1. Simple models for Prediction

Model 1 – Mean sales:

Herhangi bir makine öğrenimi bilginiz olmasa bile, bir ürün için satışları tahmin etmeniz gerekiyorsa, bunun son birkaç günün ortalaması olacağını söyleyebilirsiniz. (ya da hafta ve ay ortalaması)

Başlamak iyi bir fikir ama aynı zamanda bir soruyu da gündeme getiriyor. Fakat bu model ne kadar iyi olabilir?

Modelimizin ne kadar iyi olduğunu değerlendirebileceğimiz çeşitli yollar vardır. En yaygın yol, Ortalama Kare Hatasıdır (Mean Squared Error). Nasıl hesaplandığını inceleyelim.

Prediction error

Bir modelin ne kadar iyi olduğunu değerlendirmek için yanlış tahminlerin etkisini düşünelim. Satışların olabileceklerinden daha yüksek olacağını tahmin edersek, mağaza gereksiz düzenlemeler yapmak için çok para harcayacak ve bu da fazla envantere yol açacaktır. Öte yandan, çok düşük tahmin edersem, satış fırsatını kaybederim.

Hatayı hesaplamanın en basit yolu, tahmin edilen ve gerçek değerler arasındaki farkı hesaplamak olacaktır. Ancak, onları basitçe toplarsak, sıfıra eşitlenebilir, bu yüzden eklemeden önce bu hataların karesini alırız. Ayrıca, ortalama bir hatayı hesaplamak için kareler toplamını veri noktalarının sayısına böleriz.

özetle tahmin edilen değer - gerçek değer = hata tüm hataların karelerinin toplamı / veri noktası sayısı

In [1]:
```
# importing required libraries
import pandas as pd
from sklearn.metrics import mean_squared_error
```

In [2]:
```
# read the train and test dataset
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')
```

In [3]:
```
print(train_data.head(3).T)
```

```
                                        0           1           2
Item_Weight                       6.80000    15.600000    12.911575
Item_Visibility                   0.03749     0.172597     0.054721
Item_MRP                         48.60340   114.851800   107.825400
Outlet_Establishment_Year      2004.00000  1997.000000  1985.000000
Item_Outlet_Sales               291.62040  2163.184200  2387.558800
Item_Fat_Content_LF               0.00000     0.000000     0.000000
Item_Fat_Content_Low Fat          1.00000     1.000000     1.000000
Item_Fat_Content_Regular          0.00000     0.000000     0.000000
Item_Fat_Content_low fat          0.00000     0.000000     0.000000
Item_Fat_Content_reg              0.00000     0.000000     0.000000
Item_Type_Baking Goods            1.00000     0.000000     0.000000
Item_Type_Breads                  0.00000     0.000000     0.000000
Item_Type_Breakfast               0.00000     0.000000     0.000000
Item_Type_Canned                  0.00000     0.000000     0.000000
Item_Type_Dairy                   0.00000     0.000000     0.000000
Item_Type_Frozen Foods            0.00000     0.000000     1.000000
Item_Type_Fruits and Vegetables   0.00000     1.000000     0.000000
Item_Type_Hard Drinks             0.00000     0.000000     0.000000
Item_Type_Health and Hygiene      0.00000     0.000000     0.000000
Item_Type_Household               0.00000     0.000000     0.000000
Item_Type_Meat                    0.00000     0.000000     0.000000
Item_Type_Others                  0.00000     0.000000     0.000000
Item_Type_Seafood                 0.00000     0.000000     0.000000
Item_Type_Snack Foods             0.00000     0.000000     0.000000
Item_Type_Soft Drinks             0.00000     0.000000     0.000000
Item_Type_Starchy Foods           0.00000     0.000000     0.000000
Outlet_Size_High                  0.00000     0.000000     0.000000
Outlet_Size_Medium                0.00000     0.000000     1.000000
Outlet_Size_Small                 1.00000     1.000000     0.000000
Outlet_Location_Type_Tier 1       0.00000     1.000000     0.000000
Outlet_Location_Type_Tier 2       1.00000     0.000000     0.000000
Outlet_Location_Type_Tier 3       0.00000     0.000000     1.000000
Outlet_Type_Grocery Store         0.00000     0.000000     0.000000
Outlet_Type_Supermarket Type1     1.00000     1.000000     0.000000
Outlet_Type_Supermarket Type2     0.00000     0.000000     0.000000
Outlet_Type_Supermarket Type3     0.00000     0.000000     1.000000
```

In [4]:
```
# shape of the dataset
print('\nShape of training data :',train_data.shape)
print('\nShape of testing data :',test_data.shape)
```

```
Shape of training data : (1364, 36)

Shape of testing data : (341, 36)
```

In [5]:
```
# Now, we need to predict the missing target variable in the test data using the
# target variable - Item_Outlet_Sales

train_y = train_data['Item_Outlet_Sales']

test_y = test_data['Item_Outlet_Sales']
```

In [6]:
```
print('Mean of Target Variable : ',train_y.mean())
```

```
Mean of Target Variable :  2237.647877272729
```

In [7]:
```
# predict the target on the test dataset
predict_train = [train_y.mean() for i in range(train_y.shape[0])]

# Root Mean Squared Error on training dataset
rmse_train = mean_squared_error(train_y,predict_train)**(0.5)
print('\nRMSE on train dataset : ', rmse_train)
```

```
RMSE on train dataset :  1768.4984457265218
```

In [8]:
```
# predict the target on the testing dataset
predict_test = [train_y.mean() for i in range(test_y.shape[0])]

# Root Mean Squared Error on testing dataset
rmse_test = mean_squared_error(test_y,predict_test)**(0.5)
print('\nRMSE on test dataset : ', rmse_test)
```

```
RMSE on test dataset :  1528.70790904839
```

Model 2 – Average Sales by Location:

Bir ürünün satışında konumun hayati bir rol oynadığını biliyoruz. Örneğin, Delhi'de araba satışlarının Varanasi'deki satışlarından çok daha yüksek olacağını varsayalım. Bu nedenle, 'Outlet_Location_Type' sütununun verilerini kullanalım.

Yani temel olarak, her bir lokasyon tipi için ortalama satışları hesaplayalım ve buna göre tahmin edelim.

Aynı şeyi tahmin ettiğimizde, önceki durumumuzdan daha az olan mse = 28,75,386 elde ederiz. Böylece bir özellik[konum] kullanarak hatayı azalttığımızı fark edebiliriz.

Şimdi, satışların bağlı olacağı birden fazla özellik varsa ne olur? Bu bilgiyi kullanarak satışları nasıl tahmin ederiz? Lineer regresyon imdadımıza yetişiyor.

2. Linear Regression

Doğrusal regresyon, tahmine dayalı modelleme için en basit ve en yaygın kullanılan istatistiksel tekniktir. Temel olarak bize, hedef (dependent variables) değişkenimizin (Bizim için verimizde bu değer "sales") bağlı olduğu bağımsız değişkenler (independent variables) olarak adlandırılan özelliklerimize (features) sahip olduğumuz bir denklem verir.

Peki denklem neye benziyor? Doğrusal regresyon denklemi şöyle görünür

$$Y = \theta_1 X_1 + \theta_2 X_2 + \dots \theta_n X_n$$

Burada bağımlı değişkenimiz (dependent variable - Sales) Y ile gösteriliyor, X değerleri ise bağımsız değişkenleri (independent variables) temsil ediyor ve tüm teta sembolleri ise katsayıları ifade etmektedir. Katsayılar (Teta ile gösterilen değerler) temel olarak özelliklerin önemlerine göre atanan sayısal değerlerdir. Örneğin, bir öğenin satışının mağazanın boyutuna kıyasla konum türüne daha fazla bağımlı olacağına inanıyorsak, yer türü katsayısı mağaza büyüklüğü katsayısından daha fazla olacaktır.

O halde, öncelikle lineer regresyonu tek bir öznitelikle, yani tek bir bağımsız değişkenle anlamaya çalışalım. Bu nedenle denklemimiz şu şekilde olur,

$$Y = \theta_1 * X + \theta_0$$

Bu denklem, düz bir çizgiyi temsil eden basit bir lineer regresyon denklemi olarak adlandırılır; burada Teta sıfır kesişim, Teta-1 ise doğrunun eğimidir. Sales ve MRP arasındaki aşağıdaki çizime bir göz atın.

In [9]:
```
sns.regplot(x="Item_MRP",y="Item_Outlet_Sales",data=test_data,ci=None)
```

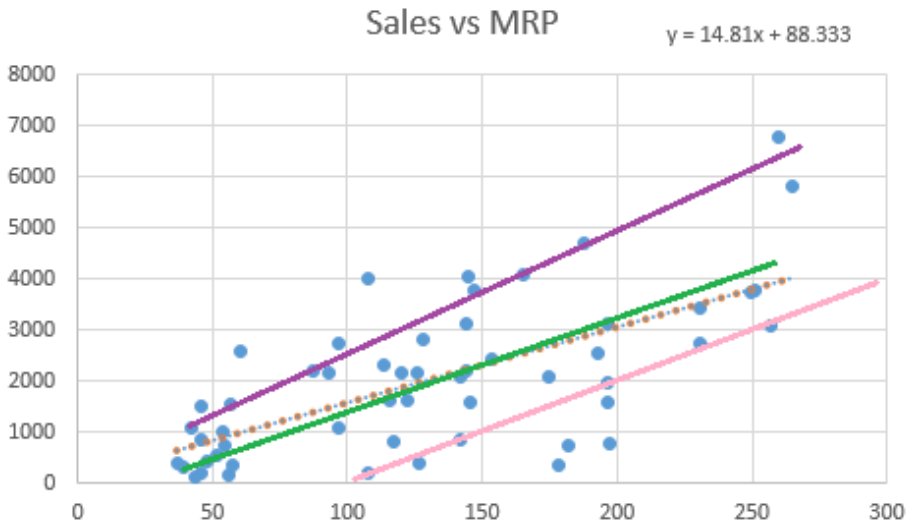
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-dc41d5cb9db2> in <module>
----> 1 sns.regplot(x="Item_MRP",y="Item_Outlet_Sales",data=test_data,ci=None)

