In this module, you will learn how to understand data and learn about how to use the libraries in Python to help you import data from multiple sources. You will then learn how to perform some basic tasks to start exploring and analyzing the imported data set.

**Learning Objectives**

* Analyze Python data using a dataset
* Identify three Python libraries and describe their uses
* Read data using Python's Pandas package
* Demonstrate how to import and export data in Python

# The Problem

0:00

In this video, we'll be talking about data analysis and

the scenario in which we'll be playing the data analyst or data scientist.

Play video starting at ::8 and follow transcript0:08

But before we begin, talking about the problem, used car prices,

we should first understand the importance of data analysis.

As you know, data is collected everywhere around us.

Whether it's collected manually by scientists or collected digitally,

every time you click on a website, or your mobile device.

But data does not mean information.

Data analysis and, in essence,data science, helps

us unlock the information and insights from raw data to answer our questions.

So data analysis plays an important role by helping us to discover useful

information from the data, answer questions, and even predict the future or

the unknown.

Play video starting at ::47 and follow transcript0:47

So let's begin with our scenario.

Let's say we have a friend named Tom.

And Tom wants to sell his car.

But the problem is he doesn't know how much he should sell his car for.

Tom wants to sell his car for as much as he can.

But he also wants to set the price reasonably, so

someone would want to purchase it.

So the price he sets should represent the value of the car.

Play video starting at :1:10 and follow transcript1:10

How can we help Tom determine the best price for his car?

Let's think like data scientists and clearly define some of his problems.

For example, is there data on the prices of other cars and their characteristics?

What features of cars affect their prices?

Color?

Brand?

Does horsepower also effect the selling price, or perhaps something else?

As a data analyst or

data scientist, these are some of the questions we can start thinking about.

To answer these questions, we're going to need some data.

In the next videos, we'll be going into how to understand the data, how to import

it into Python, and how to begin looking into some basic insights from the data.

[music]

# Understanding the Data

0:00

In this video, we'll be looking at the dataset on used car prices.

The dataset used in this course is an open dataset by Jeffrey C. Schlemmer.

This dataset is in CSV format,

which separates each of the values with commas,

making it very easy to import in most tools or applications.

Each line represents a row in the dataset.

In the hands-on lab for this module,

you'll be able to download and use the CSV file.

Do you notice anything different about the first row?

Sometimes the first row is a header,

which contains a column name for each of the 26 columns.

But in this example, it's just another row of data.

So, here's the documentation on what each of the 26 columns represent.

There are a lot of columns and I'll just go through a few of the column names,

but you can also check out the link at the bottom of

the slide to go through the descriptions yourself.

The first attribute, symboling,

corresponds to the insurance risk level of a car.

Cars are initially assigned a risk factor symbol associated with their price.

Then, if an automobile is more risky,

this symbol is adjusted by moving it up the scale.

A value of plus three indicates that the auto is risky.

Minus three, that is probably pretty safe.

The second attribute, normalized-losses,

is the relative average loss payment per insured vehicle year.

This value is normalized for all autos within

a particular size classification, two door small,

station wagons, sports specialty,

etc., and represents the average loss per car per year.

The values range from 65 to 256.

The other attributes are easy to understand.

If you would like to check out more details,

refer to the link at the bottom of the slide.

Okay, after we understand the meaning of each feature,

we'll notice that the 26 attribute is price.

This is our target value or label in other words.

This means price is the value that we want to predict from the dataset and

the predictors should be all the other variables listed like symboling,

normalized-losses, make, and so on.

Thus, the goal of this project is to predict price in

terms of other car features. Just a quick note.

This dataset is actually from 1985.

So, the car prices for the models may seem a little low.

But just bear in mind that the goal of this exercise is to learn how to analyze the data.

[music]

# Python Packages for Data Science

0:00

In order to do data analysis in Python, we should first tell you a little bit about

the main packages relevant to analysis in Python.

Play video starting at ::8 and follow transcript0:08

A Python library is a collection of functions and

methods that allow you to perform lots of actions without writing any code.

The libraries usually contain built in modules providing different

functionalities which you can use directly.

And there are extensive libraries offering a broad range of facilities.

We have divided the Python data analysis libraries into three groups.

The first group is called scientific computing libraries.

Pandas offers data structure and tools for effective data manipulation and analysis.

It provides facts, access to structured data.

The primary instrument of Pandas is the two dimensional table consisting of column

and row labels, which are called a data frame.

It is designed to provid easy indexing functionality.

The NumPy library uses arrays for its inputs and outputs.

It can be extended to objects for matrices and

with minor coding changes, developers can perform fast array processing.

SciPy includes functions for

some advanced math problems as listed on this slide, as well as data visualization.

Play video starting at :1:14 and follow transcript1:14

Using data visualization methods is the best way to communicate with others,

showing them meaningful results of analysis.

These libraries enable you to create graphs, charts and maps.

The Matplotlib package is the most well known library for data visualization.

It is great for making graphs and plots.

The graphs are also highly customizable.

Another high level visualization library is Seaborn.

It is based on Matplotlib.

It's very easy to generate various plots such as heat maps,

time series and violin plots.

Play video starting at :1:49 and follow transcript1:49

With machine learning algorithms,

we're able to develop a model using our data set and obtain predictions.

The algorithmic libraries tackles the machine learning tasks from basic

to complex.

Here we introduce two packages,

the Scikit-learn library contains tools statistical modeling,

including regression, classification, clustering, and so on.

This library is built on NumPy, SciPy and Matplotib.

Statsmodels is also a Python module that allows users to explore data,

estimate statistical models and perform statistical tests.

[music]

# Importing and Exporting Data in Python

0:00

In this video, we'll look at how to read

any data using python's pandas package.

Once we have our data in Python,

then we can perform all the subsequent data analysis procedures we need.

Data acquisition is a process of loading

and reading data into notebook from various sources.

To read any data using Python's pandas package,

there are two important factors to consider,

format and file path.

Format is the way data is encoded.

We can usually tell different encoding schemes

by looking at the ending of the file name.

Some common encodings are: CSV,

JSON, XLSX, HDF and so forth.

The path tells us where the data is stored.

Usually, it is stored either on the computer

we are using or online on the internet.

In our case, we found a dataset of used cars

which was obtained from the web address shown on the slide.

When Jerry entered the web address in his web browser,

he saw something like this.

Each row is one datapoint.

A large number of properties are associated with each datapoint.

Because the properties are separated from each other by commas,

we can guess the data format is CSV,

which stands for comma separated values.

At this point, these are just numbers and don't mean much to humans,

but once we read in this data we can try to make more sense out of it.

In pandas, the read\_CSV method can read in files with

columns separated by commas into a pandas data frame.

Reading data in pandas can be done quickly in three lines.

First, import pandas, then define a variable with

a file path and then use the read\_ CSV method to import the data.

However, read\_CSV assumes the data contains a header.

Our data on used cars has no column headers.

So, we need to specify read\_CSV to

not assign headers by setting header to none.

After reading the dataset,

it is a good idea to look at the data frame to get

a better intuition and to ensure that

everything occurred the way you expected.

Since printing the entire dataset may take up

too much time and resources to save time,

we can just use dataframe.head to show the first n rows of the data frame.

Similarly, dataframe.tail shows the bottom end rows of data frame.

Here, we printed out the first five rows of data.

It seems that the dataset was read successfully.

We can see that pandas automatically set the column header as a list of

integers because we set header equals none when we read the data.

It is difficult to work with the data frame

without having meaningful column names.

However, we can assign column names in pandas.

In our present case,

it turned out that we have the column names in a separate file online.

We first put the column names in a list called headers,

then we set df.columns equals

headers to replace the default integer headers by the list.

If we use the head method

introduced in the last slide to check the dataset,

we see the correct headers inserted at the top of each column.

At some point in time,

after you've done operations on your dataframe you may

want to export your pandas dataframe to a new CSV file.

You can do this using the method to\_CSV.

To do this, specify the file path

which includes the file name that you want to write to.

For example, if you would like to save dataframe

df as automobile.CSV to your own computer,

you can use the syntax df.to\_CSV.

Play video starting at :3:45 and follow transcript3:45

For this course, we will only read and save CSV files.

However, pandas also supports importing and

exporting of most data file types with different dataset formats.

The code syntax for reading and saving other data formats

is very similar to read or save CSV file.

Each column shows a different method to read and

save files into a different format.

[music]

# Getting Started Analyzing Data in Python

0:00

In this video, we introduce some simple Pandas methods that

all data scientists and analysts should know

when using Python, Pandas and data.

At this point, we assume that the data has been loaded.

It's time for us to explore the dataset.

Pandas has several built-in methods that can be used to understand

the datatype or features or to

look at the distribution of data within the dataset.

Using these methods, gives an overview of the dataset and also point out

potential issues such as the wrong data type of

features which may need to be resolved later on.

Data has a variety of types.

The main types stored in Pandas' objects are object,

float, Int, and datetime.

The data type names are somewhat different from those in native Python.

This table shows the differences and similarities between them.

Some are very similar such as the numeric data types, int and float.

