Data Analysis with Python

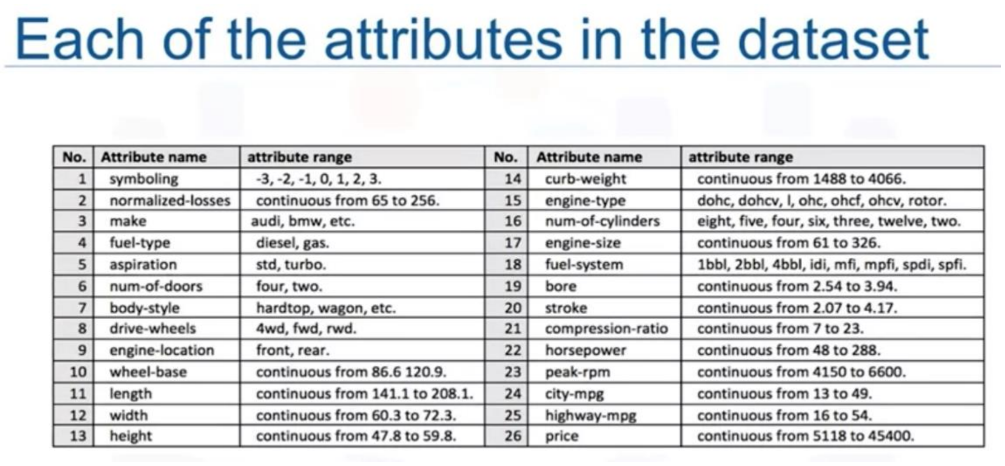
Notes of IBM Data Science Professional Certificate Courses on Coursera

View on GitHub

Data Analysis with Python

Datasets

Understanding Datasets



No.	Attribute name	attribute range	No.	Attribute name	attribute range
1	symboling	-3, -2, -1, 0, 1, 2, 3.	14	curb-weight	continuous from 1488 to 4066.
2	normalized-losses	continuous from 65 to 256.	15	engine-type	dohc, dohcv, l, ohc, ohcf, ohcv, rotor.
3	make	audi, bmw, etc.	16	num-of-cylinders	eight, five, four, six, three, twelve, two.
4	fuel-type	diesel, gas.	17	engine-size	continuous from 61 to 326.
5	aspiration	std, turbo.	18	fuel-system	1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
6	num-of-doors	four, two.	19	bore	continuous from 2.54 to 3.94.
7	body-style	hardtop, wagon, etc.	20	stroke	continuous from 2.07 to 4.17.
8	drive-wheels	4wd, fwd, rwd.	21	compression-ratio	continuous from 7 to 23.
9	engine-location	front, rear.	22	horsepower	continuous from 48 to 288.
10	wheel-base	continuous from 86.6 120.9.	23	peak-rpm	continuous from 4150 to 6600.
11	length	continuous from 141.1 to 208.1.	24	city-mpg	continuous from 13 to 49.
12	width	continuous from 60.3 to 72.3.	25	highway-mpg	continuous from 16 to 54.
13	height	continuous from 47.8 to 59.8.	26	price	continuous from 5118 to 45400.

Data source: https://archive.ics.uci.edu/ml/machine-learning-databases/autos/

Exporting to different formats in Python

Data Format	Read	Save
`csv`	`pd.read_csv()`	`df.to_csv()`
`json`	`pd.read_json()`	`df.to_json()`
`Excel`	`pd.read_excel()`	`df.to_excel()`
`sql`	`pd.read_sql()`	`df.to_sql()`

Basic insights from the data

- Understand your data before you begin any analysis
- Should check:
    - data types
        - `df.dtypes`
    - data distribution
        - `df.describe()`
        - `df.describe(include="all")`, provides full summary statistics
            - `unique`
            - `top`
            - `freq`
- Locate potential issues with the data
    - potential info and type mismatch
    - compatibility with python methods

Writing code using DB-API

```
from dbmodule import connect

#Create connection object
connection = connect('databasename','username','pswd')

#Create a cursor object
cursor = connection.cursor()

#Run queries
cursor.execute('select * from mytable')
results = cursor.fetchall()

#Free resources
Cursor.close()
connection.close()
```

Preprocessing Data in Python

- Identify and handle missing values
- Data formatting
- Data normalization (centering / scaling)
- Data binning
- Turning categorical values to numeric variables

How to deal with missing data

- Check with the data collection source
- Drop the missing values
    - drop the variable
    - drop the data entry
- Replace the missing values

    - replace it with an average (of similar datapoints)
    - replace it by frequency
    - replace it based on other functions
- Leave it as missing data

```
df.dropna(subset=["price"], axis=0, inplace=True)
```

is equivalent to

```
df = df.dropna(subset=["price"], axis=0)
```

Data Formatting in Python

Non-formatted:

- confusing
- hard to aggregate
- hard to compare

Formatted:

- more clear
- easy to aggregate
- easy to compare

Correcting data types

- use `df.dtypes()` to identify data type
- use `df.astype()` to convert data type
    - e.g. `df["price"] = df["price"].astype("int")`

Data Normalization in Python

Approaches for normalization:

- Simple feature scaling: $x_{new} = x_{old}/x_{max}$
    - `df["length"] = df["length"] / df["length"].max()`
- Min-Max: $x_{new} = (x_{old}-x_{min})/(x_{max}-x_{min})$
    - `df["length"] = (df["length"]-df["length"].min()) / (df["length"].max()-df["length"].min())`
- Z-score: $x_{new} = (x_{old}-\mu)/\sigma$
    - `df["length"] = (df["length"]-df["length"].mean()) / df["length"].std()`

Binning

Binning

- Binning: Grouping of values into "bins"
- Converts numeric into categorical variables
- Group a set of numerical values into a set of "bins"
- "price" is a feature range from 5,000 to 45,500
  (in order to have a **better representation** of price)

price: 5000, 10000,12000,12000, 30000, 31000, 39000, 44000,44500

bins: low Mid High

```
bins = np.linspace(min(df["price"]), max(df["price"]), 4)
group_names = ["Low", "Medium", "High"]
df["price-binned"] = pd.cut(df["price"], bins, labels=group_names, include_lowest=True)
```

Turning categorical variables into quantitative variables in Python

Categorical → Numeric

Solution:
- Add dummy variables for each unique category
- Assign 0 or 1 in each category

Car	Fuel	...	gas	diesel
A	gas	...	1	0
B	diesel	...	0	1
C	gas	...	1	0
D	gas	...	1	0

"One-hot encoding"

Dummy variables in Python pandas

- Use pandas.get_dummies() method.
- Convert categorical variables to dummy variables (0 or 1)

fuel
gas
diesel
gas
gas

→

gas	diesel
1	0
0	1
1	0
1	0

```
pd.get_dummies(df['fuel'])
```

Exploratory Data Analysis (EDA)

- Question:
  - “What are the characteristics which have the most impact on the car price?”
- Preliminary step in data analysis to:
  - Summarize main characteristics of the data
  - Gain better understanding of the data set
  - Uncover relationships between variables
  - Extract important variables

Learning Objectives:

- Descriptive Statistics
- GroupBy
- Correlation
- Correlation - Statistics

Descriptive Statistics - Describe()

- Summarize statistics using pandas `describe()` method
  - `df.describe()`
- Summarize categorical data is by using the `value_counts()` method
- Box Plot
- Scatter Plot
  - each observation represented as a point
  - scatter plot show the relationship between two variables
    - predictor/independent variables on x-axis
    - target/dependent variables on y-axis

Grouping data

`groupby`

- use `df.groupby()` method:
  - can be applied on categorical variables
  - group data into categories
  - single or multiple variables

Groupby()- Example

```
df_test = df[['drive-wheels', 'body-style', 'price']]
df_grp = df_test.groupby(['drive-wheels', 'body-style'], as_index=False).mean()
df_grp
```

	drive-wheels	body-style	price
0	4wd	convertible	20239.229524
1	4wd	sedan	12647.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	23949.600000
9	rwd	hardtop	24202.714286
10	rwd	hatchback	14337.777778
11	rwd	sedan	21711.833333

A table of this form isn't the easiest to read and also not very easy to visualize.

