- MODULE #1: Importing Datasets
- MODULE #2: Data Wrangling
- MODULE #3: Exploratory Data Analysis
- MODULE #4: Model Development
- MODULE #5: Model Evaluation
- MODULE #6: Final Assignment

❖ MODULE #1: Importing Datasets

The Problem

- Why Data Analysis?
    - Data is everywhere
    - Data analysis/data science helps us answer questions from data
    - Data analysis plays an important role in:
        - Discovering useful information
        - Answering questions
- Tom wants to sell his car
    - How much money should he sell his car for?
        - The price he sets should not be too high, but not too low either
- Estimated used car prices: How can we help Tom determine the best price for his car?
    - Is there data on the prices of other cars and their characteristics?
    - What features of cars affect their prices?
        - Color? Brand? Horsepower? Something else?
    - Asking the right questions in terms of data

Understanding the Data

- CSV
    - Separates each of the values with commas
    - Each line represents a row in the dataset
    - Sometimes the first row is a header, which contains a column name for each of the 26 columns.
- Name of the attribute that we want to predict: target (label)

Python Packages for Data Science

- A Python library is a collection of functions and methods that allows you to perform lots of actions without writing your code
- Scientific Computing Libraries
    - Pandas (Data structures & tools)
    - NumPy (Arrays & matrices)
    - SciPy (Integrals, solving differential equations, optimizations)
- Visualization Libraries
    - Matplotlib (plots & graphs, most popular)
    - Seaborn (plots: heat maps, time series, violin plots)
- Algorithmic Libraries
    - Scikit-learn (Machine Learning: regression, classification, clustering, etc.)

- Statsmodels (Explore data, estimate statistical models and perform statistical tests)

Importing and Exporting Data in Python

- Importing Data
  - Process of loading and reading data into Python from various resources
  - Two important properties:
    - Various formats: .csv, .json, .xlsx, .hdf, etc.
  - File Path of dataset
    - Computer: /Desktop/mydata.csv
    - Internet: https://archive.ics.uci.edu/autos/imports-85.data
- Importing a CSV into Python
  - import pandas
    url = “link”
    df = pd.read_csv(url)
- Importing a CSV without a header
  - import pandas
    url = “link”
    df = pd.read_csv(url, header = None)
- Printing the dataframe in Python
  - df prints the entire dataframe (not recommended for large datasets)
  - df.head(n) shows the first n rows of data frame
  - df.tail(n) shows the bottom n rows of data frame
- Adding headers
  - Replace default header (by df.columns = headers)
    -
```
headers = ["symboling","normalized-losses","make","fuel-type","aspiration","num-of-doors","body-style",
"drive-wheels","engine-location","wheel-base","length","width","height","curb-weight","engine-type",
"num-of-cylinders","engine-size","fuel-system","bore","stroke","compression-ratio","horsepower","peak-
rpm","city-mpg","highway-mpg","price"]
```
    - df.head()
    -

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	h
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	1
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	1
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	1
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	1
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	1

- Exporting a Pandas dataframe to CSV
  - Preserve progress anytime by saving modified dataset using
    -
```
path="C:/Windows/.../ automobile.csv"
df.to_csv(path)
```
- 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()

Getting Started Analyzing Data in Python

- Basic insights from the data
  - Understand your data before you begin any analysis
  - Should check:
    - Data Types
    - Data Distribution
  - Locate potential issues with the data
- Data Types

Pandas Type	Native Python Type	Description
object	string	numbers and strings
int64	int	Numeric characters
float64	float	Numeric characters with decimals
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	time data.

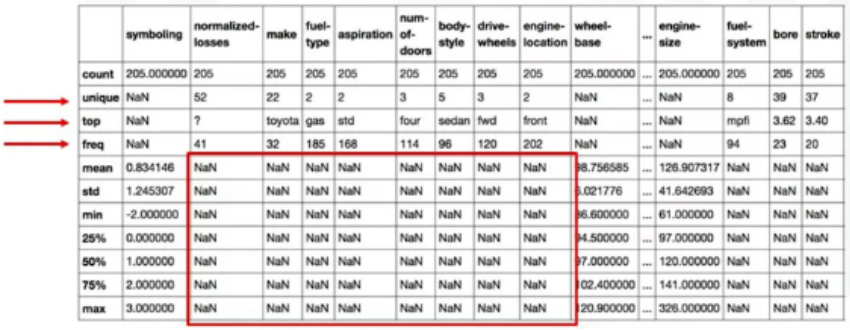
  - Why check data types?
    - Potential info and type mismatch
    - Compatibility with Python methods
  - In pandas, we use dataframe.dtypes to check data types
    - df.dtypes

```
width              float64
height             float64
curb-weight          int64
engine-type         object
num-of-cylinders    object
```

- dataframe.describe()
  - Returns a statistical summary
    - df.describe()

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	10.142537	25.219512	30.751220
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	3.972040	6.542142	6.886443
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	7.000000	13.000000	16.000000
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	8.600000	19.000000	25.000000
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	9.000000	24.000000	30.000000
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	9.400000	30.000000	34.000000
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	23.000000	49.000000	54.000000

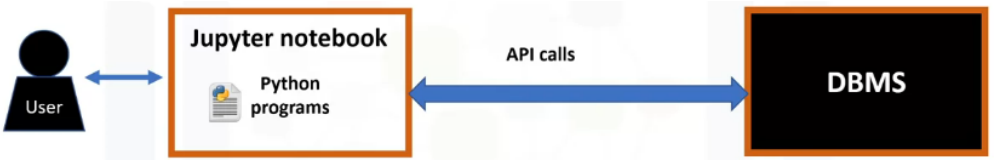
    - count, average column value (mean), standard deviation (std), maximum/minimum values and boundaries between each of the quartiles
- dataframe.describe(include="all")

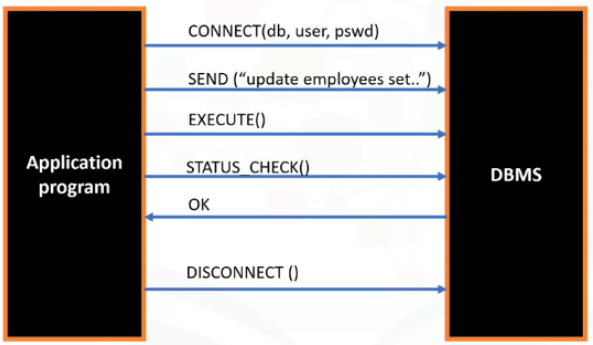
- Provides full summary statistics
  - df.describe(include="all")
  - 

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke
count	205.000000	205	205	205	205	205	205	205	205	205.000000	...	205.000000	205	205	205
unique	NaN	52	22	2	2	3	5	3	2	NaN	...	NaN	8	39	37
top	NaN	?	toyota	gas	std	four	sedan	fwd	front	NaN	...	NaN	mpfi	3.62	3.40
freq	NaN	41	32	185	168	114	96	120	202	NaN	...	NaN	94	23	20
mean	0.834146	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	98.756585	...	126.907317	NaN	NaN	NaN
std	1.245307	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6.021776	...	41.642693	NaN	NaN	NaN
min	-2.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	86.600000	...	61.000000	NaN	NaN	NaN
25%	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	94.500000	...	97.000000	NaN	NaN	NaN
50%	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	97.000000	...	120.000000	NaN	NaN	NaN
75%	2.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	102.400000	...	141.000000	NaN	NaN	NaN
max	3.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	120.900000	...	326.000000	NaN	NaN	NaN

  - Unique is the number of distinct objects in the column. Top is most frequently occurring object, and freq is the number of times the top object appears in the column
  - Some values in the table are shown here as NaN which stands for not a number. This is because that particular statistical metric cannot be calculated for that specific column data type.
- dataframe.info()
  - Provides a concise summary of your dataframe
  - This function shows the top 30 rows and bottom 30 rows of the data frame.

Accessing Databases with Python

- 

- What is a SQL API?
  - The SQL API consists of library function calls as an application programming interface, API, for the DBMS. To pass SQL statements to the DBMS, an application program calls functions in the API, and it calls other functions to retrieve query results and status information from the DBMS.
  - 

  - The basic operation of a typical SQL API is illustrated in the figure. The application program begins its database access with one or more API

calls that connect the program to the DBMS. To send the SQL statement to the DBMS, the program builds the statement as a text string in a buffer and then makes an API call to pass the buffer contents to the DBMS. The application program makes API calls to check the status of its DBMS request and to handle errors. The application program ends its database access with an API call that disconnects it from the database.