The object pandas type function's similar to string in Python,

save for the change in name.

While the datetime Pandas type,

is a very useful type for handling time series data.

There are two reasons to check data types in a dataset.

Pandas automatically assigns types based

on the encoding it detects from the original data table.

For a number of reasons,

this assignment may be incorrect.

For example, it should be awkward if

the car price column which we should

expect to contain continuous numeric numbers,

is assigned the data type of object.

It would be more natural for it to have the float type.

Jerry may need to manually change the data type to float.

The second reason, is that allows an experienced data scientists

to see which Python functions can be applied to a specific column.

For example, some math functions can only be applied to numerical data.

If these functions are applied to non-numerical data an error may result.

When the dtype method is applied to the data set,

the data type of each column is returned in a series.

A good data scientists intuition

tells us that most of the data types make sense.

They make of cars for example are names.

So, this information should be of type object.

The last one on the list could be an issue.

As bore is a dimension of an engine,

we should expect a numerical data type to be used.

Instead, the object type is used.

In later sections, Jerry will have to correct these type mismatches.

Now, we would like to check the statistical summary of

each column to learn about the distribution of data in each column.

The statistical metrics can tell the data scientist if there are

mathematical issues that may exist such

as extreme outliers and large deviations.

The data scientists may have to address these issues later.

To get the quick statistics,

we use the describe method.

It returns the number of terms in the column as count,

average column value as mean,

column standard deviation as std,

the maximum minimum values,

as well as the boundary of each of the quartiles.

By default, the dataframe.describe functions

skips rows and columns that do not contain numbers.

It is possible to make the describe method

worked for object type columns as well.

To enable a summary of all the columns,

we could add an argument.

Include equals all inside the describe function bracket.

Now, the outcome shows the summary of all the 26 columns,

including object typed attributes.

We see that for the object type columns,

a different set of statistics is evaluated,

like unique, top, and frequency.

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.

Another method you can use to check your dataset,

is the dataframe.info function.

This function shows the top 30 rows and bottom 30 rows of the data frame.

[music]

# Accessing Databases with Python

0:00

Hello, in this video you will learn how to access databases using Python.

Databases are powerful tools for data scientists.

After completing this module, you'll be able to explain the basic concepts related to using

Python to connect to databases.

This is how a typical user accesses databases using Python code written on a Jupyter notebook,

a web based editor.

There is a mechanism by which the Python program communicates with the DBMS.

The Python code connects to the database using API calls.

We will explain the basics of SQL APIs and Python DB APIs.

An application programming interface is a set of functions that you can call to get

access to some type of service.

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.

DB-API is Python's standard API for accessing relational databases.

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.

The two main concepts in the Python DB-API are connection objects and query objects.

You use connection objects to connect to a database and manage your transactions.

Cursor objects are used to run queries.

You open a cursor object and then run queries.

The cursor works similar to a cursor in a text processing system where you scroll down

in your result set and get your data into the application.

Cursors are used to scan through the results of a database.

Here are the methods used with connection objects.

The cursor() method returns a new cursor object using the connection.

The commit() method is used to commit any pending transaction to the database.

The rollback() method causes the database to roll back to the start of any pending transaction.

The close() method is used to close a database connection.

Let's walk through a Python application that uses the DB-API to query a database.

First, you import your database module by using the connect API from that module.

To open a connection to the database, you use the connection function and pass in the

parameters that is, the database name, username, and password.

The connect function returns connection object.

After this, you create a cursor object on the connection object.

The cursor is used to run queries and fetch results.

After running the queries using the cursor, we also use the cursor to fetch the results

of the query.

Finally, when the system is done running the queries, it frees all resources by closing

the connection.

Remember that it is always important to close connections to avoid unused connections taking

up resources.Thanks for watching this video.

[music]

# Lesson Summary

In this lesson, you have learned how to:

**Define the Business Problem:** Look at the data and make some high-level decision on what kind of analysis should be done

**Import and Export Data in Python:** How to import data from multiple data sources using the Pandas library and how to export files into different formats.

**Analyze Data in Python:** How to do some introductory analysis in Python using functions like **dataframe.head()** to view the first few lines of the dataset, **dataframe.info()** to view the column names and data types.

In this module, you will learn how to perform some fundamental data wrangling tasks that, together, form the pre-processing phase of data analysis. These tasks include handling missing values in data, formatting data to standardize it and make it consistent, normalizing data, grouping data values into bins, and converting categorical variables into numerical quantitative variables.

**Learning Objectives**

* Describe how to handle missing values
* Describe data formatting techniques
* Describe data normalization
* Demonstrate the use of binning
* Demonstrate the use of categotical variables

# Pre-processing Data in Python

0:00

In this video, we'll be going through

some data preprocessing techniques.

If you're unfamiliar with the term,

data preprocessing is a necessary step in data analysis.

It is the process of converting or mapping data from

one raw form into

another format to make it ready for further analysis.

Data preprocessing is often called

data cleaning or data wrangling,

and there are likely other terms.

Here are the topics that we'll be

covering in this module.

First, we'll show you how to

identify and handle missing values.

A missing value condition occurs

whenever a data entry is left empty.

Then we'll cover data formats.

Data from different sources maybe in various formats,

in different units, or in various conventions.

We will introduce some methods in

Python Pandas that can

standardize the values into the same format,

or unit, or convention.

After that, we'll cover data normalization.

Different columns of numerical data may have

very different ranges and

direct comparison is often not meaningful.

Normalization is a way to bring all data into

a similar range for more useful comparison.

Specifically, we'll focus on

the techniques of centering and scaling.

Then, we'll introduce data binning.

Binning creates bigger categories

from a set of numerical values.

It is particularly useful for

comparison between groups of data.

Lastly, we'll talk about

categorical variables and show you how to convert

categorical values into numeric variables

to make statistical modeling easier.

In Python, we usually perform operations along columns.

Each row of the column represents a sample,

I.e, a different used car in the database.

You access a column by specifying the name of the column.

For example, you can access symbolling and body style.

Each of these columns is a Panda series.

There are many ways to manipulate Dataframes in Python.

For example, you can add

a value to each entry off a column.

To add one to each symbolling entry, use this command.

This changes each value of

the Data frame column by adding one to the current value.

# Dealing with Missing Values in Python

Notes

[**Discuss**](https://www.coursera.org/learn/data-analysis-with-python/discussions/weeks/2)

0:00

In this video, we will introduce the pervasive problem of

missing values as well as strategies

on what to do when you encounter missing values in your data.

When no data value is stored for feature for a particular observation,

we say this feature has a missing value.

Usually missing value in data set appears as

question mark and a zero or just a blank cell.

In the example here,

the normalized losses feature has a missing value which is represented with NaN.

But how can you deal with missing data?

There are many ways to deal with missing values and this is regardless of Python,

R or whatever tool you use.

Of course, each situation is different and should be judged differently.

However, these are the typical options you can consider.

The first is to check if the person or group that collected

the data can go back and find what the actual value should be.

Another possibility is just to remove the data where that missing value is found.

When you drop data, you could either drop the whole

variable or just the single data entry with the missing value.

If you don't have a lot of observations with missing data,

usually dropping the particular entry is the best.

If you're removing data, you want to look

to do something that has the least amount of impact.

Replacing data is better since no data is wasted.

However, it is less accurate since we need to

replace missing data with a guess of what the data should be.

One standard for placement technique is to replace

missing values by the average value of the entire variable.

As an example, suppose we have some entries that have missing values for

the normalized losses column and the column average for entries with data is 4500.

While there is no way for us to get an accurate guess of what

the missing value is under the normalized losses column should have been,

you can approximate their values using the average value of the column 4500.

But what if the values cannot be averaged as with categorical variables?

For a variable like fuel type,

there isn't an average fuel type since the variable values are not numbers.

In this case, one possibility is to try using the mode,

the most common like gasoline.

Finally, sometimes we may find another way to guess the missing data.

This is usually because the data gathered

knows something additional about the missing data.

For example, he may know that the missing values tend to be

old cars and the normalized losses of

old cars are significantly higher than the average vehicle.

And of course, finally,

in some cases you may simply want to leave the missing data as missing data.

For one reason or another,

it may be useful to keep that observation even if some features are missing.

Now, let's go into how to drop missing values or replace missing values in Python.

To remove data that contains missing values

Panda's library has a built-in method called dropna.

Essentially, with the dropna method,

you can choose to drop rows or columns that contain missing values like NaN.

So you'll need to specify access equal zero to drop

the rows or access equals one to drop the columns that contain the missing values.

In this example, there is a missing value in the price column.

Since the price of used cars is what we're trying to predict in our upcoming analysis,

we have to remove the cars,

the rows, that don't have a listed price.

It can simply be done in one line of code using dataframe.dropna.

Setting the argument in place to true,

allows the modification to be done on the data set directly.

In place equals true,

just writes the result back into the data frame.

This is equivalent to this line of code.

Don't forget that this line of code does not change

the data frame but is a good way to

make sure that you are performing the correct operation.

To modify the data frame,

you have to set the parameter in place equal to true.