To make it easier to understand, we can transform this table to a pivot table by using the `pivot` method.

`pivot`

Pandas method - Pivot()

- One variable displayed along the columns and the other variable displayed along the rows.

```
df_pivot = df_grp.pivot(index= 'drive-wheels', columns='body-style')
```

	price				
body-style	**convertible**	**hardtop**	**hatchback**	**sedan**	**wagon**
drive-wheels					
4wd	20239.229524	20239.229524	7603.000000	12647.333333	9095.750000
fwd	11595.000000	8249.000000	8396.387755	9811.800000	9997.333333
rwd	23949.600000	24202.714286	14337.777778	21711.833333	16994.222222

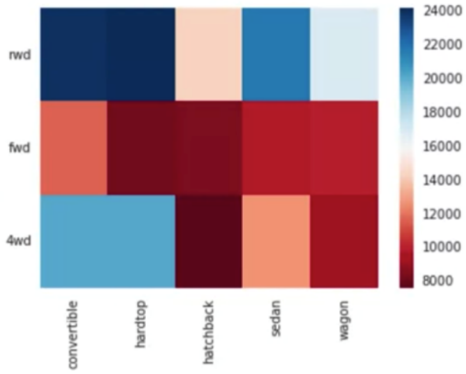
The price data now becomes a rectangular grid, which is easier to visualize. This is similar to what is usually done in Excel **spreadsheets**. Another way to represent the pivot table is using a **heat map** plot.

Heatmap

Heatmap

- Plot target variable over multiple variables

```
plt.pcolor(df_pivot, cmap='RdBu')
plt.colorbar()
plt.show()
```

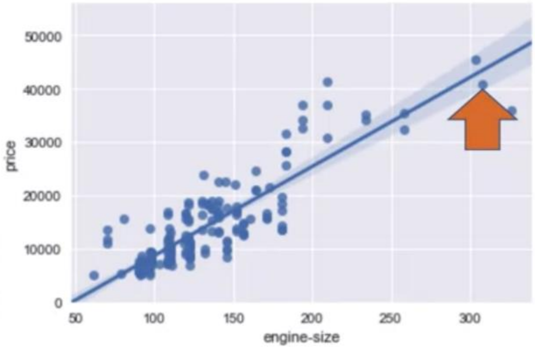


Correlation

Correlation - Positive Linear Relationship

- Correlation between two features (engine-size and price).

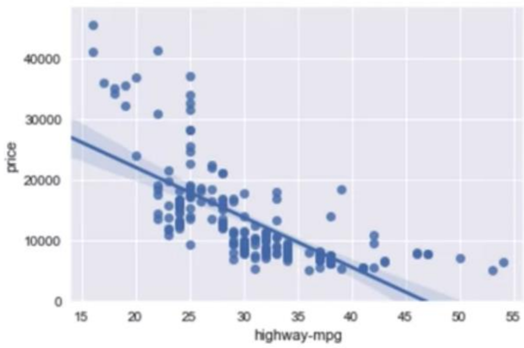
```
sns.regplot(x="engine-size", y= "price", data=df)
plt.ylim(0,)
```

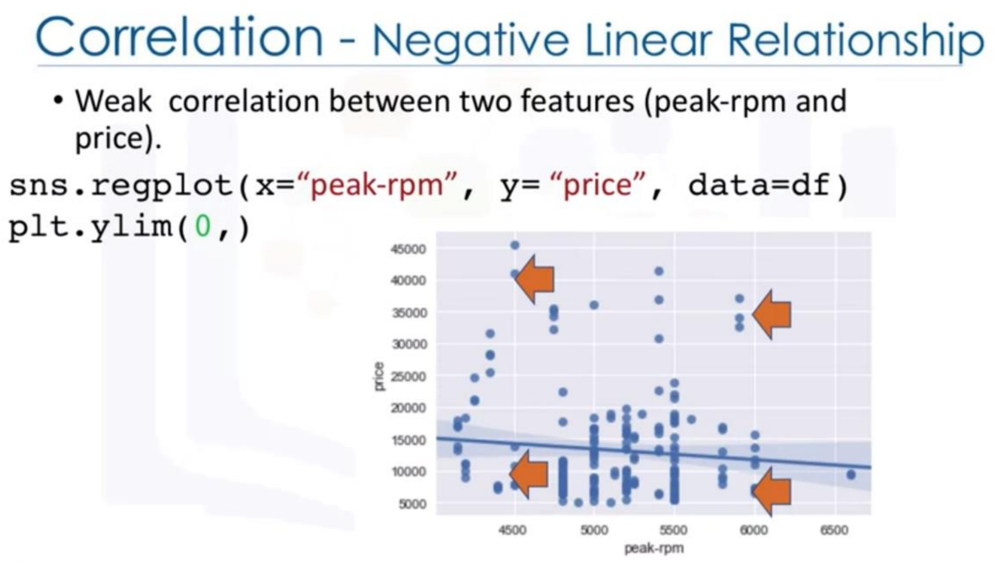


Correlation - Negative Linear Relationship

- Correlation between two features (highway-mpg and price).

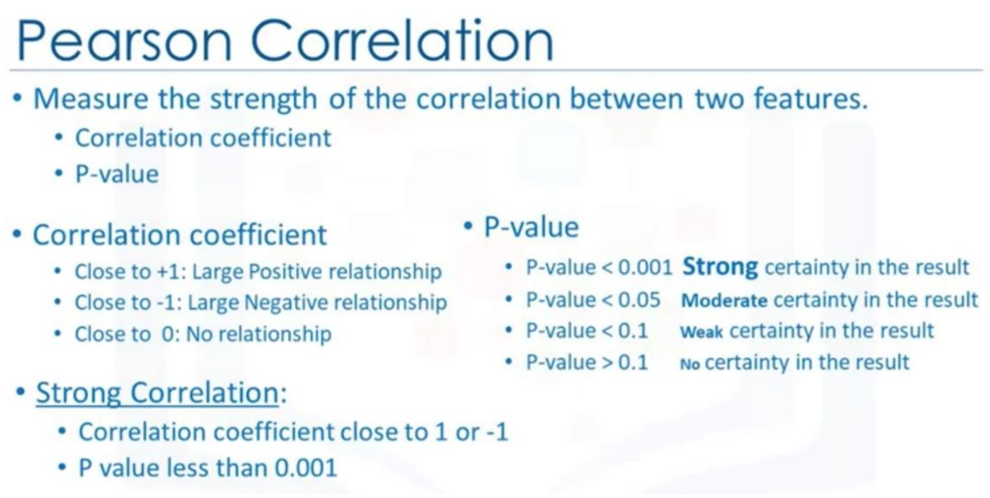
```
sns.regplot(x="highway-mpg", y= "price", data=df)
plt.ylim(0,)
```

