Uline Python Class
Corporate Data Analytics

Introduction

Hello and welcome to the course. My name is Naimesh Chaudhari and at the time of writing this manual I have been working with the analytics team within Uline for about 2.5 year. Prior to that I worked with Sears Holding in their Kmart - Grocery & Drug department. I have 2 bachelor's degrees in actuarial science & statistics along with two minors in Management and Anthropology. I am currently working on completing my online master's in data science at Indiana University - Bloomington.

Understanding that data is being collected more regularly now, this is a great time to enter the field. Being able to demystify data and provide actionable insight is invaluable to corporations. The goal of this course is to provide you with beginner level knowledge of Python, a freely available tool that can help us clean, understand, and model data.

Before we start, I would like to lay down some guidelines as to who this course is for. Our goal when developing the course was to get interested candidates a good foundation to start exploring what they can do with python in the data analytics field. We have taken some assumption regarding the candidates:

- Have former programming knowledge
- Understand Uline data
- Understand SQL
- Have job needs where python could be useful
- Have lightly or never used python before

If you fall under the above category, then this should be a valuable course for you. If on the other hand you fall under the below categories, I would urge you to reconsider

- Have used python regularly and completed projects with it
- Do not have programming knowledge
- Are weak in SQL concepts
- Don't have a job function but just want to learn

In the above situations you might find yourself overqualified, struggling with basic concepts, or forgetting what you have learned in this course very quickly.

With that being said, here are some introductory pointers about Python syntax and coding methodology.

- When validating conditions, we use two equal signs ==
- To install packages we use the *!pip install packagename function*
- To load packages, we use the *import package name* function

Course Outline

Basics of Python Programming

- Introduction & Installation
- Reading Data
- Data Structures
- Operators, Logical Statements, & Loops
- Functions

Exploratory Data Analysis

- Data Frames/Pandas
- Graphing with seaborn & matplotLib

Data Modeling

- Machine Learning with scikit-Learn
    - Data Preparation
    - Linear Regression
    - Logistic Regression
    - Decision Trees
    - Random Forest

Installation

- In your C drive create a folder called Anaconda.

- Navigate to [https://www.anaconda.com/distribution/ (https://www.anaconda.com/distribution/)](https://www.anaconda.com/distribution/)
    - Select the 64 bit windows installer
    - Save to Downloads folder

- Open downloaded file
    - Click Next, I agree, & Just me, then next.
    - In destination folder Select the C drive folder you made earlier
    - Click Install

- Delete the downloaded file in Downloads folder

- Launch Anaconda navigator by searching for it using “Search Windows”
- Launch Jupyter Notebook from Anaconda navigator.
- Congrats you are ready to use Python!

About Anaconda

With more than 13 million users, Anaconda is the world’s most popular data science platform and the foundation of modern machine learning. Anaconda Enterprise delivers data science and machine learning at speed and scale, unleashing the full potential of our customers’ data science and machine learning initiatives.

Data Structures

Python Data Structures

Understanding data structures is very important part in getting better at analyzing data. They allow us to be very flexible in how we store and behave with data. There are quite a few data structures available within Python. The built in data structures are: **Numeric, String, Boolean, Lists, Tuples and Dictionaries.**

Lists, strings, and tuples are ordered sequences of objects. Unlike strings that contain only characters, list and tuples can contain any type of objects. Tuples, like strings, are immutables. Lists are mutables so they can be extended or reduced at will. Dictionaries hold key value pairs. They allows us to store any type of object within them. Keys cannot be duplicated within dictionaries. They are mutable, so they can be extended or reduced at will. Boolean objects store true false values.

Once we move past these basic structures, we can move to more advanced objects such as Series and Data Frame which are offered by the Pandas package.

Core Data

- Numeric (No Range Limit)
  - Int - A whole number
  - Float - A decimal number
- String - An alpha phrase of x length
- Boolean - True False

Python Default Data Structures

- Lists
- Tuples
- Dictionaries

Pandas Data Object

- Data Frame

Indexing

Python index always starts at zero!

Objects & Methods

[https://thomas-cokelaer.info/tutorials/python/data_structures.html (https://thomas-cokelaer.info/tutorials/python/data_structures.html)](https://thomas-cokelaer.info/tutorials/python/data_structures.html)

Lists

Lists are ordered sequences of objects. They can contain numeric, string, or boolean data. Lists are able to extend or reduce at will.

```
In [2]: val = [1,2,3,4,5,6]
```

```
In [2]: txt = ['Naimesh','Al','Kyle','Connor']
```

```
In [3]: val
Out[3]: [1, 2, 3, 4, 5, 6]
```

Accessing Values

You can access values within a list by providing the individual index.

```
In [159]: val[0]
Out[159]: 1
```

```
In [160]: txt[3]
Out[160]: 'Connor'
```

Updating Values

You can update values within a list by providing the index.

```
In [161]: val[0] = 100
```

```
In [162]: val
Out[162]: [100, 2, 3, 4, 5, 6]
```

Remove Value from List

You can delete values within a list by providing the index.

```
In [163]: val.remove(6)
```

```
In [164]: val
Out[164]: [100, 2, 3, 4, 5]
```

```
In [165]: del val[-1]
```

```
In [166]: val
Out[166]: [100, 2, 3, 4]
```

List Operations

You can do standard operations within multiple lists

In [11]:
```
val2 = [1,2,3]
```

In [168]:
```
print(val)
print(val2)
```

```
[100, 2, 3, 4]
[1, 2, 3]
```

In [5]:
```
val + val2
```

Out[5]:
```
[1, 2, 3, 4, 5, 6, 1, 2, 3]
```

In [12]:
```
# List subtraction can only be achieved vai the use of set or loops.
list(set(val) - set(val2))
```

Out[12]:
```
[4, 5, 6]
```

In [170]:
```
txt*2
```

Out[170]:
```
['Naimesh', 'Al', 'Kyle', 'Connor', 'Naimesh', 'Al', 'Kyle', 'Connor']
```

In [171]:
```
'Al' in txt
```

Out[171]:
```
True
```

List Indexing

```
Index from rear:    -6  -5  -4  -3  -2  -1
Index from front:    0   1   2   3   4   5
                   +---+---+---+---+---+---+
                   | a | b | c | d | e | f |
                   +---+---+---+---+---+---+
Slice from front:  :   1   2   3   4   5   :
Slice from rear:   :  -5  -4  -3  -2  -1   :
```

In [172]:
```
val
```

Out[172]:
```
[100, 2, 3, 4]
```

In [173]:
```
val[0]
```

Out[173]:
```
100
```

In [174]:
```
val.index(2)
```

Out[174]:
```
1
```

In [175]:
```
val[-3:]
```
Out[175]:
```
[2, 3, 4]
```
In [176]:
```
val[2:]
```
Out[176]:
```
[3, 4]
```

List Methods

Below are some common methods used within lists

In [177]:
```
val
```
Out[177]:
```
[100, 2, 3, 4]
```
In [178]:
```
val.append(7)
val
```
Out[178]:
```
[100, 2, 3, 4, 7]
```
In [179]:
```
val.count(7)
```
Out[179]:
```
1
```
In [180]:
```
val.insert(3,200)
val
```
Out[180]:
```
[100, 2, 3, 200, 4, 7]
```
In [181]:
```
val.sort()
print(val)
val.reverse()
print(val)
```
```
[2, 3, 4, 7, 100, 200]
[200, 100, 7, 4, 3, 2]
```

Tuples

A Tuple is a sequence of immutable objects. Tuples can not be changed. They use parentheses instead of square brackets.

In [182]:
```
tup = (1,2,3,4,5)
```
In [183]:
```
tup
```
Out[183]:
```
(1, 2, 3, 4, 5)
```
In [184]:
```
tup[0]
```
Out[184]:
```
1
```

Editing Tuples

You can not edit tuples.

In [185]:
```
tup[0] = 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-185-874559a0c62a> in <module>
----> 1 tup[0] = 2

TypeError: 'tuple' object does not support item assignment
```

Tuple Operations

In [186]:
```
tup2 = ("Naimesh",'AL','KYLE','Connor')
```

In [187]:
```
tup + tup2
```

Out[187]:
```
(1, 2, 3, 4, 5, 'Naimesh', 'AL', 'KYLE', 'Connor')
```

In [188]:
```
# You can not delete specific values in tuples
del tup[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-188-53055a61f5a2> in <module>
      1 # You can not delete specific values in tuples
----> 2 del tup[0]

TypeError: 'tuple' object doesn't support item deletion
```

In [189]:
```
#You can delete an entire tuple
del tup2
```

In [190]:
```
tup2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-190-9d6435f7c59b> in <module>
----> 1 tup2

NameError: name 'tup2' is not defined
```

Dictionaries

Python dictionaries hold key value pairs. They are different than lists as dictionaries can not be indexed. You access values within them via the key. Key and values are separated by colons and commas. Keys are always uniques within a dictionary.

