable scales.Here we do so by subtracting the mean and dividing by the standard deviation.

[5]:
```
# Normalize features
def featureNormalize(X):
    X_norm = X.copy()
    mu    = np.zeros((1, X.shape[1]))
    sigma = np.zeros((1, X.shape[1]))
    for i in range(X.shape[1]):
        mu[:,i] = np.mean(X[:,i])
        sigma[:,i] = np.std(X[:,i])
        X_norm[:,i] = (X[:,i] - float(mu[:,i]))/float(sigma[:,i])
    return X_norm, mu, sigma
```

[6]:
```
X_norm, mu, sigma = featureNormalize(X)
```

[7]:
```
plt.figure(figsize=(16,8))
plt.subplot(121)
plt.scatter(X_norm[:,1],y)
plt.xlabel("Normalized number of rooms")
plt.ylabel("Price of the apartment")
plt.title("Price of the apartment vs normalized number of rooms")
plt.subplot(122)
plt.scatter(X_norm[:,0],y)
plt.xlabel("Normalized surface area")
plt.ylabel("Price of the apartment")
plt.title("Price of the apartment vs Normalized surface area")
plt.show()
```



6 Bias Term

We simply add a column vector of 1.

[8]:
```
X_padded = np.column_stack((np.ones((m,1)), X_norm))
```

7 Cost Function involving multiple variables

$$\mathbf{J}(\Theta) = \frac{1}{2m}(\mathbf{X}\Theta - \mathbf{Y})^T(\mathbf{X}\Theta - \mathbf{Y})$$

[9]:
```
def computeCost(X,y,theta):
    m=len(y)
    Cost=0.0;
    y=y.reshape(-1,1)
    Cost=1.0/2.0/m*(np.dot((X.dot(theta)-y).T,(X.dot(theta)-y)))
    return Cost;
```

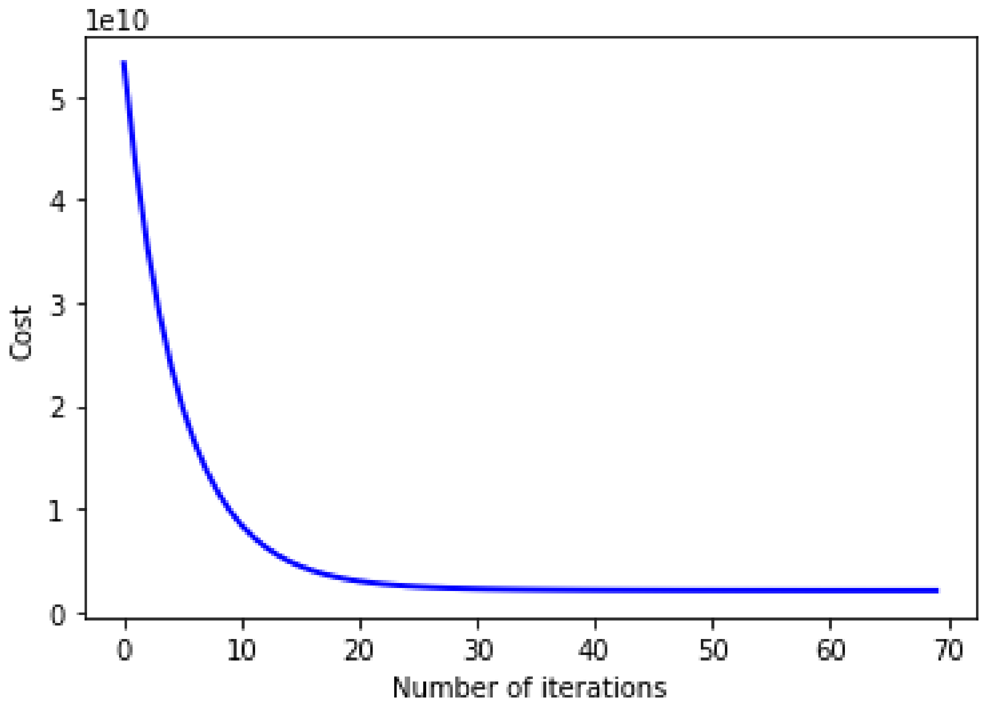
[10]:
```
def gradientDescent(X, y, alpha, num_iters):
    m,n = X.shape
    theta = np.zeros((X.shape[1],1))
    Cost_history = np.zeros((num_iters, 1))
    for i in range(num_iters):
        theta = theta - alpha*(1.0/m) * np.transpose(X).dot(X.dot(theta) - np.
↪transpose([y]))
        Cost_history[i] = computeCost(X, y, theta)
    return theta, Cost_history
```

[11]:
```
def SolveGradientDescent(X,y,alpha, num_iters):
    m,n=X.shape
    theta, Loss_history = gradientDescent(X, y, alpha, num_iters)
    plt.plot(range(Loss_history.size), Loss_history, "-b", linewidth=2 )
    plt.xlabel('Number of iterations')
    plt.ylabel('Cost')
    plt.show(block=False)
    theta.shape
    return theta
```

8 Evolution of the cost function

Try solving the same problem but without feature normalization and convince yourself the importance of the step.