- What is a DB-API?
    - It is a standard that allows you to write a single program that works with multiple kinds of relational databases instead of writing a separate program for each one. So, if you learn the DB-API functions, then you can apply that knowledge to use any database with Python.
- Concepts of the Python DB API
    - Connection Objects
        - Database connections
        - Manage transactions
    - Cursor Objects
        - Database Queries
- What are Connection methods?
    - cursor(): returns a new cursor object using the connection
    - commit(): commit any pending transaction to the database
    - rollback(): causes the database to roll back to the start of any pending transaction
    - close(): used to close a database connection
- Writing code using DB-API
    -
```
from dbmodule import connect

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

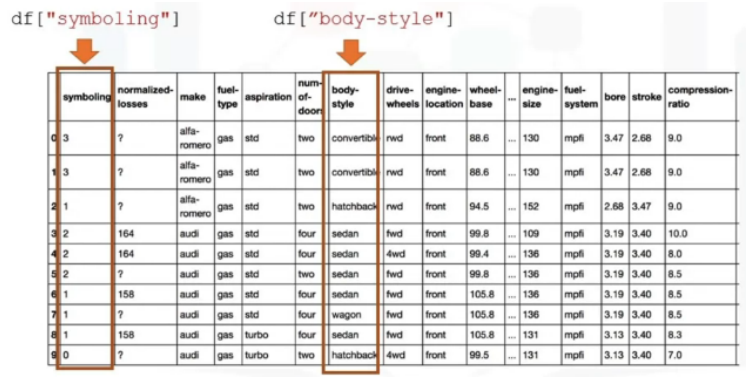
#Create a cursor object
cursor = connection.cursor()

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

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

❖ **MODULE #2: Data Wrangling**

Preprocessing Data in Python

- The process of converting or mapping data from the initial “raw” form into another format, in order to prepare the data for further analysis
- Also known as: Data cleaning, Data wrangling
- Simple Dataframe Operations
  - 

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0
5	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5
6	1	158	audi	gas	std	four	sedan	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5
7	1	?	audi	gas	std	four	wagon	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5
8	1	158	audi	gas	turbo	four	sedan	fwd	front	105.8	...	131	mpfi	3.13	3.40	8.3
9	0	?	audi	gas	turbo	two	hatchback	4wd	front	99.5	...	131	mpfi	3.13	3.40	7.0

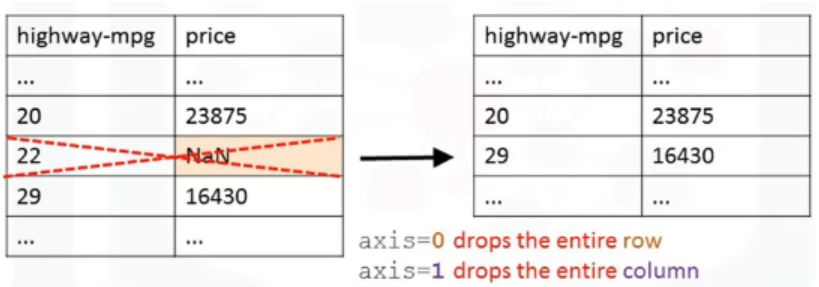
  - ```
    df["symboling"]=df["symboling"]+1
    ```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio
0	4	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0
1	4	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0
2	2	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0
5	3	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5
6	2	158	audi	gas	std	four	sedan	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5
7	2	?	audi	gas	std	four	wagon	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5
8	2	158	audi	gas	turbo	four	sedan	fwd	front	105.8	...	131	mpfi	3.13	3.40	8.3
9	1	?	audi	gas	turbo	two	hatchback	4wd	front	99.5	...	131	mpfi	3.13	3.40	7.0

Dealing with Missing Values in Python

- Missing Values
  - Missing values occur when no data value is stored for a variable (feature) in an observation
  - Could be represented as “?”, “N/A”, 0 or just a blank cell
- 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 data points)
    - Replace it by frequency
    - Replace it based on other functions
  - Leave it as missing data
- How to drop missing values in Python
    ```

- Use dataframes.dropna()
- You need to specify axis
  - axis = 0 drops the entire row
  - axis = 1 drops the entire column



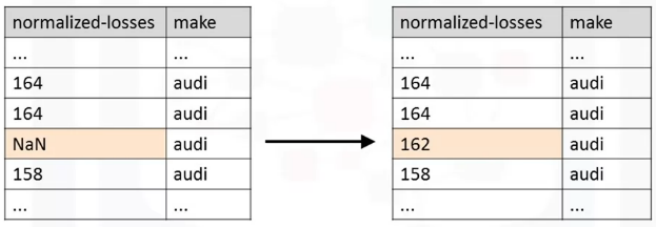

```
df.dropna(subset=["price"], axis=0, inplace = True)
```

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

- Don't Forget
  - You have to set the parameter inplace equal to true

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

- How to replace missing values in Python
  - Use dataframe.replace(missing_value, new_value)

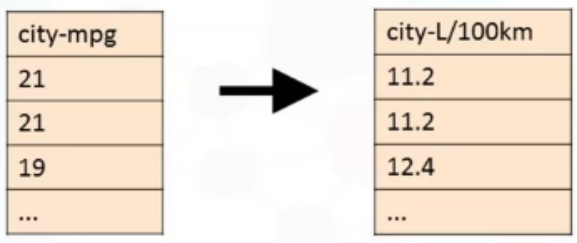



```
mean = df["normalized-losses"].mean()
df["normalized-losses"].replace(np.nan, mean)
```

Data Formatting in Python

- Data Formatting
  - Data are usually collected from different places and stored in different formats
  - Bringing data into a common standard of expression allows users to make meaningful comparison

- Applying Calculations to an entire column
  - Convert "mpg" to "L/100km" in Car dataset

- 

city-mpg
21
21
19
...

city-L/100km
11.2
11.2
12.4
...

    - df[“city-mpg”]= 235/df[“city-mpg”]
      df.rename(columns={“city_mpg”: “city-L/100km”}, inplace=True)
- Incorrect data types
    - Sometimes the wrong data type is assigned to a feature
    - ```
      df[“price”].tail(5)

      200    16845
      201    19045
      202    21485
      203    22470
      204    22625
      Name: price, dtype: object
      ```
        - Should be an int
- Data Types in Python and Pandas
    - There are many data types in pandas
        - Objects: “A”, “Hello”
        - Int64: 1,3,5
        - Float64: 2.123, 632.31
- Correcting data types
    - To identify data types:
        - Use dataframe.dtypes() to identify data type
    - To convert data types
        - Use dataframe.astype() to convert data type
    - Example: convert data type to integer in column “price”
        - df[“price”] = df[“price”].astype(“int”)

Data Normalization in Python

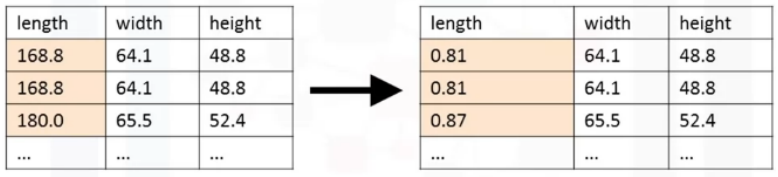
- Data Normalization
    - Uniform the features value with different range
length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
171.2	65.5	52.4
176.6	66.2	54.3
176.6	66.4	54.3
177.3	66.3	53.1
192.7	71.4	55.7
192.7	71.4	55.7
192.7	71.4	55.9
scale	[150,250]	[50,100]	[50,100]
impact	large	small	small
    - Not-normalized
      ```

        - “Age” and “income” are in different range
        - Hard to compare
        - “Income” will influence the result more
    - Normalized
        - Similar value range
        - Similar intrinsic influence on analytical model
    - 

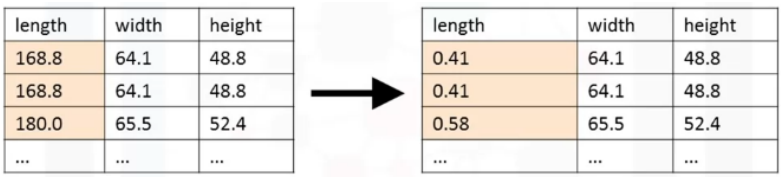
age	income
20	100000
30	20000
40	500000

Not-normalized

age	income
0.2	0.2
0.3	0.04
0.4	1

Normalized
- Methods of normalizing data
    - Simple Feature Scaling
        - $x_{new} = \frac{x_{old}}{x_{max}}$
    - Min-max
        - $x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$
    - Z-score
        - $x_{new} = \frac{x_{old} - \mu}{\sigma}$
        - Miu= average of the feature
        - Sigma= standard deviation
        - Ranges between -3 to +3
- Simple Feature Scaling in Python
    - df[“length”] = df[“length”] / df[“length”].max()
    - 

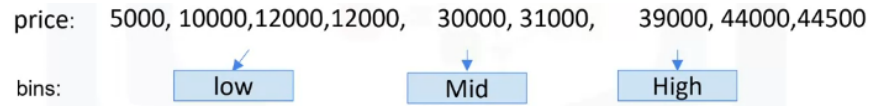
length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
180.0	65.5	52.4
...	...	...