NameError: name 'sns' is not defined
```

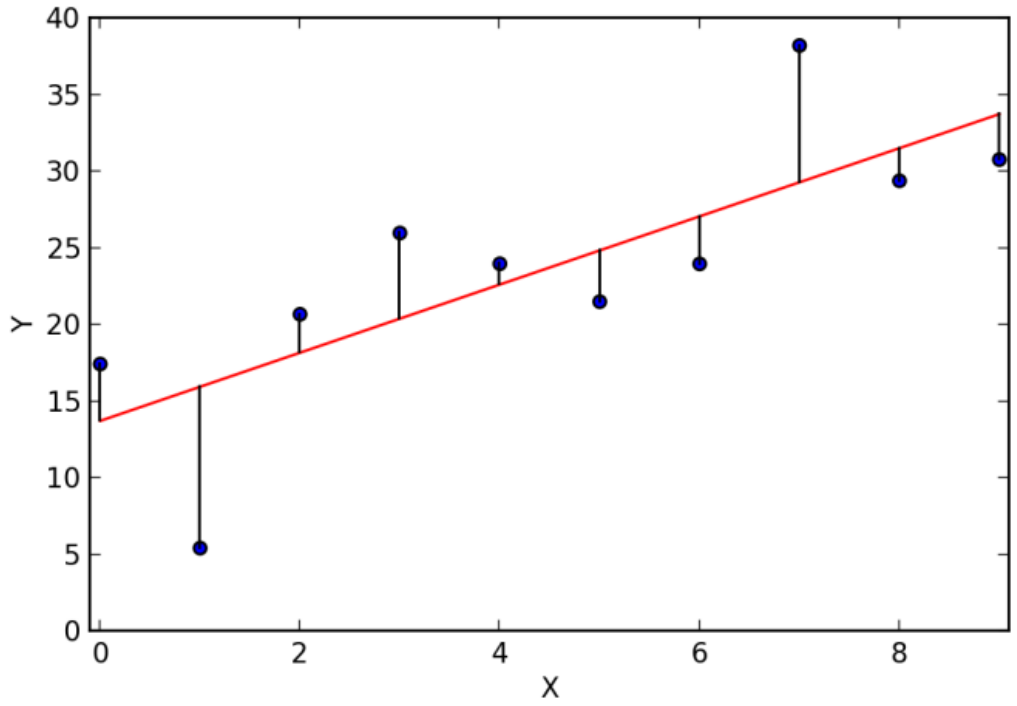
Şaşırtıcı bir şekilde, bir ürünün satışlarının MRP'sindeki artışla arttığını görebiliriz. Bu nedenle mavi çizgi, regresyon çizgimizi veya en uygun çizgiyi temsil eder. Ancak ortaya çıkan bir soru, bu çizgi nasıl çiziliyor?

3. The Line of Best Fit

Aşağıda ki grafikte görebileceğiniz gibi, MRP'lerine göre Satışları tahmin etmek için kullanılabilecek çok sayıda satır olabilir. Peki en uygun çizgiyi veya regresyon çizgisini nasıl seçersiniz?



En uygun çizginin temel amacı, tahmin edilen değerlerimizin gerçek veya gözlenen değerlerimize daha yakın olması gerektiğidir, çünkü gerçek değerlerden uzak değerleri tahmin etmenin bir anlamı yoktur. Başka bir deyişle, bizim tarafımızdan tahmin edilen değerler ile gözlemlenen değerler arasındaki ve aslında hata olarak adlandırılan farkı en aza indirme eğilimindeyiz. Hatanın grafiksel gösterimi aşağıda gösterildiği gibidir. Bu hatalara "residuals" da denir. Residuals, tahmin edilen ve gerçek değer arasındaki farkı gösteren dikey çizgilerle gösterilir.



Tamam, artık biliyoruz ki asıl amacımız hatayı bulup minimize etmek. Ama ondan önce, ilk kısımla nasıl başa çıkacağımızı, yani hatayı nasıl hesaplayacağımızı düşünelim. Hatanın bizim tarafımızdan tahmin edilen değer ile gözlemlenen değer arasındaki fark olduğunu zaten biliyoruz. Hatayı hesaplayabileceğimiz üç yolu düşünelim:

- Sum of residuals (∑(Y – h(X)))
- Sum of the absolute value of residuals (∑|Y-h(X)|)
- Sum of square of residuals (∑ (Y-h(X))2) - pratikte en çok kullanılan yöntemdir, çünkü burada daha küçük olana kıyasla daha yüksek hata değerini çok daha fazla cezalandırıyoruz, bu nedenle büyük hatalar ve küçük hatalar arasında önemli bir fark var, bu da ayırt etmeyi ve en uygun satırı seçmeyi kolaylaştırıyor .

Bu nedenle, bu artıkların karelerinin toplamı şu şekilde gösterilir:

$$SS_{residuals} = \sum_{i=1}^{m} (h(x) - y)^2$$

burada h(x) bizim tarafımızdan tahmin edilen değerdir, h(x) =Θ1*x +Θ0 , y gerçek değerlerdir ve m eğitim kümesindeki satır sayısıdır.

The cost Function

Diyelim ki, satışların daha yüksek olacağını tahmin ettiğiniz belirli bir mağazanın boyutunu büyüttünüz. Ama boyutu büyütmesine rağmen o dükkandaki satışlar o kadar artmadı. Yani dükkânı büyütmek için uygulanan maliyet, size olumsuz sonuçlar verdi.

Dolayısıyla bu maliyetleri en aza indirmemiz gerekiyor. Bu nedenle, temel olarak modelin hatasını tanımlamak ve ölçmek için kullanılan bir maliyet fonksiyonu sunuyoruz.

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Bu denkleme dikkatlice bakarsanız, matematiği kolaylaştırmak için sadece 1/2 m'lik bir faktörle çarpılarak, hataların karelerinin toplamına benzer.

Bu yüzden tahminimizi iyileştirmek için maliyet fonksiyonunu minimize etmemiz gerekiyor. Bu amaçla gradyan iniş algoritmasını kullanıyoruz. Öyleyse nasıl çalıştığına bir bakalım.

4. Gradient Descent

Bir örnek ele alalım, bu denklemin minimum değerini bulmamız gerekiyor,

Y= 5x + 4x^2. Matematikte, bu denklemin x'e göre türevini alırız, basitçe onu sıfıra eşitleriz. Bu bize bu denklemin minimum olduğu noktayı verir. Bu nedenle, bu değeri yerine koymak bize bu denklemin minimum değerini verebilir.

Gradyan inişi benzer şekilde çalışır. Maliyet fonksiyonunun minimum olacağı bir noktayı bulmak için Θ'yi yinelemeli olarak günceller. Gradyan inişini derinlemesine incelemek istiyorsanız, bu makaleyi (https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/) incelemenizi şiddetle tavsiye ederim.

5. Using Linear Regression for Prediction

Şimdi, büyük market satış problemimiz için Satışları tahmin etmek için Doğrusal Regresyon kullanmayı düşünelim.

Model 3 – Enter Linear Regression:

Yukarda bahsettiğimiz üzere, doğru özellikleri kullanmanın tahminlerimizin doğruluğunu artıracağını biliyoruz. Şimdi satışları tahmin etmek için MRP ve mağaza kuruluş yılı (store establishment year) olmak üzere iki özelliği kullanalım.

Şimdi sadece bu iki özelliği göz önünde bulundurarak python'da bir lineer regresyon modeli oluşturalım.

In [10]:
```
# importing basic libraries
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
from sklearn.model_selection import train_test_split

train = pd.read_csv('Train.csv')
test = pd.read_csv('test.csv')
```

In [11]:
```
# importing linear regressionfrom sklearn
from sklearn.linear_model import LinearRegression
lreg = LinearRegression()

# splitting into training and cv for cross validation
X = train.loc[:,['Outlet_Establishment_Year','Item_MRP']]
x_train, x_cv, y_train, y_cv = train_test_split(X,train.Item_Outlet_Sales)

# training the model
lreg.fit(x_train,y_train)

# predicting on cv
pred = lreg.predict(x_cv)

# calculating mse
mse = np.mean((pred - y_cv)**2)
```

Bu durumda, model 2'den çok daha küçük olan mse = 19,10,586.53 elde ettik. Bu nedenle, iki öznitelik yardımıyla tahmin yapmak çok daha doğrudur.

Bu lineer regresyon modelinin katsayılarına bir göz atalım.

In [12]:
```
# calculating coefficients

coeff = DataFrame(x_train.columns)
coeff['Coefficient Estimate'] = Series(lreg.coef_)
coeff
```

Out[12]:

	0	Coefficient Estimate
0	Outlet_Establishment_Year	-16.025853
1	Item_MRP	16.449667

MRP'nin yüksek bir katsayıya sahip olduğunu görebiliyoruz, yani daha yüksek fiyatlara sahip kalemlerin daha iyi satışları var.

6. Evaluating your Model – R square and adjusted R- square

Sizce model ne kadar doğru? Bunu kontrol edebilmemiz için herhangi bir değerlendirme metriğimiz var mı? Aslında R-Kare olarak bilinen bir niceliğimiz var.

R-Square: Y'deki (*dependent variable*) toplam varyasyonun ne kadarının X'teki (*independent variable*) varyasyonla açıklandığını belirler. Matematiksel olarak şu şekilde yazılabilir:

$$R - Square = 1 - \frac{\sum(Y_{actual} - Y_{predicted})\text{^}2}{\sum(Y_{actual} - Y_{mean})\text{^}2}$$

R-square değeri her zaman 0 ile 1 arasındadır; burada 0, modelin hedef değişkendeki (Y) herhangi bir kararsızlığı (variability) açıklamadığı anlamına gelir ve 1, hedef değişkendeki tam kararsızlığı (variability) açıkladığı anlamına gelir.

Şimdi yukarıdaki model için r-karesini kontrol edelim.

In [13]:
```
lreg.score(x_cv,y_cv)
```

Out[13]:
```
0.3725373936516341
```

Bu durumda R^2 %32'dir, yani satışlardaki varyansın sadece %32'si kuruluş yılı (Outlet_Establishment_Year) ve MRP (Item_MRP) ile açıklanmaktadır. Başka bir deyişle, kuruluş yılını ve MRP'yi biliyorsanız, satışları hakkında doğru bir tahminde bulunmak için %32 bilginiz olur.

Şimdi modelime bir özellik daha eklesem ne olur, modelim değerleri gerçek değerine daha yakın tahmin eder mi? R-Square'in değeri artacak mı?

Başka bir vakayı ele alalım.

Model 4 – Linear regression with more variables

Bir yerine iki değişken kullanarak, ürün satışları hakkında doğru tahminlerde bulunma becerisini geliştirdiğimizi öğrendik.

O halde, 3 numaralı durumda başka bir özelliği yani 'weight' tanıtalım. Şimdi bu üç özellik ile bir regresyon modeli oluşturalım.

In [14]:
```
X = train.loc[:,['Outlet_Establishment_Year','Item_MRP','Item_Weight']]

#splitting into training and cv for cross validation

x_train, x_cv, y_train, y_cv = train_test_split(X,train.Item_Outlet_Sales)

## training the model

lreg.fit(x_train,y_train)
```

Out[14]:
```
LinearRegression()
```

In [15]:
```
# training the model
lreg.fit(x_train,y_train)

# splitting into training and cv for cross validation
x_train, x_cv, y_train, y_cv = train_test_split(X,train.Item_Outlet_Sales)