Create a Dictionary

In [191]:
```
regions = {
    'C' : 'California'
    ,'G' : 'Georgia'
    ,'P' : 'Pennsylvania'
    ,'M' : 'Minnesota'
}


regions['N'] = 'Monterrey'
```

In [192]:
```
regions
```

Out[192]:
```
{'C': 'California',
 'G': 'Georgia',
 'P': 'Pennsylvania',
 'M': 'Minnesota',
 'N': 'Monterrey'}
```

In [193]:
```
regions['C']
```

Out[193]:
```
'California'
```

Editing Dictionaries

Dictionaries can not have multiple keys. The keys need to be unique.

In [194]:
```
regions['C'] = 'Cali'
print(regions)
```

```
{'C': 'Cali', 'G': 'Georgia', 'P': 'Pennsylvania', 'M': 'Minnesota', 'N': 'Monterrey'}
```

Deleting Dictionaries

You can delete a specific key value pair in dictionaries

In [195]:
```
#Notice c key does not exist anymore
del regions['C']
regions
```

Out[195]:
```
{'G': 'Georgia', 'P': 'Pennsylvania', 'M': 'Minnesota', 'N': 'Monterrey'}
```

In [196]:
```
#notice Region does not exist anymore
del regions
regions
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-196-fd209be63dff> in <module>
      1 #notice Region does not exist anymore
      2 del regions
----> 3 regions

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

Python Functions

Functions are a set of code or instructions that only run when they are called. We can pass different variables into a function as parameters and return multiple values. Think of functions as a box that takes in parameters and returns results.

Function Basics

In [7]:
```
def basic_fun():
    print("Hello World")
```

Notice when we run the above cell nothing happens. This is because the function has been initialized but never called. Below we will call the function to see the results.

In [8]:
```
basic_fun()
```

```
Hello World
```

Functions can also take in parameters instead of saying hello world, we can pass in a name.

In [9]:
```
def basic_fun(name):
    print(f"Hello {name}")  #<--- - -standard format to include variables into strings
```

In [10]:
```
basic_fun('Naimesh')
```

```
Hello Naimesh
```

If we do not pass in a parameter, functions will fail unless there is a default value set.

In [12]:
```
basic_fun()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-f76de505107c> in <module>
----> 1 basic_fun()

TypeError: basic_fun() missing 1 required positional argument: 'name'
```

In [13]:
```
def basic_fun(name = "World"):
    print(f"Hello {name}")
```

In [14]:
```
basic_fun()
```

```
Hello World
```

In [15]:
```
basic_fun('Naimesh')
```

```
Hello Naimesh
```

Functions can take different types of variables like list, dataframes, dictionaries etc.

In [17]:
```
def basic_fun(name = []):
    for x in name:
        print(x)
```

In [19]:
```
basic_fun(['Al','Naimesh','Kyle'])
```

```
Al
Naimesh
Kyle
```

In [ ]:
```
basic_fun()
```

Functions can return more than one value, in a tuple form.

In [28]:
```
def basic_func(num = 0):
    z = num * 2
    y = num * 5
    x = num * 10
    return x,y,z
```

In [31]:
```
basic_func(5)
```

Out[31]:
```
(50, 25, 50)
```

Lambda Functions

A lambda function can take many arguments but can only return one expression. This is a quick way to write single line functions if you need them.

In [36]:
```
basic_fun  = lambda a : a * 5
basic_fun (5)
```

Out[36]:
```
25
```

Functions can also return functions

In [37]:
```
def myfunc(n):
    return lambda a : a * n

mydoubler = myfunc(2)  #<--- Setting the value of n, and returning lambda function
mydoubler
print(mydoubler(11))
```

```
22
```

Conclusion

This is a brief introduction to functions, we will use them lightly during our sessions, but it is important to understand how they work. As you get better at coding in Python, a lot of your code will start transitioning towards them. Writing functions allows us to reuse parts of our code for different tasks.

Python Logical Statements & Loops

Operators

Arithmetics Operators

In [2]:
```
a = 5
b= 3
print(a+ b)  #Addition
print(a-b)   #Subtraction
print(a*b)   #Multiplication
print(a**b)  #Exponent
print(a/b)   #Division
print(a//b)  #remainder
print(a%b)   #full values (Modulus)
```

```
8
2
15
125
1.6666666666666667
1
2
```

Comparison Operators

In [3]:
```
print(a==b)
print(a!= b)
print(a>b)
print(a<b)
print(a>=b)
print(a<=b)
```

```
False
True
True
False
True
False
```

Assignment Operators

In [4]:
```
c = a + b
print(c)
c += a #Same as c = c+a
print(c)
c -= a
print(c)
```

```
8
13
8
```

Logical Statements

Logical statements are available in many programming languages, they allow us to provide logic within code. They allows the computer to make decisions based on a set of criterias.

If Elif Else

Notice the indentation and semicolon.

In [5]:
```
if a == 6:
    print("A is 5")
else:
    print("A is not 5")
```

```
A is not 5
```

In [6]:
```
if a == 5:
    print("A is 5")
else:
    print("A is not 5")
```

```
A is 5
```

In [7]:
```
if a < 0:
    print("A is negative")
elif a == 0:
    print("A is Zero")
else:
    print("A is greater than zero")
```

```
A is greater than zero
```

Inline If & If else

In [8]:
```
print("A is greater than zero") if a > 0 else  print("A is not greater than zero")
```

```
A is greater than zero
```

In [9]:
```
print("A is greater than six") if a > 6 else  print("A is not greatrr than six")
```

```
A is not greatrr than six
```

Python Loops

Loops are an important part of any programming language. Often times we need to iterate over multiple rows, objects, or models to get to our desired results. There are two types of loops available in Python For and While Loop. We will review both below.

For Loop

In [29]:
```
val = ['Al','Naimesh','Kyle']
for v in val:
    print(v)
```

```
Al
Naimesh
Kyle
```

In [2]:
```
for x in range(1,11):
    print(x)
```

```
1
2
3
4
5
6
7
8
9
10
```

Inline For Loops

In [30]:
```
doubles = []
for v in val:
    doubles.append(v*2)
doubles
```

Out[30]:
```
['AlAl', 'NaimeshNaimesh', 'KyleKyle']
```

In [31]:
```
[v*2 for v in val]
```

Out[31]:
```
['AlAl', 'NaimeshNaimesh', 'KyleKyle']
```

While Loops

In [32]:
```
i = 1
while i < 6:
    print(i)
    i += 1
```

```
1
2
3
4
5
```

Break & Continue

- Break - Allows us to completely exit a loop if a condition is met

In [38]:
```
for v in val:
    if v == 'Al':
        print("Breaking")
        break
    else:
        print(v)g
```

```
Breaking
```

- Continue/Pass - Allows us to move on to the next iteration if a condition is met.

In [39]:
```
for v in val:
    if v == 'Al':
        continue
    else:
        print(v)
```

```
Naimesh
Kyle
```

In [40]:
```
for v in val:
    if v == 'Al':
        pass
    else:
        print(v)
```

```
Naimesh
Kyle
```

Conclusion

Operators, conditional statements, and loops are important concepts we need to understand to be efficient programmers in Python. In most situations you will be using some combinations of these to do the task at hand. They allow us to make logical decisions that we would normally take, but now programmatically.

Reading Data

Below are techniques we can use to read in the most common file types within Uline. Some key things to note, Python has default arguments for most functions. For example when using **read_csv**, we generally don't need to define any of the parameters. For **read_excel**, if we do not define a sheet name, the function will default to the first sheet. The best way to see what the default parameters are set to is to use the help function by pressing **shift tab** inside the function. Getting data directly from SQL is the route we generally want to take as we can write our SQL scripts directly on SSMS and import the sql script into Python.

Reading Data From Files ¶

Loading Packages

In [2]:
```
import pandas as pd
```

Reading Data From CSV

In [3]:
```
df = pd.read_csv('Data\Sample.csv')
df.head()
```

Out[3]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
0	S-4783	11.25x8.75x12 275# Box 25/500	A	C1	1475
1	S-4784	12x6x6 275# Box 25/500	A	C1	1650
2	S-18180	6x6x6 275# Dw Box 15/675	A	C1	5745
3	S-18181	11.25x8.75x6 275# Dw Box 15/450	A	C1	1800
4	S-18182	12x10x10 275# Dw Box 15/300	A	C1	1995

Reading Data From Excel Files

In [4]:
```
df[df['ItemNum'].isin(['S-4125','S-4783'])]
```

Out[4]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
0	S-4783	11.25x8.75x12 275# Box 25/500	A	C1	1475
182	S-4783	11.25x8.75x12 275# Box 25/500	A	C2	2250

In [5]:
```python
df = pd.read_excel('Data\Sample.xlsx', sheet_name='Sample')
df.head(2)
```

Out[5]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
0	S-4783	11.25x8.75x12 275# Box 25/500	A	C1	1475
1	S-4784	12x6x6 275# Box 25/500	A	C1	1650

Reading Data From SQL Server

Loading Packages

In [10]:
```python
!pip install pypyodbc "C:\Uline-Python-master\packages\pypyodbc-1.3.5.2.zip"
```