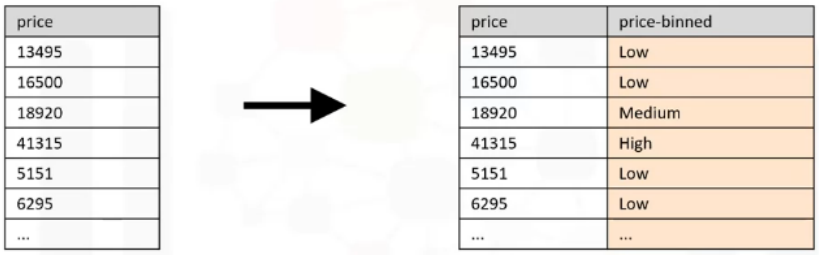
length	width	height
0.81	64.1	48.8
0.81	64.1	48.8
0.87	65.5	52.4
...	...	...
- Min-max in Python
    - df[“length”] = (df[“length”]-df[“length”].min()) / (df[“length”].max()-df[“length”].min())
    - 

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
180.0	65.5	52.4
...	...	...

length	width	height
0.41	64.1	48.8
0.41	64.1	48.8
0.58	65.5	52.4
...	...	...
- Z-score in Python
    - df[“length”] = (df[“length”]-df[“length”].mean()) / df[“length”].std()

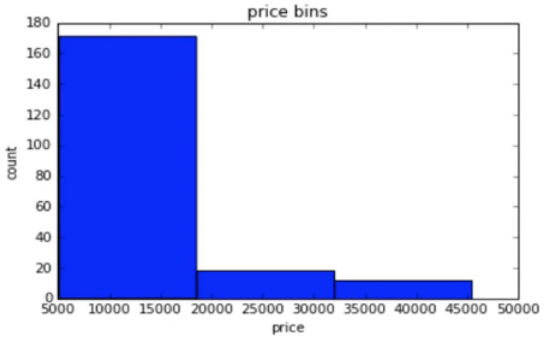
Binning in Python

- Binning
  - Grouping of values into “binss”
  - 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)
    - 

- Binning in Python pandas
  - 

price
13495
16500
18920
41315
5151
6295
...

price	price-binned
13495	Low
16500	Low
18920	Medium
41315	High
5151	Low
6295	Low
...	...

  - 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)
    - Numpy function “linspace”
    - We create a list “group_names “ that contains the different bin names.
    - We use the pandas function ”cut” to segment and sort the data values into bins.
- Visualizing binned data
  - Histograms
  - 

Turning categorical variables into quantitative variables in Python

- Categorical Variables
  - Problem: Most statistical models cannot take in the objects/strings as input

-

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

- 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

    - Also called “one-hot encoding”
- Dummy variables in Python pandas
  - Use pandas.get_dummies() method
  - Convert categorical variables to dummy variables (0 or 1)
    - pd.get_dummies(df[‘fuel’])

❖ **MODULE #3: Exploratory Data Analysis**

Exploratory Data Analysis

- 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
- Question: “What are the characteristics which have the most impact on the car price?”

Descriptive Statistics

- Descriptive Statistics
  - Describe basic features of data
  - Giving short summaries about the sample and measures of the data
- Describe()
  - Summarize statistics using pandas describe() method
  - df.describe()
    -

	Unnamed: 0	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke
count	201.000000	201.000000	164.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.00000
mean	100.000000	0.840796	122.000000	98.797015	174.200995	65.889055	53.766667	2555.666667	126.875622	3.319154	3.256766
std	58.167861	1.254802	35.442168	6.066366	12.322175	2.101471	2.447822	517.296727	41.546834	0.280130	0.316049
min	0.000000	-2.000000	65.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000
25%	50.000000	0.000000	NaN	94.500000	166.800000	64.100000	52.000000	2169.000000	98.000000	3.150000	3.110000
50%	100.000000	1.000000	NaN	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000
75%	150.000000	2.000000	NaN	102.400000	183.500000	66.600000	55.500000	2926.000000	141.000000	3.580000	3.410000
max	200.000000	3.000000	256.000000	120.900000	208.100000	72.000000	59.800000	4066.000000	326.000000	3.940000	4.170000

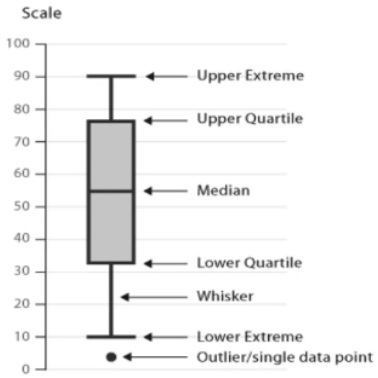
    - NaN values will be excluded
- Value_Counts()
  - Summarize the categorical data is by using the value_counts() method
  - drive_wheels_counts=df[“drive-wheels”].value_counts().to_frame()
    drive_wheels_counts.rename(columns={‘drive-wheels’:’value_counts’}, inplace=True)
    -

	value_counts
drive-wheels	
fwd	118
rwd	75
4wd	8

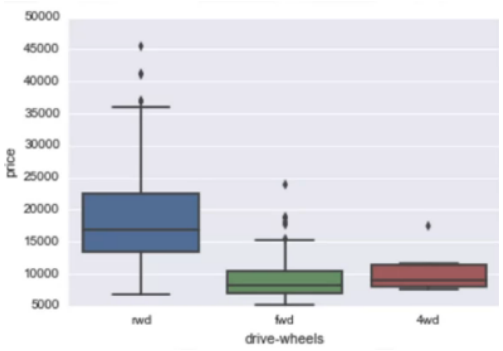
- Box Plots
  -

```
import matplotlib.pyplot as plt
import numpy as np

data = np.random.randn(100)
plt.boxplot(data)
plt.show()
```



- Example:
  - sns.boxplot(x="drive-wheels", y="price", data=df)



- 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
  - Matplotlib scatter function is used
    - y=df["price"]
      x=df["engine-size"]
      plt.scatter(x,y)

    - It's important to label your axes
    - plt.title("Scatterplot of Engine Size vs Price")
      plt.xlabel("Engine Size")
      plt.ylabel("Price")

GroupBy in Python

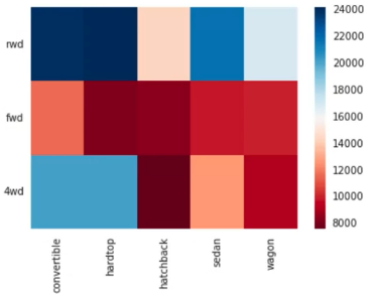
- Grouping data
  - Use Panda dataframe.Groupby() method

        - Can be applied on categorical variables
        - Group data into categories
        - Single or multiple variables
- Groupby() Example
    - df[['price','body-style']].groupby(['body-style'],as_index= False).mean()
        -

	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
12	rwd	wagon	16994.222222

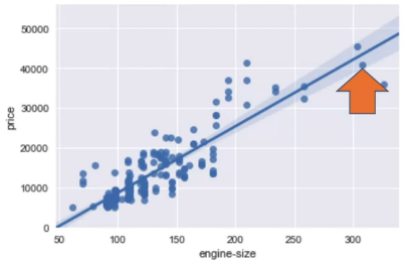
        - A table of this form isn't the easiest to read and also not very easy to visualize. To make it easier to understand, we can transform this table to a pivot table by using the pivot method.
- Pandas method - Pivot()
    - One variable displayed along the columns and the other variable displayed along the columns
    - df_pivot = df_grp.pivot(index= ‘drive-wheels’, columns=’body-style’)
        -

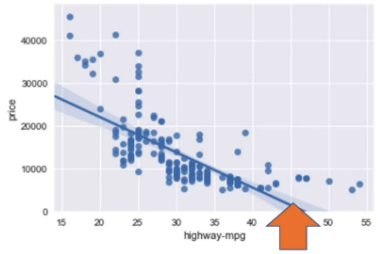
	price				
body-style	**convertible**	**hardtop**	**hatchback**	**sedan**	**wagon**
drive-wheels					
4wd	20239.229524	20239.229524	7603.000000	12647.333333	9095.750000
fwd	11595.000000	8249.000000	8396.387755	9811.800000	9997.333333
rwd	23949.600000	24202.714286	14337.777778	21711.833333	16994.222222

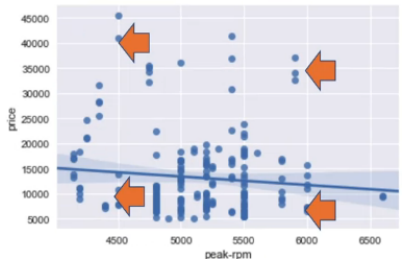
- Heatmap
    - Plot target variable over multiple variables
    - We use pyplot’s pcolor method
    - plt.pcolor(df_pivor, cmap=’RdBu’)
      plt.colorbar()
      plt.show()
        - 

    - According to the color bar, we see that the top section of the heat map seems to have higher prices than the bottom section.