# training the model
lreg.fit(x_train,y_train)

# predicting on cv
pred = lreg.predict(x_cv)

# calculating mse
mse = np.mean((pred - y_cv)**2)

mse
```

Out[15]:
```
2240927.089764429
```

In [16]:
```
## calculating coefficients

coeff = DataFrame(x_train.columns)
coeff['Coefficient Estimate'] = Series(lreg.coef_)

coeff
```

Out[16]:

	0	Coefficient Estimate
0	Outlet_Establishment_Year	-19.279751
1	Item_MRP	17.204334
2	Item_Weight	-9.365385

In [17]:
```
# calculating r-square

lreg.score(x_cv,y_cv)
```

Out[17]:
```
0.24654621683098
```

Bu nedenle mse'nin daha da azaldığını görebiliriz. R-kare değerinde bir artış olması, madde ağırlığının eklenmesinin modelimiz için faydalı olduğu anlamına mı geliyor?

Adjusted R-square

R2'nin tek dezavantajı, modelimize yeni tahminciler (X) eklenirse, R2'nin yalnızca artması veya sabit kalması, ancak asla azalmamasıdır. Modelimizin karmaşıklığını artırarak onun daha mı doğru hale getirdiğimizi yargılayamayız.

Bu yüzden “Adjusted R-Square” kullanıyoruz.

Adjusted R-Square, modeldeki tahmin edicilerin sayısı için ayarlanmış olan değiştirilmiş R-Kare biçimidir. Modelin serbestlik derecesini içerir. Adjusted R-Square, yalnızca yeni terim model doğruluğunu iyileştirdiğinde artar.

$$R^2 \text{ adjusted} = 1 - \frac{(1-R^2)(N-1)}{N-p-1}$$

- R2 = Örnek R kare
- p = Öngörücülerin sayısı
- N = toplam örnek boyutu

7. Using all the features for prediction

Şimdi tüm özellikleri içeren bir model oluşturalım. Regresyon modellerini oluştururken sadece sürekli öznitelikler (continuous features) kullandım. Bunun nedeni, kategorik değişkenleri lineer regresyon modelinde kullanılmadan önce farklı şekilde ele almamız gerektiğidir. Onları tedavi

etmek için farklı teknikler var, burada bir sıcak kodlama kullandım (kategorik bir değişkenin her sınıfını bir özellik olarak dönüştürün). Bunun dışında, çıkış boyutu için eksik değerleri de belirledim.

Data pre-processing steps for regression model

In [18]:
```
train.head().T
```

Out[18]:

	0	1	2	3	4
Item_Weight	6.80000	15.600000	12.911575	11.800000	17.8500
Item_Visibility	0.03749	0.172597	0.054721	0.098312	0.0466
Item_MRP	48.60340	114.851800	107.825400	81.461800	125.1388
Outlet_Establishment_Year	2004.00000	1997.000000	1985.000000	1998.000000	2004.0000
Item_Outlet_Sales	291.62040	2163.184200	2387.558800	161.123600	1981.4208
Item_Fat_Content_LF	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Fat_Content_Low Fat	1.00000	1.000000	1.000000	1.000000	0.0000
Item_Fat_Content_Regular	0.00000	0.000000	0.000000	0.000000	1.0000
Item_Fat_Content_low fat	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Fat_Content_reg	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Baking Goods	1.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Breads	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Breakfast	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Canned	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Dairy	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Frozen Foods	0.00000	0.000000	1.000000	0.000000	0.0000
Item_Type_Fruits and Vegetables	0.00000	1.000000	0.000000	0.000000	1.0000
Item_Type_Hard Drinks	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Health and Hygiene	0.00000	0.000000	0.000000	1.000000	0.0000
Item_Type_Household	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Meat	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Others	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Seafood	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Snack Foods	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Soft Drinks	0.00000	0.000000	0.000000	0.000000	0.0000
Item_Type_Starchy Foods	0.00000	0.000000	0.000000	0.000000	0.0000
Outlet_Size_High	0.00000	0.000000	0.000000	0.000000	0.0000
Outlet_Size_Medium	0.00000	0.000000	1.000000	0.000000	0.0000
Outlet_Size_Small	1.00000	1.000000	0.000000	0.000000	1.0000
Outlet_Location_Type_Tier 1	0.00000	1.000000	0.000000	0.000000	0.0000
Outlet_Location_Type_Tier 2	1.00000	0.000000	0.000000	0.000000	1.0000
Outlet_Location_Type_Tier 3	0.00000	0.000000	1.000000	1.000000	0.0000
Outlet_Type_Grocery Store	0.00000	0.000000	0.000000	1.000000	0.0000
Outlet_Type_Supermarket Type1	1.00000	1.000000	0.000000	0.000000	1.0000
Outlet_Type_Supermarket Type2	0.00000	0.000000	0.000000	0.000000	0.0000
Outlet_Type_Supermarket Type3	0.00000	0.000000	1.000000	0.000000	0.0000

Building the model

In [19]:
```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import matplotlib.pyplot as plt
%matplotlib inline

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

In [20]:
```
# importing linear regression
from sklearn.linear_model import LinearRegression
lreg = LinearRegression()
```

In [21]:
```
# for cross validation
from sklearn.model_selection import train_test_split

X = train.drop('Item_Outlet_Sales',1)
x_train, x_cv, y_train, y_cv = train_test_split(X,train.Item_Outlet_Sales, test_
```

In [22]:
```
# training a linear regression model on train
lreg.fit(x_train,y_train)
```

Out[22]:
```
LinearRegression()
```

In [23]:
```
# predicting on cv
pred_cv = lreg.predict(x_cv)
```

In [24]:
```
# calculating mse
mse = np.mean((pred_cv - y_cv)**2)
mse
```

Out[24]:
```
1468499.2681441572
```

In [25]:
```
# evaluation using r-square
lreg.score(x_cv,y_cv)
```

Out[25]:
```
0.5460963766280733
```

Açıkça, hem mse hem de R-kare'de büyük bir gelişme olduğunu görebiliyoruz, bu da modelimizin artık gerçek değerlere çok daha yakın değerleri tahmin edebildiği anlamına geliyor.

Selecting the right features for your model

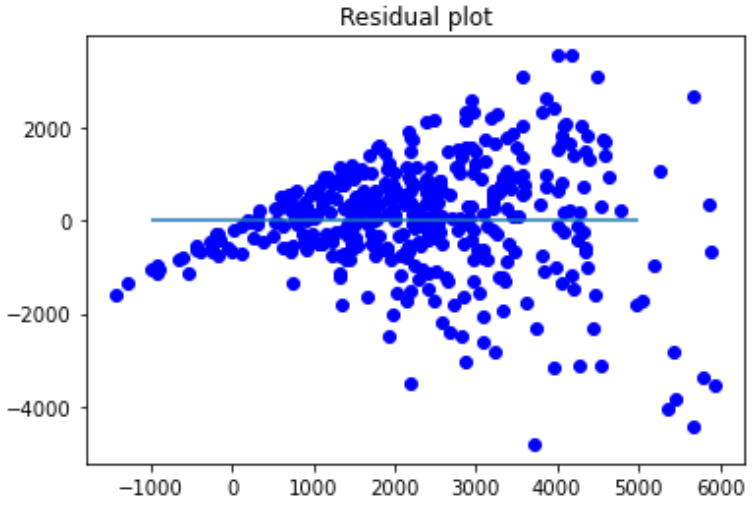
Yüksek boyutlu bir veri setimiz olduğunda, bazıları gereksiz bilgi verebildiğinden tüm değişkenleri kullanmak oldukça verimsiz olacaktır. Bize doğru bir model veren ve bağımlı değişkeni iyi açıklayabilen doğru değişken setini seçmemiz gerekir. Model için doğru değişken kümesini seçmenin birden çok yolu vardır. Bunların başında işi anlamak ve alan bilgisi gelir. Örneğin, satışları tahmin ederken, pazarlama çabalarının satışları olumlu yönde etkilemesi gerektiğini biliyoruz ve bu, modelinizde önemli bir özelliktir. Ayrıca seçeceğimiz değişkenlerin kendi aralarında ilişkili olmamasına da dikkat etmeliyiz.

Değişkenleri manuel olarak seçmek yerine, ileri veya geri seçimi kullanarak bu işlemi otomatikleştirebiliriz. İleriye doğru seçim, modeldeki en önemli tahmin edici ile başlar ve her adım için değişken ekler. Geriye doğru eleme, modeldeki tüm tahmin edicilerle başlar ve her adım için en az anlamlı değişkeni kaldırır. Seçim kriterleri, R-kare, t-stat vb. gibi herhangi bir istatistiksel ölçüye ayarlanabilir.

Interpretation of Regression Plots

In [26]:
```
# residual plot
x_plot = plt.scatter(pred_cv, (pred_cv - y_cv), c='b')
plt.hlines(y=0, xmin= -1000, xmax=5000)
plt.title('Residual plot')
```

Out[26]:
```
Text(0.5, 1.0, 'Residual plot')
```



Plot'ta huni benzeri bir şekil görebiliriz. Bu şekil **Heteroskedastisiteyi** gösterir. Hata terimlerinde sabit olmayan varyansın varlığı, değişen varyansla sonuçlanır. Hata terimlerinin (residuals) varyansının sabit olmadığını açıkça görebiliriz. Genel olarak, sabit olmayan varyans, aykırı değerlerin veya aşırı kaldıraç değerlerinin varlığında ortaya çıkar. Bu değerler çok fazla ağırlık alarak modelin performansını orantısız bir şekilde etkiler. Bu fenomen meydana geldiğinde, örneklem dışı tahmin için güven aralığı gerçekçi olamayacak kadar geniş veya dar olma eğilimindedir.

Bunu, **residual** değerlere karşı **fitted** değerler grafiğine bakarak kolayca kontrol edebiliriz. Eğer heteroskedastisite mevcutsa, grafik, yukarıda gösterildiği gibi bir huni şekli modeli sergileyecektir. Bu, model tarafından yakalanmayan verilerde doğrusal olmama işaretlerini gösterir. Varsayımların ayrıntılı bir şekilde anlaşılması ve regresyon grafiklerinin yorumlanması için [bu makaleyi (https://www.analyticsvidhya.com/blog/2016/07/deeper-regression-analysis-assumptions-plots-solutions/)](https://www.analyticsvidhya.com/blog/2016/07/deeper-regression-analysis-assumptions-plots-solutions/) incelemenizi şiddetle tavsiye ederim.

Bu doğrusal olmayan etkileri yakalamak için polinom regresyonu olarak bilinen başka bir regresyon tipine sahibiz. Öyleyse şimdi onu anlayalım.