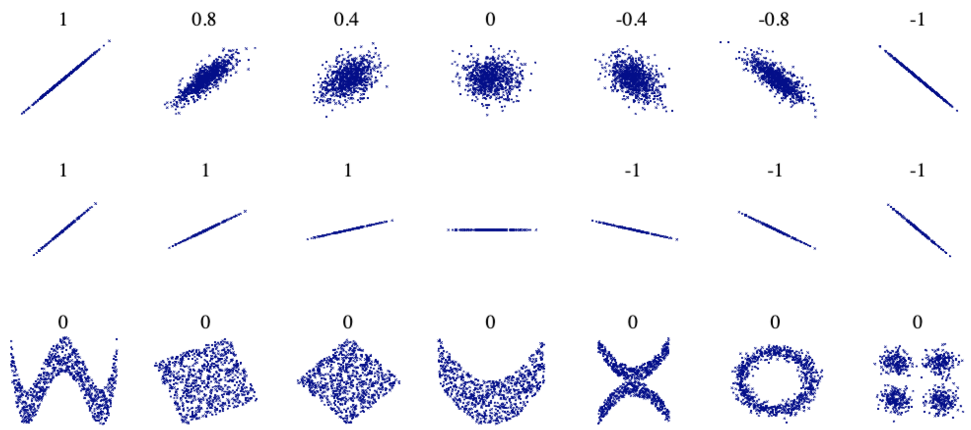




Correlation - Statistics

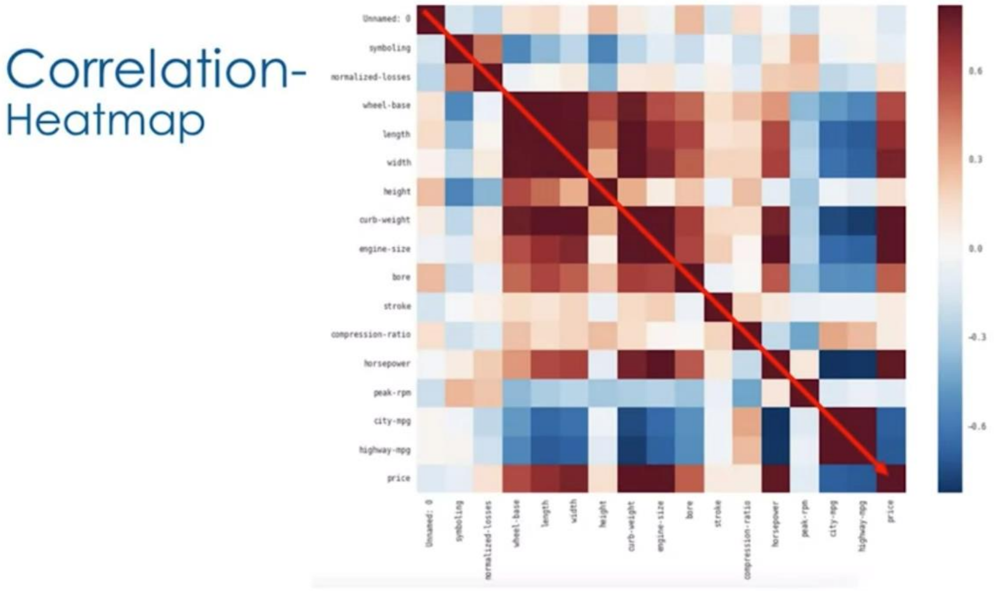
Pearson Correlation





The correlation reflects the noisiness and direction of a linear relationship (top row), but not the slope of that relationship (middle), nor many aspects of nonlinear relationships (bottom). N.B.: the figure in the center has a slope of 0 but in that case the correlation coefficient is undefined because the variance of Y is zero.

Correlation Heatmap



Association between two categorical variables: Chi-Square

Categorical variables

- use the Chi-square Test for Association (denoted as $\chi 2$)
- The test is intended to test how likely it is that an observed distribution is due to chance

Chi-Square Test of association

- The Chi-square tests a null hypothesis that the variables are independent.
- The Chi-square does not tell you the type of relationship that exists between both variables; but only that a relationship exists.

See also: [Chi-Square Test of Independence]

Categorical variables

- Is there an association between fuel-type and aspiration?

$$\chi 2 = \sum_{i=1}^{n} \frac{(O_i - \boxed{E_i})^2}{\boxed{E_i}}$$

Observed value

	Standard	Turbo	Total
diesel	7	13	20
gas	161	24	185
Total	168	37	205

$$\frac{\text{Row total} \quad * \text{ Column total}}{\text{Grand total}}$$

Categorical variables

$$\chi 2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

	Standard	Turbo	Total
diesel	7	13	20
gas	161	24	185
Total	168	37	205

Observed value

Expected value

aspiration / Fuel-type	Standard	Turbo	
Diesel	16.39	3.61	20
Gas	151.61	33.39	185
	168	37	

$$\frac{\text{Row total} \quad * \text{ Column total}}{\text{Grand total}}$$

Categorical variables

$$\chi 2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

Degree of freedom = (row-1)*(column-1)

$$\chi 2 = 29.6$$

Percentage Points of the Chi-Square Distribution

Degrees of Freedom	Probability of a larger value of x [2]							
	0.99	0.95	0.90	0.75	0.50	0.25	0.10	0.05
1	0.000	0.004	0.016	0.102	0.455	1.32	2.71	3.84
2	0.020	0.103	0.211	0.575	1.386	2.77	4.61	5.99
3	0.115	0.352	0.584	1.212	2.366	4.11	6.25	7.81
4	0.297	0.711	1.064	1.923	3.357	5.39	7.78	9.49
5	0.554	1.145	1.610	2.675	4.351	6.63	9.24	11.07
6	0.872	1.635	2.204	3.455	5.348	7.84	10.64	12.59
7	1.239	2.167	2.833	4.255	6.346	9.04	12.02	14.07
8	1.647	2.733	3.490	5.071	7.344	10.22	13.36	15.51
9	2.088	3.325	4.168	5.899	8.343	11.39	14.68	16.92

P-value < 0.05, we reject the null hypothesis that the two variables are independent and conclude that there is evidence of association between fuel-type and aspiration.

