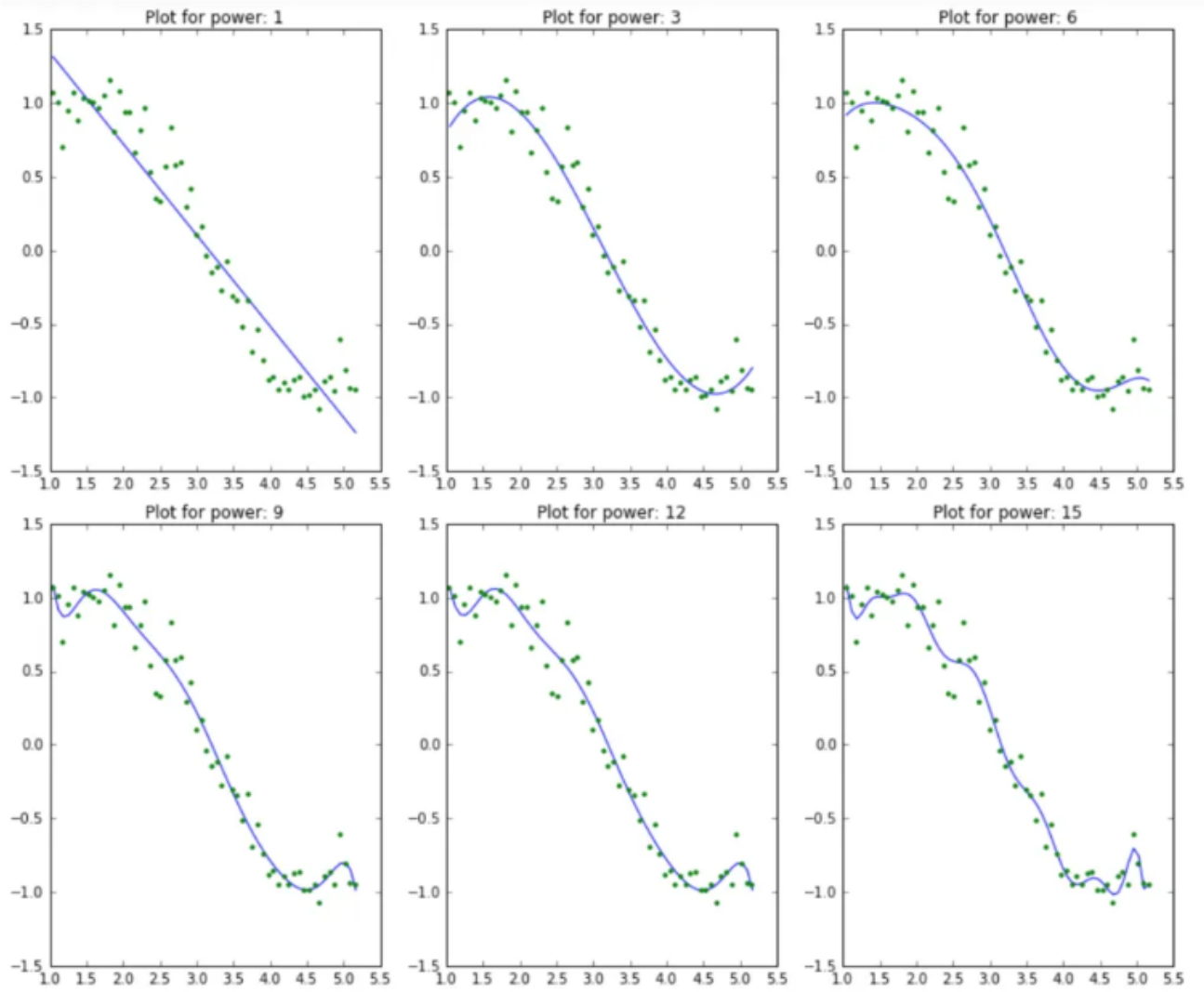
Predicting Houseprice:

In this Part, we will learn about Regularization parameters.