```
Processing c:\uline-python-master\packages\pypyodbc-1.3.5.2.zip
Requirement already satisfied: pypyodbc in c:\miniconda3\lib\site-packages (1.3.4)
Requirement already satisfied: setuptools in c:\miniconda3\lib\site-packages (from pypyo
dbc==1.3.4) (40.6.3)
```

In [7]:
```python
import pypyodbc
```

Reading Data

In [8]:
```python
cnxn = pypyodbc.connect(r"Driver={ODBC Driver 11 for SQL Server};"
                        "Server=WK-SQL2;"
                        "Database=UlineReporting;"
                        "Trusted_Connection=yes"
                       )

query = open('Data\Ses1Q1.sql', 'r')
df = pd.read_sql(query.read(), cnxn)
cnxn.close()
df.head()
```

Out[8]:

	itemnum	itemdesc	actvycode	whsenum	availqty
0	S-4896	18x12x8 275# Dw Box 15/180	A	C1	585.0
1	S-15035	12x10x8 275# Box 25/500	A	C1	2550.0
2	S-15036	12x12x10 275# Dw Box 15/150	A	C1	345.0
3	S-15037	13x13x13 275# Dw Box 15/180	A	C1	435.0
4	S-15041	24x16x16 275# Dw Box 10/90	A	C1	220.0

Above in the connect method we for the first time see "r" being used with a string. In Python there are commands associated with certain codes, they usually start with /. For example when we type / in a string, it will assume that as a new line. To avoid it interpreting string anything other then what they are, we will type r in front of it. This tells Python read the string as a raw string.

Writing Data

In Python we can write data into many different types. We will briefly go over the csv file type, but you can also write to excel using to_excel method for a dataframe.

In [9]:
```
df.to_csv("Data\TestWrite.csv")
```

Data Frames

Data frame is an object that is introduced by pandas. It is a composition of lists. You already have some examples of this in the reading data section. The methods discussed here will be the most important methods for you to remember.

Below is a link to all the methods that are available to a data frame. We will discuss some of the ones that are commonly used.

https://pandas.pydata.org/pandas-docs/stable/reference/frame.html (https://pandas.pydata.org/pandas-docs/stable/reference/frame.html)

In [3]:
```
import pandas as pd
path = r"C:\Uline-Python-master\2.Reading.Writing Data & EDA"  #notice we are utilizing
 r so read the string as raw and not interpret \ as a new line.
df = pd.read_csv(path+'\Data\Sample.csv')
```

Lets take a look at the first 10 rows of the dataframe

In [25]:
```
df.head(10)
```

Out[25]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
0	S-4783	11.25x8.75x12 275# Box 25/500	A	C1	1475
1	S-4784	12x6x6 275# Box 25/500	A	C1	1650
2	S-18180	6x6x6 275# Dw Box 15/675	A	C1	5745
3	S-18181	11.25x8.75x6 275# Dw Box 15/450	A	C1	1800
4	S-18182	12x10x10 275# Dw Box 15/300	A	C1	1995
5	S-18183	14x14x6 275# Dw Box 15/180	A	C1	1815
6	S-4730	18x18x18 275# Box 10/120	A	C1	290
7	S-11253	16x16x12 275# Dw Box 10/90	A	C1	580
8	S-4697	20x20x20 275# Box 10/120	A	C1	1100
9	S-4786	14x14x14 275# Dw Box 15/90	A	C1	525

Lets take a look at the last 10 rows of the dataframe

In [26]:
```
df.tail(10)
```

Out[26]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
280	S-4801	24x18x12 275# Box 15/120	A	C2	3885
281	S-4603	10x10x10 275# Box 25/500	A	C2	7750
282	S-15062	48x24x12 275# Dw 90/Bale	A	C2	351
283	S-4591	8x8x8 275# Box 25/750	A	C2	3375
284	S-4996	20x20x12 275# Dw Box 10/90	A	C2	160
285	S-4958	24x18x18 275# Dw Box 10/90	A	C2	2800
286	S-4725	18x12x12 275# Dw Box 15/180	A	C2	3615
287	S-4775	30x20x20 275# Dw Box 90/Bale	A	C2	1155
288	S-4794	20x16x14 275# Box 15/120	A	C2	3270
289	S-4968	18x18x12 275# Dw Box 10/90	A	C2	2110

Lets take a look at the data types of our available columns. To do this we will used the **dtypes** attribute. We can see that field that are string in nature are returned as a object data type while numeric ones are returns with their numeric type.

In [3]:
```
df.dtypes
```

Out[3]:
```
ItemNum      object
ItemDesc     object
ActvyCode    object
WhseNum      object
AvailQty      int64
dtype: object
```

Lets use the **describe** method to give us statistics about our data frame. Since most of the statistics require numeric columns, this function by default will only show you the numeric columns. To include the object columns, we need to set **include** to all. You will notice there are a few **NaN's** being displayed in the results below. **Python interprets Null's as either Nulls or NaN's** depending on which library we use. Both represent the same thing. In the results below the NaN's are used to let the use know we can not do mathematical operations on object columns, hence certain measures will be displayed as NaN's.

In [5]:
```python
df.describe(include = 'all')
```

Out[5]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
count	290	290	290	290	290.000000
unique	182	182	2	2	NaN
top	S-15062	13x13x13 275# Dw Box 15/180	A	C1	NaN
freq	2	2	286	182	NaN
mean	NaN	NaN	NaN	NaN	1135.493103
std	NaN	NaN	NaN	NaN	1489.785582
min	NaN	NaN	NaN	NaN	17.000000
25%	NaN	NaN	NaN	NaN	282.000000
50%	NaN	NaN	NaN	NaN	550.000000
75%	NaN	NaN	NaN	NaN	1407.500000
max	NaN	NaN	NaN	NaN	9300.000000

Data Frame Indexing

There are two methods we can use to index data frame: loc and iloc

Notice the difference in how loc and iloc behave.

- loc has format **[row selection][column selection]** where the first is row and second is column. Both of those can be strings if your index and columns are strings
- iloc has format **[rows,columns]**, both the rows and columns need to be numeric.
- using **colon :** tells python to get all the values

Python Pandas Selections and Indexing

.iloc selections - position based selection

data.iloc[row selection , column selection]

Integer list of rows: [0,1,2]
Slice of rows: [4:7]
Single values: 1

Integer list of columns: [0,1,2]
Slice of columns: [4:7]
Single column selections: 1

loc selections - position based selection

data.loc[<row selection>],[<column selection>]

Index/Label value: 'john'
List of labels: ['john', 'sarah']

Named column: 'first_name'
List of column names: ['first_name', 'age']

Column Selection

There is a new string code we are suing here, notice \n creates a new line.

In [9]:
```python
# Lets try to select the first columns in the Data Frame, 3 ways to do this
print('Using Column Selector \n')
print(df['ItemNum'].head())


print('Using iloc \n')
print(df.iloc[:,0].head())   #Notice here we used colon to identify all rows, and 0 to i
dentify the 0th column.


print('Using Loc \n')
print(df.loc[:]['ItemNum'].head()) #Notice here we used colon to identify all rows, and
 the column name to do the column selection
```

```
Using Column Selector 

0     S-4783
1     S-4784
2    S-18180
3    S-18181
4    S-18182
Name: ItemNum, dtype: object
Using iloc 

0     S-4783
1     S-4784
2    S-18180
3    S-18181
4    S-18182
Name: ItemNum, dtype: object
Using Loc 

0     S-4783
1     S-4784
2    S-18180
3    S-18181
4    S-18182
Name: ItemNum, dtype: object
```

Row Selection

In [11]:
```
print(df.iloc[0,:])
print('\n')
print(df.loc[0])
```

```
ItemNum                            S-4783
ItemDesc       11.25x8.75x12 275# Box 25/500
ActvyCode                               A
WhseNum                                C1
AvailQty                             1475
Name: 0, dtype: object