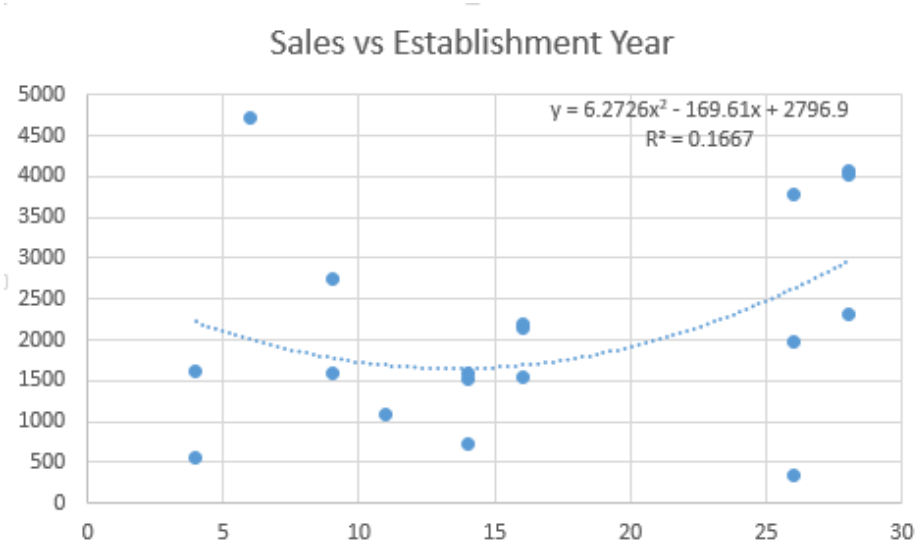
8. Polynomial Regression

Polinom regresyon, bağımsız değişkenin maksimum gücünün 1'den büyük olduğu bir başka regresyon şeklidir. Bu regresyon tekniğinde en uygun doğru, düz bir doğru değil, bir eğri şeklindedir.

İkinci dereceden regresyon veya ikinci dereceden polinomlu regresyon, aşağıdaki denklemle verilir:

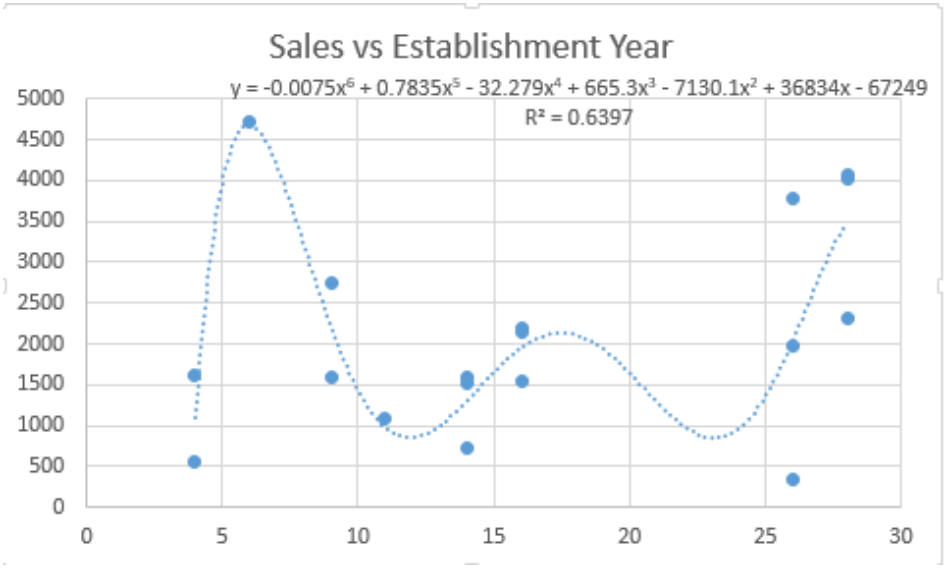
Y =Θ1 +Θ2*x* +Θ3x2

Şimdi aşağıda verilen şemaya bir göz atın.

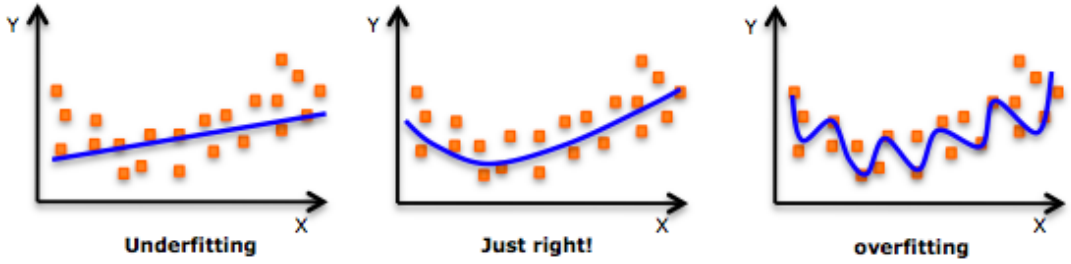


Açıkça ikinci dereceden denklem, verilere basit doğrusal denklemden daha iyi uyuyor. Bu durumda, ikinci dereceden regresyonun R-kare değerinin basit lineer regresyondan daha büyük olacağını düşünüyorsunuz? Kesinlikle evet, çünkü ikinci dereceden regresyon, verilere lineer regresyondan daha iyi uyuyor. İkinci dereceden ve kübik polinomlar yaygın olmakla birlikte, daha yüksek dereceli polinomlar da ekleyebilirsiniz.

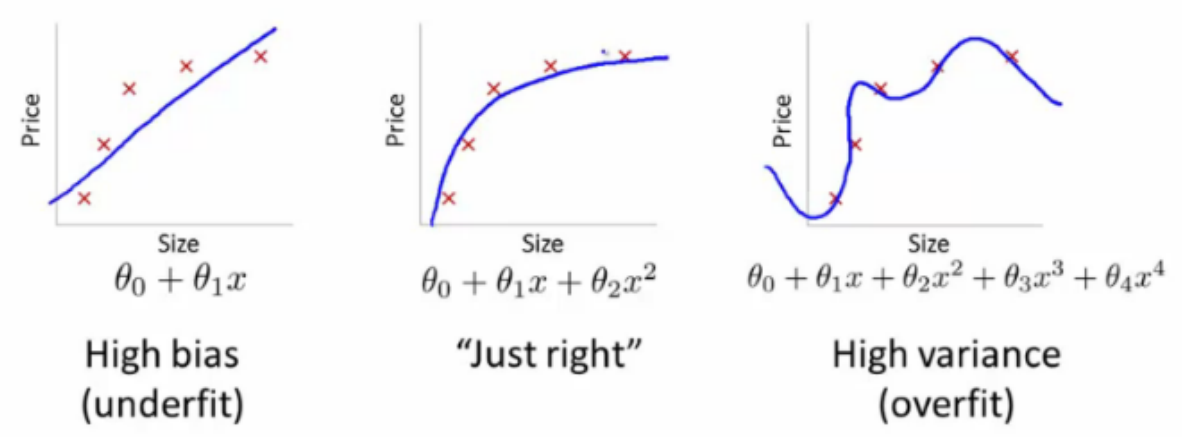
Aşağıdaki şekil 6. dereceden bir polinom denkleminin davranışını göstermektedir.



Öyleyse, veri kümesine uyması için daha yüksek dereceli polinomları kullanmanın her zaman daha iyi olduğunu düşünüyor musunuz? Üzgünüm hayır. Temel olarak, eğitim verilerimize iyi uyan ancak eğitim setinin ötesindeki değişkenler arasındaki gerçek ilişkiyi tahmin edemeyen bir model oluşturduk. Bu nedenle modelimiz test verilerinde düşük performans gösteriyor. Bu soruna **aşırı uydurma (over-fitting)** denir. Ayrıca modelin yüksek varyansa ve düşük yanlılığa sahip olduğunu söylüyoruz.



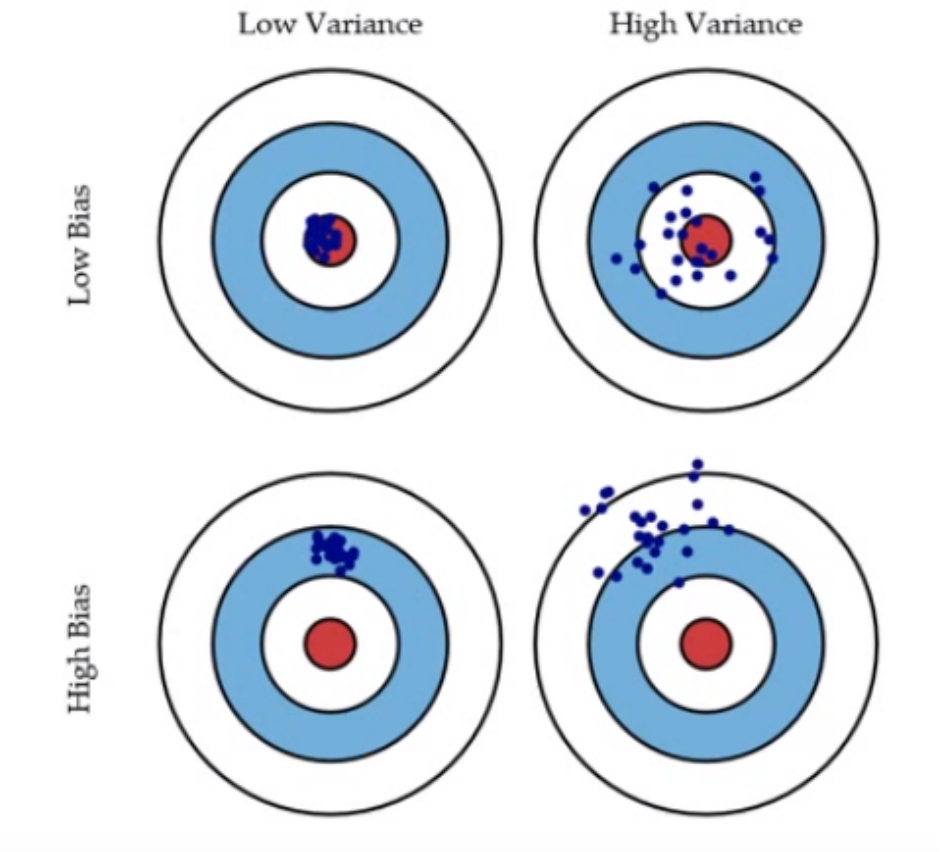
Benzer şekilde, **yetersiz uyum (underfitting)** adı verilen başka bir sorunumuz var, modelimiz eğitim verilerine uymadığında veya yeni veriler üzerinde genelleme yapmadığında ortaya çıkıyor.



Yüksek önyargı ve düşük varyansa sahip olduğumuzda modelimiz yetersizdir.