You should always check the documentation if

you are not familiar with the function or method.

The pandas web page has lots of useful resources.

To replace missing values like NaNs with actual values,

Pandas library has a built-in method called replace which can be

used to fill in the missing values with the newly calculated values.

As an example, assume that we want to replace the missing values of

the variable normalized losses by the mean value of the variable.

Therefore, the missing value should be replaced

by the average of the entries within that column.

In Python, first we calculate the mean of the column.

Then we use the method replace to

specify the value we would like to be replaced as the first parameter,

in this case NaN.

The second parameter is the value we would

like to replace it with i.e the mean in this example.

This is a fairly simplified way of replacing missing values.

There are of course other techniques such as replacing

missing values for the average of the group instead of the entire data set.

So, we've gone through two ways in Python to deal with missing data.

We learnt to drop problematic rows or columns containing

missing values and then we learnt how to replace missing values with other values.

But don't forget the other ways to deal with missing data.

You can always check for a higher quality data set or source

or in some cases you may want to leave the missing data as missing data.

# Data Formatting in Python

0:00

In this video, we'll look at the problem of data with different formats,

units and conventions and the pandas methods that help us deal with these issues.

Data is usually collected from different places by

different people which may be stored in different formats.

Data formatting means bringing data into a common standard of

expression that allows users to make meaningful comparisons.

As a part of dataset cleaning,

data formatting ensures the data is consistent and easily understandable.

For example, people may use different expressions to represent New York City,

such as uppercase N uppercase Y,

uppercase N lowercase y,

uppercase N uppercase Y and New York.

Sometimes, this unclean data is a good thing to see.

For example, if you're looking at the different ways people tend to write New York,

then this is exactly the data that you want.

Or if you're looking for ways to spot fraud,

perhaps writing N.Y. is more likely

to predict an anomaly than if someone wrote out New York in full.

But perhaps more often than not,

we just simply want to treat them all as the same entity or

format to make statistical analyses easier down the road.

Referring to our used car dataset,

there's a feature named city-miles per gallon in the dataset,

which refers to a car fuel consumption in miles per gallon unit.

However, you may be someone who lives in a country that uses metric units.

So, you would want to convert those values to

liters per 100 kilometers, the metric version.

To transform miles per gallon to liters per 100 kilometers,

we need to divide 235 by each value in the city-miles per gallon column.

In Python, this can easily be done in one line of code.

You take the column and set it to equal to 235,

divide it by the entire column.

In the second line of code,

rename column name from city-miles per gallon to

city-liters per 100 kilometers using the data frame rename method.

For a number of reasons,

including when you import a dataset into Python,

the data type may be incorrectly established.

For example, here we noticed the assigned data type to the price feature is object.

Although the expected data type should really be an integer or float type.

It is important for later analysis to explore

the features data type and convert them to the correct data types.

Otherwise, the developed models later on may behave strangely,

and totally valid data may end up being treated like missing data.

There are many data types in pandas.

Objects can be letters or words.

Int64 are integers and floats are real numbers.

There are many others that we will not discuss.

To identify features data type,

in Python we can use the dataframe.dtypes

method and check the data type of each variable in a data frame.

In the case of wrong data types,

the method dataframe.astype can be

used to convert a data type from one format to another.

For example, using astype int for the price column,

you can convert the object column into an integer type variable.

# Data Normalization in Python

Notes

[**Discuss**](https://www.coursera.org/learn/data-analysis-with-python/discussions/weeks/2)

0:00

In this video, we'll be talking about data normalization.

An important technique to understand in data pre-processing.

When we take a look at the used car data set,

we notice in the data that the feature length ranges from 150-250,

while feature width and height ranges from 50-100.

We may want to normalize these variables so that the range of the values is consistent.

This normalization can make some statistical analyses easier down the road.

By making the ranges consistent between variables,

normalization enables a fair comparison between the different features,

making sure they have the same impact.

It is also important for computational reasons.

Here is another example that will help you understand why normalization is important.

Consider a data set containing two features, age and income.

Where age ranges from 0-100,

while income ranges from 0-20,000 and higher.

Income is about 1,000 times larger than age and ranges from 20,000-500,000.

So, these two features are in very different ranges.

When we do further analysis,

like linear regression for example,

the attribute income will intrinsically

influence the result more due to its larger value.

But this doesn't necessarily mean it is more important as a predictor.

So, the nature of the data biases

the linear regression model to weigh income more heavily than age.

To avoid this, we can normalize

these two variables into values that range from zero to one.

Compare the two tables at the right.

After normalization, both variables now

have a similar influence on the models we will build later.

There are several ways to normalize data.

I will just outline three techniques.

The first method called simple feature scaling just

divides each value by the maximum value for that feature.

This makes the new values range between zero and one.

The second method called min-max takes

each value X\_old subtract it from the minimum value of that feature,

then divides by the range of that feature.

Again, the resulting new values range between zero and one.

The third method is called z-score or standard score.

In this formula for each value you subtract the mu which is the average of the feature,

and then divide by the standard deviation sigma.

The resulting values hover around zero,

and typically range between negative three and positive three but can be higher or lower.

Following our earlier example,

we can apply the normalization method on the length feature.

First, we use the simple feature scaling method,

where we divide it by the maximum value in the feature.

Using the pandas method max,

this can be done in just one line of code.

Here's the min-max method on the length feature.

We subtract each value by the minimum of that column,

then divide it by the range of that column.

The max minus the min.

Finally, we apply the z-score method on length feature to normalize the values.

Here we apply the mean and STD method on the length feature.

Mean method will return the average value of the feature in the data set,

and STD method will return the standard deviation of the features in the data set.

# Binning in Python

0:00

In this video, we’ll talk about binning as a method of data pre-processing.

Binning is when you group values together into bins. For example, you can bin “age”

into [0 to 5], [6 to 10], [11 to 15] and so on.

Sometimes, binning can improve accuracy of the predictive models.

In addition, sometimes we use data binning to group a set of numerical values into a

smaller number of bins to have a better understanding of the data distribution.

As example, “price” here is an attribute range from 5,000 to 45,500.

Using binning, we categorize the price into three bins: low price, medium price, and high

prices.

In the actual car dataset, ”price" is a numerical variable ranging from 5188 to 45400,

it has 201 unique values.

We can categorize them into 3 bins: low, medium, and high-priced cars.

In Python we can easily implement the binning: We would like 3 bins of equal binwidth, so

we need 4 numbers as dividers that are equal distance apart.

First we use the numpy function “linspace” to return the array “bins” that contains

4 equally spaced numbers over the specified interval of the price.

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.

You can then use histograms to visualize the distribution of the data after they’ve been

divided into bins.

This is the histogram that we plotted based on the binning that we applied in the price

feature.

From the plot, it is clear that most cars have a low price, and only very few cars have

high price.

# Turning categorical variables into quantitative variables in Python

Notes

[**Discuss**](https://www.coursera.org/learn/data-analysis-with-python/discussions/weeks/2)

0:00

In this video, we'll discuss how to turn

categorical variables into quantitative variables in Python.

Most statistical models cannot take in objects or strings as input

and for model training only take the numbers as inputs.

In the car data set,

the fuel type feature as a categorical variable has two values,

gas or diesel, which are in string format.

For further analysis, Jerry has to convert

these variables into some form of numeric format.

We encode the values by adding new features corresponding to

each unique element in the original feature we would like to encode.

In the case where the feature fuel has two unique values,

gas and diesel, we create two new features, gas and diesel.

When a value occurs in the original feature,

we set the corresponding value to one in the new feature.

The rest of the features are set to zero.

In the fuel example,

for car B, the fuel value is diesel.

Therefore we set the feature diesel equal to one and the gas feature to zero.

Similarly, for Car D,

the fuel value is gas.

Therefore we set the feature gas equal to one and the feature diesel equal to zero.

This technique is often called "One-hot encoding.

In Pandas, we can use get\_dummies

method to convert categorical variables to dummy variables.

In Python, transforming categorical variables to dummy variables is simple.

Following the example, pd.get\_dummies method gets

the fuel type column and creates the data frame dummy\_variable\_1.

The get\_dummies method automatically generates a list of numbers,

each one corresponding to a particular category of the variable.

# Lesson Summary

In this lesson, you have learned how to:

**Identify and Handle Missing Values:** Drop rows with incomplete information and impute missing data using the mean values.

**Understand Data Formatting:** Wrangle features in a dataset and make them meaningful for data analysis.

**Apply normalization to a data set:** By understanding the relevance of using feature scaling on your data and how normalization and standardization have varying effects on your data analysis.

In this module, you will learn what is meant by exploratory data analysis, and you will learn how to perform computations on the data to calculate basic descriptive statistical information, such as mean, median, mode, and quartile values, and use that information to better understand the distribution of the data. You will learn about putting your data into groups to help you visualize the data better, you will learn how to use the Pearson correlation method to compare two continuous numerical variables, and you will learn how to use the Chi-square test to find the association between two categorical variables and how to interpret them.