ItemNum                            S-4783
ItemDesc       11.25x8.75x12 275# Box 25/500
ActvyCode                               A
WhseNum                                C1
AvailQty                             1475
Name: 0, dtype: object
```

List of Column Names

In [14]:
```
#lets try and select all column names in a data frame
df.columns
```

Out[14]:
```
Index(['ItemNum', 'ItemDesc', 'ActvyCode', 'WhseNum', 'AvailQty'], dtype='object')
```

Data Frame Index

In [12]:
```
#Lets try and select all the indexes in the data frame
df.index
```

Out[12]:
```
RangeIndex(start=0, stop=290, step=1)
```

Data Frame Sorting

In [6]:
```
df.sort_values(by = ['ItemNum','WhseNum'], ascending = [False,True]).head(5)
```

Out[6]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
171	S-4996	20x20x12 275# Dw Box 10/90	A	C1	1960
284	S-4996	20x20x12 275# Dw Box 10/90	A	C2	160
85	S-4995	20x16x16 275# Dw Box 10/90	A	C1	160
228	S-4995	20x16x16 275# Dw Box 10/90	A	C2	770
176	S-4968	18x18x12 275# Dw Box 10/90	A	C1	390

Data Frame Filtering

Single Condition Filtering

In [13]:
```
#Filtering data frame to a specific condition
print(df[df['ItemNum'] == 'S-18924'])

print('\n')
df[df['ActvyCode']== 'A'].head()
```

```
     ItemNum                   ItemDesc ActvyCode WhseNum  AvailQty
35   S-18924  24x6x6 275# Dw Box 15/360         A      C1      1020
```

Out[13]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
0	S-4783	11.25x8.75x12 275# Box 25/500	A	C1	1475
1	S-4784	12x6x6 275# Box 25/500	A	C1	1650
2	S-18180	6x6x6 275# Dw Box 15/675	A	C1	5745
3	S-18181	11.25x8.75x6 275# Dw Box 15/450	A	C1	1800
4	S-18182	12x10x10 275# Dw Box 15/300	A	C1	1995

Multiple Selection Criterias

In [14]:
```
df[df['ItemNum'].isin(['S-4783','S-18180'])]
```

Out[14]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
0	S-4783	11.25x8.75x12 275# Box 25/500	A	C1	1475
2	S-18180	6x6x6 275# Dw Box 15/675	A	C1	5745
182	S-4783	11.25x8.75x12 275# Box 25/500	A	C2	2250

In [15]:
```
df[df['AvailQty'].between(10,200)].head()
```

Out[15]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
10	S-12613	30x24x12 275# Dw Box 10/90	A	C1	80
18	S-19775	36x18x12 275# Dw Box 90/Bale	A	C1	96
21	S-14231	18x14x14 275# Dw Box 15/90	A	C1	180
23	S-14233	24x24x36 275# Dw Box 90/Bale	A	C1	160
24	S-14234	36x24x12 275# Dw Box 90/Bale	A	C1	134

In [16]:
```
df[(df['AvailQty'].between(10,200)) & (df['ItemNum'] == 'S-12613')]
```

Out[16]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
10	S-12613	30x24x12 275# Dw Box 10/90	A	C1	80

Data Frame Aggregation

- Step 1: Create a Dictionary, where the key is a columns you want to aggregate and value is the aggregation you want to perform.
- Step 2 Create a list with the columns you want to group by
- Step 3: Apply the groupby method for the list you created in step 2
- Step 4: Apply the agg method, with the dictionary you created in step one

All the above steps can be done in one steps

In [12]:
```
tot = {
    'AvailQty': ['mean','sum']
   ,'ItemNum' : ['count']
}

gbcolumns = ['ActvyCode','WhseNum']

df2 = df.groupby(by = gbcolumns).agg(tot)
df2

#When we do the group by, the groped columns get converted to indexes and can no longer
 be referenced by column selection.
```

Out[12]:

		AvailQty		ItemNum
		mean	sum	count
ActvyCode	WhseNum			
A	C1	720.376404	128227	178
	C2	1830.518519	197696	108
I	C1	842.500000	3370	4

Selecting Newly Created Columns

In [13]:
```
df2.columns #Notice The multi Index and codes
```

Out[13]:
```
MultiIndex(levels=[['AvailQty', 'ItemNum'], ['count', 'mean', 'sum']],
           codes=[[0, 0, 1], [1, 2, 0]])
```

In [14]:
```
df2.columns.values #Notice values gives us a array of tuples with the column and aggrega
tion
```

Out[14]:
```
array([('AvailQty', 'mean'), ('AvailQty', 'sum'), ('ItemNum', 'count')],
      dtype=object)
```

In [15]:
```
df2['AvailQty'][['mean','sum']]
```

Out[15]:

		mean	sum
ActvyCode	**WhseNum**		
A	**C1**	720.376404	128227
	C2	1830.518519	197696
I	**C1**	842.500000	3370

In [16]:
```
    # Flattening Columns. Here we are using an inline for statement to combine the value
s of the tuples above.
['_'.join(col) for col in df2.columns.values]
```

Out[16]:
```
['AvailQty_mean', 'AvailQty_sum', 'ItemNum_count']
```

In [17]:
```
df2.columns  = ['_'.join(col) for col in df2.columns.values]
```

Selecting Columns from aggregated data sets

In [18]:
```
#Notice The index is always showing
df2['AvailQty_mean']
```

Out[18]:
```
ActvyCode  WhseNum
A          C1          720.376404
           C2         1830.518519
I          C1          842.500000
Name: AvailQty_mean, dtype: float64
```

In [36]:
```python
#Notice we can not select the index columns anymore!
df2['ActvyCode']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
C:\miniconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, metho
d, tolerance)
   2656             try:
-> 2657                 return self._engine.get_loc(key)
   2658             except KeyError:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_
item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_
item()

KeyError: 'ActvyCode'

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<ipython-input-36-d3104fc8f148> in <module>
      1 #Notice we can not select the index columns anymore!
----> 2 df2['ActvyCode']

C:\miniconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   2925             if self.columns.nlevels > 1:
   2926                 return self._getitem_multilevel(key)
-> 2927             indexer = self.columns.get_loc(key)
   2928             if is_integer(indexer):
   2929                 indexer = [indexer]

C:\miniconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, metho
d, tolerance)
   2657                 return self._engine.get_loc(key)
   2658             except KeyError:
-> 2659                 return self._engine.get_loc(self._maybe_cast_indexer(key))
   2660         indexer = self.get_indexer([key], method=method, tolerance=tolerance)
   2661         if indexer.ndim > 1 or indexer.size > 1:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_
item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_
item()

KeyError: 'ActvyCode'
```

In [37]:
```python
#But we can select them using loc! As we can select lables through loc.
df2.loc['A']
df2.loc['A','C2']
```

Out[37]:
```
AvailQty_mean      1830.518519
AvailQty_sum     197696.000000
ItemNum_count       108.000000
Name: (A, C2), dtype: float64
```

In [38]:
```python
# We can reset  the index if need be, after we should be able to reference them accordin
gly
df2.reset_index()['ActvyCode']
```

Out[38]:
```
0    A
1    A
2    I
Name: ActvyCode, dtype: object
```

Data Frame Editing

Column Name Editing

If you have columns or indexes that have characters like # in them, it would be a good idea to rename them.

In [24]:
```python
df.columns = ['ItemNum', 'ItemDesc', 'ActvyCode', 'WhseNum', 'AvailQty']
df.columns
```

Out[24]:
```
Index(['ItemNum', 'ItemDesc', 'ActvyCode', 'WhseNum', 'AvailQty'], dtype='object')
```

Index Name Editing

In [23]:
```python
df2.index.names = ['ActvCode', 'Whse']
df2.index
```

Out[23]:
```
MultiIndex(levels=[['A', 'I'], ['C1', 'C2']],
           codes=[[0, 0, 1], [0, 1, 0]],
           names=['ActvCode', 'Whse'])
```

Data frame editing is allowed if we have the index of the row that needs to be edited. We can do this in many ways. Examples are below.

iterrows is a common method used to not only get the index of the row, but also its values

IterRows Indexing

In [29]:
```python
for index, row in df.head().iterrows():
    df.loc[index ,'AvailQty']  = 9999

print(df.head())
```

```
   ItemNum                       ItemDesc ActvyCode WhseNum  AvailQty
0   S-4783    11.25x8.75x12 275# Box 25/500         A      C1      9999
1   S-4784           12x6x6 275# Box 25/500         A      C1      9999
2  S-18180          6x6x6 275# Dw Box 15/675         A      C1      9999
3  S-18181  11.25x8.75x6 275# Dw Box 15/450         A      C1      9999
4  S-18182       12x10x10 275# Dw Box 15/300         A      C1      9999
```

In [30]:
```python
for index, row in df.head().iterrows():
    df.iloc[index, 4] = 999

print(df.head())
```

```
   ItemNum                       ItemDesc ActvyCode WhseNum  AvailQty
0   S-4783    11.25x8.75x12 275# Box 25/500         A      C1       999
1   S-4784           12x6x6 275# Box 25/500         A      C1       999
2  S-18180          6x6x6 275# Dw Box 15/675         A      C1       999
3  S-18181  11.25x8.75x6 275# Dw Box 15/450         A      C1       999
4  S-18182       12x10x10 275# Dw Box 15/300         A      C1       999
```

Assigning Values without iterrow

In [39]:
```python
#Assigning a Single value
df.iloc[0:5, 4] = 99
df.head()
```

Out[39]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
0	S-4783	11.25x8.75x12 275# Box 25/500	A	C1	99
1	S-4784	12x6x6 275# Box 25/500	A	C1	99
2	S-18180	6x6x6 275# Dw Box 15/675	A	C1	99
3	S-18181	11.25x8.75x6 275# Dw Box 15/450	A	C1	99
4	S-18182	12x10x10 275# Dw Box 15/300	A	C1	99

In [40]:
```python
#Assigning a list of values
tst = [9,9,9,9,9]
df.iloc[0:5, 4] = tst
df.head()
```

Out[40]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
0	S-4783	11.25x8.75x12 275# Box 25/500	A	C1	9
1	S-4784	12x6x6 275# Box 25/500	A	C1	9
2	S-18180	6x6x6 275# Dw Box 15/675	A	C1	9
3	S-18181	11.25x8.75x6 275# Dw Box 15/450	A	C1	9
4	S-18182	12x10x10 275# Dw Box 15/300	A	C1	9

Assigning values with a search criteria

In [41]:
```
#This will not work, because when we apply methods it gives up a 2nd df, while we need t
o edit the original one.
#To accomplish this we need to get the index of the rows we are trying to edit.