Categorical variables

$$\chi 2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

	Standard	Turbo	Total
diesel	7	13	20
gas	161	24	185
Total	168	37	205

```
scipy.stats.chi2_contingency(cont_table, correction = True)
```

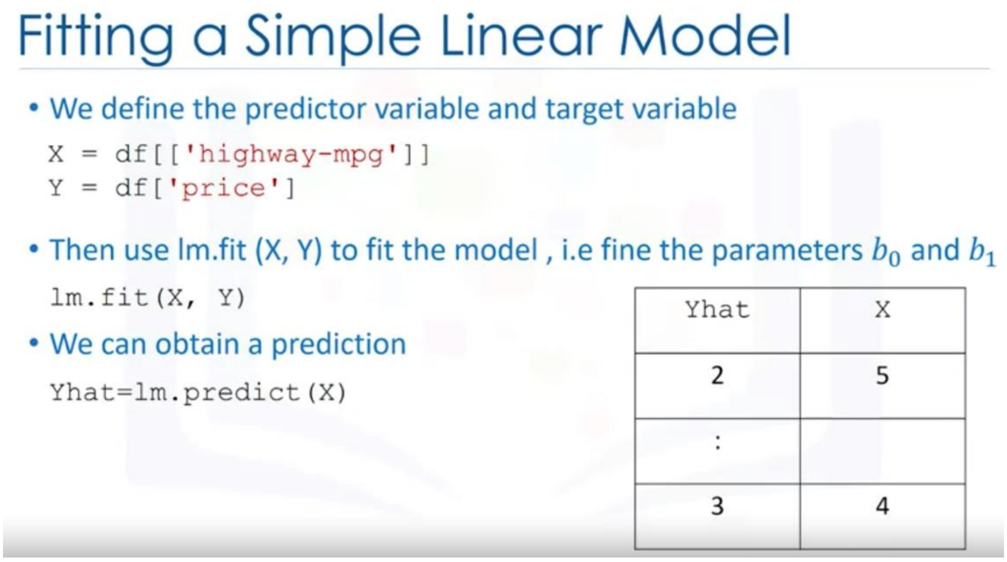
```
(29.605759385109046,
 5.2947382636786724e-08,
 1,
 array([[ 16.3902439,   3.6097561],
        [151.6097561,  33.3902439]]))
```

P-value of < 0.05, we reject the null hypothesis that the two variables are independent and conclude that there is evidence of association between fuel-type and aspiration.

Model Development

- simple linear regression
- multiple linear regression
- polynomial regression

Linear Regression and Multiple Linear Regression



SLR – Estimated Linear Model

- We can view the intercept (b_0): `lm.intercept_` 38423.305858
- We can also view the slope (b_1): `lm.coef_` -821.73337832
- The Relationship between Price and Highway MPG is given by:
- **Price = 38423.31 - 821.73 * highway-mpg**

$$\hat{Y} = b_0 + b_1 x$$

Fitting a Multiple Linear Model Estimator

1. We can extract the for 4 predictor variables and store them in the variable Z

```
Z = df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']]
```

2. Then train the model as before:

```
lm.fit(Z, df['price'])
```

3. We can also obtain a prediction

```
Yhat=lm.predict(X)
```

x_1	x_2	x_3	x_4
3	5	-4	3
:	:	:	:
2	4	2	-4

Yhat
2
:
3

MLR – Estimated Linear Model

1. Find the intercept (b_0)

```
lm.intercept_
-15678.742628061467
```

2. Find the coefficients (b_1, b_2, b_3, b_4)

```
lm.coef_
array([52.65851272 ,4.69878948,81.95906216 , 33.58258185])
```

The Estimated Linear Model:

- **Price** = -15678.74 + (52.66) * **horsepower** + (4.70) * **curb-weight** + (81.96) * **engine-size** + (33.58) * **highway-mpg**

Model Evaluation using Visualization

Regression Plot

Regression plot gives us a good estimate of:

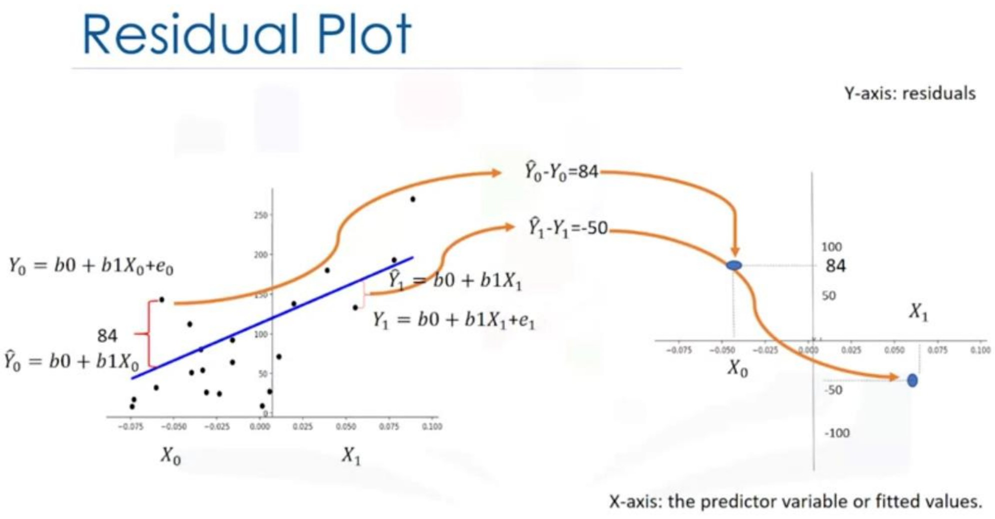
- the relationship between two variables
- the strength of the correlation
- the direction of the relationship (positive or negative)

Regression plot shows us a combination of:

- the scatterplot: where each point represents a different `y`
- the fitted linear regression line (ŷ)

```
import seaborn as sns

sns.regplot(x="highway-mpg", y="price", data=df)
plt.ylim(0,)
```

Residual Plot

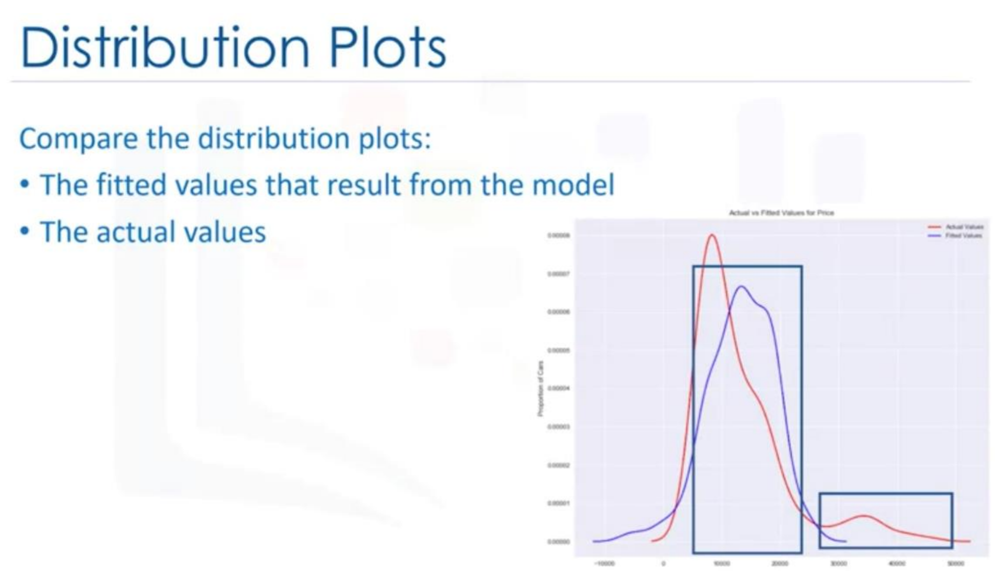


We expect to see the results to have **zero mean**, distributed **evenly** around the `x` axis with similar variance.

```
import seaborn as sns

sns.residplot(df["highway-mpg"], df["price"])
```