Correlation

- What is Correlation?
  - Measures to what extent different variables are interdependent
  - For example:
    - Lung cancer → Smoking
    - Rain → Umbrella
  - Correlation doesn't imply causation
- Positive Linear Relationship
  - Correlation between two features (engine-size and price)
  - sns.regplot(x="engine-size", y="price", data=df)
    plt.ylim(0,)
    - 

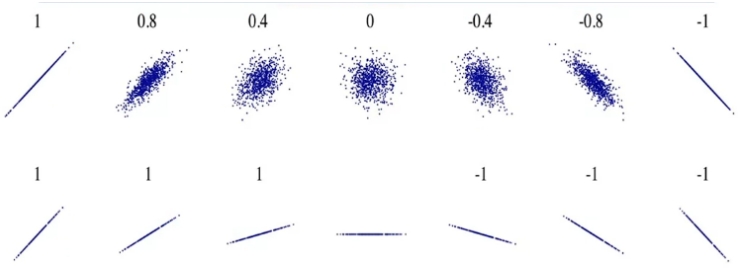
- Negative Linear Relationship
  - Correlation between two features (highway-mpg and price)
    - 

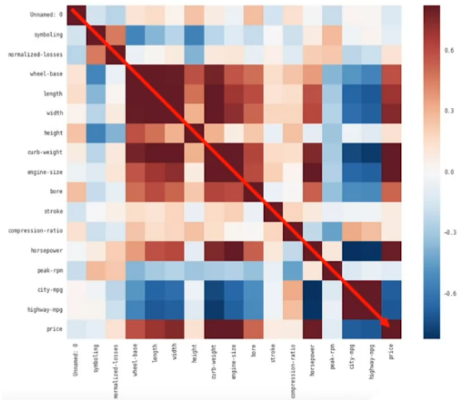
    - Although this relationship is negative the slope of the line is steep which means that the highway miles per gallon is still a good predictor of price.
  - Weak correlation between two features (peak-rpm and price)
    - 

    - For example, both low peak RPM and high values of peak RPM have low and high prices. Therefore, we cannot use RPM to predict the values

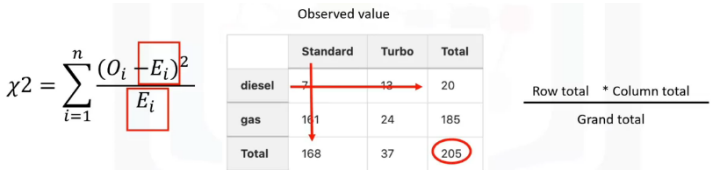
Correlation - Statistics

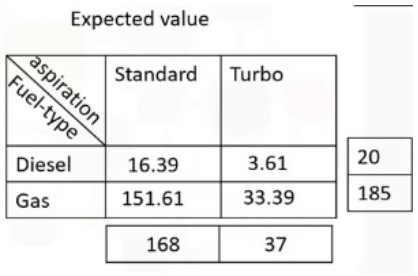
- Pearson Correlation

- Measure the strength of the correlation between two features
  - Correlation coefficient
  - P-value
- Correlation coefficient
  - Close to +1: Largo Positive relationship
  - Close to -1: Large Negative relationship
  - Close to 0: No relationship
- P-value
  - P-value < 0.001: Strong certainty in the result
  - P-value < 0.05: Moderate certainty in the result
  - P-value < 0.1: Weak certainty in the result
  - P-value > 0.1: No certainty in the result
- Strong Correlation
  - Correlation coefficient close to 1 or -1
  - P value less than 0.001
- 

- Example
  - pearson_coef, p_value = stats.pearsonr(df['horsepower'], df['price'])
    - Pearson correlation: 0.81
    - P-value: 9.35 e-48
    - Strong positive correlation
- Heatmap
  - 
  - This correlation heatmap gives us a good overview of how the different variables are related to one another and, most importantly, how these variables are related to price.

Association between two categorical variables: Chi-Square

- Categorical variables
    - We use the Chi-square Test for Association (denoted as x2)
    - The test is intended to test how likely it is that an observed distribution is due to chance
- Chi-Square Test for 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
- Categorical variables
    - Is there an association between fuel-type and aspiration?
        - In our case the crosstab or contingency table shows us the counts in each category
        - 

        - 

        - ```
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.