[12]:
```
theta_GD=SolveGradientDescent(X=X_padded,y=y,alpha=0.1,num_iters=70)
print('Theta computed from gradient descent: ',theta_GD.T)
```



```
Theta computed from gradient descent:  [[340199.36423635 106961.59217624
-4092.86897639]]
```

9 Make predictions using the trained model

Remember that the input features were normalized so we need to do the same with the new input features on which we want to make make predictions.

```
[13]: house_norm_padded = np.array([1, (1650-mu[0,0])/sigma[0,0], (3-mu[0,1])/
       ↪sigma[0,1]])
      price_GD = np.array(house_norm_padded).dot(theta_GD)
      print("Predicted price of a 1650 sq-ft, 3 br house (using gradient descent):",␣
       ↪price_GD)
```

```
Predicted price of a 1650 sq-ft, 3 br house (using gradient descent):
[293415.1732852]
```

10 Solution using Normal Equation

Generally for very large problems due to memory requirements gradient descent algorithm is used for minimization but since here we are working with relatively small dataset with smaller number of feature vectors X, we have used direct method for finiding the minima. This involves invoking the closed-form solution to linear regression. Convince yourself with a little calculation that:

$$= (\mathbf{X}^{\mathbf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathbf{T}}\mathbf{Y}$$

```
[14]: # Solve using direct method
      def normalEqn(X, y):
          theta = np.zeros((X.shape[1], 1))
          theta = np.linalg.pinv(np.transpose(X).dot(X)).dot(np.transpose(X).dot(y))
          return theta
```

```
[15]: theta_Normal = normalEqn(X_padded, y)
      print("Theta calculated by Normal Equation ",theta_Normal)
      price_Normal = np.array(house_norm_padded).dot(theta_Normal)
      print("Predicted price of a 1650 sq-ft, 3 br house (using normal equation):",␣
       ↪price_Normal)
```

```
Theta calculated by Normal Equation  [340412.65957447 109447.79646964
-6578.35485416]
Predicted price of a 1650 sq-ft, 3 br house (using normal equation):
293081.4643348961
```

11 Regression using scikit-learn library in two lines

```
[16]: from sklearn import linear_model
      lr = linear_model.LinearRegression()
      lr.fit(X_padded,y)
      print("Theta calculated by SK-learn regression ",lr.coef_)
      print("Predicted price ",lr.predict(house_norm_padded.reshape(1,-1)))
```

```
Theta calculated by SK-learn regression  [     0.          109447.79646964
-6578.35485416]
Predicted price  [293081.4643349]
```

12 Example 2

Data Description

- date : yyyy-mm-dd format
- calendar_code : 0 or 1 (a code describing certain calendar events)
- request_count : an integer (the number of support requests received on that date)
- site_count : an integer (the number of sites operating on that date)
- max_temp : a float (max temperature for that day in degrees Celsius)
- min_temp : a float (min temperature for that day in degrees Celsius)
- precipitation : a float (millimeters of precipitation on that date)
- events : a string (description of weather events on that date)

Our aim is to predict the request_count when the other paeameters are given.

```
[17]: import pandas as pd
      import numpy as np
      from sklearn.preprocessing import LabelEncoder, OneHotEncoder
      from sklearn.model_selection import train_test_split
      from sklearn import linear_model
      import matplotlib.pyplot as plt
      import seaborn as sns
```

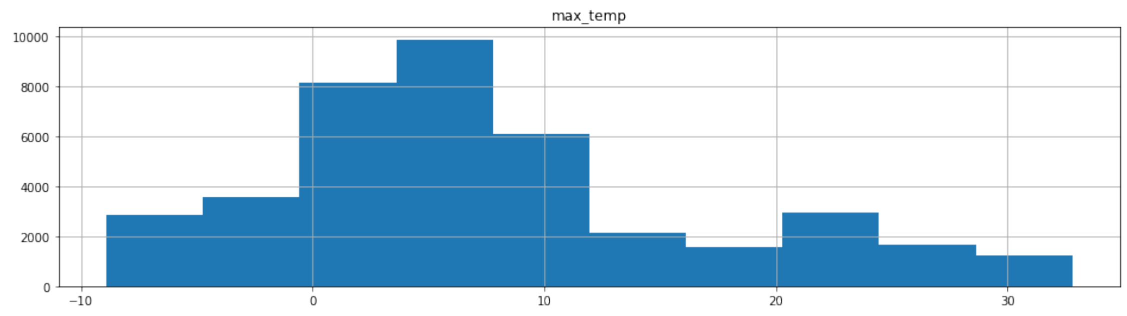
```
[18]: #ALso remember to parse the date column. This will be helpful in the next step
      training_data=pd.read_csv('../data/Met/Met_train.
       ↪csv',sep=',',parse_dates=['date'])
      training_data.head()
```