1. Whay Regularization
2. Laso Vs Ridge
3. How lambda affects equation

Why Regularization:

1. As variables complexity increases, the model starts to overfit. It produces very high coefficients



The output looks like:

	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10	coef_x_11	c
model_pow_1	3.3	2	-0.62	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_2	3.3	1.9	-0.58	-0.006	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_3	1.1	-1.1	3	-1.3	0.14	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_4	1.1	-0.27	1.7	-0.53	-0.036	0.014	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_5	1	3	-5.1	4.7	-1.9	0.33	-0.021	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_6	0.99	-2.8	9.5	-9.7	5.2	-1.6	0.23	-0.014	NaN	NaN	NaN	NaN	NaN	N
model_pow_7	0.93	19	-56	69	-45	17	-3.5	0.4	-0.019	NaN	NaN	NaN	NaN	N
model_pow_8	0.92	43	-1.4e+02	1.8e+02	-1.3e+02	58	-15	2.4	-0.21	0.0077	NaN	NaN	NaN	N
model_pow_9	0.87	1.7e+02	-6.1e+02	9.6e+02	-8.5e+02	4.6e+02	-1.6e+02	37	-5.2	0.42	-0.015	NaN	NaN	N
model_pow_10	0.87	1.4e+02	-4.9e+02	7.3e+02	-6e+02	2.9e+02	-87	15	-0.81	-0.14	0.026	-0.0013	NaN	N
model_pow_11	0.87	-75	5.1e+02	-1.3e+03	1.9e+03	-1.6e+03	9.1e+02	-3.5e+02	91	-16	1.8	-0.12	0.0034	N
model_pow_12	0.87	-3.4e+02	1.9e+03	-4.4e+03	6e+03	-5.2e+03	3.1e+03	-1.3e+03	3.8e+02	-80	12	-1.1	0.062	-(
model_pow_13	0.86	3.2e+03	-1.8e+04	4.5e+04	-6.7e+04	6.6e+04	-4.6e+04	2.3e+04	-8.5e+03	2.3e+03	-4.5e+02	62	-5.7	0
model_pow_14	0.79	2.4e+04	-1.4e+05	3.8e+05	-6.1e+05	6.6e+05	-5e+05	2.8e+05	-1.2e+05	3.7e+04	-8.5e+03	1.5e+03	-1.8e+02	1
model_pow_15	0.7	-3.6e+04	2.4e+05	-7.5e+05	1.4e+06	-1.7e+06	1.5e+06	-1e+06	5e+05	-1.9e+05	5.4e+04	-1.2e+04	1.9e+03	-:

It is clearly evident that the **size of coefficients increase exponentially with increase in model complexity**. I hope this gives some intuition into why putting a constraint on the magnitude of coefficients can be a good idea to reduce model complexity.

Lets try to understand this even better.

What does a large coefficient signify? It means that we're putting a lot of emphasis on that feature, i.e. the particular feature is a good predictor for the outcome. When it becomes too large, the algorithm starts modelling intricate relations to estimate the output and ends up overfitting to the particular training data.

1. Reduces Model Overfit/Variance: Regularization decreases model complexity by introducing bias in the form of loss function

Lasso Vs Ridge Regression:

Lasso and Ridge regression penalizes increasing model complexity differently. Below are the loss function. Lasso is different than ridge in the following characteristics

1. Robust to outliers
2. Can 0 out variables and hence can act as variable selection.

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \sum_{n=1}^{N} \frac{1}{2}(y_n - \beta x_n)^2 + \lambda \sum_{i=1}^{p} \beta_i^2$$

$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \sum_{n=1}^{N} \frac{1}{2}(y_n - \beta x_n)^2 + \lambda \sum_{i=1}^{p} |\beta_i|$$

In [350]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import skew, norm
from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax
from scipy.stats.stats import pearsonr