**Hands-on Lab: ANOVA**

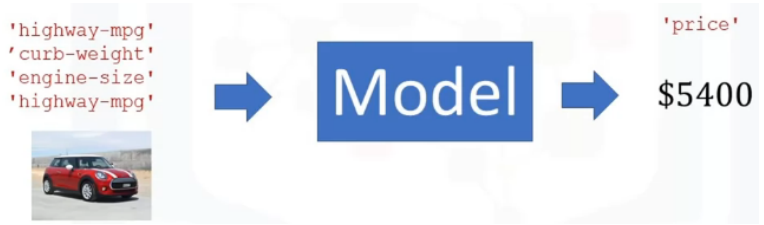
- ANOVA: Analysis of Variance
    - The Analysis of Variance (ANOVA) is a statistical method used to test whether there are significant differences between the means of two or more groups. ANOVA returns two parameters:
        - F-test score: ANOVA assumes the means of all groups are the same, calculates how much the actual means deviate from the assumption, and reports it as the F-test score. A larger score means there is a larger difference between the means.
```

        - P-value: P-value tells how statistically significant our calculated score value is.
    - If our price variable is strongly correlated with the variable we are analyzing, we expect ANOVA to return a sizable F-test score and a small p-value.

❖ **MODULE #4: Model Development**

Model Development

- A model can be thought of as a mathematical equation used to predict a value given one or more other values
- Relating one or more independent variables to dependent variables
  - 

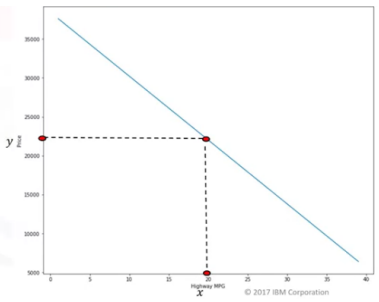
- The more relevant data you have the more accurate your model is
  - 

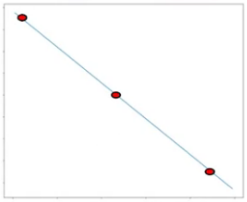
- To understand why more data is important consider the following situation:
  - You have two almost identical cars
  - Pink cars sell for significantly less
    - If your models' independent variables or features do not include color, your model will predict the same price for cars that may sell for much less.

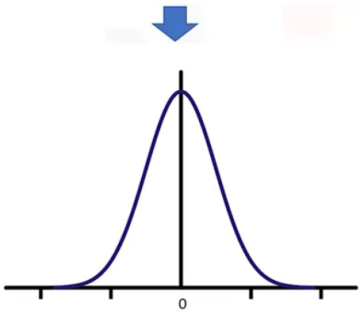
Linear Regression and Multiple Linear Regression

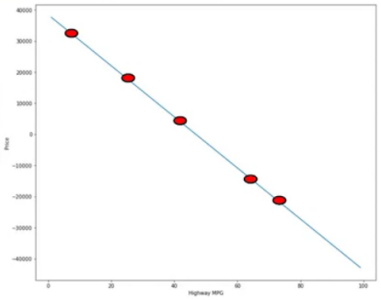
- Introduction
  - Simple Linear Regression will refer to one independent variable to make a prediction
  - Multiple Linear Regression will refer to multiple independent variables to make a prediction
- Simple Linear Regression
  - The predictor (independent) variable - x
  - The target (dependent) variable - y
    - $y = b_0 + b_1 x$ b_0: the intercept, b_1 : the slope
  - What kind of relationship would we like between the predictor variable x and the dependent variable y?
    - Linear
- Simple Linear Regression: Prediction

- $y = 38423 - 821x$
  $= 38423 - 821\ (20)$
  $= 22\ 003$



- Simple Linear Regression: Fit
  - 
    $X = \begin{bmatrix} 0 \\ 20 \\ 40 \end{bmatrix} \quad Y = \begin{bmatrix} 38423 \\ 22003 \\ 5583 \end{bmatrix}$
  - Noise typically refers to the random variation or error present in the relationship between the independent variable (predictor) and the dependent variable (response).
  - The figure on the left shows the distribution of the noise. The vertical axis shows the value added and the horizontal axis illustrates the probability that the value will be added.

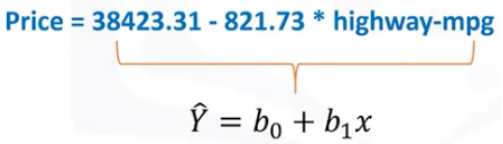




- Fitting a Simple Linear Model Estimator
  - X: Predictor variable
    Y: Target variable
  - Import linear_model from scikit-learn
    - from sklearn.linear_model import Linear Regression
  - Create a Linear Regression Object using the constructor
    - lm=LinearRegression()
- Fitting a Simple Linear Model
  - We define the predictor variable and target variable
    - X = df[[‘highway-mpg’]]
    - Y = df[‘price’]
  - Then use lm.fit(X,Y) to fit the model, i.e fine the parameters b0 and b1
    - lm.fit(X,Y)
  - We can obtain a predictor
    - Yhat=lm.predict(X)

-

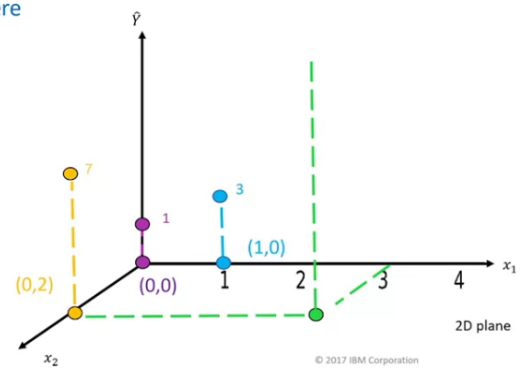
Yhat	X
2	5
:	
3	4

    - The output is an array. The array has the same number of samples as the input x.
- SLR - Estimated Linear Model
    - We can view the intercept (b0):
        - lm.intercept_
    - We can also view the slope (b1):
        - lm.coef_
    - The relationship between Price and Highway MPG is given by:
        - Price = (38423.31 - 821.73) * highway-mpg
        - 

- Multiple Linear Regression (MLR)
    - This method is used to explain the relationship between:
        - One continuous target (Y) variable
        - Two or more predictor (X) variables
    - $\hat{Y} = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + b_4 x_4$
        - b0: intercept (X=0)
        - b1, b2, bn: coefficient of parameters of x1, x2, xn
    - • This is shown below where

$\hat{Y} = 1 + 2x_1 + 3x_2$

n	x_1	x_2	$\hat{Y}$
1	0	0	1
2	0	2	7
3	1	0	3
4	3	2	13



- Fitting a Multiple Linear Model Estimator
    - We can extract the for 4 predictor variables and store them in the variable Z
        - Z = df[[‘horsepower’, ‘curb-weight’, ‘engine-size’, ‘highway-mpg’]]
    - Then train the model as before:

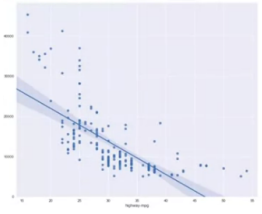
        - lm.fit(Z, df['price'])
    - We can also obtain a prediction
        - Yhat=lm.predict(X)
    - 

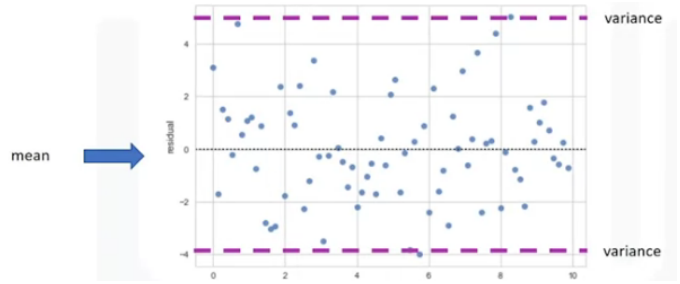
x_1	x_2	x_3	x_4
3	5	-4	3
:	:	:	:
2	4	2	-4

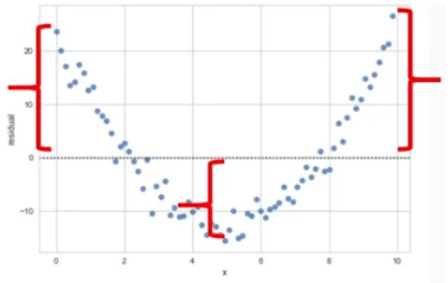
Yhat
2
:
3

        - The output is an array with the same number of elements as number of samples
- MLR – Estimated Linear Model
    - Find the intercept (b0)
        - lm.intercept_
        - ```
          lm.intercept_
          -15678.742628061467
          ```
    - Find the coefficients (b1,b2,b3,b4)
        - lm.coef_
        - ```
          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**
        - $\hat{Y} = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4$

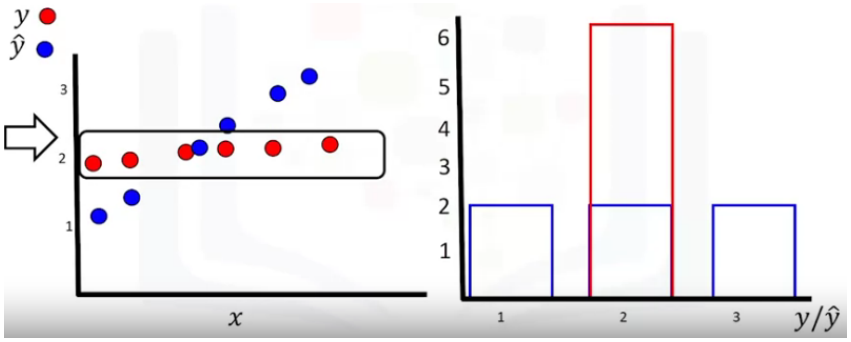
Model Evaluation using Visualization

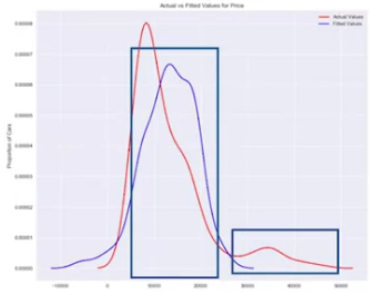
- Regression Plot
    - Why use a regression plot? It gives us a good estimate of:
        - The relationship between two variables
        - The strength of the correlation
        - The direction of the relationship (positive or negative)
    - Regression Plot shows us a combination of:
        - The scatterplot: where each point represents a different y
        - The fitted regression line (y^)
        - Horizontal axis is the independent variable, vertical axis the dependent variable
        - 

    - How to plot?
        - import seaborn as sns
        - sns.regplot(x='highway-mpg',y='price',data=df)
          plt.ylim(0,)
          plt.show()
- Residual Plot
    - Residual plots in Python are a graphical tool used to assess the goodness of fit of a regression model, particularly in linear regression. These plots are a crucial part of the model evaluation process and can help you identify potential issues with your regression model. Residuals are the differences between the observed values and the predicted values from your regression model.
    - 

    - Look at the spread of the residuals:
        - Randomly spread out around x-axis then a linear model is appropriate
    - 

        - In the final location, the error is large. The residuals are not randomly separated. This suggests the linear assumption is incorrect. This plot suggests a nonlinear function.
    - We can use seaborn to create a residual plot
        - 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.

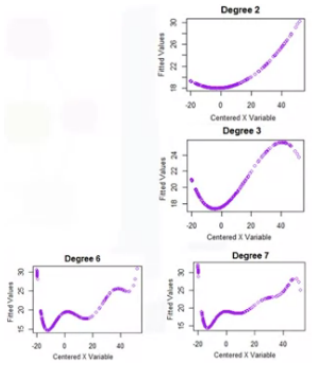
- 

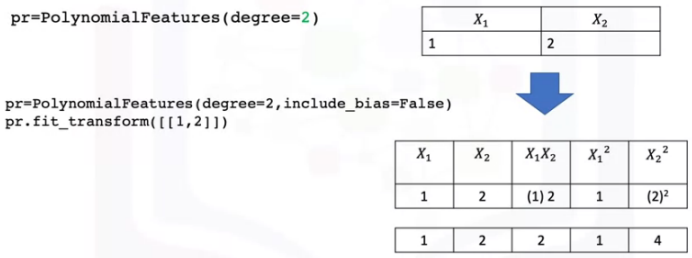
    - Dependent variable y^ (blue): count and plot the numbers that are approximately equal to one, two, three
    - Target value y (red): all target values are approximately equal to two
  - A histogram is for discrete values
  - Compare the distribution plots:
    - The dependent variable or feature is price
    - The fitted values that result from the model (blue)
    - The actual values (red)
    - 
    - The actual values are in red. We see the predicted values for prices in the range from 40,000 to 50,000 are inaccurate. The prices in the region from 10,000 to 20,000 are much closer to the target value.
  - 

    - Parameter → Price
    - Histogram → False (we want a distribution plot, not an histogram)

Polynomial Regression and Pipelines

- Polynomial Regressions
  - A special case of the general linear regression model
  - Used for describing curvilinear relationships

- Curvilinear relationships:
  - By squaring or setting higher-order terms of the predictor variables
- Quadratic
- Cubic
- Higher order
- 

- Polynomial Regression
  - Calculate Polynomial of 3rd order
    - f=np.polyfit(x,y,3)
    - p=np.polyld(f)
  - We can print out the model
    - print(p)
- Polynomial Regression with more than one dimension
  - $\hat{Y} = b0 + b1\,X_1 + b2\,X_2 + b3\,X_1\,X_2 + b4(X_1)^2 + b5(X_2)^2 + ..$
  - The “preprocessing” library in scikit-learn
    - from sklearn.preprocessing import PolynomialFeatures
      pr=PolynomialFeatures(degree=2, include_bias=False)
  - How would you create a second order polynomial transform object pr:
    - pr=PolynomialFeatures(degree=2)
  - Applying the method we transform the data, we now have a new set of features that are a transformed version of our original features.
    - 

- Pre-processing
  - In scikit-learn, preprocessing refers to a set of techniques and tools used to prepare your data for machine learning. Data preprocessing is a crucial step in the machine learning pipeline because the quality and format of your data can significantly impact the performance and results of your machine learning models. Scikit-learn provides various preprocessing methods and classes to help you clean, transform, and

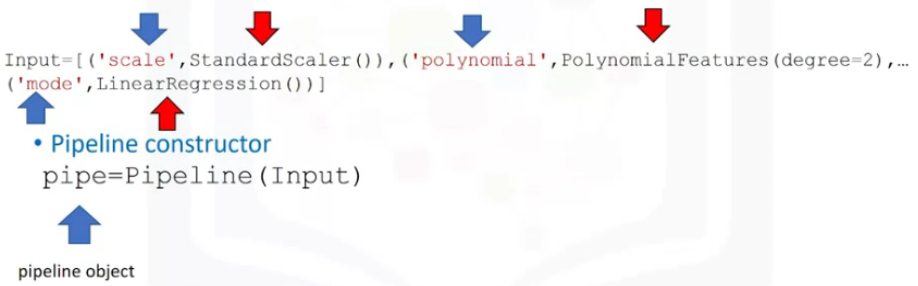
preprocess your data.

- For example we can Normalize the each feature simultaneously

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

- Pipelines
  - There are many steps to getting a prediction
    - Normalization (transformation p.1)
    - Polynomial transform (transformation p.2)
    - Linear regression (prediction)

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



    - We create a list of tuples, the first element in the tuple contains the name of the estimator model. The second element contains a model constructor. We input the list in the pipeline constructor. We now have a pipeline object