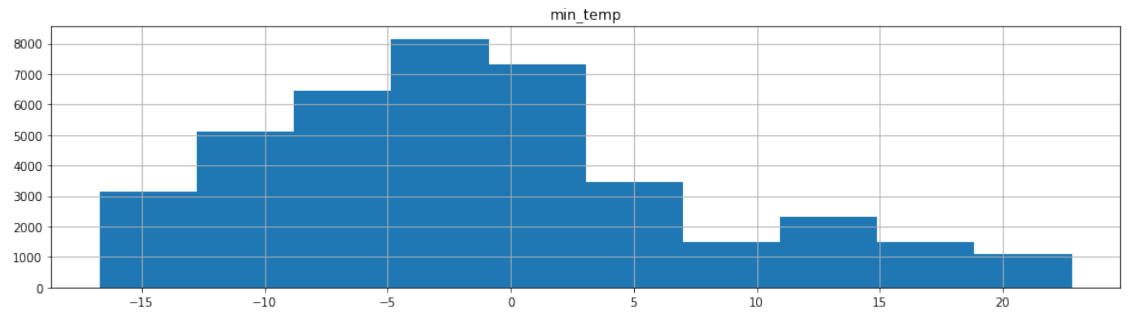
```
[18]:         date  calendar_code  request_count  site_count  max_temp  min_temp  \
      0 2014-09-01            0.0            165           6      30.6      22.8
      1 2014-09-02            1.0            138           7      32.8      22.8
      2 2014-09-03            1.0            127           7      29.4      18.3
      3 2014-09-04            1.0            174           7      29.4      17.2
      4 2014-09-05            1.0            196           7      30.6      21.7

         precipitation             events
```

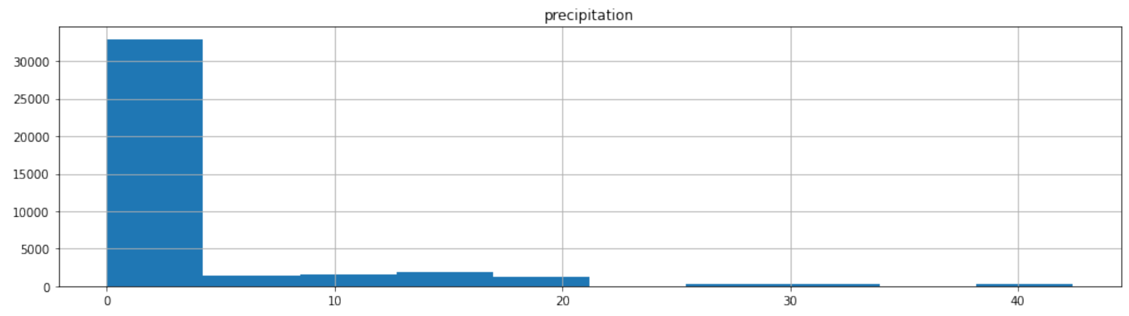
```
0         0.0               Rain
1        15.5  Rain-Thunderstorm
2         0.0               None
3         0.0               None
4         0.0                Fog
```

13 Data Exploration

[19]:
```
training_data.
 ↪hist('max_temp',weights=training_data['request_count'],figsize=(16,4))
training_data.
 ↪hist('min_temp',weights=training_data['request_count'],figsize=(16,4))
training_data.
 ↪hist('precipitation',weights=training_data['request_count'],figsize=(16,4))
plt.show()
```



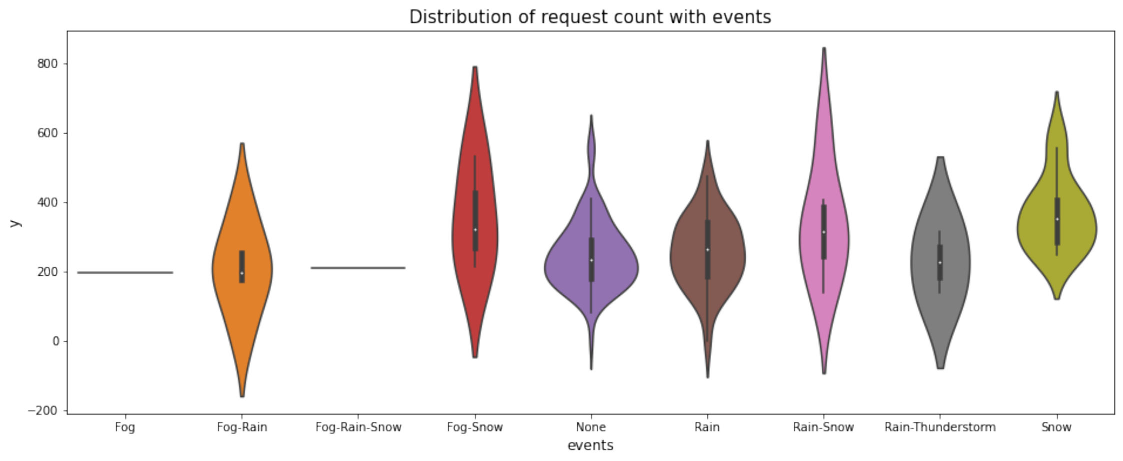




From the above histograms we see that most of the request comes when 1. maximum temperature is below 10C 2. min temperature is below 2C 3. When there is zero precipitation