#The below example will not work
tst = [99,99,99,99,99]
df[df['ActvyCode'] == 'A'].head().iloc[0:5,4] = tst
df[df['ActvyCode'] == 'A'].head()
```

Out[41]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
0	S-4783	11.25x8.75x12 275# Box 25/500	A	C1	9
1	S-4784	12x6x6 275# Box 25/500	A	C1	9
2	S-18180	6x6x6 275# Dw Box 15/675	A	C1	9
3	S-18181	11.25x8.75x6 275# Dw Box 15/450	A	C1	9
4	S-18182	12x10x10 275# Dw Box 15/300	A	C1	9

In [34]:
```
#this will work
ind = df[df['ActvyCode'] == 'A'].head().index.values
df.loc[ind,'AvailQty'] = tst
df.head()
```

Out[34]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
0	S-4783	11.25x8.75x12 275# Box 25/500	A	C1	99
1	S-4784	12x6x6 275# Box 25/500	A	C1	99
2	S-18180	6x6x6 275# Dw Box 15/675	A	C1	99
3	S-18181	11.25x8.75x6 275# Dw Box 15/450	A	C1	99
4	S-18182	12x10x10 275# Dw Box 15/300	A	C1	99

In [35]:
```
df[df['WhseNum'] == 'C2'].head()
```

Out[35]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
182	S-4783	11.25x8.75x12 275# Box 25/500	A	C2	2250
183	S-4730	18x18x18 275# Box 10/120	A	C2	870
184	S-11253	16x16x12 275# Dw Box 10/90	A	C2	3570
185	S-4786	14x14x14 275# Dw Box 15/90	A	C2	4155
186	S-12613	30x24x12 275# Dw Box 10/90	A	C2	870

In [36]:
```
ind = df[df['WhseNum'] == 'C2'].head().index.values
df.loc[ind,'AvailQty'] = tst
df[df['WhseNum'] == 'C2'].head()
```

Out[36]:

	ItemNum	ItemDesc	ActvyCode	WhseNum	AvailQty
182	S-4783	11.25x8.75x12 275# Box 25/500	A	C2	99
183	S-4730	18x18x18 275# Box 10/120	A	C2	99
184	S-11253	16x16x12 275# Dw Box 10/90	A	C2	99
185	S-4786	14x14x14 275# Dw Box 15/90	A	C2	99
186	S-12613	30x24x12 275# Dw Box 10/90	A	C2	99

Python EDA (Graphing)

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

In [2]:
```
df = pd.read_csv('Data\Fraud.csv')
df.drop(['custnum', 'ordernum'], axis = 1, inplace = True)
```

In [3]:
```
df.head()
```

Out[3]:

	device	frtpct	ltdsales	ltdord	ordtype	timeblock	fbitemflg	carttime	invcamt	ipcountry	...	taxflg
0	Mobile	20%	0.0	0	INTERNET	6PM	0	49	1018.1	US	...	taxable
1	Mobile	20%	0.0	0	INTERNET	6PM	0	6	92.6	US	...	taxable
2	Mobile	20%	0.0	0	INTERNET	6PM	0	0	679.0	US	...	taxable
3	Mobile	20%	0.0	0	INTERNET	3PM	0	19	240.0	US	...	taxable
4	unknown	20%	0.0	0	PHONE ORDER	3PM	0	0	91.0	Other	...	taxable

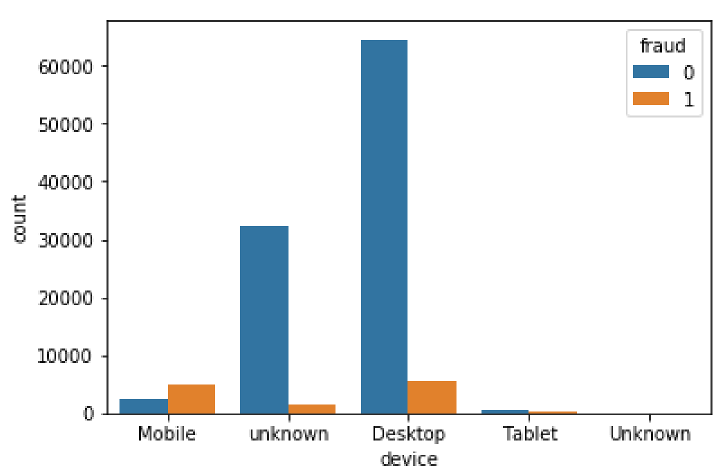
5 rows × 25 columns

What is the breakout of device types?

Below we are setting two attributes. One for the X label and second for the Hue. We are also referencing columns in a different way. As long as the column names do not have a space we can reference them using the below notation. Also notice we applied hue on the fraud column, this will split the data between the uniques values of the fraud column. We did not specify a Y because this is a counts plot.

In [4]:
```
sns.countplot(x = df.device, hue  = df.fraud)  #Notice unknows spelled differently, Shou
ld we combine it?
```

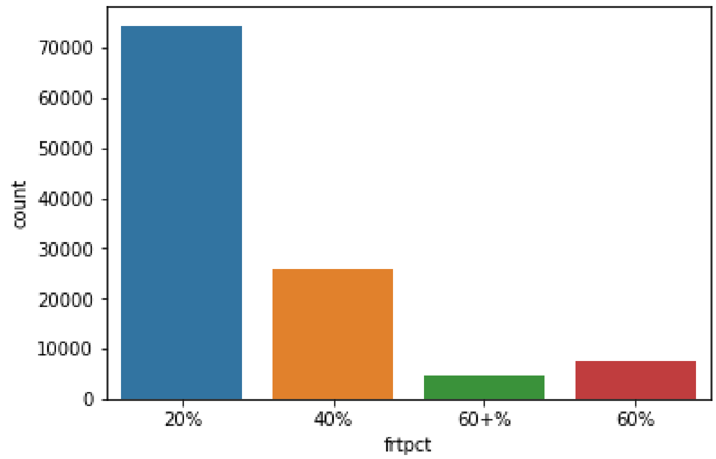
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x1f71c806978>



What is the breakout of frtpct?

```
In [5]: sns.countplot(x = df.frtpct)
```

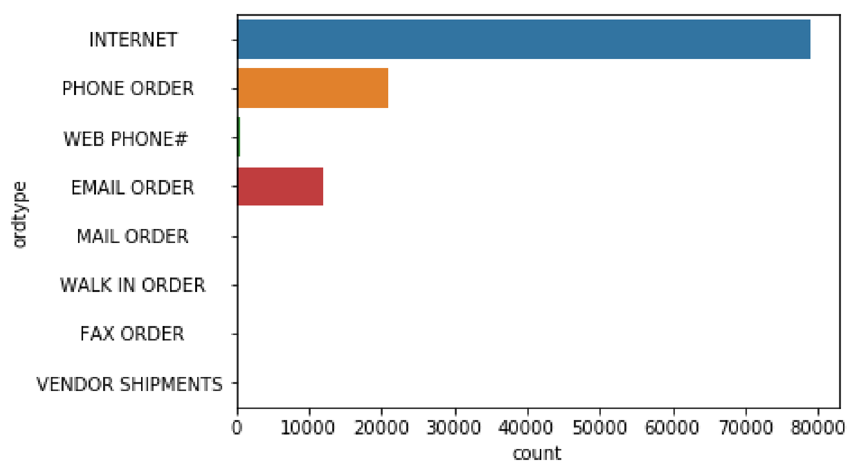
```
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x1f71e5056d8>
```



What is the breakout of OrderType?