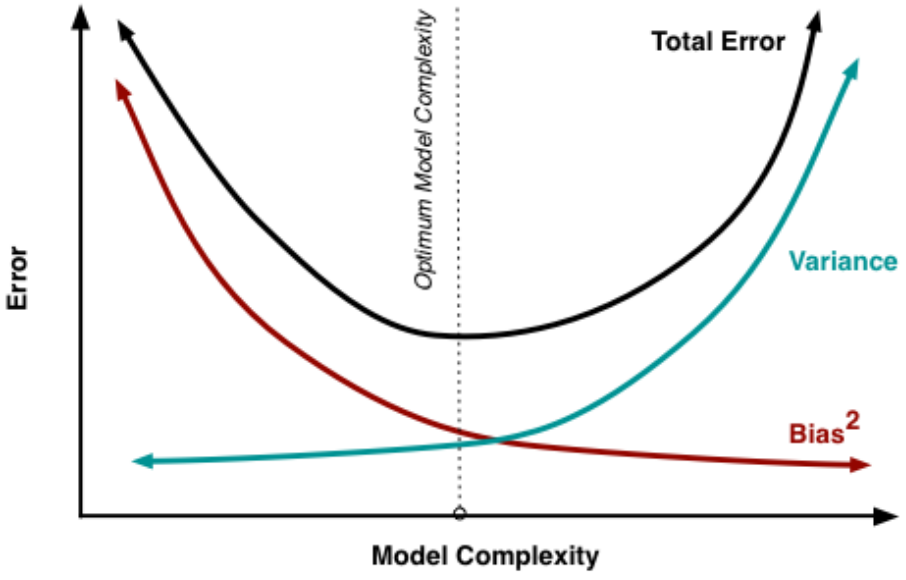
9. Bias and Variance in regression models

Bu önyargı ve varyans aslında ne anlama geliyor? Bunu bir okçuluk hedefi örneği ile anlayalım.



Diyelim ki çok doğru bir modelimiz var, bu nedenle modelimizin hatası düşük olacak, bu da ilk şekilde gösterildiği gibi düşük bir sapma ve düşük varyans anlamına geliyor. Tüm veri noktaları, boğa gözüne sığar. Benzer şekilde, varyans artarsa, veri noktamızın yayılmasının arttığını ve bunun daha az doğru tahminle sonuçlandığını söyleyebiliriz. Ve önyargı arttıkça, tahmin edilen değerimiz ile gözlemlenen değerler arasındaki hata artar.

Şimdi bu önyargı ve varyans, mükemmel bir modele sahip olmak için nasıl dengeleniyor? Aşağıdaki resme bir bakın ve anlamaya çalışın.



Modelimize daha fazla parametre ekledikçe, karmaşıklığı artar, bu da artan varyans ve azalan yanlılık, yani aşırı uyum ile sonuçlanır. Bu yüzden, sapmadaki azalmanın varyanstaki artışa eşit olduğu modelimizde bir optimum nokta bulmamız gerekiyor. Pratikte bu noktayı bulmanın analitik bir yolu yoktur. Peki yüksek varyans veya yüksek önyargı ile nasıl başa çıkılır?

Underfitting veya **yüksek sapmanın (high bias)** üstesinden gelmek için, model karmaşıklığının artması ve böylece yüksek sapmanın azaltılması için temel olarak modelimize yeni parametreler ekleyebiliriz.

Şimdi, bir regresyon modeli için Overfitting'in üstesinden nasıl gelebiliriz?

Temel olarak overfitting üstesinden gelmek için iki yöntem vardır,

- Model karmaşıklığını azaltın
- Düzenlileştirme (Regularization)

Burada, Düzenlileştirme hakkında ayrıntılı olarak ve modelinizi daha genel hale getirmek için nasıl kullanılacağını tartışıyor olacağız.

10. Regularization

Modeliniz hazır, çıktınızı tahmin ettiniz. Öyleyse neden düzenlileştirmeyi incelemeniz gerekiyor? Bu gerekli mi?

Diyelim ki bir yarışmaya katıldınız ve bu problemde sürekli bir değişkeni tahmin etmeniz gerekiyor. Böylece doğrusal regresyon uyguladınız ve çıktınızı tahmin ettiniz. İşte! Skor tablosundasın. Ama bekle, hala lider panosunda senin üstünde birçok insan olduğunu görüyorsun. Ama her şeyi doğru yaptın, o zaman bu nasıl mümkün olabilir?

> "Her şey mümkün olduğunca basitleştirilmeli, ama daha basit değil. - Albert Einstein"

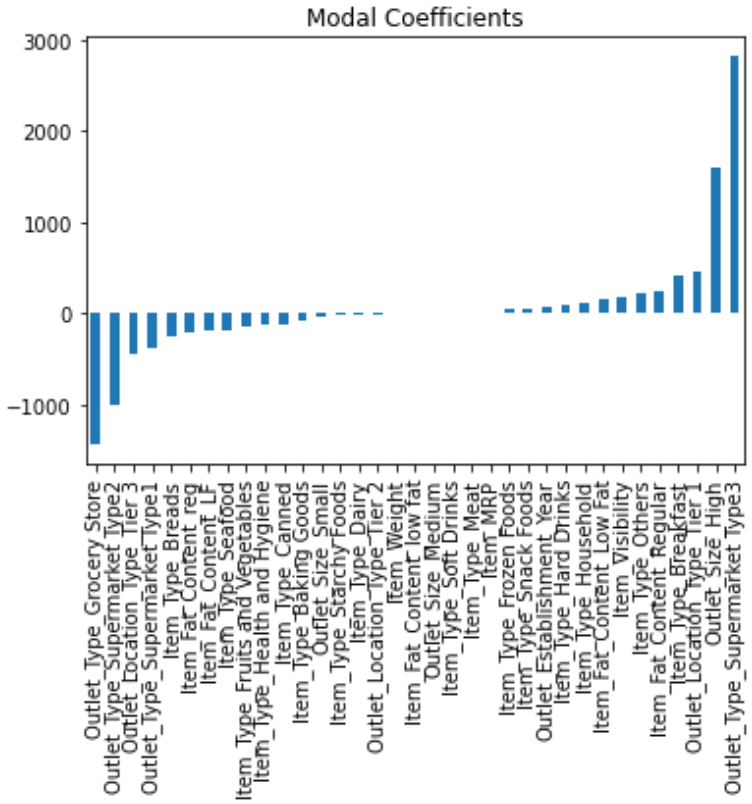
Bizim yaptığımız daha basitti, bunu herkes yaptı, şimdi basitleştirmeye bakalım. Bu nedenle, düzenlileştirme yardımıyla kodumuzu optimize etmeye çalışacağız.

Düzenlemede yaptığımız şey normalde aynı sayıda özniteliği tutmak, ancak j katsayılarının büyüklüğünü azaltmaktır. Katsayıları azaltmak bize nasıl yardımcı olacak?

Yukarıdaki regresyon modelimizde özellik katsayılarına bir göz atalım.

In [27]:
```
# checking the magnitude of coefficients
predictors = x_train.columns
coef = Series(lreg.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Modal Coefficients')
```

Out[27]: <AxesSubplot:title={'center':'Modal Coefficients'}>



Outlet_Identifier_OUT027 ve Outlet_Type_Supermarket_Type3(son 2) katsayılarının diğer katsayılara göre çok daha yüksek olduğunu görebiliriz. Bu nedenle, bir öğenin toplam satışları daha çok bu iki özellik tarafından yönlendirilecektir.

Modelimizdeki katsayıların büyüklüğünü nasıl azaltabiliriz? Bu amaçla, bu sorunun üstesinden gelmek için düzenlileştirmeyi kullanan farklı tipte regresyon tekniklerine sahibiz. Öyleyse onları tartışalım.

11. Ridge Regression

Önce yukarıdaki problemimize uygulayalım ve lineer regresyon modelimizden daha iyi performans gösterip göstermediğini kontrol edelim.

In [28]:
```
from sklearn.linear_model import Ridge

## training the model
ridgeReg = Ridge(alpha=0.05, normalize=True)
ridgeReg.fit(x_train,y_train)
pred = ridgeReg.predict(x_cv)

# calculating mse
mse = np.mean((pred_cv - y_cv)**2)
mse
```

Out[28]:
```
1468499.2681441572
```

In [29]:
```
## calculating score
ridgeReg.score(x_cv,y_cv)
```

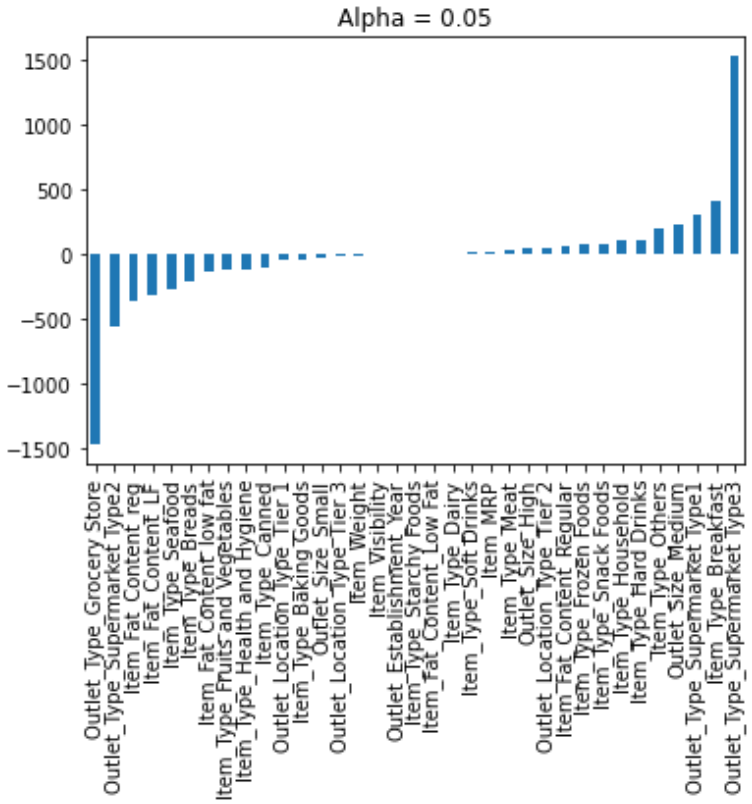
Out[29]:
```
0.5480702249101943
```

Yani R-Square değeri arttırıldığı için modelimizde ufak bir gelişme olduğunu görebiliyoruz. Ridge'in hiperparametresi olan alpha değerinin model tarafından otomatik olarak öğrenilmediği, bunun yerine manuel olarak ayarlanması gerektiği anlamına geldiğine dikkat edin.

Burada alfa = 0.05'i ele aldık. Ama farklı alfa değerlerini düşünelim ve her durum için katsayıları çizelim.