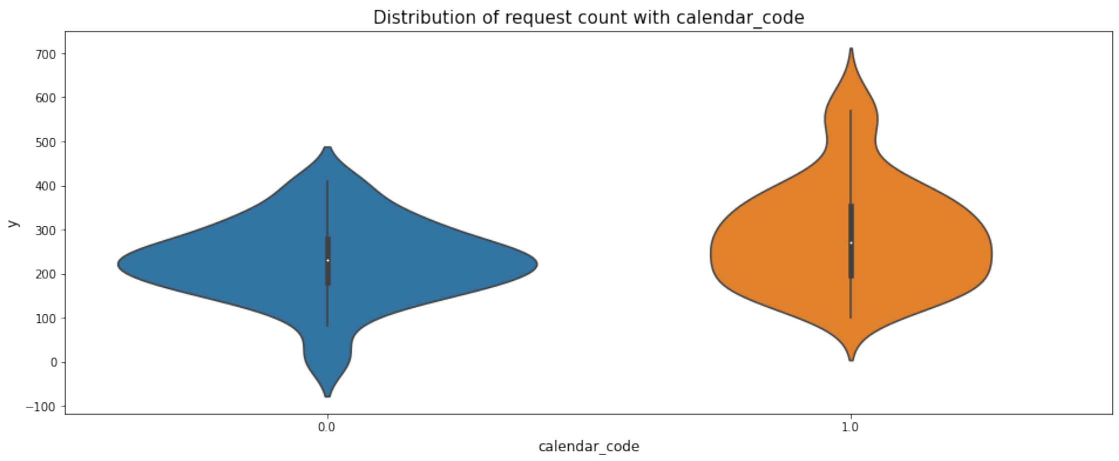
We use violin plot for dependence on the categorical variables (https://blog.modeanalytics.com/violin-plot-examples/)

[20]:
```
var_name = "events"
col_order = np.sort(training_data[var_name].unique()).tolist()
plt.figure(figsize=(16,6))
sns.violinplot(x=var_name, y='request_count', data=training_data,␣
 ↪order=col_order)
plt.xlabel(var_name, fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of request count with "+var_name, fontsize=15)
plt.show()
```



[21]:
```
var_name = "calendar_code"
col_order = np.sort(training_data[var_name].unique()).tolist()
plt.figure(figsize=(16,6))
```

```
sns.violinplot(x=var_name, y='request_count', data=training_data,␣
↪order=col_order)
plt.xlabel(var_name, fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of request count with "+var_name, fontsize=15)
plt.show()
```



[22]:
```
training_data['day_of_week'] = training_data['date'].dt.dayofweek
training_data['week_day'] = training_data['date'].dt.day_name()
training_data.head()
```

[22]:
```
        date  calendar_code  request_count  site_count  max_temp  min_temp  \
0 2014-09-01            0.0            165           6      30.6      22.8
1 2014-09-02            1.0            138           7      32.8      22.8
2 2014-09-03            1.0            127           7      29.4      18.3
3 2014-09-04            1.0            174           7      29.4      17.2
4 2014-09-05            1.0            196           7      30.6      21.7

   precipitation             events  day_of_week   week_day
0            0.0               Rain            0     Monday
1           15.5  Rain-Thunderstorm            1    Tuesday
2            0.0               None            2  Wednesday
3            0.0               None            3   Thursday
4            0.0                Fog            4     Friday
```

14 var_name = "week_day"

col_order = np.sort(training_data[var_name].unique()).tolist() plt.figure(figsize=(16,6)) sns.violinplot(x=var_name, y='request_count', data=training_data, order=col_order) plt.xlabel(var_name, fontsize=12) plt.ylabel('y', fontsize=12) plt.title("Distribution of request count with"+var_name, fontsize=15) plt.show()

15 Dealing with categorical variables

In most of the machine learning algorithms barringa a few (ex. Decision Tree) we need numerical values for the input features. In the current data events are categorical variable which should be converted into a unique numerical identifiers. This will result in an additional column "events_code"

[24]:
```
training_data['events_code'] = pd.Categorical(training_data["events"]).codes
training_data.head()
```

[24]:
```
         date  calendar_code  request_count  site_count  max_temp  min_temp  \
0  2014-09-01            0.0            165           6      30.6      22.8
1  2014-09-02            1.0            138           7      32.8      22.8
2  2014-09-03            1.0            127           7      29.4      18.3
3  2014-09-04            1.0            174           7      29.4      17.2
4  2014-09-05            1.0            196           7      30.6      21.7

   precipitation             events  day_of_week   week_day  events_code
0            0.0               Rain            0     Monday            5
1           15.5  Rain-Thunderstorm            1    Tuesday            7
2            0.0               None            2  Wednesday            4
3            0.0               None            3   Thursday            4
4            0.0                Fog            4     Friday            0
```

Since request count is the target variable, we store it separately as "y"