**Learning Objectives**

* Implement descriptive statistics
* Demonstrate the basics of grouping
* Describe data correlation processes
* Describe why and how to apply the Chi-Squared test

# Exploratory Data Analysis

0:00

In this module we're going to cover the basics of exploratory data analysis using python.

Exploratory data analysis or in short EDA is an approach to analyze data in order to

summarize main characteristics of the data gain better understanding of the data set,

uncover relationships between different variables, and extract important variables for the problem

we're trying to solve. The main question we are trying to answer in this module is

what are the characteristics that have the most impact on the car price? We will be going through

a couple of different useful exploratory data analysis techniques in order to answer this

question: In this module you will learn about descriptive statistics, which describe basic

features of a data set and obtains a short summary about the sample and measures of the data. Basic

of grouping data using group by and how this can help to transform our data set, the correlation

between different variables, and lastly advanced correlation. Where we'll introduce you to

various correlation statistical methods namely pearson correlation and correlation heat maps

# Descriptive Statistics

0:00

In this video, we'll be talking about Descriptive Statistics.

When you begin to analyze data, it's important to first explore your data

before you spend time building complicated models.

One easy way to do so, is to calculate some Descriptive Statistics for your data.

Descriptive statistical analysis helps to describe basic features of a data set, and

obtains a short summary about the sample and measures of the data.

Let's show you a couple different useful methods.

Play video starting at ::30 and follow transcript0:30

One way in which we can do this is by using the describe function in pandas.

Using the describe function and applying it on your data frame, the describe

function automatically computes basic statistics for all numerical variables.

It shows the mean, the total number of data points, the standard deviation,

the quartiles and the extreme values.

Any NAN values are automatically skipped in these statistics.

This function will give you a clear idea of the distribution of your different

variables.

Play video starting at :1:2 and follow transcript1:02

You could have also categorical variables in your data set.

These are variables that can be divided up into different categories or groups, and

have discrete values.

For example, in our data set we have the drive system as a categorical variable,

which consists of the categories, forward wheel drive, rear wheel drive and

four wheel drive.

One way you can summarize the categorical data,

is by using the function value\_counts.

We can change the name of the column to make it easier to read.

We see that we have 118 cars in the front wheel drive category.

75 cars in the rear wheel drive category, and

8 cars in the four wheel drive category.

Play video starting at :1:44 and follow transcript1:44

Box plots are a great way to visualize numeric data,

since you can visualize the various distributions of the data.

The main features that the box plot shows, are the median of the data,

which represents where the middle data point is.

The upper quartile shows where the 75th percentile is.

The lower quartile shows where the 25th percentile is.

The data between the upper and

lower quartile represents the interquartile range.

Next you have the lower and upper extremes.

These are calculated as 1.5 times the interquartile range,

above the 75th percentile, and as 1.5 times the IQR below the 25th percentile.

Finally, box plots also display outliers as individual dots that

occur outside the upper and lower extremes.

With box plots, you can easily spot outliers, and

also see the distribution and skewness of the data.

Box plots make it easy to compare between groups.

In this example, using box plot we can see the distribution of different categories

of the drive wheels feature over price feature.

We can see that the distribution of price between the rear wheel drive, and

the other categories are distinct.

But the price for front wheel drive and

four wheel drive are almost indistinguishable.

Often times we tend to see continuous variables in our data.

These data points are numbers contained in some range.

For example, in our data set price and engine size are continuous variables.

What if we want to understand the relationship between engine size and

price.

Could engine size possibly predict the price of a car?

One good way to visualize this is using a scatter plot.

Each observation in the scatter plot is represented as a point.

This plot shows the relationship between two variables.

The predictor variable, is the variable that you are using to predict an outcome.

In this case our predictor variable is the engine size.

The target variable is the variable that you are trying to predict.

In this case, our target variable is the price.

Since this would be the outcome.

In a scatter plot, we typically set the predictor variable on the x-axis or

horizontal axis, and we set the target variable on the y-axis or vertical axis.

Play video starting at :4:3 and follow transcript4:03

In this case, we will thus plot the engine size on the x-axis and

the price on the y-axis.

We are using, the matplotlib functions scatter here, taking in x and y variable.

Something to note is that it's always important to label your axes, and

write a general plot title, so that you know what you're looking at.

Now how is the variable engine size related to price?

From the scatter plot, we see that as the engine size goes up,

the price of the car also goes up.

This is giving us an initial indication that there is a positive

linear relationship between these two variables.

Play video starting at :4:39 and follow transcript4:39

[MUSIC]

# GroupBy in Python

Notes

[**Discuss**](https://www.coursera.org/learn/data-analysis-with-python/discussions/weeks/3)

0:00

In this video, we'll cover the basics of

grouping and how this can help to transform our dataset.

Assume you want to know, is there

any relationship between the different types of drive system,

forward, rear, and four-wheel drive,

and the price of the vehicles?

If so, which type of drive system adds the most value to a vehicle?

It would be nice if we could group all the data by the different types of

drive wheels and compare the results of these different drive wheels against each other.

In Pandas, this can be done using the group by method.

The group by method is used on categorical variables,

groups the data into subsets according to the different categories of that variable.

You can group by a single variable or you can group by

multiple variables by passing in multiple variable names.

As an example, let's say we are interested in finding the average price of vehicles and

observe how they differ between different types of

body styles and drive wheels variables.

To do this, we first pick out the three data columns we are interested in,

which is done in the first line of code.

We then group the reduced data according to

drive wheels and body style in the second line.

Since we are interested in knowing how the average price differs across the board,

we can take the mean of each group and

append it this bit at the very end of the line too.

The data is now grouped into subcategories and

only the average price of each subcategory is shown.

We can see that according to our data,

rear wheel drive convertibles and rear wheel drive hard hardtops have

the highest value while four wheel drive hatchbacks have the lowest value.

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.

In the previous table,

both drive wheels and body style were listening columns.

A pivot table has one variable displayed along

the columns and the other variable displayed along the rows.

Just with one line of code and by using the Panda's pivot method,

we can pivot the body style variable so it is displayed

along the columns and the drive wheels will be displayed along the rows.

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.

Heat map takes a rectangular grid of data and assigns

a color intensity based on the data value at the grid points.

It is a great way to plot the target variable over multiple variables and

through this get visual clues with

the relationship between these variables and the target.

In this example, we use pyplot's p color method to

plot heat map and convert the previous pivot table into a graphical form.

We specify the red-blue color scheme.

In the output plot,

each type of body style is numbered along

the x-axis and each type of drive wheels is numbered along the y-axis.

The average prices are plotted with varying colors based on their values.

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

0:00

In this video, we'll talk about the correlation between different variables.

Correlation is a statistical metric for

measuring to what extent different variables are interdependent.

In other words, when we look at two variables over time,

if one variable changes how does this affect change in the other variable?

For example, smoking is known to be correlated to lung cancer

since you have a higher chance of getting lung cancer if you smoke.

In another example, there is a correlation between umbrella and

rain variables where more precipitation means more people use umbrellas.

Also, if it doesn't rain people would not carry umbrellas.

Therefore, we can say that umbrellas and rain are

interdependent and by definition they are correlated.

It is important to know that correlation doesn't imply causation.

In fact, we can say that umbrella and rain are correlated but we would not have

enough information to say whether the umbrella

caused the rain or the rain caused the umbrella.

In data science we usually deal more with correlation.

Let's look at the correlation between engine size and price.

This time we'll visualize these two variables using

a scatter plot and an added linear line called a regression line,

which indicates the relationship between the two.

The main goal of this plot is to see whether the engine size has any impact on the price.

In this example, you can see that the straight line through the data points is very

steep which shows that there's a positive linear relationship between the two variables.

With increase in values of engine size,

values of price go up as well and the slope of the line is positive.

So there is a positive correlation between engine size and price.

We can use seaborn.regplot to create the scatter plot.

As another example, now let's look at the relationship between

highway miles per gallon to see its impact on the car price.

As we can see in this plot,

when highway miles per gallon value goes up the value price goes down.

Therefore there is a negative linear relationship

between highway miles per gallon 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.

These two variables are said to have a negative correlation.

Finally, we have an example of a weak correlation.

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

0:00

In this video, we'll introduce you to various correlations statistical methods.

One way to measure the strength of the correlation between

continuous numerical variable is by using a method called Pearson correlation.

Pearson correlation method will give you

two values: the correlation coefficient and the P-value.

So how do we interpret these values?

For the correlation coefficient,

a value close to 1 implies a large positive correlation,

while a value close to negative 1 implies a large negative correlation,

and a value close to zero implies no correlation between the variables.

Next, the P-value will tell us how

certain we are about the correlation that we calculated.

For the P-value, a value less than.001

gives us a strong certainty about the correlation coefficient that we calculated.

A value between.001 and.05 gives us moderate certainty.

A value between.05 and.1 will give us a weak certainty.

And a P-value larger than.1 will give us no certainty of correlation at all.

We can say that there is a strong correlation when

the correlation coefficient is close to 1 or negative 1,

and the P-value is less than.001.

The following plot shows data with different correlation values.

In this example, we want to look at the correlation

between the variable's horsepower and car price.