# Models
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor,
AdaBoostRegressor, BaggingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.linear_model import Lasso, LassoCV
from sklearn.linear_model import regressor
from sklearn.linear_model import ElasticNet, ElasticNetCV
from sklearn.svm import SVR
from mlxtend.regressor import StackingCVRegressor
import lightgbm as lgb
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

# Stats
from scipy.stats import skew, norm
from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax

# Misc
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA


# Ignore useless warnings
import warnings
warnings.filterwarnings(action="ignore")
pd.options.display.max_seq_items = 8000
pd.options.display.max_rows = 8000
```

In [286]:
```
df=pd.read_csv('C:/Users/thand/Downloads/house-prices-advanced-regression-techniques/train.csv')
df.head()
```

Out[286]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Util
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	Al
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	Al
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	Al
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	Al
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	Al

5 rows × 81 columns

In [387]:
```
linearModel = LinearRegression()
linearModel.fit(x_train, y_train)

# Evaluating the Linear Regression model
print(linearModel.score(x_test, y_test))
```

```
-90553149601138.48
```

In [371]:
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score
from statistics import mean

# List to maintain the different cross-validation scores
cross_val_scores_ridge = []

# List to maintain the different values of alpha
alpha = []
for i in range(1,9):
    ridgeModel = Ridge(alpha = i * 0.25)
    ridgeModel.fit(x_train, y_train)
    scores = cross_val_score(ridgeModel, x_train, y_train, cv = 10)
    avg_cross_val_score = mean(scores)*100
    cross_val_scores_ridge.append(avg_cross_val_score)
    alpha.append(i * 0.25)

for i in range(0, len(alpha)):
    print(str(alpha[i])+' : '+str(cross_val_scores_ridge[i]))
```

```
0.25 : 89.0902428548502
0.5 : 89.43320368427688
0.75 : 89.59964503576103
1.0 : 89.70437934053672
1.25 : 89.77957546999163
1.5 : 89.83795009672872
1.75 : 89.8855716552063
2.0 : 89.92572190954031
```

In [388]:
```
#choosing alpha=2
# Building and fitting the Ridge Regression model
ridgeModelChosen = Ridge(alpha = 2)
ridgeModelChosen.fit(x_train, y_train)

# Evaluating the Ridge Regression model
print(ridgeModelChosen.score(x_test, y_test))
#print(ridgeModelChosen.coef_)
```

```
0.8761256811603585
```

In [380]:
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso

# List to maintain the cross-validation scores
cross_val_scores_lasso = []

# List to maintain the different values of Lambda
Lambda = []

# Loop to compute the cross-validation scores
for i in range(1, 9):
    lassoModel = Lasso(alpha = i * 0.25, tol = 0.0925)
    lassoModel.fit(x_train, y_train)
    scores = cross_val_score(lassoModel, x_train, y_train, cv = 10)
    avg_cross_val_score = mean(scores)*100
    cross_val_scores_lasso.append(avg_cross_val_score)
    Lambda.append(i * 0.25)

#Loop to print the different values of cross-validation scores
for i in range(0, len(alpha)):
    print(str(alpha[i])+' : '+str(cross_val_scores_lasso[i]))
```

```
0.25 : 75.34702291104158
0.5 : 61.390420374845235
0.75 : 61.237956344974066
1.0 : 61.17867381435005
1.25 : 61.04278149614899
1.5 : 60.857510337039166
1.75 : 60.702719846996125
2.0 : 60.682150088703466
```

In [389]:
```
# Building and fitting the Lasso Regression Model
lassoModelChosen = Lasso(alpha = 2, tol = 0.0925)
lassoModelChosen.fit(x_train, y_train)

# Evaluating the Lasso Regression model
print(lassoModelChosen.score(x_test, y_test))
#print(lassoModelChosen.sparse_coef_)
```

```
0.6513700711368502
```

In [ ]:
```

```