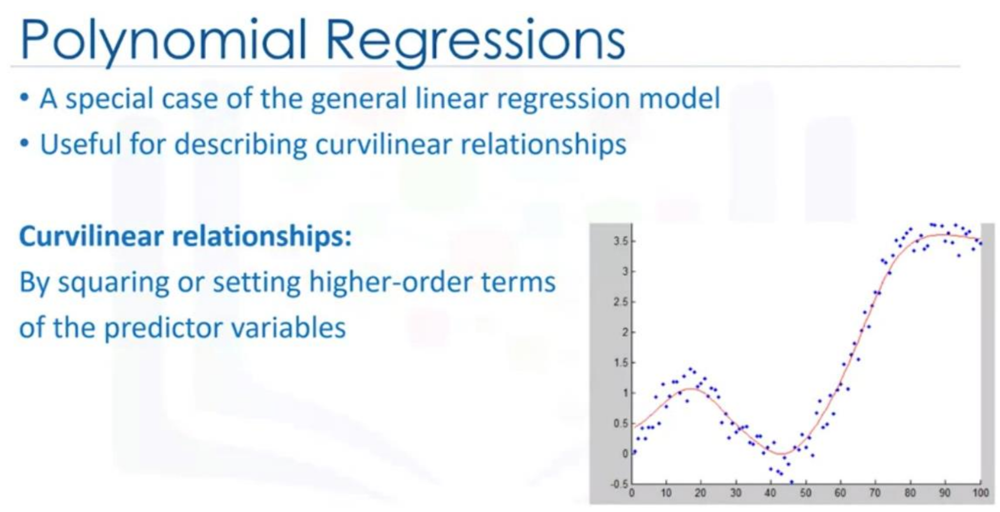
Distribution Plots

A distribution plot counts the predicted value versus the actual value. These plots are extremely useful for visualizing models with more than one independent variable or feature.




```
import seaborn as sns

ax1 = sns.distplot(df["price"], hist=False, color="r", label="Actual Value")

sns.distplot(Yhat, hist=False, color="b", label="Fitted Value", ax=ax1)
```

Polynomial Regression and Pipelines



Polynomial Regression

- Quadratic – 2nd order

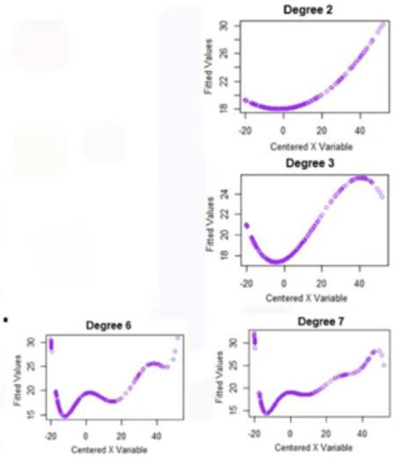
$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2$$

- Cubic – 3rd order

$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2 + b_3 (x_1)^3$$

- Higher order

$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2 + b_3 (x_1)^3 + ..$$



Polynomial Regression

1. Calculate Polynomial of 3rd order

```
f=np.polyfit(x,y,3)

p=np.poly1d(f)
```

2. We can print out the model

```
print (p)
```

$$-1.557(x_1)^3 + 204.8(x_1)^2 + 8965\, x_1 + 1.37\text{x}10^5$$

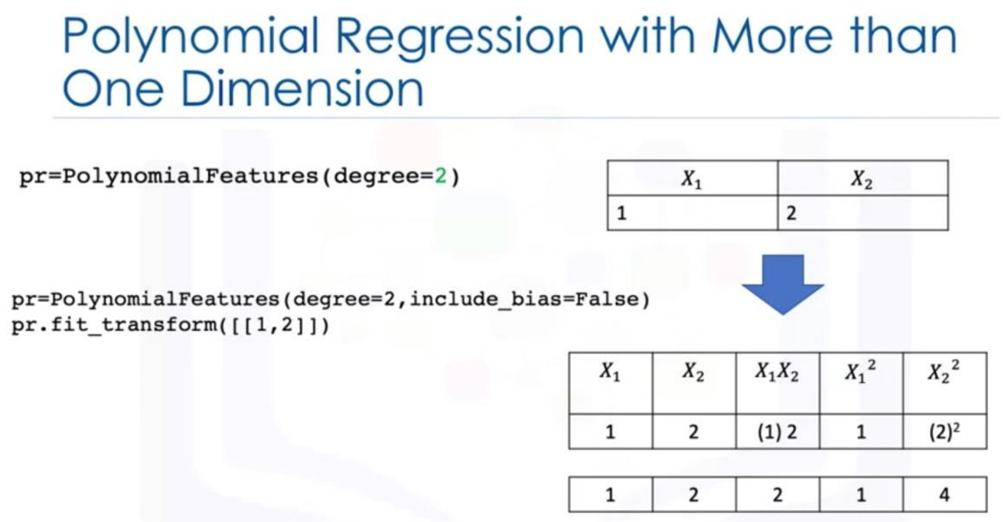
Polynomial Regression with More than One Dimension

- We can also have multi dimensional polynomial linear regression

$$\hat{Y} = b0 + b1\, X_1 + b2\, X_2 + b3\, X_1 X_2 + b4(X_1)^2 + b5(X_2)^2 + ..$$

Numpy's polyfit function cannot perform this type of regression. We use the preprocessing library in scikit-learn to create a polynomial feature object.

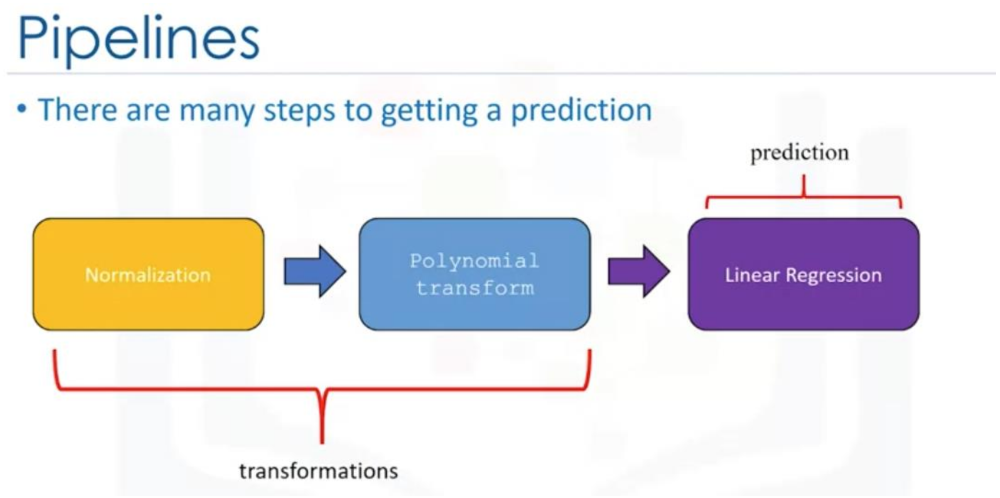
```
from sklearn.preprocessing import PolynomialFeatures

pr = PolynomialFeatures(degree=2, include_bias=False)
x_poly = pr.fit_transform(x[['horsepower', 'curb-weight']])
```



As the dimension of the data gets larger, we may want to normalize multiple features in scikit-learn. Instead we can use the preprocessing module to simplify many tasks. For example, we can standardize each feature simultaneously. We import `StandardScaler`.

```
from sklearn.preprocessing import StandardScaler
SCALE = StandardScaler()
SCALE.fit(x_data[['horsepower', 'highway-mpg']])
x_scale = SCALE.transform(x_data[['horsepower', 'highway-mpg']])
```