See how easy you can calculate the Pearson correlation using the SI/PI stats package?

We can see that the correlation coefficient is approximately.8, and this is close to 1.

So there is a strong positive correlation.

We can also see that the P-value is very small, much smaller than.001.

And so we can conclude that we are certain about the strong positive correlation.

Taking all variables into account,

we can now create a heatmap that indicates

the correlation between each of the variables with one another.

The color scheme indicates the Pearson correlation coefficient,

indicating the strength of the correlation between two variables.

We can see a diagonal line with a dark red color,

indicating that all the values on this diagonal are highly correlated.

This makes sense because when you look closer,

the values on the diagonal are the correlation of all variables with themselves,

which will be always 1.

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

0:07

In this video, we will learn how to find out if there is a relationship between two categorical

variables. When dealing with the relationships between

two categorical variables, we can’t use the same correlation method for continuous

variables, we will have to employ the use of chi square

test for the association.

The Chi-square test is intended to test how likely it is that an observed distribution

is due to chance. It measures how well the observed distribution

of data fits with the distribution that is expected if the variables are independent.

Before we go into an example, let's look through some important points.

The Chi-square tests a null hypothesis that the variables are independent. The test compares

the observed data to the values that the model expects if the data was distributed in different

categories by chance. Anytime the observed data doesn't fit within the model of the expected

values, the probability that the variables are dependent becomes stronger, thus proving

the null hypothesis incorrect. The Chi-square does not tell you the type

of relationship that exists between both variables, but only that a relationship exists.

We will use the cars dataset. Assuming we want to test the relationship between fuel-type

and aspiration; these are categorical variables. It is either the fuel-type of the car, gas

or diesel, and the aspiration is that either the car is standard or turbo. To do this we

will find the observed counts of cars in each category. This can be done by creating a crosstab

using the pandas library. A crosstab is a table showing the relationship between two

or more variables. When the table only shows the relationship between two categorical variables,

the crosstab is also known as a contingency table. In our case the crosstab or contingency

table shows us the counts in each category:

A standard car with diesel fuel,

a standard car with gas fuel, a turbo car with diesel fuel,

or a turbo car with gas fuel. The formula for chi-square is given as follows:

The summation of the observed value i.e., the counts in each group, minus the expected

value, all squared, divided by the expected value.

Expected values are based on the given totals, that is, what can we say individual cells would

be if we did not know the observed values? To calculate the expected value of a standard

car with diesel, we take the row total which is twenty multiplied

by the column total, one hundred and sixty-eight,

divided by the grand total of two hundred and five.

This will give you sixteen point three nine.

If we do the same thing for turbo cars with gas fuel, we will take the

row total, one hundred and eighty-five, multiplied by

column total, thirty-seven, and we divide by

the grand total, two hundred and five, to get

thirty-three point three nine. If we repeat the same procedure for all of

them we get these values.

If we took the row totals, column totals, and grand total

we will get the same values as the totals as the observed values.

Now going back to this formula, if we took a summation of all the observed, minus the

expected values, all squared, divided by the expected value, we will get a chi-square value

of twenty-nine point six.

On the chi-square table

we check on the degree of freedom equals one row and find the value closest to twenty-nine point six.

Here we can see that twenty-nine point six will fall in between a p-value less

than zero point zero five. Therefore, we can say the p-value is less than

zero point zero five. Since the p-value is less than point zero five, we reject the

null hypothesis that the two variables are independent and therefore we conclude that

there is an association between fuel type and aspiration.

To do this in Python we will use the chi square contingency function in the SciPy dot statistics

package. The function will print out the chi-square

test value twenty-nine point six, the second value is the p-value which is very close to

zero and a degree of freedom of one. If you remember, the chi-square table did not give an exact

p-value but a range in which it falls. Python will give the exact p-value. We can see the

same results as in our previous slides. It also prints out the expected values which

we also calculated by hand. Since the p-value is close to zero, we reject

the null hypothesis that the two variables are independent and conclude that there is

evidence of association between fuel-type and aspiration.

# Lesson Summary

In this lesson, you have learned how to:

**Describe Exploratory Data Analysis:** By summarizing the main characteristics of the data and extracting valuable insights.

**Compute basic descriptive statistics:** Calculate the mean, median, and mode using python and use it as a basis in understanding the distribution of the data.

**Create data groups:** How and why you put continuous data in groups and how to visualize them.

**Define correlation as the linear association between two numerical variables:** Use Pearson correlation as a measure of the correlation between two continuous variables

**Define the association between two categorical variables:** Understand how to find the association of two variables using the Chi-square test for association and how to interpret them.

In this module, you will learn how to define the explanatory variable and the response variable and understand the differences between the simple linear regression and multiple linear regression models. You will learn how to evaluate a model using visualization and learn about polynomial regression and pipelines. You will also learn how to interpret and use the R-squared and the mean square error measures to perform in-sample evaluations to numerically evaluate our model. And lastly, you will learn about prediction and decision making when determining if our model is correct.

**Learning Objectives**

* Describe how to process linear regression in Python
* Apply model evaluation using visualization in Python
* Apply polynomial regression techniques to Python
* Evaluate a data model by using visualization
* Describe the use of R-squared and MSE for in-sample evaluation
* Apply prediction and decision making to Python model creation

# Model Development

0:00

In this video we will examine

model development by trying to predict the price of a car using our dataset.

In this module, you'll learn about simple and multiple linear regression,

model evaluation using visualization,

polynomial regression and pipelines,

R-squared and MSE for in-sample evaluation,

prediction and decision making,

and how you can determine a fair value for a used car.

A model or estimator can be thought of as

a mathematical equation used to predict the value given one or more other values.

Relating one or more independent variables or features to dependent variables.

For example, you input a car model's

highway miles per gallon as the independent variable or feature,

the output of the model or dependent variable is the price.

Usually, the more relevant data you have,

the more accurate your model is.

For example, you input multiple independent variables or features to your model.

Therefore, your model may predict a more accurate price for the car.

To understand why more data is important,

consider the following situation.

You have two almost identical cars.

Pink cars sell for significantly less.

You want to use your model to determine the price of two cars,

one pink, one red.

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.

In addition to getting more data,

you can try different types of models.

In this course, you will learn about simple linear regression,

multiple linear regression and polynomial regression.

# Linear Regression and Multiple Linear Regression

0:00

In this video, we'll be talking about

simple linear regression and multiple linear regression.

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 or SLR is

a method to help us understand the relationship between two variables.

The predictor independent variable x and the target dependent variable y.

We would like to come up with a linear relationship between the variables shown here.

The parameter b\_0 is the intercept.

The parameter b\_1 is the slope.

When we fit or train the model,

we will come up with these parameters.

This step requires lots of math,

so we will not focus on this part.

Let's clarify the prediction step.

It's hard to figure out how much a car costs,

but the highway miles per gallon is in the owner's manual.

If we assume there is a linear relationship between these variables,

we can use this relationship to formulate a model to determine the price of the car.

If the highway miles per gallon is 20,

we can input this value into the model to obtain a prediction of $22,000.

In order to determine the line,

we take data points from our data set marked in red here.

We then use these training points to fit our model.

The results of the training points are the parameters.

We usually store the data points into data frame or numpy arrays.

The value we would like to predict is called the target that we store in the array y.

We store the independent variable in the data frame or array x.

Each sample corresponds to a different row in each data frame or array.

In many cases, many factors influence how much people pay for a car.

For example, make or how old the car is.

In this model, this uncertainty is taken into account by

assuming a small random value is added to the point on the line.

This is called noise.

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.

Usually a small positive value is added or a small negative value.

Sometimes, large values are added.

But for the most part,

the values added are near zero.

We can summarize the process like this.

We have a set of training points.

We use these training points to fit or train the model and get parameters.

We then use these parameters in the model. We now have a model.

We use the hat on the y to denote the model is an estimate.

We can use this model to predict values that we haven't seen.

For example, we have no car with 20 highway miles per gallon.

We can use our model to make a prediction for the price of this car.

But don't forget our model is not always correct.

We can see this by comparing the predicted value to the actual value.

We have a sample for 10 highway miles per

gallon but the predictive value does not match the actual value.

If the linear assumption is correct,

this error is due to the noise.

But there can be other reasons.

To fit the model in Python,

first we import linear model from

sklearn then create a linear regression object using the constructor.

We define the predictor variable and target variable then use the method

fit to fit the model and find the parameters b\_0 and b\_1.

The input are the features and the targets.

We can obtain prediction using the method predict.

The output is an array.

The array has the same number of samples as the input x.

The intercept b\_0 is an attribute of the object lm.

The slope b\_1 is also an attribute of the object lm.

The relationship between price and highway miles per

gallon is given by this equation in bold,

price equals 38,423.31 minus

821.73 times highway miles per gallon,

like the equation we discussed before.

Multiple linear regression is used to explain the relationship between

one continuous target y variable and two or more predictor x variables.

If we have for example 4 predictor variables then b\_0

intercept x equal zero b \_1 the coefficient or parameter of x\_1,

b\_2 the coefficient of parameter x\_2 and so on.