```
Pipe.fit(df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']],y)
yhat=Pipe.predict(X[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']])
```

    - We can train the pipeline by applying the train method to the pipeline object. We can also produce a prediction as well. The method normalizes the data, performs a polynomial transform, then outputs a prediction.

Measures for In-Sample Evaluation

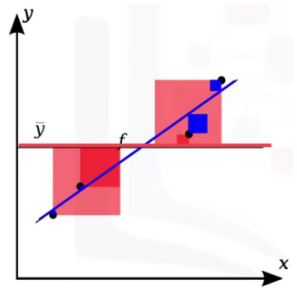
- A way to numerically determine how good the model fits on dataset
- Two important measures to determine the fit of a model:

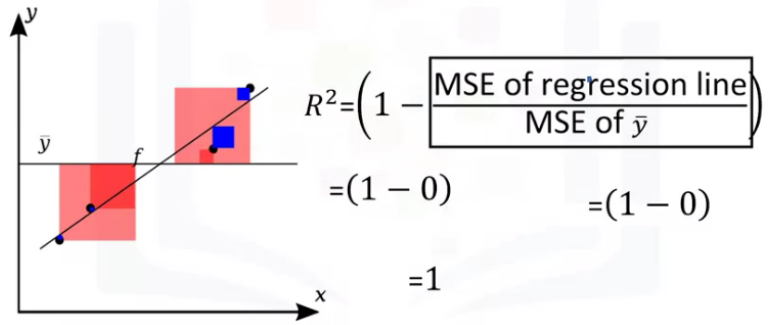
    - Mean Squared Error (MSE)
    - R-squared (R^2)
- Mean Squared Error (MSE)
    - For example for sample 1:
        - Actual value is 150, and the predicted value is 50. If we subtract, we get 100
        - We then square the number. We then take the Mean or average of all the errors by adding them all together and dividing by the number of samples.
        - In python we can measure the MSE as follows

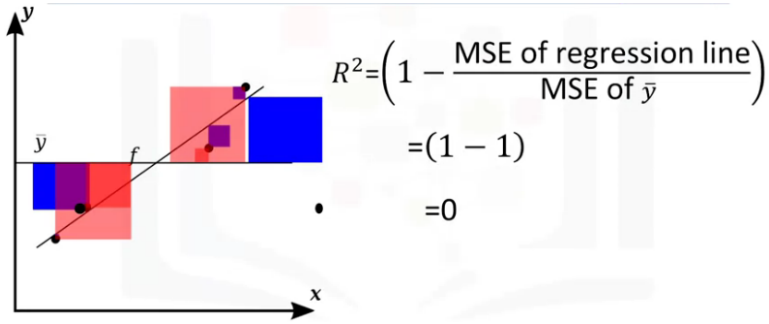
```
from  sklearn.metrics import mean_squared_error

mean_squared_error(df['price'],Y_predict_simple_fit)