[25]:
```
y=training_data["request_count"]
```

Drop the redundant columns now "date","events","request_count"

[26]:
```
training_data = training_data.
 ↪drop(["date","events","request_count","week_day"],axis=1)
training_data.head()
```

[26]:
```
   calendar_code  site_count  max_temp  min_temp  precipitation  day_of_week  \
0            0.0           6      30.6      22.8            0.0            0
1            1.0           7      32.8      22.8           15.5            1
2            1.0           7      29.4      18.3            0.0            2
3            1.0           7      29.4      17.2            0.0            3
4            1.0           7      30.6      21.7            0.0            4

   events_code
0            5
1            7
2            4
3            4
4            0
```

16 One-Hot-Encoding

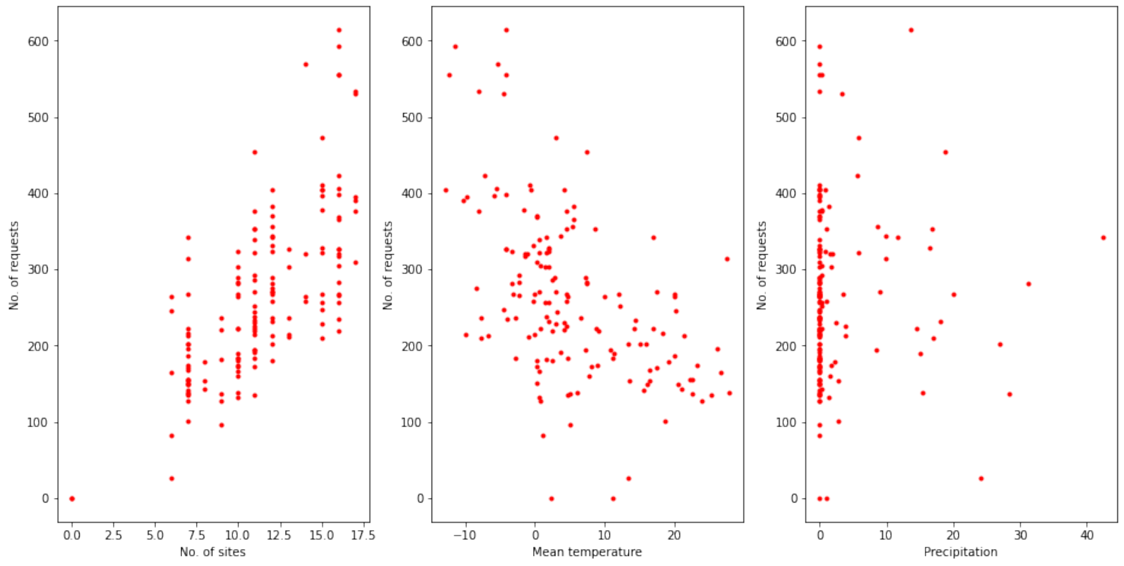
The numerical values of day_of_week, events_code and calender code do not signify anything so they need to be one-hot-encoded to be used as a feature input vector.

[27]:
```
training_data= pd.
↪get_dummies(training_data,columns=["calendar_code","events_code","day_of_week"],prefix=["ca
training_data.head()
```

[27]:
```
   site_count  max_temp  min_temp  precipitation  calendar_0.0  calendar_1.0  \
0           6      30.6      22.8            0.0             1             0
1           7      32.8      22.8           15.5             0             1
2           7      29.4      18.3            0.0             0             1
3           7      29.4      17.2            0.0             0             1
4           7      30.6      21.7            0.0             0             1

   event_0  event_1  event_2  event_3  …  event_6  event_7  event_8  week_0  \
0        0        0        0        0  …        0        0        0       1
1        0        0        0        0  …        0        1        0       0
2        0        0        0        0  …        0        0        0       0
3        0        0        0        0  …        0        0        0       0
4        1        0        0        0  …        0        0        0       0

   week_1  week_2  week_3  week_4  week_5  week_6
0       0       0       0       0       0       0
1       1       0       0       0       0       0
2       0       1       0       0       0       0
3       0       0       1       0       0       0
4       0       0       0       1       0       0

[5 rows x 22 columns]
```

Call the feature vectors X

[28]:
```
X=training_data.values
plt.figure(figsize=(16,8))
plt.subplot(131)
plt.plot(X[:,0],y[:],'r.')
plt.xlabel("No. of sites")
plt.ylabel("No. of requests")
plt.subplot(132)
plt.plot((X[:,1]+X[:,2])/2.0,y[:],'r.')
plt.xlabel("Mean temperature")
plt.ylabel("No. of requests")
plt.subplot(133)
plt.plot(X[:,3],y[:],'r.')
plt.xlabel("Precipitation")
plt.ylabel("No. of requests")
plt.show()
```



17 Task 1: Feature Engineering

It appeats that the no of requests has some kind of a quadratic dependence on the mean temperature so in addition to max and min temperature we should construct a new feature $((minx+maxx)/2)^2$. Feature engineering is one way of brining in domain knowledge or experience into the analysis. Add a new feature vector to the X matrix