We can simplify our code by using a pipeline library.




```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline

Input = [('scale', StandardScaler()), ('polynomial', PolynomialFeatures(degree=2),...),
('model', LinearRegression())]
pipe = Pipeline(Input)
pipe.fit(df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']], y)
yhat = pipe.predict(X[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']])
```

Measures for In-Sample Evaluation

Measures for In-Sample Evaluation

- A way to numerically determine how good the model fits on dataset
- Two important measures to determine the fit of a model:
    - Mean Squared Error (MSE)
    - R-squared (R^2)

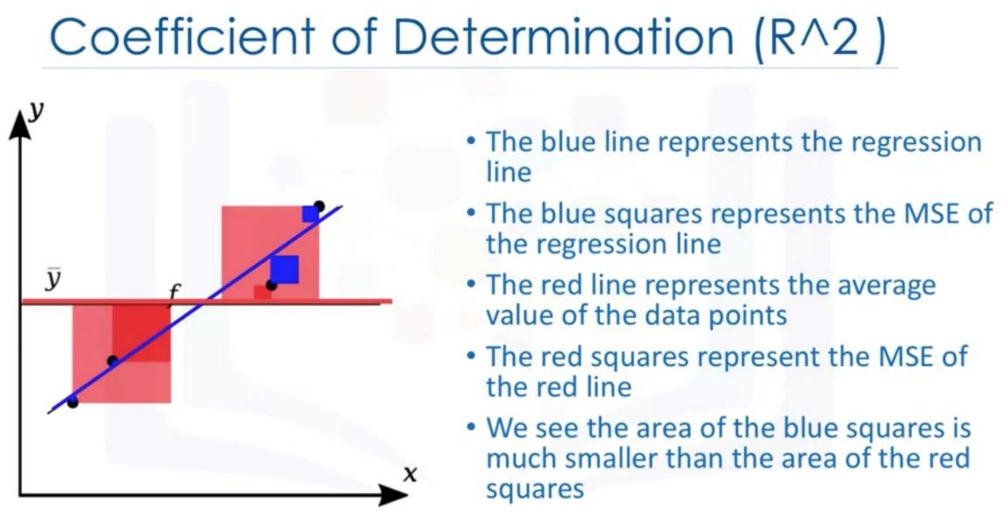
Mean Squared Error (MSE)

```
from sklearn.metrics import mean_square_error

mean_square_error(df['price'], Y_predict_simple_fit)
```

R-squared

- The [Coefficient of Determination]() or R-squared (R^2)
- Is a measure to determine how close the data is to the fitted regression line.
- R^2: the percentage of variation of the target variable (Y) that is explained by the linear model.
- think about as comparing a regression model to a simple model i.e. the mean of the data points

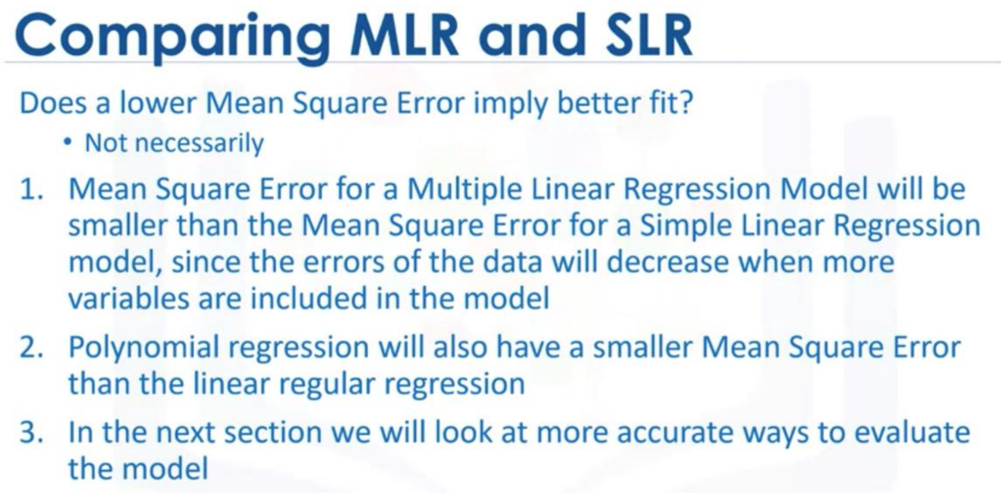
R^2=(1-(MSE of regression line)/(MSE of the average of the data))



- Generally the values of the MSE are between 0 and 1
- We can calculate the R^2 as follows

```python
X = df[['highway-mpg']]
Y = df['price']
lm.fit(X, Y)
lm.score(X, Y)  # 0.496591188
```

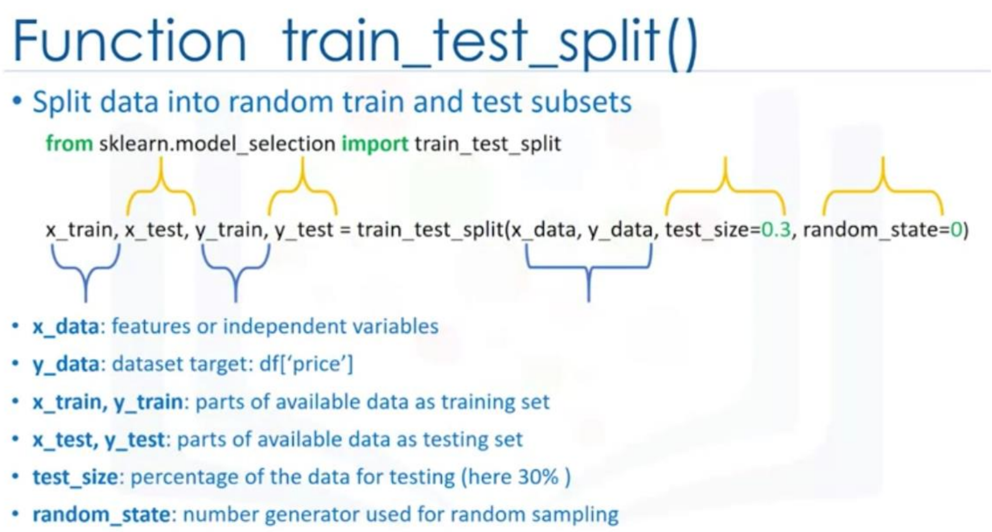
We can say that approximately **49.695%** of the variation of price is explained by this simple linear model.



Model Evaluation and Refinement

Training/Testing Sets

- Split dataset into:
  - Training set (70%)
  - Testing set (30%)
- Build and train the model with a training set
- Use testing set to assess the performance of a predictive model
- When we have completed testing our model we should use all the data to train the model to get the best performance



Function `cross_val_score()`

One of the most common out of sample evaluation metrics is cross-validation.

- In this method, the dataset is split into K equal groups.
- Each group is referred to as a fold. For example, four folds. Some of the folds can be used as a training set which we use to train the model and the remaining parts are used as a test set, which we use to test the model.
- For example, we can use three folds for training, then use one fold for testing. This is repeated until each partition is used for both training and testing.
- At the end, we use the average results as the estimate of out-of-sample error.
- The evaluation metric depends on the model, for example, the r squared.

The simplest way to apply cross-validation is to call the `cross_val_score` function, which performs multiple out-of-sample evaluations.