3163502.944639888
```

    - How would you calculate the mean squared error between your predicted values Yhat and actual values y?
        - from sklearn.metrics import mean_squared_error mean_squared_error(Y,Yhat)
- R-squared (R^2)
    - The Coefficient of Determination or R squared R^2
    - It's 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
    - $$R^2 = \left(1 - \frac{\text{MSE of regression line}}{\text{MSE of the average of the data}}\right)$$
- Coefficient of Determination (R^2)
    - The blue line represents the regression line
    - The blue squares represents the MSE of the regression line
    - The red line represents the average value of the data points
    - The red squares represent the MSE of the red line
    - We see the area of the blue squares is much smaller than the area of the red squares
    - 

- 

  - Performs well
- 

  - Doesn't perform well
- R-squared R^2
  - Generally the values of the MSE are between 0 and 1
  - We can calculate the E^2 as follows
    ```
    X = df[['highway-mpg']]
    Y = df['price']

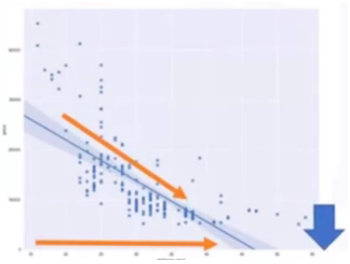
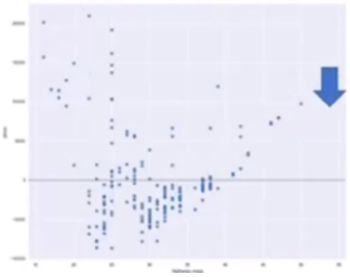
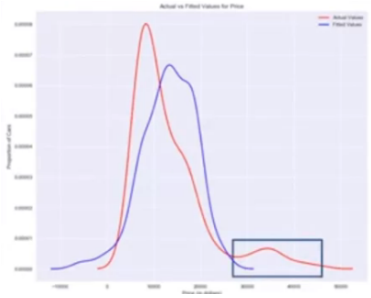
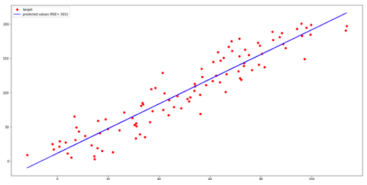
    lm.fit(X, Y)

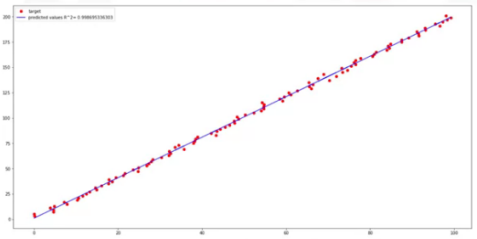
    lm.score(X,y)
    0.496591188
    ```

Prediction and Decision Making

- Decision Making: Determining a Good Model Fit
  - To determine final best fit, we look at a combination of:
    - Do the predicted values make sense
    - Visualization
    - Numerical measures for evaluation
    - Comparing Models
- Do the predicted values make sense
  - First we train the model
    - lm.fit(df[‘highway-mpg’,df[‘prices’])
  - Let's predict the price of a car with 30 highway-mpg
    - lm.predict(np.array(30.0).reshape(-1,1))
  - Result: $13771.30
    - lm.coef_

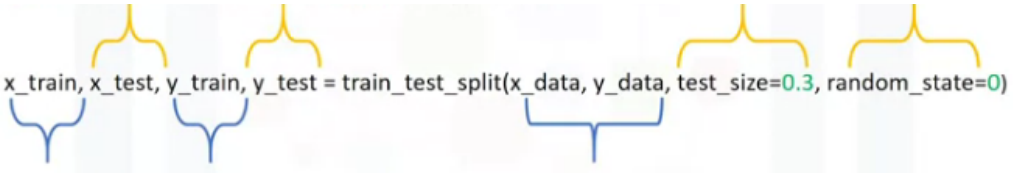
    - Price = 38423.31 - 821.73 * highway-mpg
- Do the predicted values make sense
    - First we import numpy
        - import numpy as np
    - We use the numpy function arrange to generate a sequence from 1 to 100
        - new_input = np.arrange(1,101,1).reshape(-1,1)
        - 

    - We can predict new values
        - yhat=lm.predict(new_input)
- Visualization
    - Simply visualizing your data with a regression
        - 
        - Non-linear behavior
- Residual Plot
    - 
- Distribution Plot
    - A distribution plot is a good method for multiple linear regression.
        - 
- Numerical measures for Evaluation
    - Mean square error
        - 

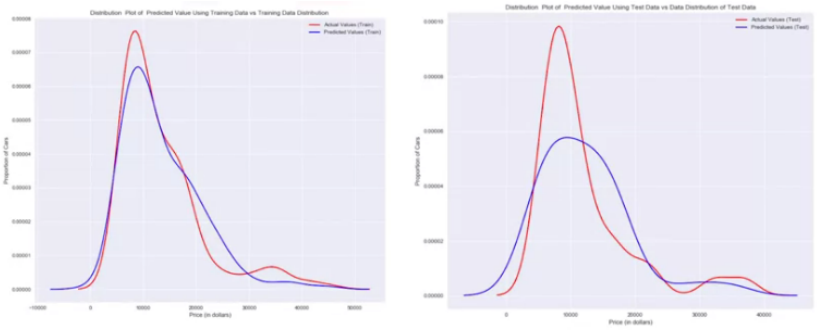
    - R squared
        - It tells you how well your line fits into the model
        - An r squared of 0.9986 the model appears to be a good fit.
        - 
- Comparing MLR and SLR
    - Does a lower Mean Square Error imply better fit?
        - Not necessarily
    - Mean Square Error for a Multiple Linear Regression Model will be smaller than the Mean Square Error for a Simple Linear Regression model, since the errors of the data will decrease when more variables are included in the model
    - Polynomial regression will also have a smaller Mean Square Error than the linear regular regression

❖ **MODULE #5: Model Evaluation**

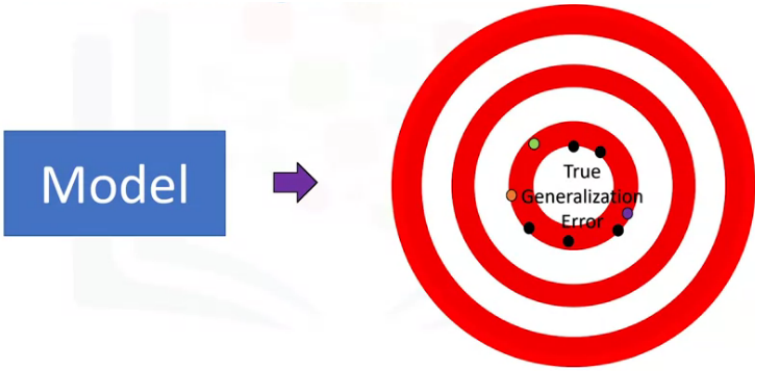
Model Evaluation and Refinement

- Model Evaluation
  - In-sample evaluation tells us how well our model will fit the data used to train it
  - Problem?
    - It does not tell us how well the trained model can be used to predict new data
  - Solution?
    - In-sample data or training data
    - Out-of-sample evaluation or test set
- Training/Testing Sets
  - When we split a dataset, usually the larger portion of data is used for training and a smaller part is used for testing
  - Split dataset into:
    - Training set (70%)
    - Training 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 train_test_split()
  - Split data into random train and test subsets
  - 

    - x_data: features or independent variables
    - y_data: dataset target df['price']
    - The output is an array. x_train, y_train: parts of available data as training set
    - x_test, y_test: parts of available data as testing set
    - test_size: percentage of the data for testing (here 30%)
    - random_state: number generator used for random sampling
- Generalization Performance
  - Generalization error is measure of how well our data does at predicting previously unseen data
  - The error we obtain using our testing data is an approximation of this error

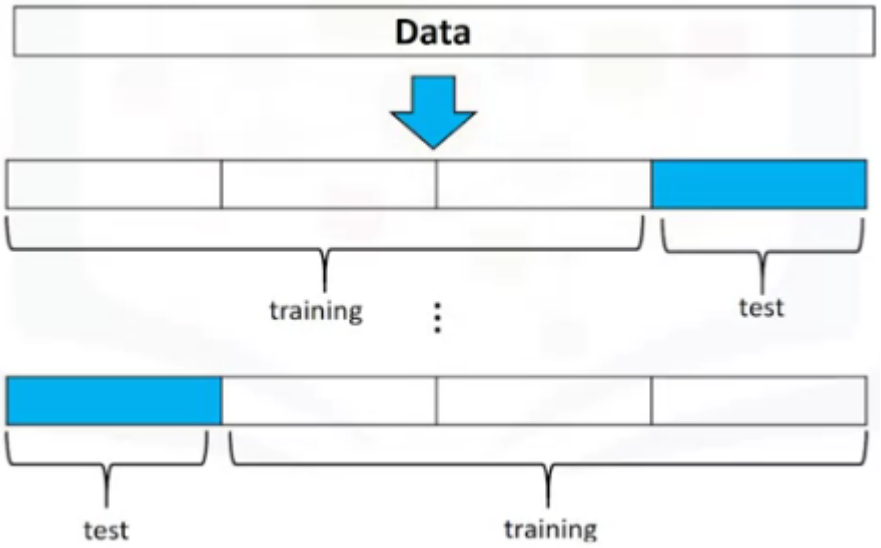


- We see the distributions are somewhat similar. If we generate the same plot using the test data, we see the distributions are relatively different. The difference is due to a generalization error and represents what we see in the real world.



- Using a lot of data for training, gives us an accurate means of determining how well our model will perform in the real world. But the precision of the performance will be low. Let's clarify this with an example. The center of this bull's eye represents the correct generalization error. Let's say we take a random sample of the data using 90 percent of the data for training and 10 percent for testing. The first time we experiment, we get a good estimate of the training data. If we experiment again training the model with a different combination of samples, we also get a good result. But, the results will be different relative to the first time we run the experiment. Repeating the experiment again with a different combination of training and testing samples, the results are relatively close to the generalization error, but distinct from each other. Repeating the process, we get a good approximation of the generalization error, but the precision is poor i.e. all the results are extremely different from one another.

- 

    - If we use fewer data points to train the model and more to test the model, the accuracy of the generalization performance will be less, but the model will have good precision. The figure above demonstrates this. All our error estimates are relatively close together, but they are further away from the true generalization performance. To overcome this problem, we use cross-validation. One of the most common out of sample evaluation metrics is cross-validation.
- Cross Validation
    - Most common out-of-sample evaluation metrics
    - More effective use of data (each observation is used for both training and testing)
    - 

    - 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.
- Function cross_val_score()
    - It's the simplest way to apply cross validation, which performs multiple out-of-sample evaluations. Imported from sklearn.

```
from sklearn.model_selection import cross_val_score

scores= cross_val_score(lr, x_data, y_data, cv=3)

np.mean(scores)
```

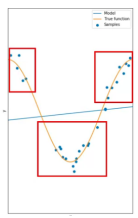
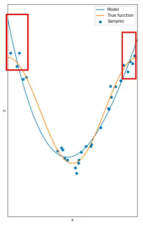
    - Type of model (lr)
    - x_data is the predictor variable data
    - y_data is the target variable data
    - cv=3 data is split into three equal partitions
    - The function returns an array of scores, one for each partition that was chosen as the testing set.
    - We can average the result together to estimate out of sample r squared using the mean function NumPy
- 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

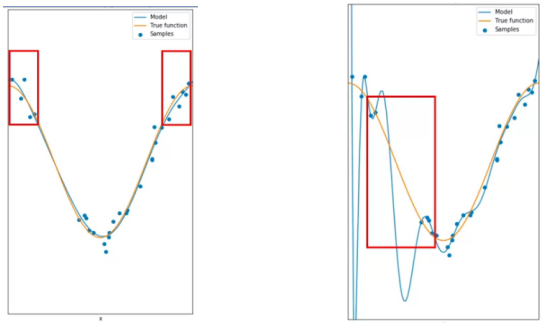
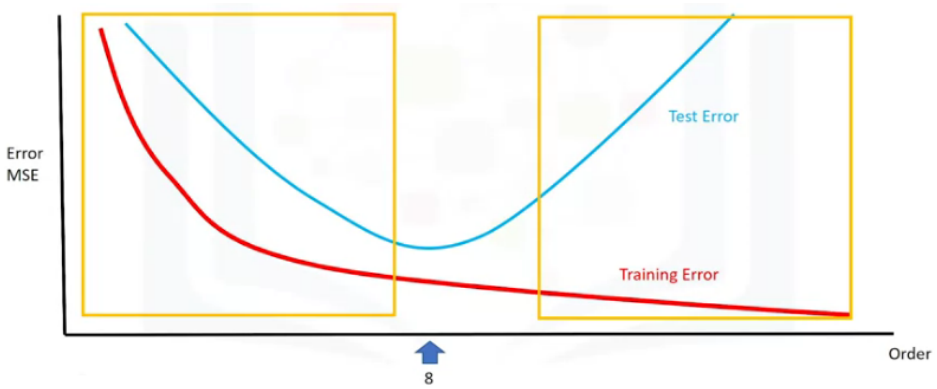
yhat= cross_val_predict (lr2e, x_data, y_data, cv=3)
```

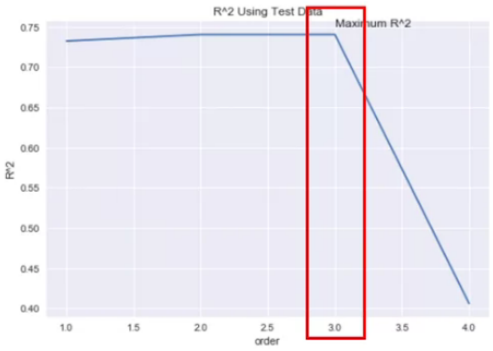
- What function randomly splits a dataset into training and testing subsets?
    - train_test_split

Overfitting, Underfitting and Model Selection

- Model Selection
    - Underfitting: the model is too simple to fit the data, there are many errors. If we increase the order of the polynomial, the model fits better, but the model is still not flexible enough and exhibits underfitting.
        - $y = b_0 + b_1 x$
        - 
        - $y = b_0 + b_1 x + b_2 x^2$
        - 
        - (linear function) – (quadratic function)
    - Overfitting: the model is too flexible and fits the noise rather than the function
        - Explained: If we increase the order of the polynomial, the model fits better, but the model is still not flexible enough and exhibits

underfitting. The model does extremely well at tracking the training point but performs poorly at estimating the function. This is especially apparent where there is little training data. The estimated function oscillates without tracking the function.