[30]:
```
X = np.column_stack((X, ((X[:,1]-X[:,2])/2)**2))
```

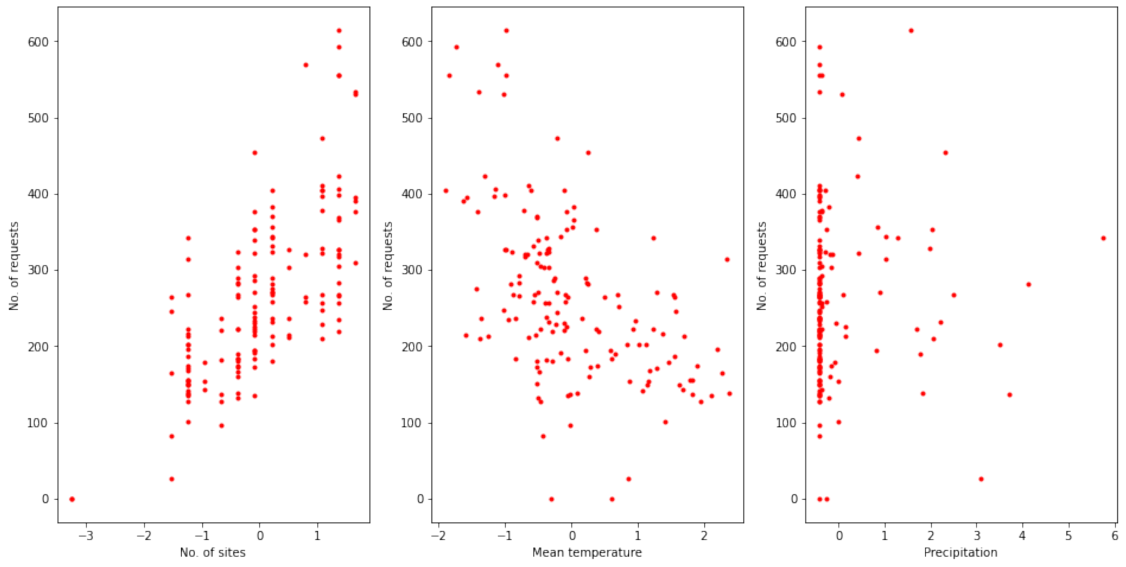
18 Splitting the data into training and test set

Generally we can always tune the hyperparameters so much that the model performs well on the data but it fails to generalize on the unseen data. To avoid this we split the available data into traing and test sets. We want somewhat similar performance on both the sets.

19 Task 2: Split the data into training and test and call them

X_train, X_test, y_train, y_test

[31]:
```
X_norm, mu, sigma = featureNormalize(X)
plt.figure(figsize=(16,8))
plt.subplot(131)
plt.plot(X_norm[:,0],y[:],'r.')
plt.xlabel("No. of sites")
plt.ylabel("No. of requests")
plt.subplot(132)
plt.plot((X_norm[:,1]+X_norm[:,2])/2.0,y[:],'r.')
plt.xlabel("Mean temperature")
plt.ylabel("No. of requests")
plt.subplot(133)
plt.plot(X_norm[:,3],y[:],'r.')
plt.xlabel("Precipitation")
plt.ylabel("No. of requests")
plt.show()
np.mean(X_norm[:,2])



X_train = X_norm[:100]
X_test = X_norm[100:160]
y_train = y[:100]
y_test = y[100:160]
```



20 Task 3: Conduct a MLR on the dataset and report MSE both on the training and the test data

[32]:
```
lr = linear_model.LinearRegression()
lr.fit(X_train,y_train)
print("Theta calculated by SK-learn regression ",lr.coef_)

y_pred_train = lr.predict(X_train)
y_pred_test = lr.predict(X_test)
```

```python
mse_test = np.square((y_pred_test-y_test)).mean()
print("MSE for testing set is ", mse_test)

mse_train = np.square((y_pred_train-y_train)).mean()
print("MSE for training set is ", mse_train)
```

```
Theta calculated by SK-learn regression  [ 6.59669680e+01 -1.89000501e+02
1.69483236e+02 -7.42193916e+00
 -1.87326707e+01  1.87326707e+01 -7.09141760e-01 -1.74801388e+00
  5.11590770e-13 -4.17828763e-01 -1.09958691e+01  2.94266656e+00
  6.68295513e-01  5.25237957e+00  1.35897134e+01 -9.36358741e+00
 -1.21231038e+01 -1.52724091e+00 -6.73754307e+00  4.00044838e+00
  1.71786687e+01  8.66385753e+00  2.50098393e+01  2.50098393e+01]
MSE for testing set is  6055.432535349294
MSE for training set is  1754.3156281012211
```