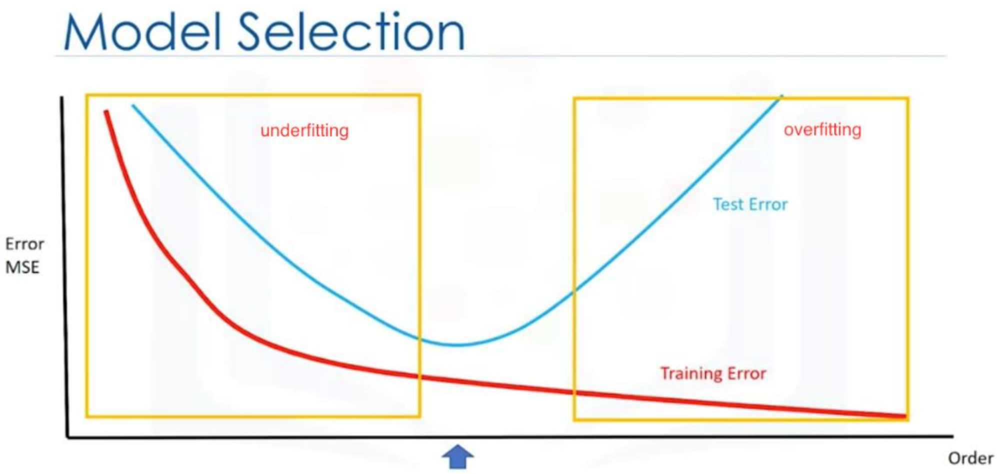
```
from sklearn.model_selection import cross_val_score

score = cross_val_score(lr, x_data, y_data, cv=3)
np.mean(scores)
```

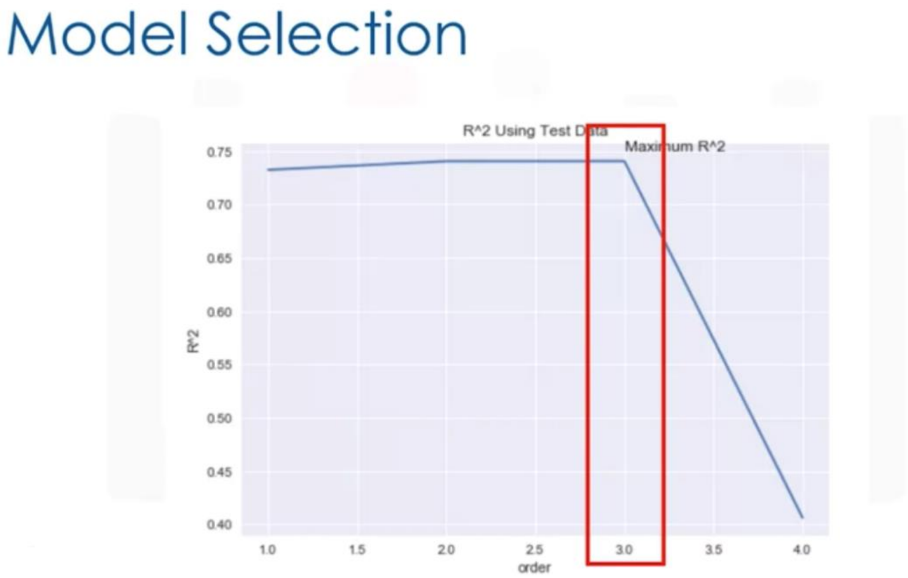
Function `cross_val_predict()`

- It returns the prediction that was obtained for each element when it was in the test set
- Has a similar interface to `cross_val_score()`

```
from sklearn.model_selection import cross_val_predict

yhat = cross_val_predict(lr2e, x_data, y_data, cv=3)
```

Overfitting, Underfitting and Model Selection



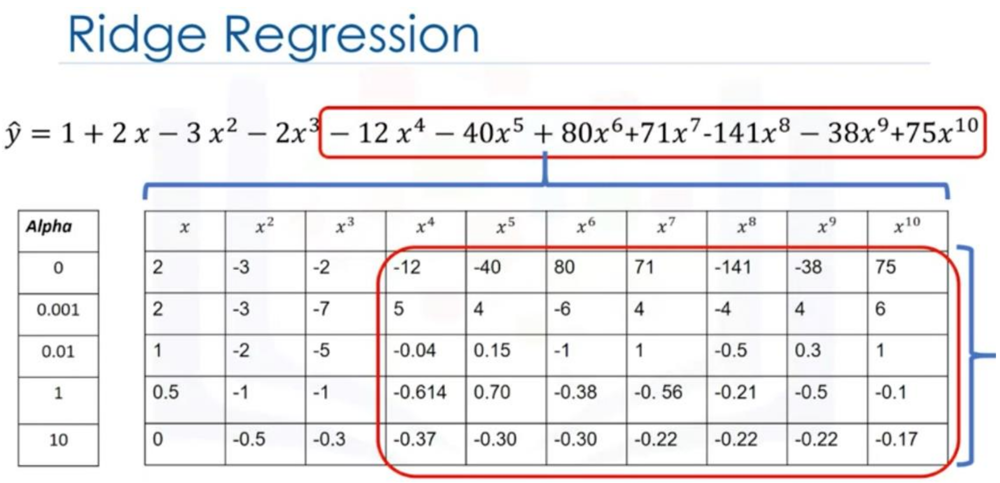


Calculate different R-squared values as follows:

```
Rsqu_test = []
order = [1,2,3,4]

for n in order:
  pr = PolynomialFeatures(degree=n)
  x_train_pr = pr.fit_transform(x_train[['horsepower']])
  x_test_pr = pr.fit_transform(x_test[['horsepower']])
  lr.fit(x_train_pr, y_train)
  Rsqu_test.append(lr.score(x_test_pr, y_test))
```

Ridge Regression

Ridge regression is a regression that is employed in a Multiple regression model when Multicollinearity occurs. Multicollinearity is when there is a strong relationship among the independent variables. Ridge regression is very common with polynomial regression.

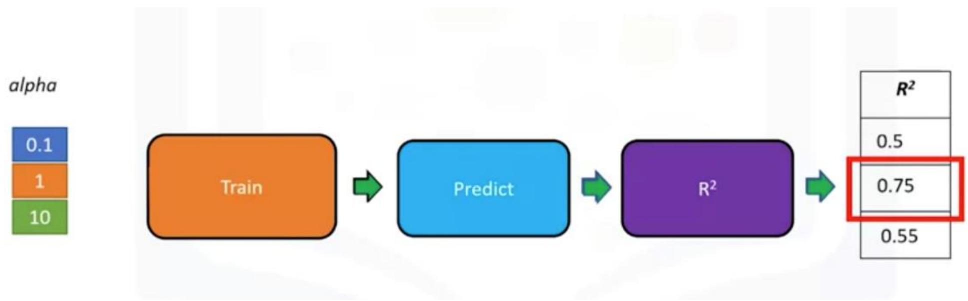


$$\hat{y} = 1 + 2\,x - 3\,x^2 - 2x^3 \boxed{- 12\,x^4 - 40x^5 + 80x^6 + 71x^7 - 141x^8 - 38x^9 + 75x^{10}}$$

Alpha	x	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9	x^{10}
0	2	-3	-2	-12	-40	80	71	-141	-38	75
0.001	2	-3	-7	5	4	-6	4	-4	4	6
0.01	1	-2	-5	-0.04	0.15	-1	1	-0.5	0.3	1
1	0.5	-1	-1	-0.614	0.70	-0.38	-0. 56	-0.21	-0.5	-0.1
10	0	-0.5	-0.3	-0.37	-0.30	-0.30	-0.22	-0.22	-0.22	-0.17

The column corresponds to the different polynomial coefficients, and the rows correspond to the different values of alpha.