```
In [6]: sns.countplot(y = df.ordtype)
```

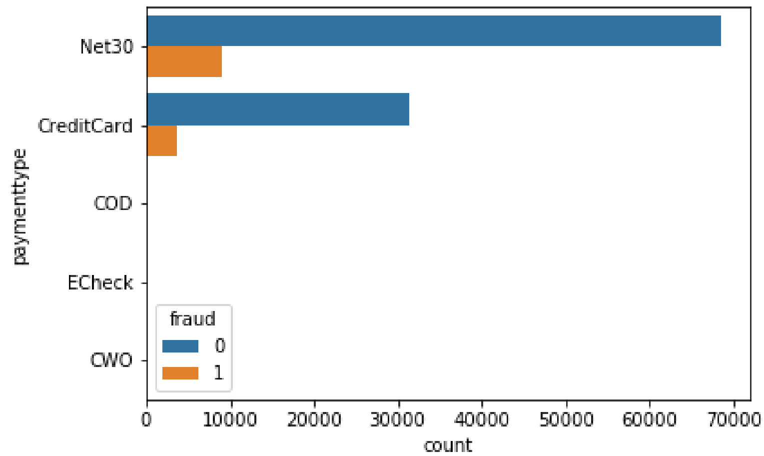
```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1f71e5701d0>
```



What is the breakout of payment type?

In [7]:
```
sns.countplot(y = df.paymenttype, hue = df.fraud)
```

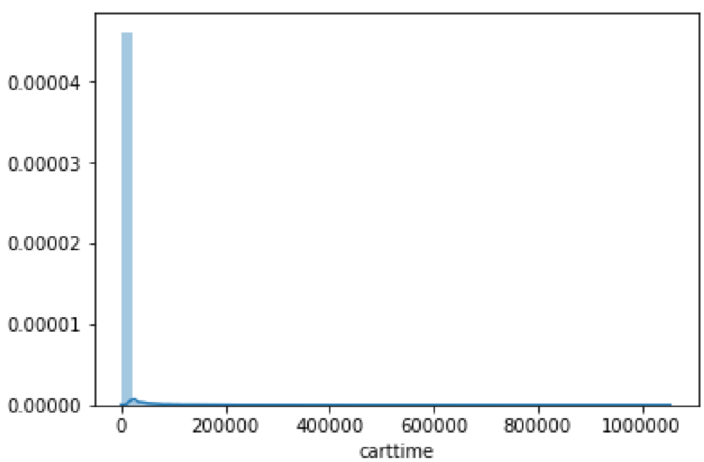
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1f71e5d33c8>



What is the distribution of CartTime?

In [8]:
```
sns.distplot(df.carttime)
```

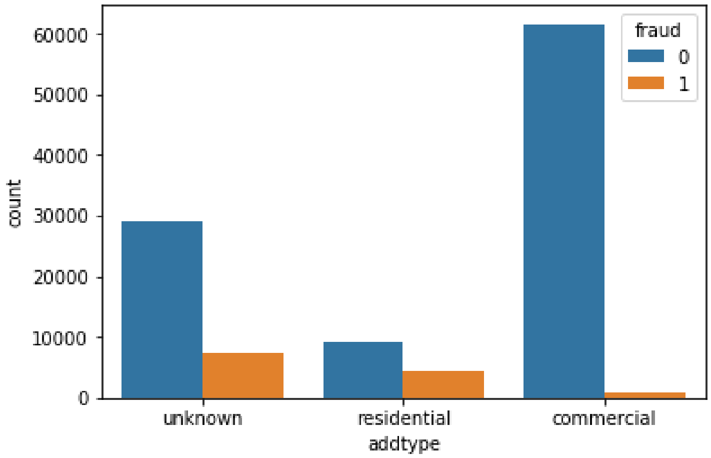
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1f71e669a58>



What is the breakout of add type?

In [9]:
```
sns.countplot(df.addtype, hue = df.fraud)
```

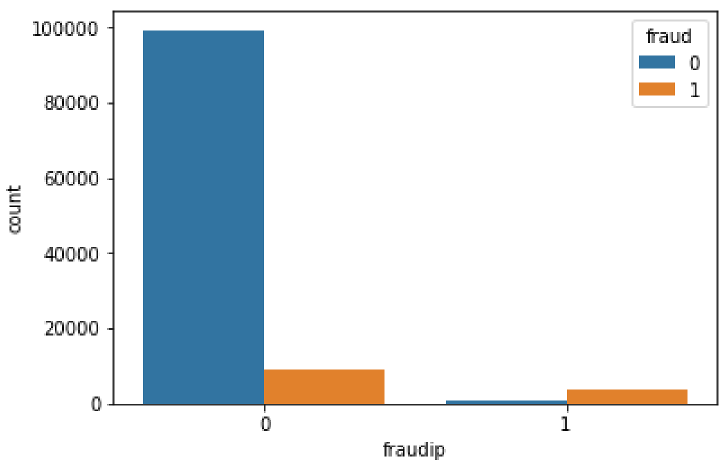
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1f71e745400>



What is the breakout of fraud id?

In [10]:
```
sns.countplot(df.fraudip, hue = df.fraud)
```

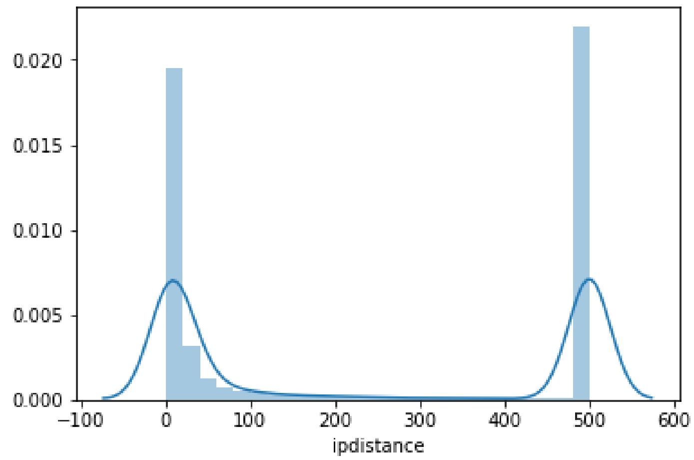
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1f71e7abda0>



What is the distribution of ip distance?

In [11]:
```
sns.distplot(df.ipdistance)
```

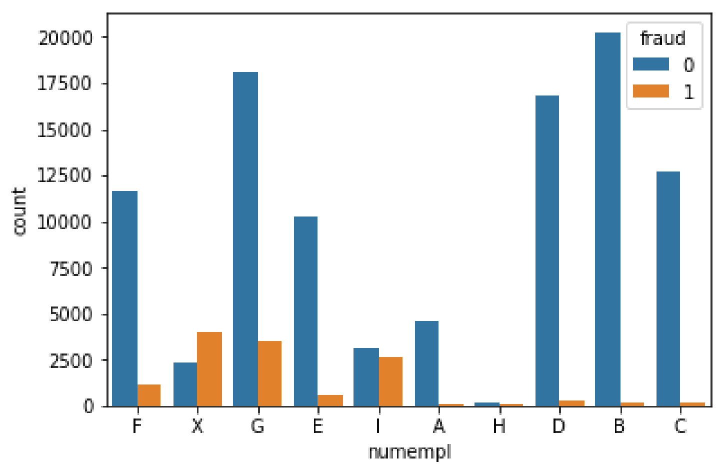
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1f71e801be0>



What is the breakout of number of employees?

In [12]:
```
sns.countplot(df.numempl, hue = df.fraud )
```

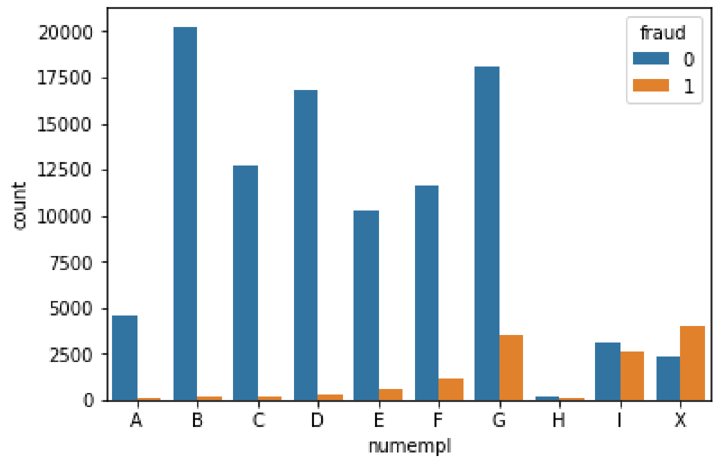
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1f71e8935c0>



In [15]:
```
## Lets Sort The Codes Ascendingly
sns.countplot(df['numempl'].sort_values(ascending=True), hue = df.fraud )

#This is the first time we are using the sort_values method, this allows us to sort valu
es within a df by ascendingly or descendingly.
```

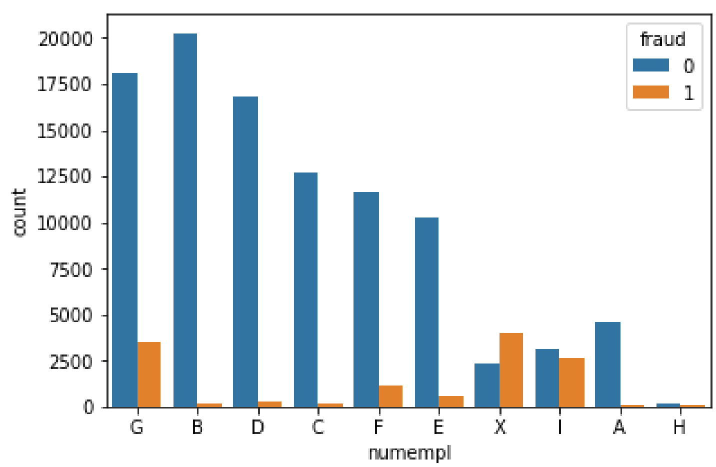
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1f71e9bc9e8>



In [17]:
```
## Lets Sort the Codes By Value Counts
sns.countplot(df['numempl'], hue = df.fraud, order = df['numempl'].value_counts().index
)