If there are only two variables then we can visualize the values.

Consider the following function:

The variables x\_1 and x\_2 can be visualized on a 2D plane.

Let's do an example on the next slide.

The table contains different values of the predictor variables x\_1 and x\_2.

The position of each point is placed on the 2D plane color-coded accordingly.

Each value of the predictor variables x\_1 and x\_2 will be mapped to a new value y, y hat.

The new values of y y hat are mapped in

the vertical direction with height proportional to the value that y hat takes.

We can fit the multiple linear regression as follows.

We can extract the four predictor variables and store

them in the variable z then train the model

as before using the method train with

the features or dependent variables and the targets colon.

We can also obtain a prediction using the method predict.

In this case, the input is an array or data frame with four columns.

The number of rows corresponds to the number of samples.

The output is an array with the same number of elements as number of samples.

The intercept is an attribute of the object and the coefficients are also attributes.

It is helpful to visualize the equation,

replacing the dependent variable names with actual names.

This is identical to the form we discussed earlier.

# Model Evaluation using Visualization

0:00

In this video, we'll look at Model Evaluation using Visualization.

Regression plots are a good estimate of the relationship between two variables,

the strength of the correlation,

and the direction of the relationship (positive or negative).

The horizontal axis is the independent variable.

The vertical axis is the dependent variable.

Each point represents a different target point.

The fitted line represents the predicted value.

There are several ways to plot a regression plot.

A simple ways to use regplot from the seaborn library.

First, "import seaborn."

Then use the "regplot" function.

The parameter x is the name of the column that

contains the independent variable or feature.

The parameter y, contains the name of the column that

contains the name of the dependent variable or target.

The parameter data is the name of the dataframe.

The result is given by the plot.

The residual plot represents the error between the actual value.

Examining the predicted value and actual value we see a difference.

We obtain that value by subtracting the predicted value,

and the actual target value.

We then plot that value on

the vertical axis with the independent variable as the horizontal axis.

Similarly, for the second sample,

we repeat the process.

Subtracting the target value from the predicted value.

Then plotting the value accordingly.

Looking at the plot gives us some insight into our data.

We expect to see the results to have zero mean,

distributed evenly around the x axis with similar variance.

There is no curvature.

This type of residual plot suggests a linear plot is appropriate.

In this residual plot, there is a curvature.

The values of the error change with x.

For example, in the region,

all the residual errors are positive.

In this area, the residuals are negative.

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 will deal with this in the next section.

In this plot, we see the variance of the residuals increases with x.

Therefore, our model is incorrect.

We can use seaborn to create a residual plot.

First, "import seabourn."

We use the "residplot" function.

The first parameter is a series of dependent variable or feature.

The second parameter is a series of dependent variable or target.

We see in this case, the residuals have a curvature.

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.

Let's look at a simplified example.

We examined the vertical axis.

We then count and plot the number of

predicted points that are approximately equal to one.

We then, count and plot the number of

predicted points that are approximately equal to two.

We repeat the process.

For predicted points, they are approximately equal to three.

Then we repeat the process for the target values.

In this case, all the target values are approximately equal to two.

The values of the targets and predicted values are continuous.

A histogram is for discrete values.

Therefore, pandas will convert them to a distribution.

The vertical axis is scaled to make the area under the distribution equal to one.

This is an example of using a distribution plot.

The dependent variable or feature is price.

The fitted values that result from the model are in blue.

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.

In this example, we use multiple features or independent variables.

Comparing it to the plot on the last slide,

we see predicted values are much closer to the target values.

Here's the code to create a distribution plot.

The actual values are used as a parameter.

We wanted distribution instead of a histogram.

So we want the hist parameters set to false.

The color is red. The label is also included.

The predicted values are included for the second plot.

The rest of the parameters are set accordingly.

# Polynomial Regression and Pipelines

0:00

In this video, we will cover polynomial regression and pipelines.

What do we do when a linear model is not the best fit for our data?

Let's look into another type of regression model.

The polynomial regression.

We transform our data into a polynomial,

then use linear regression to fit the parameter.

Then we will discuss pipelines.

Pipelines are a way to simplify your code.

Polynomial regression is a special case of the general linear regression.

This method is beneficial for describing curvilinear relationships.

What is a curvilinear relationship?

It's what you get by squaring or setting higher order terms of

the predictor variables in the model transforming the data.

The model can be quadratic,

which means that the predictor variable in the model is squared.

We use a bracket to indicate it as an exponent.

This is a second order polynomial regression,

with a figure representing the function.

The model can be cubic,

which means that the predictor variable is cubed.

This is the third order polynomial regression.

We see by examining the figure that the function has more variation.

There also exists higher order polynomial regressions.

When a good fit hasn't been achieved by second or third order.

We can see in figures how much the graphs change,

when we change the order of the polynomial regression.

The degree of the regression makes

a big difference and can result in a better fit If you pick the right value.

In all cases, the relationship between the variable and the parameter is always linear.

Let's look at an example from our data where we generate a polynomial regression model.

In Python we do this by using the polyfit function.

In this example, we develop a third order polynomial regression model base.

We can print out the model.

Symbolic form for the model is given by the following expression:

negative 1.557 x1 cubed plus 204.8 x one

squared plus 8965 x1 plus 1.37 times 10 to the power of five.

We could also have multi-dimensional polynomial linear regression.

The expression can get complicated.

Here are just some of the terms for a two dimensional second order polynomial.

Numpy's polyfit function cannot perform this type of regression.

We use the preprocessing library in scikit-learn to create a polynomial feature object.

The constructor takes the degree of the polynomial as a parameter.

Then we transform the features into

a polynomial feature with the fit underscore transform method.

Let's do a more intuitive example.

Consider the features shown here.

Applying the method we transform the data,

we now have a new set of features that are

a transformed version of our original features.

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. We train the object,

fit the scale object,

then transform the data into a new data frame on array x\_scale.

There are more normalization methods available in

the preprocessing library as well as other transformations.

We can simplify our code by using a pipeline library.

There are many steps to getting a prediction.

For example, normalization, polynomial transform, and linear regression.

We simplify the process using a pipeline.

Pipeline sequentially perform a series of transformations.

The last step carries out a prediction.

First we import all the modules we need,

then we import the library pipeline.

We create a list of tuples,

the first element in the tuple contains the name of the estimator model.

The second element contains model constructor.

We input the list in the pipeline constructor.

We now have a pipeline object.

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.

[music]

# Measures for In-Sample Evaluation

0:00

Now that we’ve seen how we can evaluate a model by using visualization, we want to

numerically evaluate our models.

Let’s look at some of the measures that we use for in-sample evaluation.

These measures are a way to numerically determine how good the model fits on our data.

Two important measures that we often use to determine the fit of a model are: Mean Square

Error (MSE), and R-squared.

To measure the MSE, we find the difference between the actual value y and the predicted

value yhat then square it.

In this case, the actual value is 150; the predicted value is 50. Subtracting these points

we get 100.

We then square the number.

We then take the Mean or average of all the errors by adding then all together and dividing

by the number of samples.

To find the MSE in Python, we can import the “mean\_Squared\_error” from “scikit-learn.metrics”.

The “mean\_Squared\_error” function gets two inputs: the actual value of target variable

and the predicted value of target variable.

R-squared is also called the coefficient of determination. It’s a measure to determine

how close the data is to the fitted regression line. So how close is our actual data to our

estimated model?

Think about it as comparing a regression model to a simple model, i.e., the mean of the data

points. If the variable x is a good predictor our model should perform much better than

just the mean.

In this example the average of the data points 𝑦 | is 6.

Coefficient of Determination R^2 is 1 minus the ratio of the MSE of the regression

lined divided by the MSE of the average of the data points. For the most part, it takes

values between 0 and 1.

Let’s look at a case where the line provides a relatively good fit.

The blue line represents the regression line.

The blue squares represent 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.

In this case, because the line is a good fit, the Mean squared error is small, therefore

the numerator is small.

The Mean squared error of the line is relatively large, as such the numerator is large.

A small number divided by a larger number is an even smaller number. Taken to an extreme

this value tends to zero.

If we Plug in this value from the previous slide for R^2, we get a value near one, this

means the line is a good fit for the data. Here is an example of a line that does not

fit the data well.

If we just examine the area of the red squares compared to the blue squares, we see the area

is almost identical.

The ratio of the areas is close to one.

In this case the R^2 is near zero.

This line performs about the same as just using the average of the data points, therefore,

this line did not perform well.

We find the R-squared value in Python by using the score method, in the linear regression

object.

From the value that we get from this example, we can say that approximately 49.695% of the

variation of price is explained by this simple linear model.

Your R^2 value is usually between 0 and 1. If your R^2 is negative, it can be due to over fitting

that we will discuss in the next module.

[music]

# Prediction and Decision Making

0:07

In this video our final topic will be on prediction and decision making.

How can we determine if our model is correct?

Play video starting at ::15 and follow transcript0:15

The first thing you should do is make sure your model results make sense. You should always use

visualization, numerical measures for evaluation, and comparing between different models.