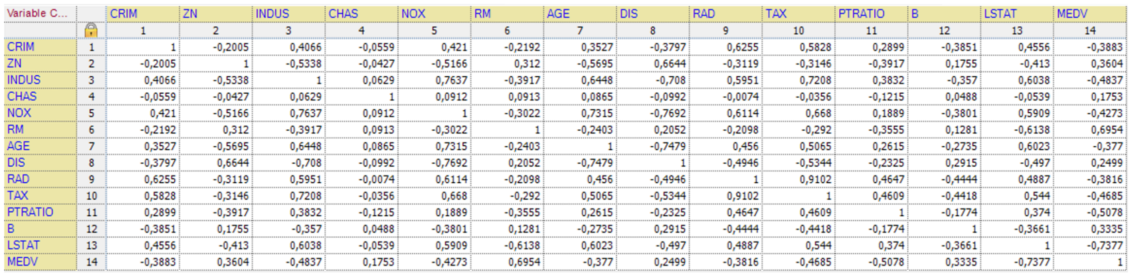
[ ]:
```python
# Unscrambler
```

21 MLR on housing data in unscrambler

21.1 Correlation matrix: Are the correlations as expected?

Yes, the correlations are as expected. Here are some examples:

- The amount of crime is negativly correlated with the average number of rooms per house.
- The NOX concentration is positivly correlated with the proportion of non-retail business per acres.
- The distance form five Boston employment centers are negativly correlated with the amount of industry in the area.



Variable C...		CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
CRIM	1	1	-0,2005	0,4066	-0,0559	0,421	-0,2192	0,3527	-0,3797	0,6255	0,5828	0,2899	-0,3851	0,4556	-0,3883
ZN	2	-0,2005	1	-0,5338	-0,0427	-0,5166	0,312	-0,5695	0,6644	-0,3119	-0,3146	-0,3917	0,1755	-0,413	0,3604
INDUS	3	0,4066	-0,5338	1	0,0629	0,7637	-0,3917	0,6448	-0,708	0,5951	0,7208	0,3832	-0,357	0,6038	-0,4837
CHAS	4	-0,0559	-0,0427	0,0629	1	0,0912	0,0913	0,0865	-0,0992	-0,0074	-0,0356	-0,1215	0,0488	-0,0539	0,1753
NOX	5	0,421	-0,5166	0,7637	0,0912	1	-0,3022	0,7315	-0,7692	0,6114	0,668	0,1889	-0,3801	0,5909	-0,4273
RM	6	-0,2192	0,312	-0,3917	0,0913	-0,3022	1	-0,2403	0,2052	-0,2098	-0,292	-0,3555	0,1281	-0,6138	0,6954
AGE	7	0,3527	-0,5695	0,6448	0,0865	0,7315	-0,2403	1	-0,7479	0,456	0,5065	0,2615	-0,2735	0,6023	-0,377
DIS	8	-0,3797	0,6644	-0,708	-0,0992	-0,7692	0,2052	-0,7479	1	-0,4946	-0,5344	-0,2325	0,2915	-0,497	0,2499
RAD	9	0,6255	-0,3119	0,5951	-0,0074	0,6114	-0,2098	0,456	-0,4946	1	0,9102	0,4647	-0,4444	0,4887	-0,3816
TAX	10	0,5828	-0,3146	0,7208	-0,0356	0,668	-0,292	0,5065	-0,5344	0,9102	1	0,4609	-0,4418	0,544	-0,4685
PTRATIO	11	0,2899	-0,3917	0,3832	-0,1215	0,1889	-0,3555	0,2615	-0,2325	0,4647	0,4609	1	-0,1774	0,374	-0,5078
B	12	-0,3851	0,1755	-0,357	0,0488	-0,3801	0,1281	-0,2735	0,2915	-0,4444	-0,4418	-0,1774	1	-0,3661	0,3335
LSTAT	13	0,4556	-0,413	0,6038	-0,0539	0,5909	-0,6138	0,6023	-0,497	0,4887	0,544	0,374	-0,3661	1	-0,7377
MEDV	14	-0,3883	0,3604	-0,4837	0,1753	-0,4273	0,6954	-0,377	0,2499	-0,3816	-0,4685	-0,5078	0,3335	-0,7377	1

21.2 Multiple linear regression (MLR):

21.2.1 ANOVA table:

The ANOVA (Analysis Of Variance) table shows many different massures when it comes to the variance of each variable and between the different variables in the data set. Performed on a regular data set, this table tells how the different variables contribute to the total variance of the data set. And if its the withiin variance of each variable that is the most important or the variance between the mean of the groups.