#This is the first time we are using the value_counts method, which takes a column and c
ounts the number of unique values. Notice it sorts from highest to lowest
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1f71ea4b240>



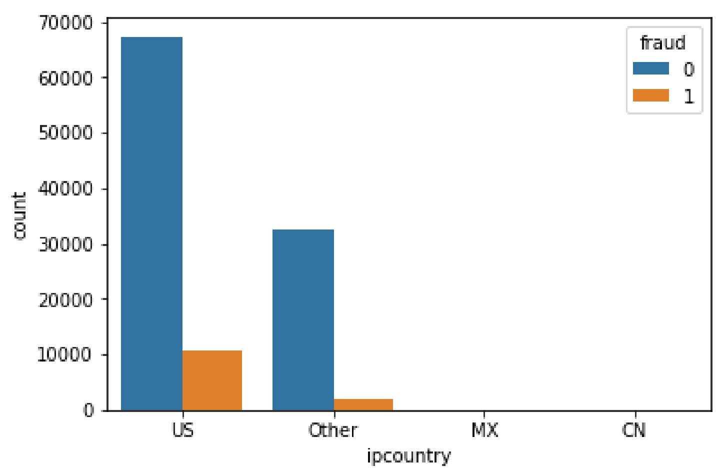
In [16]:
```
df['numempl'].value_counts()
```

Out[16]:
```
G    21610
B    20439
D    17044
C    12871
F    12834
E    10855
X     6291
I     5778
A     4579
H      280
Name: numempl, dtype: int64
```

What is the distribution of IP country?

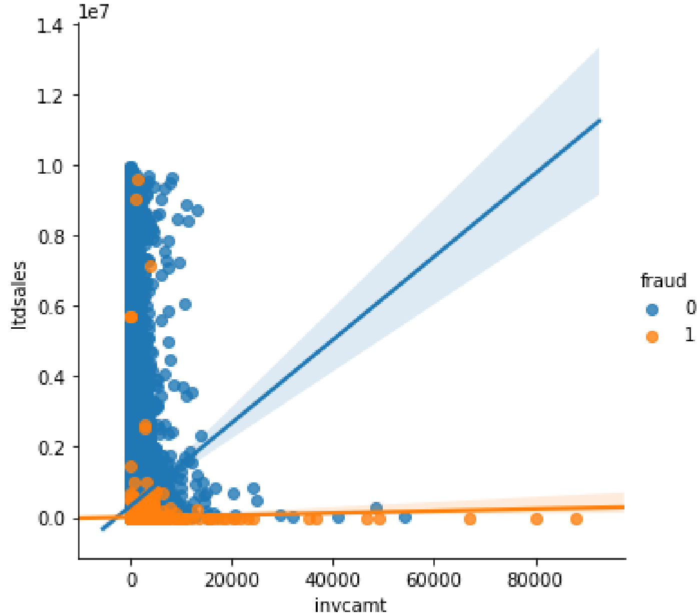
In [19]:
```
sns.countplot(df.ipcountry, hue = df.fraud)
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f71eaf5f98>
```



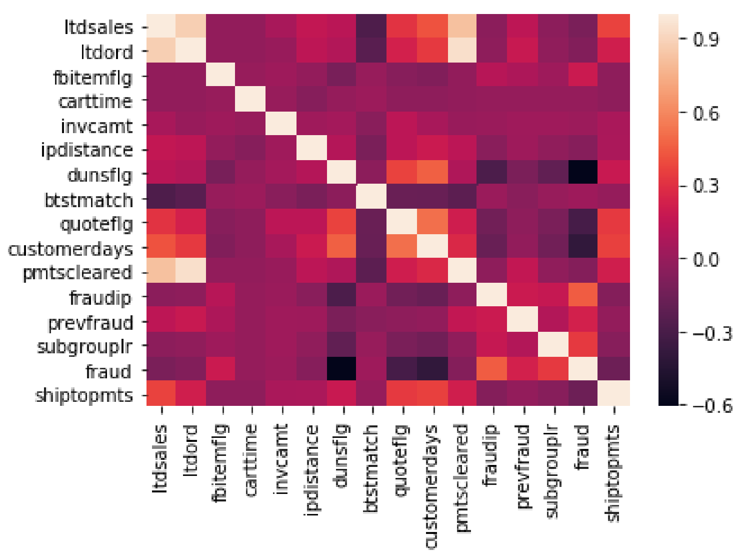
Is there a relationship between ltd sales and incvamt, when it relates to fraud?

In [20]:
```
sns.lmplot(y='ltdsales', x = 'invcamt', hue = 'fraud', data = df)
```
Out[20]: <seaborn.axisgrid.FacetGrid at 0x1f71eb70ac8>



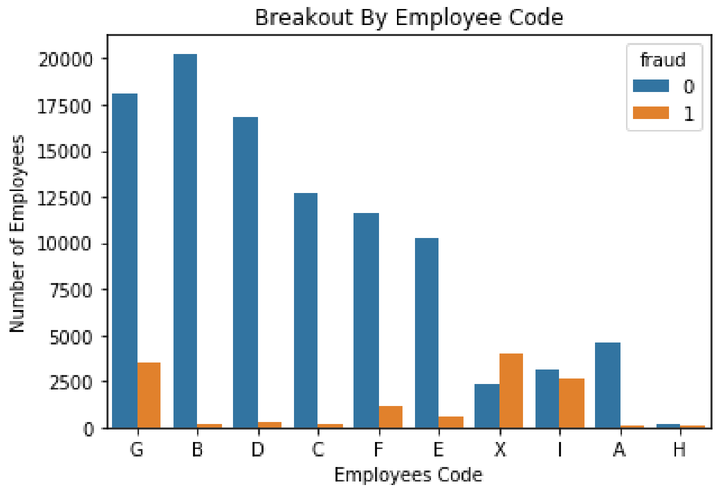
What is the correlation between variables?

In [21]:
```
sns.heatmap(df.corr())
```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1f71edbc7f0>



Adding titles and axis labels.

In [28]:
```
sns.countplot(df['numempl'], hue = df.fraud, order = df['numempl'].value_counts().index
)
plt.title("Breakout By Employee Code")
plt.xlabel("Employees Code")
plt.ylabel("Number of Employees")
plt.show()
```



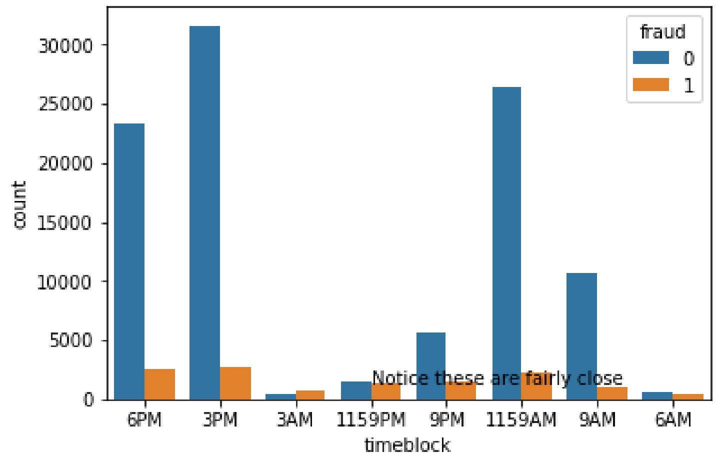
In [51]:
```
df[df['fraud'] == 1]['timeblock'].value_counts()
#3PM 34276
#3pm(1) 2664
#3pm(0) 31612
```

Out[51]:
```
3PM       2664
6PM       2532
1159AM    2283
9PM       1565
1159PM    1297
9AM       1025
3AM        791
6AM        423
Name: timeblock, dtype: int64
```

Adding text annotation

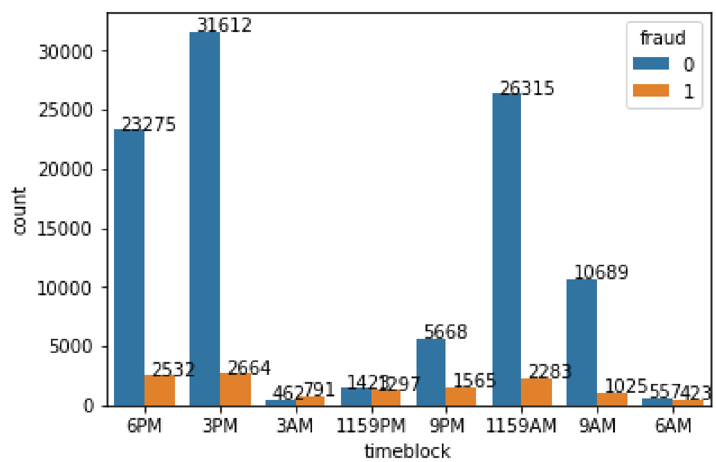
In [64]:
```
sns.countplot(df.timeblock, hue = df.fraud)
plt.annotate("Notice these are fairly close", xy = (3,1297 ) )
```

Out[64]: Text(3, 1297, 'Notice these are fairly close')



Adding data point annotation

In [76]:
```
ax  = sns.countplot(df.timeblock, hue = df.fraud)
for p in ax.patches:
    ax.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.08, p.get_height()+1))
```



Data Preparation

During this workbook we are going to discuss some important concepts that are essential to successfully building machine learning models. Considering that all models are mathematical operations, we need to convert every data point into a numeric field. To accomplish this this we will to convert each category, into a binary field. If we do not do this, we could have improper results.