Let's look at an example of prediction. If you recall we trained the model using the fit

method. Now, we want to find out what the price would be for a car that has a highway miles per

gallon of 30. Plugging this value into the predict method gives us a resulting price of

$13,771.30 this seems to make sense. For example, the value is not negative, extremely high, or

extremely low. We can look at the coefficients by examining the coeff underscore attribute. If you

recall the expression for the simple linear model that predicts price from highway miles per gallon.

This value corresponds to the multiple of the highway miles per gallon feature.

As such an increase of one unit in highway miles per gallon

the value of the car decreases approximately 821 dollars. This value also seems reasonable

Play video starting at :1:20 and follow transcript1:20

sometimes your model will produce values that don't make sense.

For example, if we plot the model out for highway miles per gallon in the ranges of 0 to 100 we

get negative values for the price. This could be because the values in that range are not realistic

the linear assumption is incorrect or we don't have data for cars in that range. In this case

it is unlikely that a car will have fuel mileage in that range so our model seems valid. To

generate a sequence of values in a specified range import numpy then use the numpy arrange function

to generate the sequence the sequence starts at 1 and increments by 1 until we reach 100.

The first parameter is the starting point of the sequence. The second parameter is the endpoint

plus one of the sequence the final parameter is the step size between elements in the sequence.

In this case it's one so we increment the sequence one step at a time from one to two and so on.

We can use the output to predict new values the output is a numpy array many of the values are

negative. Using a regression plot to visualize your data is the first method you should try.

See the labs for examples of how to plot polynomial regression

for this example the effect of the independent variable is evident in this case. The data

trends down as the dependent variable increases the plot also shows some non-linear behavior.

Play video starting at :2:51 and follow transcript2:51

Examining the residual plot we see in this case, the residuals have a curvature suggesting

non-linear behavior. A distribution plot is a good method for multiple linear regression.

For example, we see the predicted values for prices in the range from thirty thousand to fifty

thousand are inaccurate this suggests a non-linear model may be more suitable or we need more data in

this range. The mean square error is perhaps the most intuitive numerical measure for determining

if a model is good or not. Let's see how different measures of mean square error impact the model.

The figure shows an example of a mean square error of 3495. This example has a mean square

error of three thousand six hundred and fifty two. The final plot has a mean square error of twelve

thousand eight hundred and seventy. As the square error increases the targets get further from the

predicted points. As we discussed r squared is another popular method to evaluate your model.

It tells you how well your line fits into the model. r squared values range from zero to one.

r squared tells us what percent of the variability in the dependent variable is accounted for by

the regression on the independent variable. An r squared of 1 means that all movements of another

dependent variable are completely explained by movements in the independent variables.

In this plot we see the target points in red and the predicted line in blue.

An r squared of 0.9986 the model appears to be a good fit. That means that more than 99

of the variability of the predicted variable is explained by the independent variables. This model

has an r squared of 0.9226 there still is a strong linear relationship model is still a good fit.

An r squared of 0806 of the data we can visually see that the values are scattered around the line.

They are still close to the line and we can say that 80 percent of the variability of the

predicted variable is explained by the independent variables. And an r squared 0.61 means that

approximately 61 percent of the observed variation can be explained by the independent variables.

An acceptable value for r squared depends on what field you are studying and what your

use case is. Falcon Miller 1992 suggests that an acceptable r squared value should be at least 0.1.

Does a lower mean square error imply better fit? Not necessarily MSE4 and MLR model will be smaller

than the MSE for an SLR model. Since the errors of the data will decrease when more variables are

included in the model. Polynomial regression will also have a smaller MSE than regular regression

in the next section we will look at more accurate ways to evaluate the model.

# Lesson Summary

In this lesson, you have learned how to:

**Define the explanatory variable and the response variable:** Define the response variable (y) as the focus of the experiment and the explanatory variable (x) as a variable used to explain the change of the response variable. Understand the differences between Simple Linear Regression because it concerns the study of only one explanatory variable and Multiple Linear Regression because it concerns the study of two or more explanatory variables.

**Evaluate the model using Visualization:** By visually representing the errors of a variable using scatterplots and interpreting the results of the model.

**Identify alternative regression approaches:** Use a Polynomial Regression when the Linear regression does not capture the curvilinear relationship between variables and how to pick the optimal order to use in a model.

**Interpret the R-square and the Mean Square Error:** Interpret R-square (x 100) as the percentage of the variation in the response variable y  that is explained by the variation in explanatory variable(s) x. The Mean Squared Error tells you how close a regression line is to a set of points. It does this by taking the average distances from the actual points to the predicted points and squaring them.

# Model Evaluation and Refinement

0:00

Model evaluation tells us how our model performs in the real world.

In the previous module,

we talked about in-sample evaluation.

In-sample evaluation tells us how well our model fits the data already given to train it.

It does not give us an estimate of how well the train model can predict new data.

The solution is to split our data up,

use the in-sample data or training data to train the model.

The rest of the data, called Test Data,

is used as out-of-sample data.

This data is then used to approximate, how the model performs in the real world.

Separating data into training and testing sets is an important part of model evaluation.

We use the test data to get an idea how our model will perform in the real world.

When we split a dataset,

usually the larger portion of data is used for

training and a smaller part is used for testing.

For example, we can use 70 percent of the data for training.

We then use 30 percent for testing.

We use training set to build a model and discover predictive relationships.

We then use a testing set to evaluate model performance.

When we have completed testing our model,

we should use all the data to train the model.

A popular function, in the scikit-learn package for

splitting datasets, is the train test split function.

This function randomly splits a dataset into training and testing subsets.

From the example code snippet,

this method is imported from sklearn.cross-validation.

The input parameters y\_data is the target variable.

In the car appraisal example,

it would be the price and x\_data,

the list of predictive variables.

In this case, it would be all the other variables in

the car dataset that we are using to try to predict the price.

The output is an array.

x\_train and y\_train the subsets for training.

x\_test and y\_test the subsets for testing.

In this case, the test size is a percentage of the data for the testing set.

Here, it is 30 percent.

The random state is a random seed for random data set splitting.

Generalization error is a 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.

This figure shows the distribution of the actual values in

red compared to the predicted values from a linear regression in blue.

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.

Play video starting at :3:10 and follow transcript3:10

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 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.

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.

This method is imported from sklearn's model selection package.

We then use the function cross\_val\_score.

The first input parameters,

the type of model we are using to do the cross-validation.

In this example, we initialize a linear regression model or object

lr which we passed the cross\_val\_score function.

The other parameters are x\_data,

the predictive variable data,

and y\_data, the target variable data.

We can manage the number of partitions with the cv parameter.

Here, cv equals three,

which means the data set 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 NnumPi.

Let's see an animation,

let's see the result of the score array in the last slide.

First, we split the data into three folds.

We use two folds for training,

the remaining fold for testing.

The model will produce an output.

We will use the output to calculate a score.

In the case of the r squared i.e.

coefficient of determination, we will store that value in an array.

We will repeat the process using two folds for training and one fold for testing.

Save the score, then use

a different combination for training and the remaining fold for testing.

We store the final result.

The cross\_val\_score function returns

a score value to tell us the cross-validation result.

What if we want a little more information?

What if we want to know the actual predicted values

supplied by our model before the r squared values are calculated?

To do this, we use the cross\_ val\_predict function.

The input parameters are exactly the same as the cross\_val\_score function,

but the output is a prediction.

Let's illustrate the process.

First, we split the data into three folds.

We use two folds for training,

the remaining fold for testing.

The model will produce an output, and we will store it in an array.

We will repeat the process using two folds for training, one for testing.

The model produces an output again.

Finally, we use the last two folds for training.

Then we use the testing data.

This final testing fold produces an output.

These predictions are stored in an array.

[music]

# Overfitting, Underfitting and Model Selection

0:00

If you recall, in the last module,

we discussed polynomial regression.

In this section, we will discuss how to pick

the best polynomial order and problems

that arise when selecting the wrong order polynomial.

Consider the following function,

we assume the training points come from a polynomial function plus some noise.

The goal of Model Selection is to determine the order of

the polynomial to provide the best estimate of the function y(x).

If we try and fit the function with a linear function,

the line is not complex enough to fit the data.

As a result, there are many errors.

This is called underfitting,

where the model is too simple to fit the data.

If we increase the order of the polynomial,

the model fits better, but the model is

still not flexible enough and exhibits underfitting.

This is an example of the 8th order polynomial used to fit the data.

We see the model does well at fitting the data and estimating

the function even at the inflection points.

Increasing it to a 16th order polynomial,

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 not tracking the function.

This is called overfitting,

where the model is too flexible and fits the noise rather than the function.

Let's look at a plot of the mean square error for

the training and testing set of different order polynomials.

The horizontal axis represents the order of the polynomial.

The vertical axis is the mean square error.

The training error decreases with the order of the polynomial.

The test error is a better means of estimating the error of a polynomial.

The error decreases 'till the best order of the polynomial

is determined. Then the error begins to increase.

We select the order that minimizes the test error.

In this case, it was eight.

Anything on the left would be considered underfitting.