In [30]:
```
predictors = x_train.columns
coef = Series(ridgeReg.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Alpha = 0.05')
```

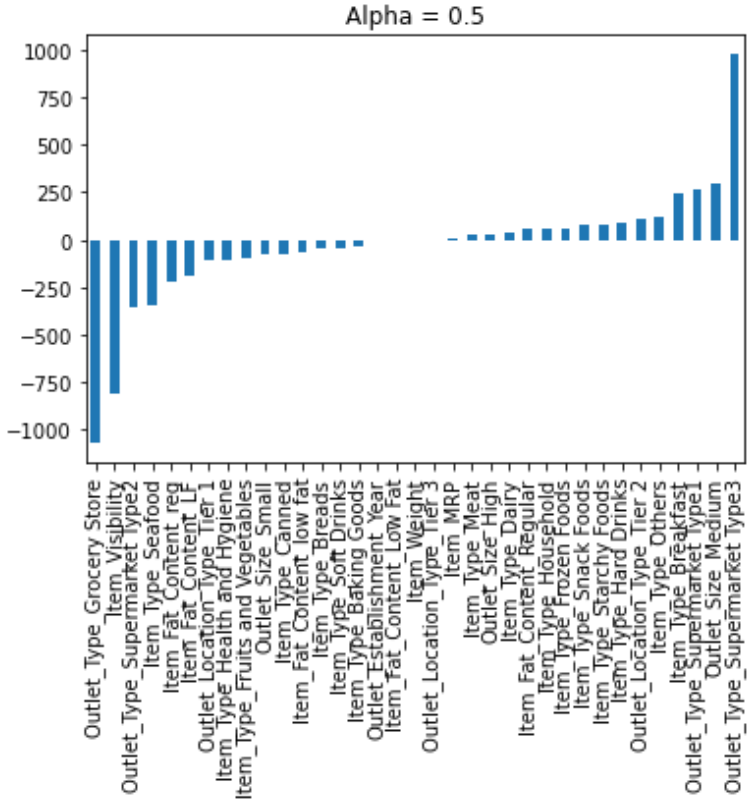
Out[30]: <AxesSubplot:title={'center':'Alpha = 0.05'}>



In [31]:
```
## training the model
ridgeReg = Ridge(alpha=0.5, normalize=True)
ridgeReg.fit(x_train,y_train)
pred = ridgeReg.predict(x_cv)

predictors = x_train.columns
coef = Series(ridgeReg.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Alpha = 0.5')
```

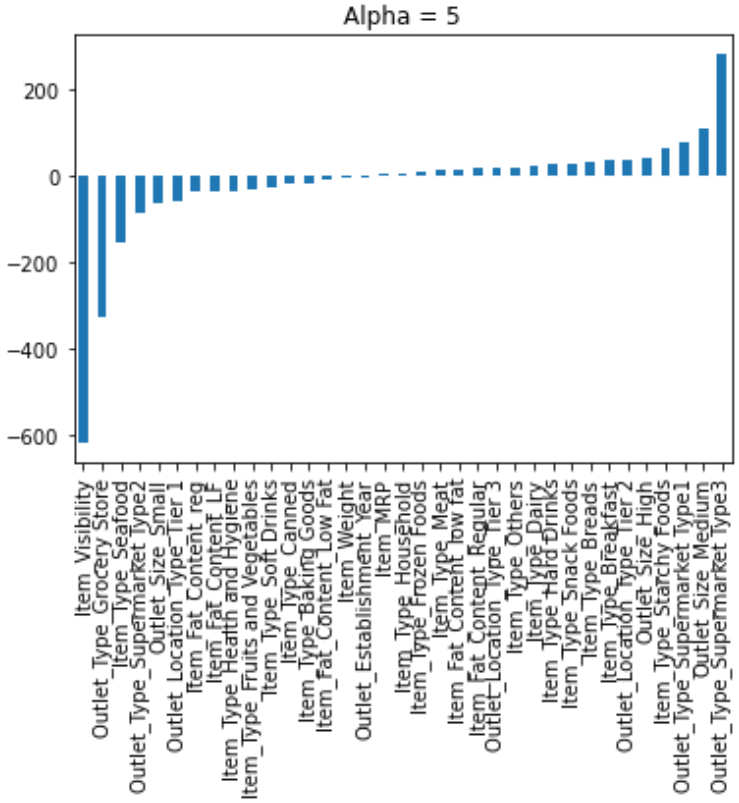
Out[31]:
```
<AxesSubplot:title={'center':'Alpha = 0.5'}>
```



In [32]:
```
## training the model
ridgeReg = Ridge(alpha=5, normalize=True)
ridgeReg.fit(x_train,y_train)
pred = ridgeReg.predict(x_cv)

predictors = x_train.columns
coef = Series(ridgeReg.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Alpha = 5')
```

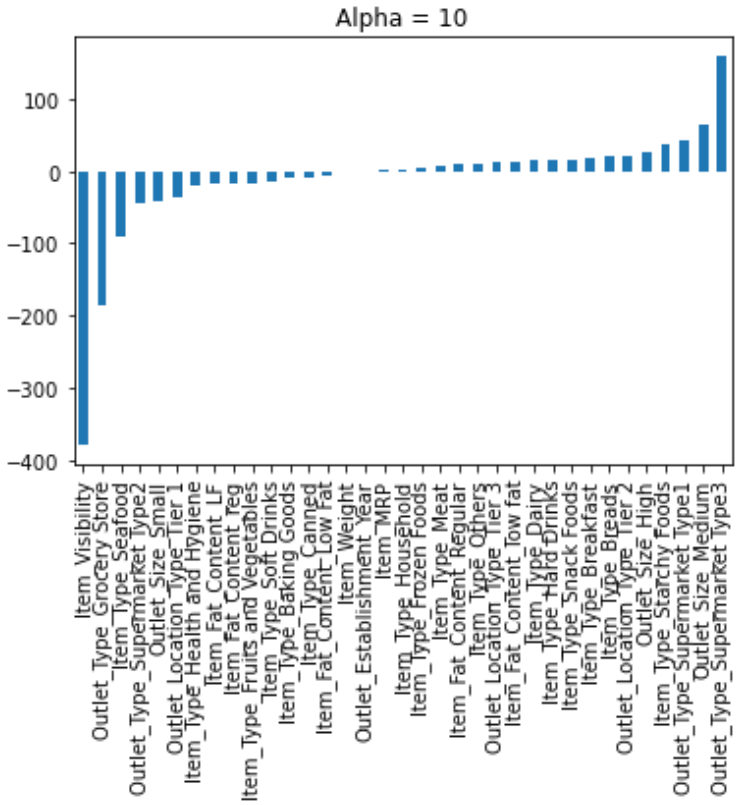
Out[32]:
```
<AxesSubplot:title={'center':'Alpha = 5'}>
```



In [33]:
```
## training the model
ridgeReg = Ridge(alpha=10, normalize=True)
ridgeReg.fit(x_train,y_train)
pred = ridgeReg.predict(x_cv)

predictors = x_train.columns
coef = Series(ridgeReg.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Alpha = 10')
```

Out[33]:
```
<AxesSubplot:title={'center':'Alpha = 10'}>
```



Görüyorsunuz ki, alfa değerini artırdıkça katsayıların büyüklüğü azalır, burada değerler sıfıra ulaşır ancak mutlak sıfır olmaz.

Ancak her alfa için R-kare hesaplarsanız, alfa=0.05'te R-kare değerinin maksimum olacağını göreceğiz. Bu yüzden, onu bir dizi değer üzerinden yineleyerek ve bize en düşük hatayı vereni kullanarak akıllıca seçmeliyiz.

Şimdi nasıl uygulanacağına dair bir fikriniz var ama şimdi matematik tarafına da bir göz atalım. Şimdiye kadarki fikrimiz, maliyet fonksiyonunu temel olarak minimize etmekti, öyle ki, tahmin edilen değerler istenen sonuca çok daha yakın olur.

Şimdi ridge regression için maliyet fonksiyonuna tekrar bir göz atın.

$$\min\left(\left|\left|Y - X(\theta)\right|\right|_2^2 + \lambda||\theta||_2^2\right)$$

Burada dikkat ederseniz ceza süresi olarak bilinen ekstra bir terim ile karşılaşıyoruz. Burada verilen λ, aslında ridge function'unda alfa parametresi ile gösterilir. Yani alfa değerlerini değiştirerek, temel olarak ceza terimini kontrol ediyoruz. Alfa değerleri ne kadar yüksek olursa, ceza da o kadar büyük olur ve bu nedenle katsayıların büyüklüğü azalır.

Important Points:

- Parametreleri küçültür, bu nedenle çoğunlukla çoklu doğrusallığı önlemek için kullanılır.
- Katsayı küçülmesi ile model karmaşıklığını azaltır.
- L2 düzenleme tekniğini kullanır. (bu makalenin ilerleyen kısımlarında tartışacağım)

Şimdi de düzenlileştirmeyi kullanan başka bir tür regresyon tekniğini ele alalım.

12. Lasso regression

LASSO (Least Absolute Shrinkage Selector Operator), ridge'e oldukça benzer, ancak büyük market problemimizde uygulayarak aralarındaki farkı anlayalım.

In [34]:
```
from sklearn.linear_model import Lasso

lassoReg = Lasso(alpha=0.3, normalize=True)
lassoReg.fit(x_train,y_train)
pred = lassoReg.predict(x_cv)

# calculating mse
mse = np.mean((pred_cv - y_cv)**2)
mse
```

Out[34]:
```
1468499.2681441572
```

In [35]:
```
lassoReg.score(x_cv,y_cv)
```

Out[35]:
```
0.5475492762129966
```

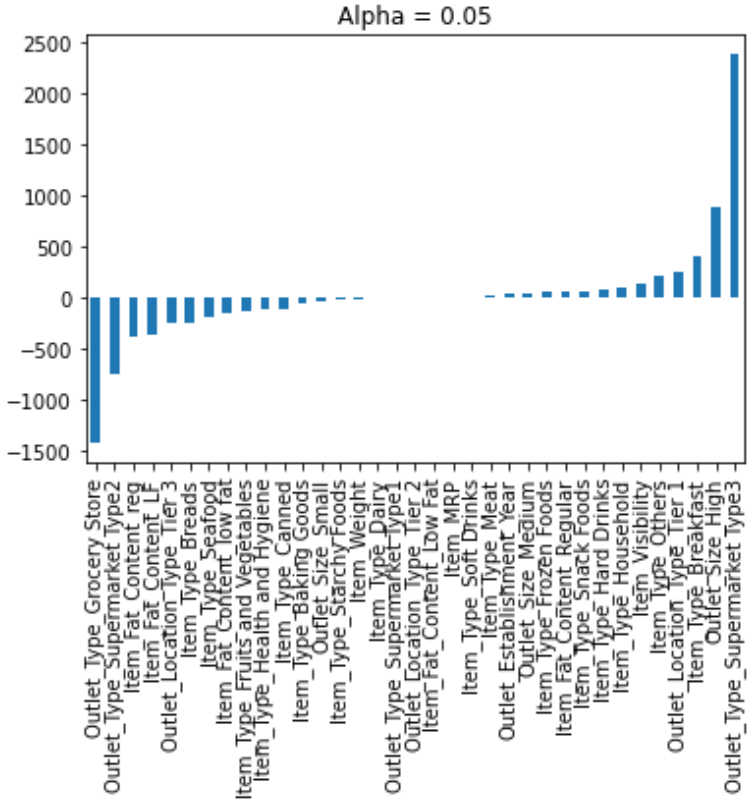
Görüldüğü gibi modelimiz için hem **mse** hem de **R-square** değeri arttırılmıştır. Bu nedenle, lasso modeli hem linear hem de ridge daha iyi tahmin ediyor.