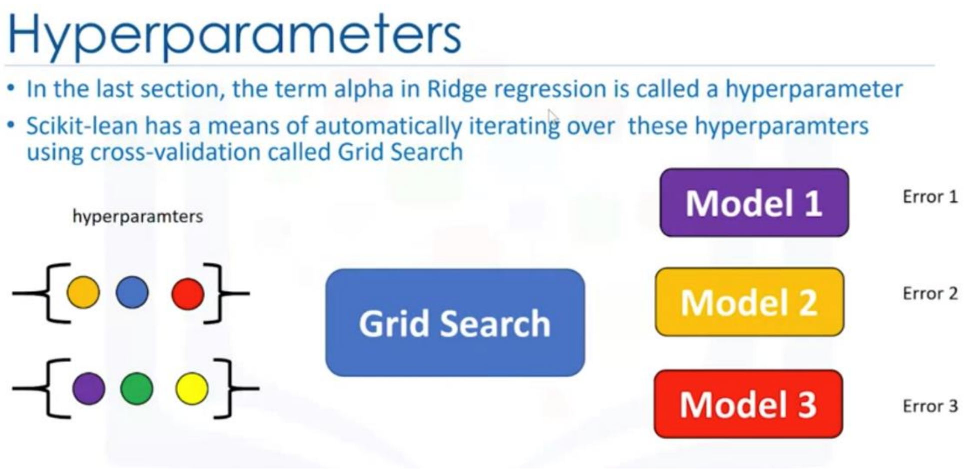
- As alpha increases, the parameters get smaller. This is most evident for the higher order polynomial features.
- But Alpha must be selected carefully.
    - If alpha is too large, the coefficients will approach zero and underfit the data.
    - If alpha is zero, the overfitting is evident.



Grid Search

- The term alpha in Ridge regression is called a **hyperparameter**.
- Scikit-learn has a means of automatically iterating over these hyperparameters using cross-validation called **Grid Search**.

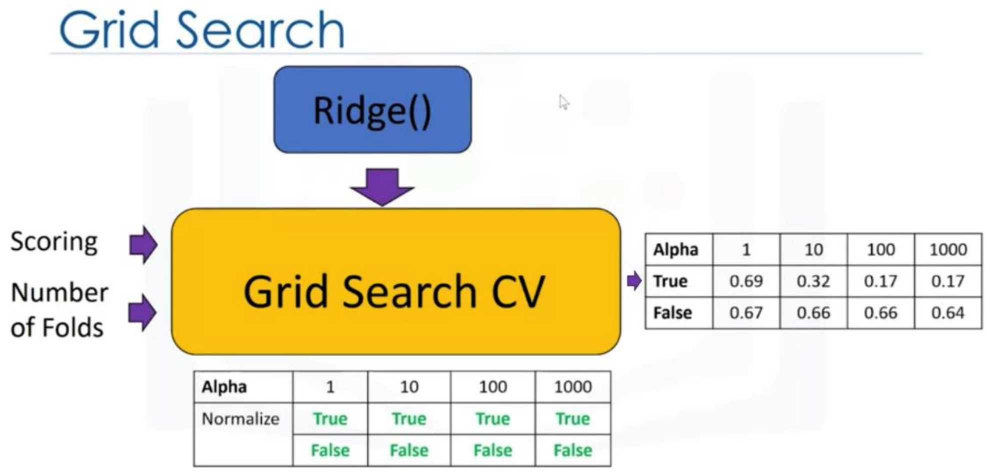
Grid Search takes the model or objects you would like to train and different values of the hyperparameters. It then calculates the mean square error or R-squared for various hyperparameter values, allowing you to choose the best values.



Use the validation dataset to pick the best hyperparameters.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

parameters1 = [{'alpha': [0.001, 0.1, 1, 10, 100, 1000, 10000, 100000]}]

RR = Ridge()
Grid1 = GridSearchCV(RR, parameters1, cv=4)
Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']], y_data)
Grid1.best_estimator_

scores = Grid1.cv_results_
scores['mean_test_score']
```

What are the advantages of Grid Search is how quickly we can test **multiple parameters**.




```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

parameters2 = [{'alpha': [0.001, 0.1, 1, 10, 100], 'normalize': [True, False]}]

RR = Ridge()
Grid1 = GridSearchCV(RR, parameters2, cv=4)
```

```python
Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']], y_data)
Grid1.best_estimator_

scores = Grid1.cv_results_

for param, mean_val, mean_test in zip(scores['params'], scores['mean_test_score'], scores['mean_train_score']):
  print(param, "R^2 on test data:", mean_val, "R^2 on train data:", mean_test)
```

```
{'alpha': 0.001, 'normalize': True} R^2 on tesst data: 0.66605547293 R^2 on train data: 0.814001968709
{'alpha': 0.001, 'normalize': False} R^2 on tesst data: 0.665488366584 R^2 on train data: 0.814002698797
{'alpha': 0.1, 'normalize': True} R^2 on tesst data: 0.694175625356 R^2 on train data: 0.810546768311
{'alpha': 0.1, 'normalize': False} R^2 on tesst data: 0.665488937796 R^2 on train data: 0.814002698794
{'alpha': 1, 'normalize': True} R^2 on tesst data: 0.690486934584 R^2 on train data: 0.749104440368
{'alpha': 1, 'normalize': False} R^2 on tesst data: 0.665494127178 R^2 on train data: 0.814002698472
{'alpha': 10, 'normalize': True} R^2 on tesst data: 0.321376875232 R^2 on train data: 0.341856042902
{'alpha': 10, 'normalize': False} R^2 on tesst data: 0.665545680812 R^2 on train data: 0.8140026666
{'alpha': 100, 'normalize': True} R^2 on tesst data: 0.0170551710263 R^2 on train data: 0.0496044796826
{'alpha': 100, 'normalize': False} R^2 on tesst data: 0.666029359996 R^2 on train data: 0.813999791851
{'alpha': 1000, 'normalize': True} R^2 on tesst data: -0.0301961745066 R^2 on train data: 0.005184451599
{'alpha': 1000, 'normalize': False} R^2 on tesst data: 0.668968215369 R^2 on train data: 0.813870488264
{'alpha': 10000, 'normalize': True} R^2 on tesst data: -0.0351687400461 R^2 on train data: 0.000520784757979
{'alpha': 10000, 'normalize': False} R^2 on tesst data: 0.673346359342 R^2 on train data: 0.812583743226
{'alpha': 100000, 'normalize': True} R^2 on tesst data: -0.0356685844558 R^2 on train data: 5.2101975528e-05
{'alpha': 100000, 'normalize': False} R^2 on tesst data: 0.657818838432 R^2 on train data: 0.789541446486
{'alpha': 100000, 'normalize': True} R^2 on tesst data: -0.0356685844558 R^2 on train data: 5.2101975528e-05
{'alpha': 100000, 'normalize': False} R^2 on tesst data: 0.657818838432 R^2 on train data: 0.789541446486
```