Anything on the right is overfitting.

If we select the best order of the polynomial,

we will still have some errors.

If you recall the original expression for the training points we see a noise term.

This term is one reason for the error.

This is because the noise is random, and we can't predict it.

This is sometimes referred to as an irreducible error.

There are other sources of errors as well.

For example, our polynomial assumption may be wrong.

Our sample points may have come from a different function.

For example, in this plot,

the data is generated from a sine wave.

The polynomial function does not do a good job at fitting the sine wave.

For real data, the model may be too difficult to

fit or we may not have the correct type of data to estimate the function.

Let's try different order polynomials on the real data using horsepower.

The red points represent the training data.

The green points represent the test data.

If we just use the mean of the data,

our model does not perform well.

A linear function does fit the data better.

A second order model looks similar to the linear function.

A third order function also appears to increase,

like the previous two orders.

Here, we see a fourth order polynomial.

At around 200 horsepower,

the predicted price suddenly decreases.

This seems erroneous.

Let's use R-squared to see if our assumption is correct.

The following is a plot of the R-squared value.

The horizontal axis represents the order polynomial models.

The closer the R-squared is to one, the more accurate the model is.

Here, we see the R-squared is optimal when the order of the polynomial is three.

The R-squared drastically decreases when the order is increased to four,

validating our initial assumption.

We can calculate different R-squared values as follows.

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.

[music]

# Ridge Regression Introduction

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. The next video shows how Ridge regression is used to regularize and reduce the standard errors to avoid over-fitting a regression model

# Ridge Regression

0:00

In this video, we'll discuss ridge regression.

Ridge regression prevents overfitting.

In this video, we will focus on polynomial regression for visualization,

but overfitting is also a big problem when

you have multiple independent variables, or features.

Consider the following fourth order polynomial in orange.

The blue points are generated from this function.

We can use a tenth order polynomial to fit the data.

The estimated function in blue does a good job at approximating the true function.

In many cases real data has outliers.

For example, this point shown here does not appear to come from the function in orange.

If we use a tenth order polynomial function to fit the data,

the estimated function in blue is incorrect,

and is not a good estimate of the actual function in orange.

If we examine the expression for the estimated function,

we see the estimated polynomial coefficients have a very large magnitude.

This is especially evident for the higher order polynomials.

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.

Let's see how different values of alpha change the model.

This table represents the polynomial coefficients for different values of alpha.

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.

For alpha equal to 0.001,

the overfitting begins to subside.

For Alpha equal to 0.01,

the estimated function tracks the actual function.

When alpha equals one,

we see the first signs of underfitting.

The estimated function does not have enough flexibility.

At alpha equals to 10,

we see extreme underfitting.

It does not even track the two points.

In order to select alpha,

we use cross validation.

To make a prediction using ridge regression,

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.

In order to determine the parameter alpha,

we use some data for training.

We use a second set called validation data.

This is similar to test data,

but it is used to select parameters like alpha.

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.

We train the model again,

make a prediction using the validation data,

then calculate the R-squared and store the values of R-squared.

We repeat the process for a different alpha value,

training the model, and making a prediction.

We select the value of alpha that maximizes the R-squared.

Note that we can use other metrics to select the value of alpha,

like mean squared error.

The overfitting problem is even worse if we have lots of features.

The following plot shows the different values of R-squared on the vertical axis.

The horizontal axis represents different values for alpha.

We use several features from our used car data

set and a second order polynomial function.

The training data is in red and validation data is in blue.

We see as the value for alpha increases,

the value of R-squared increases and converges at approximately 0.75.

In this case, we select the maximum value of alpha because

running the experiment for higher values of alpha have little impact.

Conversely, as alpha increases,

the R-squared on the test data decreases.

This is because the term alpha prevents overfitting.

This may improve the results in the unseen data,

but the model has worse performance on the test data.

See the lab on how to generate this plot.

[music]

# Grid Search

0:00

Grid Search allows us to scan through multiple free parameters with few lines of code.

Parameters like the alpha term discussed in

the previous video are not part of the fitting or training process.

These values are called hyperparameters.

Scikit-learn has a means of automatically

iterating over these hyperparameters using cross-validation.

This method is 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.

Let the small circles represent different hyperparameters.

We start off with one value for hyperparameters and train the model.

We use different hyperparameters to train the model.

We continue the process until we have exhausted the different free parameter values.

Each model produces an error.

We select the hyperparameter that minimizes the error.

To select the hyperparameter,

we split our dataset into three parts,

the training set, validation set, and test set.

We train the model for different hyperparameters.

We use the R-squared or mean square error for each model.

We select the hyperparameter that minimizes the mean squared

error or maximizes the R-squared on the validation set.

We finally test our model performance using the test data.

This is the scikit-learn web page,

where the object constructor parameters are given.

It should be noted that the attributes of an object are also called parameters.

We will not make the distinction even though some of

the options are not hyperparameters per se.

In this module, we will focus on

the hyperparameter alpha and the normalization parameter.

The value of your Grid Search is a Python list that contains a Python dictionary.

The key is the name of the free parameter.

The value of the dictionary is the different values of the free parameter.

This can be viewed as a table with various free parameter values.

We also have the object or model.

The Grid Search takes on the scoring method.

In this case, R-squared the number of folds,

the model or object,

and the free parameter values.

Some of the outputs include the different scores for different free parameter values.

In this case, the R-squared along with a free parameter values that have the best score.

First, we import the libraries we need,

including GridSearchCV, the dictionary of parameter values.

We create a ridge regression object or model.

We then create a GridSearchCV object.

The inputs are the ridge regression object,

the parameter values, and the number of folds.

We will use R-squared.

This is the default scoring method. We fit the object.

We can find the best values for the free parameters using the attribute best estimator.

We can also get information like the mean score on

the validation data using the attribute CV result.

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

For example, ridge regression has the option to normalize the data.

To see how to standardize, see module four.

The term alpha is the first element in the dictionary.

The second element is the normalized option.

The key is the name of the parameter.

The value is the different options in

this case because we can either normalize the data or not.

The values are True or False respectively.

The dictionary is a table or grid that contains two different values.

As before, we need the ridge regression object or model.

The procedure is similar except that we have

a table or grid of different parameter values.

The output is the score for all the different combinations of parameter values.

The code is also similar.

The dictionary contains the different free parameter values.

We can find the best value for the free parameters.

The resulting scores of the different free parameters are stored in

this dictionary, Grid1.cv\_results \_.

We can print out the score for the different free parameter values.

The parameter values are stored as shown here.

See the course labs for more examples.

[music]

# Lesson Summary

In this lesson, you have learned how to:

**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.

Congratulations! You have now completed all the modules for this course. In this last module, you will complete the final assignment that will be graded by your peers. In this final assignment, you will assume the role of a Data Analyst working at a real estate investment trust organization who wants to start investing in residential real estate. You will be given a dataset containing detailed information about house prices in the region based on a number of property features, and it will be your job to analyze and predict the market price of houses given that information.

**Learning Objectives**

* Create a Jupyter notebook
* Apply data analysis and modeling techniques to housing price data

# Project Scenario

In this assignment, you are a Data Analyst working at a Real Estate Investment Trust. The Trust would like to start investing in Residential real estate. You are tasked with determining the market price of a house given a set of features. You will analyze and predict housing prices using attributes or features such as square footage, number of bedrooms, number of floors, and so on. A template notebook is provided in the lab; your job is to complete the ten questions. Some hints to the questions are given in the template notebook.

**Dataset Used in this Assignment**

The dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015. It was taken from [here](https://www.kaggle.com/harlfoxem/housesalesprediction?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-wwwcourseraorg-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDA0101ENSkillsNetwork20235326-2022-01-01). It was also slightly modified for the purposes of this course.

For this project, you will utilize JupyterLab running on the Cloud in Skills Network Labs environment.

**Notebook URL:** Alternatively, you can work on your local machine or any other environment of choice, by downloading this link : [Notebook link House Sales](https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/FinalModule_Coursera/House_Sales_in_King_Count_USA.ipynb)

# Lab for Final Project

For this final project, you will utilize Juoyterlite running on the Cloud in Skills Network Labs environment. To launch the lab notebook in a new browser tab, check the box below and click on the Open Tool button.

To work on your local machine or in any other environment, download the lab notebook (.ipynb file) by clicking [HERE](https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/FinalModule_Coursera/House_Sales_in_King_Count_USA.ipynb) .

As you are **completing this notebook**, take and save the screenshots of the final outputs of your solutions (e.g., final charts, tables, calculation results etc.). They will need to be shared in the following Peer Review section of the Final Project module.

Once you have **completed the lab,**you can download the notebook by navigating to "File" and clicking on "Download" button. This will save the (.ipynb) file on your computer. Once saved, you can upload "House\_Sales\_in\_King\_Count\_USA.ipynb" file to "My Submission" tab, of the "Peer-graded Assignment" section.

In the next, **"Peer Review"** section of the course, follow the instructions to submit the downloaded "House\_Sales\_in\_King\_Count\_USA.ipynb" file, along with your screenshots.