Yine alfa değerini değiştirelim ve katsayıları nasıl etkilediğini görelim.

In [36]:
```
## training the model
lassoReg = Lasso(alpha=0.05, normalize=True)
lassoReg.fit(x_train,y_train)
pred = lassoReg.predict(x_cv)

predictors = x_train.columns
coef = Series(lassoReg.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Alpha = 0.05')
```

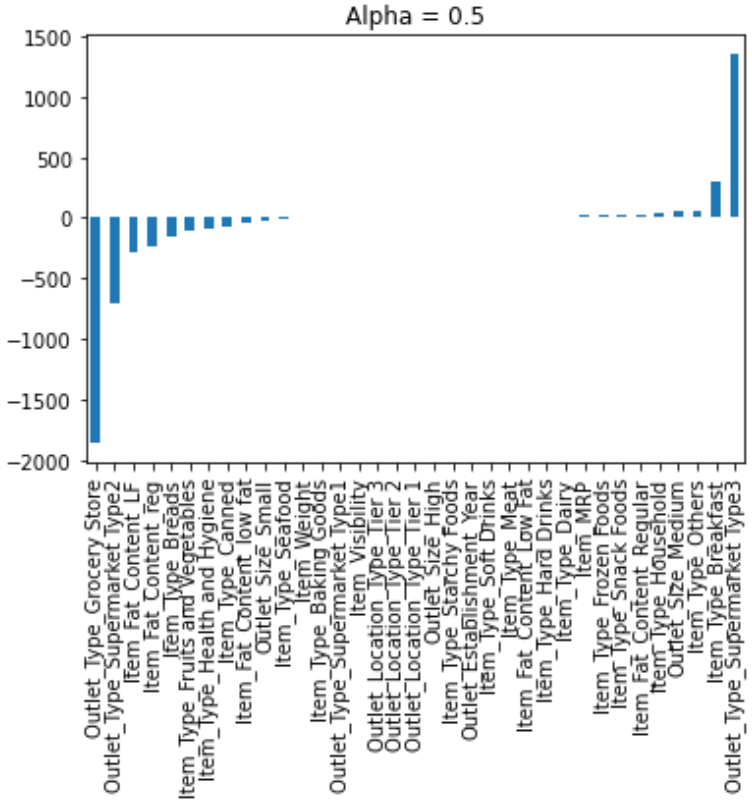
Out[36]:
```
<AxesSubplot:title={'center':'Alpha = 0.05'}>
```



In [37]:
```
## training the model
lassoReg = Lasso(alpha=0.5, normalize=True)
lassoReg.fit(x_train,y_train)
pred = lassoReg.predict(x_cv)

predictors = x_train.columns
coef = Series(lassoReg.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Alpha = 0.5')
```

Out[37]:
```
<AxesSubplot:title={'center':'Alpha = 0.5'}>
```



Böylece, alfanın küçük değerlerinde bile katsayıların büyüklüğünün çok azaldığını görebiliriz. Çizimlere bakarak, ridge ve lasso arasındaki farkı anlayabilir misiniz?

Alfa değerini artırdıkça katsayıların sıfıra yaklaştığını görebiliriz, ancak lasso durumunda görürseniz daha küçük alfalarda bile katsayılarımız mutlak sıfırlara düşüyor. Bu nedenle, lasso yalnızca bazı özellikleri seçerken diğerlerinin katsayılarını sıfıra indirir. Bu özellik, özellik seçimi olarak bilinir ve çıkıntı durumunda yoktur.

Kement regresyonunun arkasındaki matematik, tetanın karelerini eklemek yerine, yalnızca ridge farkınınkine benzer şekilde sessizdir, Θ mutlak değerini ekleyeceğiz.

$$\min\left(\left|\left|Y - X\theta\right|\right|_2^2 + \lambda||\theta||_1\right)$$

Burada da λ, değeri Kement işlevindeki alfaya eşit olan hipermetredir.

Important Points:

- L1 düzenleme tekniğini kullanır (bu makalenin ilerleyen bölümlerinde tartışılacaktır)
- Özellik seçimini otomatik olarak yaptığı için genellikle daha fazla özelliğimiz olduğunda kullanılır.

Artık temel bir ridge ve lasso regresyonu anlayışına sahip olduğunuza göre, büyük bir veri setimizin olduğu bir örnek düşünelim, diyelim ki 10.000 özelliği var. Ve bazı bağımsız özelliklerin diğer bağımsız özelliklerle ilişkili olduğunu biliyoruz. O zaman düşünün, hangi regresyonu kullanırsınız, Rigde veya Lasso?

tek tek tartışalım. Buna ridge regresyon uygularsak, tüm özellikleri koruyacak ancak katsayıları küçültecektir. Ancak sorun şu ki, 10.000 özellik olduğu için model hala karmaşık kalacak ve bu nedenle düşük model performansına yol açabilir.

Ridge yerine bu probleme lasso regresyonu uygularsak ne olur? Lasso regresyonundaki temel sorun, ilişkili değişkenlere sahip olduğumuzda, yalnızca bir değişkeni tutması ve diğer ilişkili değişkenleri sıfıra ayarlamasıdır. Bu, muhtemelen modelimizde daha düşük doğrulukla sonuçlanan bir miktar bilgi kaybına yol açacaktır.

O zaman bu sorunun çözümü nedir? Aslında, temelde ridge ve lasso regresyonunun bir melezi olan, **elastic net regresyon** olarak bilinen başka bir tür regresyonumuz var. Öyleyse onu anlamaya çalışalım.

LINEAR, RIDGE AND LASSO REGRESSION

In [38]:
```
# importing requuired libraries
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso, Ridge
```

In [39]:
```
# read test and train file
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

In [41]:
```
print(train.head(3).T)
```

```
                                        0           1           2
Item_Weight                       6.80000    15.600000   12.911575
Item_Visibility                   0.03749     0.172597    0.054721
Item_MRP                         48.60340   114.851800  107.825400
Outlet_Establishment_Year      2004.00000  1997.000000  1985.000000
Item_Outlet_Sales               291.62040  2163.184200  2387.558800
Item_Fat_Content_LF               0.00000     0.000000    0.000000
Item_Fat_Content_Low Fat          1.00000     1.000000    1.000000
Item_Fat_Content_Regular          0.00000     0.000000    0.000000
Item_Fat_Content_low fat          0.00000     0.000000    0.000000
Item_Fat_Content_reg              0.00000     0.000000    0.000000
Item_Type_Baking Goods            1.00000     0.000000    0.000000
Item_Type_Breads                  0.00000     0.000000    0.000000
Item_Type_Breakfast               0.00000     0.000000    0.000000
Item_Type_Canned                  0.00000     0.000000    0.000000
Item_Type_Dairy                   0.00000     0.000000    0.000000
Item_Type_Frozen Foods            0.00000     0.000000    1.000000
Item_Type_Fruits and Vegetables   0.00000     1.000000    0.000000
Item_Type_Hard Drinks             0.00000     0.000000    0.000000
Item_Type_Health and Hygiene      0.00000     0.000000    0.000000
Item_Type_Household               0.00000     0.000000    0.000000
Item_Type_Meat                    0.00000     0.000000    0.000000
Item_Type_Others                  0.00000     0.000000    0.000000
Item_Type_Seafood                 0.00000     0.000000    0.000000
Item_Type_Snack Foods             0.00000     0.000000    0.000000
Item_Type_Soft Drinks             0.00000     0.000000    0.000000
Item_Type_Starchy Foods           0.00000     0.000000    0.000000
Outlet_Size_High                  0.00000     0.000000    0.000000
Outlet_Size_Medium                0.00000     0.000000    1.000000
Outlet_Size_Small                 1.00000     1.000000    0.000000
Outlet_Location_Type_Tier 1       0.00000     1.000000    0.000000
Outlet_Location_Type_Tier 2       1.00000     0.000000    0.000000
Outlet_Location_Type_Tier 3       0.00000     0.000000    1.000000
Outlet_Type_Grocery Store         0.00000     0.000000    0.000000
Outlet_Type_Supermarket Type1     1.00000     1.000000    0.000000
Outlet_Type_Supermarket Type2     0.00000     0.000000    0.000000
Outlet_Type_Supermarket Type3     0.00000     0.000000    1.000000
```

In [42]:
```
#splitting into training and test
## try building model with the different features and compare the result.
X = train.loc[:,['Outlet_Establishment_Year','Item_MRP']]
x_train, x_cv, y_train, y_cv = train_test_split(X,train.Item_Outlet_Sales,rando
```

In [43]:
```
# Trainig Linear Regression Model
lreg = LinearRegression()
#training the model
lreg.fit(x_train,y_train)
```

Out[43]:
```
LinearRegression()
```

In [44]:
```
#predicting on cv
pred = lreg.predict(x_cv)
```

In [45]:
```
#calculating mse
mse = np.mean((pred - y_cv)**2)
print('Mean Sqaured Error = ',mse )
```

```
Mean Sqaured Error =  1981482.7752175191
```

In [46]:
```
#Let us take a look at the coefficients of this linear regression model.
# calculating coefficients
coeff = DataFrame(x_train.columns)
```

In [47]:
```
coeff['Coefficient Estimate'] = Series(lreg.coef_)
print(coeff)
```

```
                           0  Coefficient Estimate
0  Outlet_Establishment_Year            -21.180115
1                   Item_MRP             17.018057
```

In [48]:
```
# Model performance on Test data
print(lreg.score(x_cv,y_cv))
```

```
0.3109901925596644
```

In [49]:
```
# Training Ridge Regression Model

ridge = Ridge()
ridge.fit(x_train,y_train)
pred1 = ridge.predict(x_cv)
mse_1 = np.mean((pred1-y_cv)**2)

print('Mean Squared Error = ',mse_1)
```

```
Mean Squared Error =  1981482.0637939738
```

In [50]:
```
# calculating coefficients
coeff = DataFrame(x_train.columns)
coeff['Coefficient Estimate'] = Series(ridge.coef_)
print(coeff)
```

```
                           0  Coefficient Estimate
0  Outlet_Establishment_Year            -21.179817
1                   Item_MRP             17.018053
```

In [51]:
```
# Model performance on Test data
print(ridge.score(x_cv,y_cv))
```