- 
  - (8th order polynomial) – (16th order polynomial)
- Error MSE – Order
  - 

  - Izquierda underfitting y derecha overfitting
- R^2
  - 

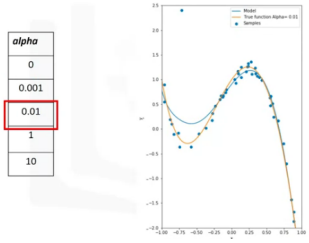
  - The R-squared is optimal when the order of the polynomial is three

```
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))
```

- First, we create an empty list to store the values. We create a list containing different polynomial orders. We then iterate through the list using a loop. We create a polynomial feature object with the order of the polynomial as a parameter. We transform the training and test data into a polynomial using the fit transform method. We fit the regression model using the transform data. We then calculate the R-squared using the test data and store it in the array.

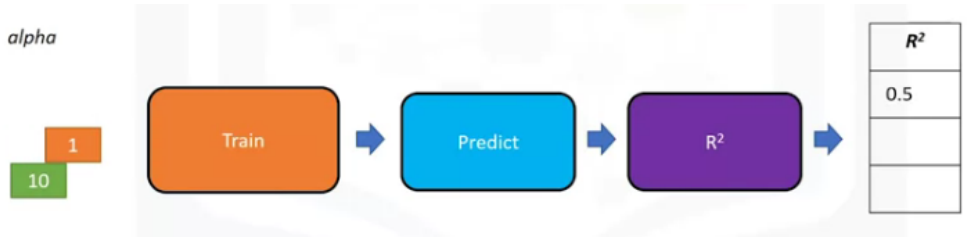
Ridge Regression

- 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.
- Ridge regression
  - Controls the magnitude of these polynomial coefficients by introducing the parameter alpha. Alpha is a parameter we select before fitting or training the model
  - Each row in the following table represents an increasing value of alpha

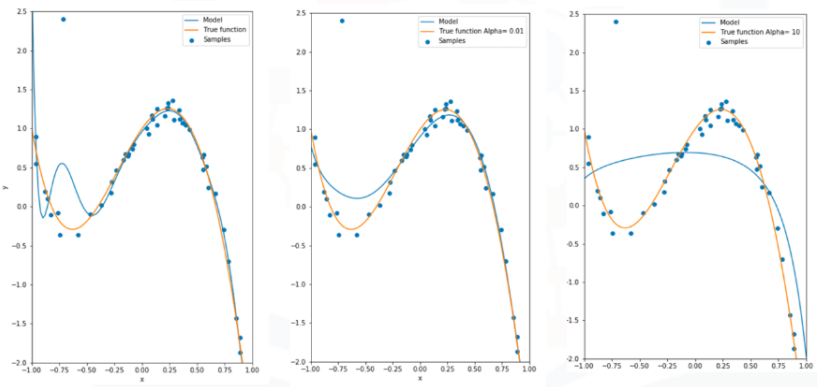



```
from sklearn.linear_model import Ridge
RidgeModel=Ridge(alpha=0.1)
RidgeModel.fit(X,y)
Yhat=RidgeModel.predict(X)
```

        - Import ridge from sklearn.linear_models. Create a ridge object using the constructor. The parameter alpha is one of the arguments of the constructor. We train the model using the fit method. To make a prediction, we use the predict method.



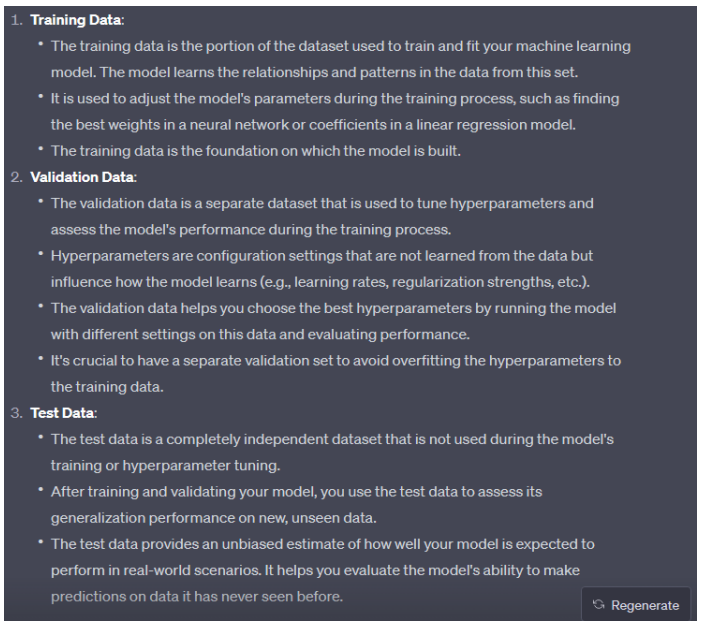
    - Validation data: We start with a small value of alpha. We train the model, make a prediction using the validation data, then calculate the R-squared and store the values. Repeat the value for a larger value of alpha.

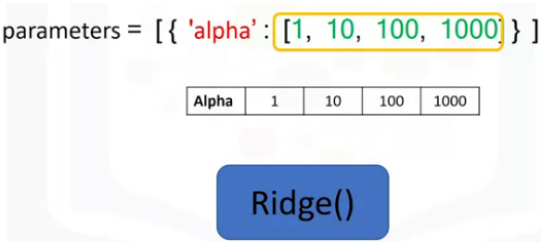


    - Lowest value in alpha is a)

Grid Search

- Grid Search allows us to scan through multiple free parameters with few lines of code.
- Hyperparameters
    - In the last section, 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
    - What data do we use to pick the best hyperparameter?
        - Validation data

- 
1. **Training Data**:
   - The training data is the portion of the dataset used to train and fit your machine learning model. The model learns the relationships and patterns in the data from this set.
   - It is used to adjust the model's parameters during the training process, such as finding the best weights in a neural network or coefficients in a linear regression model.
   - The training data is the foundation on which the model is built.
2. **Validation Data**:
   - The validation data is a separate dataset that is used to tune hyperparameters and assess the model's performance during the training process.
   - Hyperparameters are configuration settings that are not learned from the data but influence how the model learns (e.g., learning rates, regularization strengths, etc.).
   - The validation data helps you choose the best hyperparameters by running the model with different settings on this data and evaluating performance.
   - It's crucial to have a separate validation set to avoid overfitting the hyperparameters to the training data.
3. **Test Data**:
   - The test data is a completely independent dataset that is not used during the model's training or hyperparameter tuning.
   - After training and validating your model, you use the test data to assess its generalization performance on new, unseen data.
   - The test data provides an unbiased estimate of how well your model is expected to perform in real-world scenarios. It helps you evaluate the model's ability to make predictions on data it has never seen before.

- Grid Search
  - Python list that contains a Python dictionary
  - Object or model Ridge()
    - 

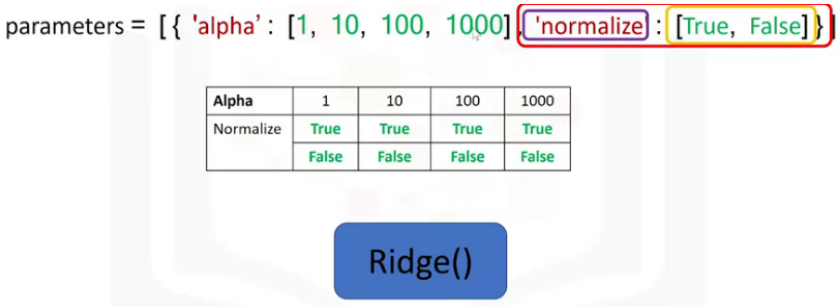
    -
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

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

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']
```
  - Grid Search has the option to normalize the data
    - 

    -

```
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)

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  inzip(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
```

LESSON SUMMARY

- **Identify over-fitting and under-fitting in a predictive model:** Overfitting occurs when a function is too closely fit to the training data points and captures the noise of the data. Underfitting refers to a model that can't model the training data or capture the trend of the data.
- **Apply Ridge Regression to linear regression models:** Ridge regression is a regression that is employed in a Multiple regression model when Multicollinearity occurs.
- **Tune hyper-parameters of an estimator using Grid search:** Grid search is a time-efficient tuning technique that exhaustively computes the optimum values of hyperparameters performed on specific parameter values of estimators.

❖ **MODULE #6: Final Assignment**