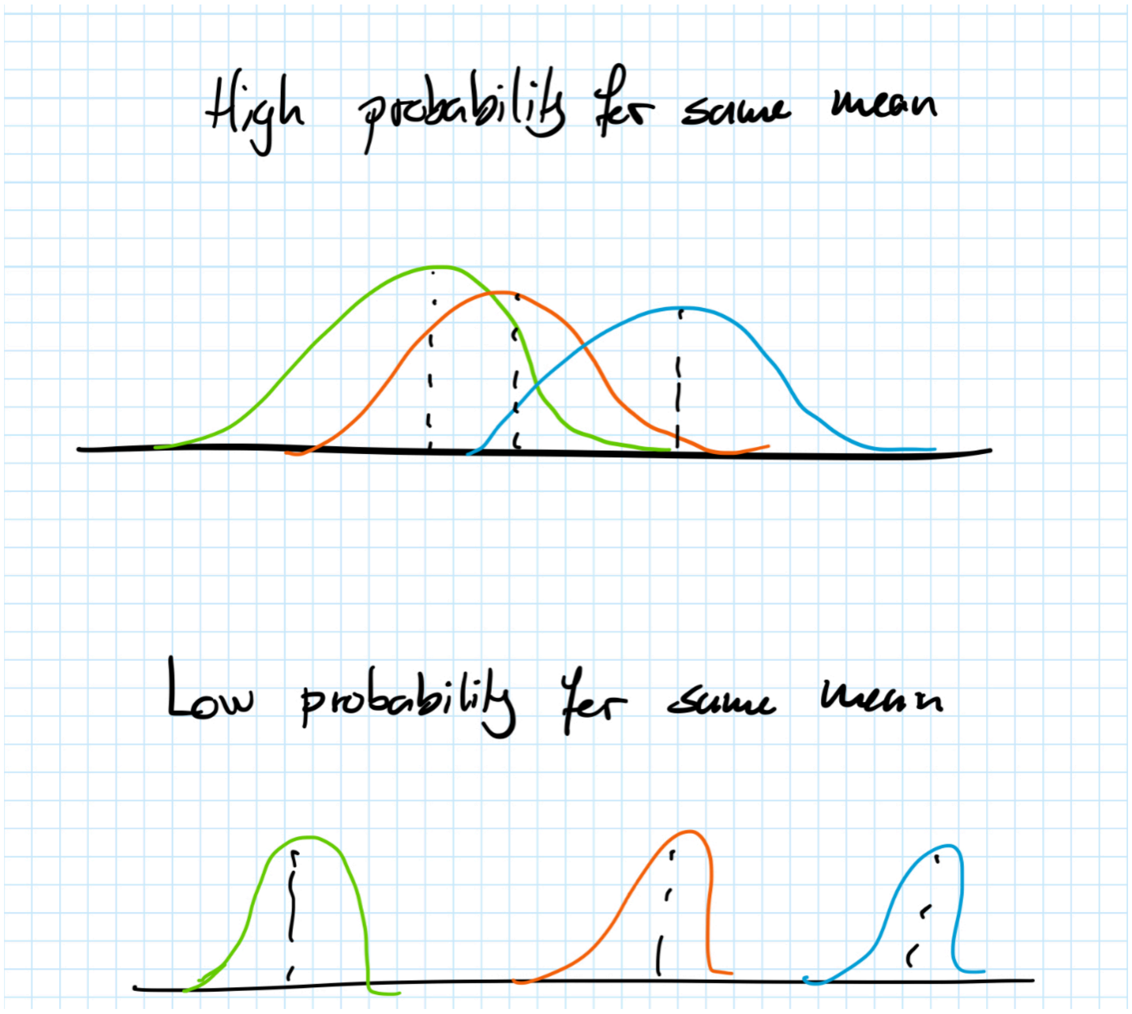
The first column is the SS (Squared Sum) this column consists of different sums of squares describing the variability between different parts of the data. The first row, called model, is the squared sum over the within variance of all different variables. The error row is the squares sum over the mean of the different variables. The last row, asjusted total, is the sum of the above two.

The second column is the degrees of freedom.

The third column is the MSE, this is the squared sum divided by the degrees of freedom. This tells how much the variables varys on average.

The fourth column is the F-ratio. This is a measure of how much of the variance of the model that is caused by variance within each variable compared to the variance between the mean of the different variables. This also exists for each variable and is here a measure of how much variance each variable contributed with compared to the variance between the mean of the different variables. The value in this column is F-distributed.

The fifth column is the p-value. This value is extracted from the F-distributon table. This value gives the probability of that what we hace measured here can happen if all variables has the same mean. In other words: if the variance between differnt the mean of the different variables are big compared to the internal variance of each variable the probability of the variables having the same mean is big. On the other side, if the variables has small variance between the differnt means and big internal variance, the probability for them having the same mean is bigger. This is somewhat explained by the below drawing :)



For the ANOVA table in the MLR the variance is described after the influence of the different thetas (multipliers corresponding to the different variables). Thus the F-ratio of each variable is an indication on how much the variance of the variable is used in the MLR. This is beacause the F-ratio is the squared sum within the variable divided by the squared sum over the different variables. Thus a high F-ratio indicates that the variable is important to the MLR.

This can also be seen in the p-value. In this case the value of the p-value is somewhat a probability for the theta of a variable actually being 0 or the variance of the variable not being important at all. Thus one can conclude that the varinace for the AGE, DIS and RAM all are important for this model. As seen later this also corresponds with them having a higher weighting.

By lastly taking a look at the F-ratio for the whole model this has a value of 174. Thus the variance within the different variables are much more influential than the variance between the mean of the different groups. This also gives a p-value of 0. Thus we can conclude that the model we have made almost certainly explains a pattern between some of the variables and the price.

```
[ ]:
```