```
0.3109904399389526
```

In [52]:
```
# Training Lasso Regression Model

lasso = Lasso()
lasso.fit(x_train,y_train)
pred2 = lasso.predict(x_cv)
mse_2 = np.mean((pred2-y_cv)**2)

print('Mean Squared Error = ',mse_2)
```

```
Mean Squared Error =  1981447.9970456203
```

In [53]:
```
# calculating coefficients
coeff = DataFrame(x_train.columns)
coeff['Coefficient Estimate'] = Series(lasso.coef_)
print(coeff)
```

```
                           0  Coefficient Estimate
0  Outlet_Establishment_Year            -21.165732
1                   Item_MRP             17.017830
```

In [54]:
```
# Model performance on Test data
print(lasso.score(x_cv,y_cv))
```

```
0.311002285776836
```

13. Elastic Net Regression

Teori kısmına geçmeden önce bunu big mart satış probleminde de uygulayalım. Ridge ve lasso daha iyi performans gösterecek mi? Hadi kontrol edelim!

In [55]:
```
from sklearn.linear_model import ElasticNet
ENreg = ElasticNet(alpha=1, l1_ratio=0.5, normalize=False)
ENreg.fit(x_train,y_train)
pred_cv = ENreg.predict(x_cv)

#calculating mse
mse = np.mean((pred_cv - y_cv)**2)
mse
```

Out[55]:
```
1981105.6704098866
```

In [56]:
```
ENreg.score(x_cv,y_cv)
```

Out[56]:
```
0.31112132108328316
```

Böylece hem ridge'ten hem de lasso'dan çok daha küçük olan R-Kare değerini elde ederiz. Neden olduğunu düşünebilir misin? Bu düşüşün arkasındaki sebep, temelde çok sayıda özelliğe sahip olmamamızdı. Elastik regresyon, genellikle büyük bir veri kümemiz olduğunda iyi çalışır.

Not, burada alpha ve l1_ratio olmak üzere iki parametremiz vardı. İlk önce elastik nette neler olduğunu ve ridge'den ve lasso'dan nasıl farklı olduğunu tartışalım.

Elastik net, temel olarak hem L1 hem de L2 düzenlemesinin bir birleşimidir. Yani elastik net'i biliyorsanız, parametreleri ayarlayarak hem Ridge hem de Lasso'yu uygulayabilirsiniz. Dolayısıyla hem L1 hem de L2 ceza terimini kullanır, bu nedenle denklemi aşağıdaki gibi görünür:

$$\min\left(\left|\left|Y - X\theta\right|\right|_2^2 + \lambda_1||\theta||_1 + \lambda_2||\theta||_2^2\right)$$

Peki L1 ve L2 ceza süresini kontrol etmek için lambdaları nasıl ayarlayacağız? Bir örnekle anlayalım. Bir göletten balık yakalamaya çalışıyorsunuz. Ve sadece bir ağınız var, o zaman ne yapardınız? Ağını rastgele atar mısın? Hayır, aslında etrafta yüzen bir balık görene kadar bekleyeceksiniz, ardından tüm balık grubunu toplamak için ağı o yöne atacaksınız. Bu nedenle, ilişkili olsalar bile, yine de tüm gruplarına bakmak istiyoruz.

Elastik regresyon da benzer şekilde çalışır. Diyelim ki, bir veri setinde bir grup bağıntılı bağımsız değişkenimiz var, o zaman elastik ağ basitçe bu bağıntılı değişkenlerden oluşan bir grup oluşturacaktır. Şimdi, bu grubun değişkenlerinden herhangi biri güçlü bir tahmin edici ise (bağımlı değişkenle güçlü bir ilişkiye sahip olmak anlamına gelir), o zaman tüm grubu model oluşturmaya dahil edeceğiz, çünkü diğer değişkenleri (kementte yaptığımız gibi) ihmal etmek yorumlama yeteneği açısından bazı bilgilerin kaybolmasına neden olarak zayıf bir model performansına yol açar.

Yani yukarıdaki koda bakarsanız modeli tanımlarken alpha ve l1_ratio tanımlamamız gerekiyor. Alfa ve l1_ratio, L1 ve L2 cezasını ayrı ayrı kontrol etmek isterseniz buna göre ayarlayabileceğiniz parametrelerdir. Aslında, bizde var

> Alpha = a + b ve l1_ratio = a / (a+b)

Burada, sırasıyla L1 ve L2 terimlerine atanan a ve b ağırlıkları. Dolayısıyla, alpha ve l1_ratio değerlerini değiştirdiğimizde, a ve b, L1 ve L2 arasındaki değiş tokuşu kontrol edecek şekilde buna göre ayarlanır:

> a * (L1 term) + b* (L2 term)

Alfa (veya a+b) = 1 olsun ve şimdi aşağıdaki durumları ele alalım:

- l1_ratio=1 ise, dolayısıyla l1_ratio formülüne bakarsak, l1_ratio'nun ancak a=1 ise 1'e eşit olabileceğini görebiliriz, bu da b=0 anlamına gelir. Bu nedenle, bir kement cezası olacaktır.
- Benzer şekilde, l1_ratio = 0 ise, a=0 anlamına gelir. O zaman ceza sırt cezası olacak.
- 0 ile 1 arasındaki l1_ratio için ceza, sırt ve kement kombinasyonudur.

Öyleyse alpha ve l1_ratio'yu ayarlayalım ve aşağıda verilen katsayı grafiklerinden anlamaya çalışalım.

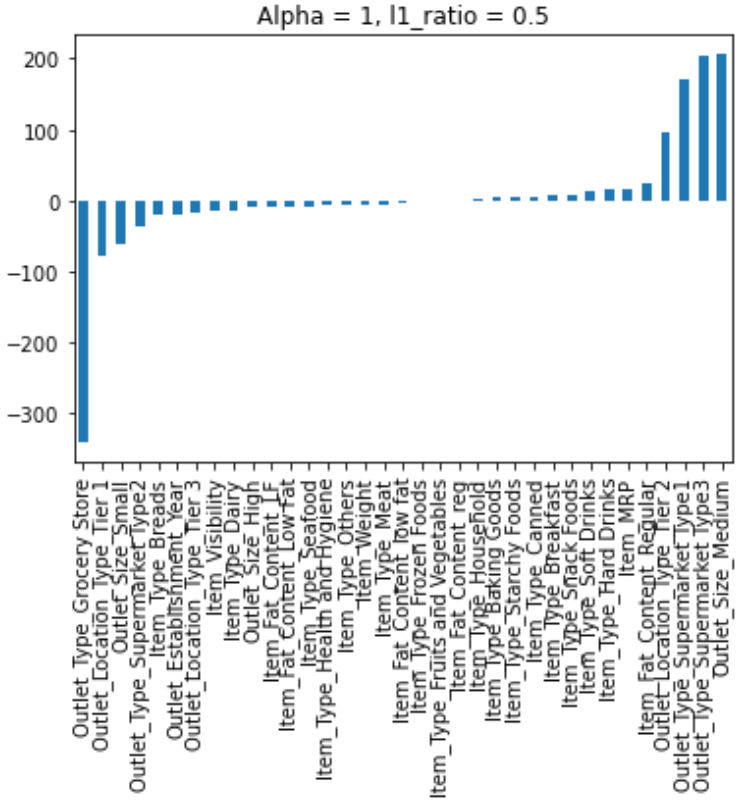
In [57]:
```
# for cross validation
from sklearn.model_selection import train_test_split

X = train.drop('Item_Outlet_Sales',1)
x_train, x_cv, y_train, y_cv = train_test_split(X,train.Item_Outlet_Sales, test_

## training the model
from sklearn.linear_model import ElasticNet
ENreg = ElasticNet(alpha=1, l1_ratio=0.5, normalize=False)
ENreg.fit(x_train,y_train)
pred_cv = ENreg.predict(x_cv)

predictors = x_train.columns
coef = Series(ENreg.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Alpha = 1, l1_ratio = 0.5')
```

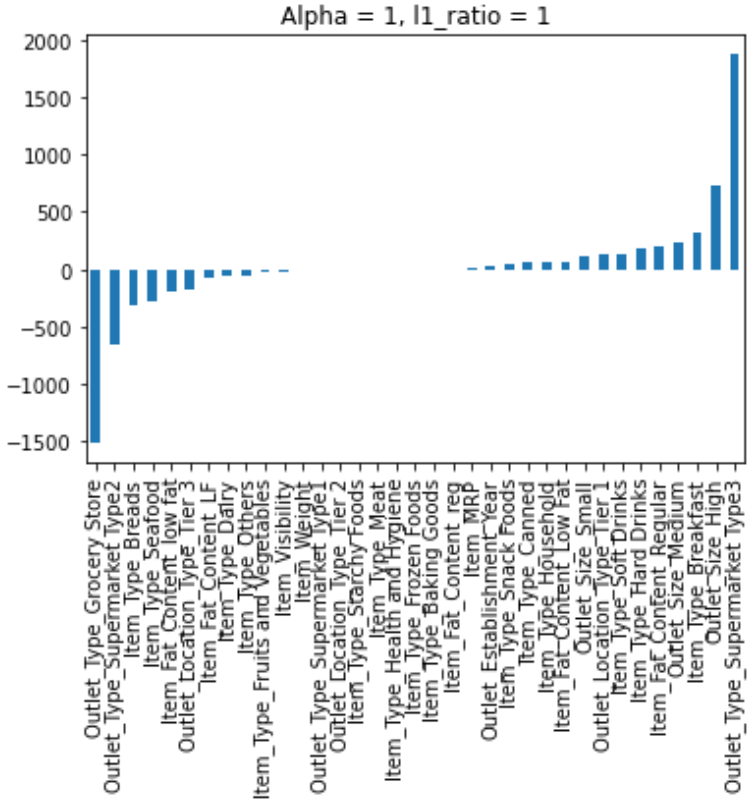
Out[57]: <AxesSubplot:title={'center':'Alpha = 1, l1_ratio = 0.5'}>



In [58]:
```
## training the model
ENreg = ElasticNet(alpha=1, l1_ratio=1, normalize=False)
ENreg.fit(x_train,y_train)
pred_cv = ENreg.predict(x_cv)

predictors = x_train.columns
coef = Series(ENreg.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Alpha = 1, l1_ratio = 1')
```

Out[58]:
```
<AxesSubplot:title={'center':'Alpha = 1, l1_ratio = 1'}>
```



Artık ridge, lasso ve elasticnet regresyonu hakkında temel bilgilere sahipsiniz. Ancak bu sırada, temelde iki tür düzenlileştirme olan L1 ve L2 terimiyle karşılaştık. Özetlemek gerekirse, temel olarak lasso ve ridge, sırasıyla L1 ve L2 düzenlemesinin doğrudan uygulamasıdır.