Along with this we also need to consider numerical data transformations, primarily normalizations. You will be introduced to a new python package called scikit-learn. This is a widely used machine learning package in the field. It has a defined work flow pipeline that is intuitive to follow. The package is well documented. Link to their homepage is below.

[https://scikit-learn.org/stable/ (https://scikit-learn.org/stable/)](https://scikit-learn.org/stable/)

One Hot Encoding

One hot encoding is the technical term used to convert categorical columns into binary numerical columns. Below is the how the scikit-learn developers describe the requirements.

"The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are encoded using a one-hot (aka ‘one-of-K’ or ‘dummy’) encoding scheme. This creates a binary column for each category and returns a sparse matrix or dense array."

There are two terms here we do not know of. Sparse Matrix & Dense array. A sparse matrix is a matrix where most of the values are 0's. A dense matrix is where most of the values are non zeros.

[https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html)](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html)

In [40]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [41]:
```
df = pd.read_csv('Data\Fraud.csv')
```

In all ML processes, we need to determine,

- Which columns we are trying the predict
- Which columns should be removed
- Which are numerical columns
- Which are categorical columns
- Which are flag columns

We will use df.columns, df.describe, and df['columns name'].unique() to help use parse them.

In [42]:
```
df.describe()
```

Out[42]:

	custnum	ltdsales	ltdord	ordernum	fbitemflg	carttime	invcamt
count	1.125810e+05	1.125810e+05	112581.000000	1.125810e+05	112581.000000	1.125810e+05	112581.000000
mean	6.689032e+06	3.275062e+05	853.523943	5.381436e+07	0.012844	2.616138e+03	427.731354
std	5.344784e+06	1.078934e+06	3785.025709	4.184373e+07	0.112602	2.057020e+04	909.593000
min	3.000000e+00	0.000000e+00	0.000000	1.000022e+07	0.000000	0.000000e+00	1.200000
25%	1.556559e+06	1.480660e+03	7.000000	1.286362e+07	0.000000	0.000000e+00	85.000000
50%	5.636208e+06	2.657990e+04	86.000000	1.591999e+07	0.000000	0.000000e+00	208.000000
75%	1.196843e+07	1.628023e+05	385.000000	9.650303e+07	0.000000	6.000000e+00	484.450000
max	1.541459e+07	9.976137e+06	57555.000000	1.000000e+08	1.000000	1.054358e+06	88188.000000

In [43]:
```
df.columns
```

Out[43]:
```
Index(['custnum', 'device', 'frtpct', 'ltdsales', 'ltdord', 'ordernum',
       'ordtype', 'timeblock', 'fbitemflg', 'carttime', 'invcamt', 'ipcountry',
       'ipdistance', 'numempl', 'dunsflg', 'btstmatch', 'quoteflg', 'taxflg',
       'paymenttype', 'customerdays', 'pmtscleared', 'addtype', 'fraudip',
       'prevfraud', 'subgrouplr', 'fraud', 'shiptopmts'],
      dtype='object')
```

In [57]:
```
# In the EDA section we noticed that device had unknown twice one with a capital U and o
ne withought lets combine the two before moving forward.
df['device'] = df['device'].str.lower()

#We could do multiple other tranformations here. For example:
#Combining [OrderType, Paymenttype] that have low counts into other category
```

In [45]:
```
df['shiptopmts'].unique()
```

Out[45]:
```
array([   0,    3,    1, ...,  966, 1063,  954], dtype=int64)
```

In [46]:
```
y = ['fraud']
remove = ['custnum','ordernum']
flags = ['fbitemflg','dunsflg','quoteflg','btstmatch','fraudip','prevfraud']
cat_columns = ['device','frtpct','ordtype','timeblock','ipcountry','numempl','taxflg','p
aymenttype','addtype']
numeric_columns = ['ltdsales','ltdord','carttime','invcamt','ipdistance','customerdays',
'pmtscleared','shiptopmts','subgrouplr']
```

Let's one hot encode all the categorical columns

In scikit-learn, we build models using the objects provided in the library. We initiate the model with the fit method. If it is a transformation object we then transform the data. The reason why we fit and transform is because we can do similar transformation to unseen data. If you just transform you will loose all the transformation variables. If the object is a predictive algorithm then we can apply the predict method after fit.

In [47]:
```python
end = OneHotEncoder(sparse= False)
end.fit(df[cat_columns])
new_cat = end.transform(df[cat_columns])
new_cat  #Notice this returns an array. How go we get the column names and convert the a
rray to a dataframe?
```

Out[47]:
```
array([[0., 1., 0., ..., 0., 0., 1.],
       [0., 1., 0., ..., 0., 0., 1.],
       [0., 1., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.]])
```

In [48]:
```python
cat_names = end.get_feature_names(input_features = cat_columns)
cat_col_end = pd.DataFrame(new_cat, columns = cat_names)
cat_col_end.head()
```

Out[48]:

	device_desktop	device_mobile	device_tablet	device_unknown	frtpct_20%	frtpct_40%	frtpct_60%	frtpct_6
0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	
1	0.0	1.0	0.0	0.0	1.0	0.0	0.0	
2	0.0	1.0	0.0	0.0	1.0	0.0	0.0	
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	1.0	0.0	0.0	

5 rows × 48 columns

In [49]:
```python
cat_col_end.columns
```

Out[49]:
```
Index(['device_desktop', 'device_mobile', 'device_tablet', 'device_unknown',
       'frtpct_20%', 'frtpct_40%', 'frtpct_60%', 'frtpct_60+%',
       'ordtype_EMAIL ORDER     ', 'ordtype_FAX ORDER       ',
       'ordtype_INTERNET        ', 'ordtype_MAIL ORDER      ',
       'ordtype_PHONE ORDER     ', 'ordtype_VENDOR SHIPMENTS',
       'ordtype_WALK IN ORDER   ', 'ordtype_WEB PHONE#      ',
       'timeblock_1159AM', 'timeblock_1159PM', 'timeblock_3AM',
       'timeblock_3PM', 'timeblock_6AM', 'timeblock_6PM', 'timeblock_9AM',
       'timeblock_9PM', 'ipcountry_CN', 'ipcountry_MX', 'ipcountry_Other',
       'ipcountry_US', 'numempl_A', 'numempl_B', 'numempl_C', 'numempl_D',
       'numempl_E', 'numempl_F', 'numempl_G', 'numempl_H', 'numempl_I',
       'numempl_X', 'taxflg_notax', 'taxflg_taxable', 'paymenttype_COD',
       'paymenttype_CWO', 'paymenttype_CreditCard', 'paymenttype_ECheck',
       'paymenttype_Net30', 'addtype_commercial', 'addtype_residential',
       'addtype_unknown'],
      dtype='object')
```

Min Max Scaler

Notice our numerical columns are in many difference scales, in situations likes this it is highly recommended to normalize the columns. To do this we will use the MinMaxScaler object from the scikit learn package. Documentation below.

In [50]:
```
df[numeric_columns].head()
```

Out[50]:

	ltdsales	ltdord	carttime	invcamt	ipdistance	customerdays	pmtscleared	shiptopmts	subgrouplr
0	0.0	0	49	1018.1	13	0	0	0	3.840077
1	0.0	0	6	92.6	2	0	0	0	2.236480
2	0.0	0	0	679.0	1	0	0	0	0.754100
3	0.0	0	19	240.0	7	0	0	0	2.533910
4	0.0	0	0	91.0	500	0	0	0	2.821576

In [51]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [52]:
```
scl = MinMaxScaler()
scl.fit(df[numeric_columns])
dat = scl.transform(df[numeric_columns])
num_col_scl = pd.DataFrame(dat, columns = numeric_columns)
num_col_scl.head()
```

```
C:\miniconda3\lib\site-packages\sklearn\preprocessing\data.py:334: DataConversionWarnin
g: Data with input dtype int64, float64 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

Out[52]:

	ltdsales	ltdord	carttime	invcamt	ipdistance	customerdays	pmtscleared	shiptopmts	subgrouplr
0	0.0	0.0	0.000046	0.011531	0.026	0.0	0.0	0.0	0.008180
1	0.0	0.0	0.000006	0.001036	0.004	0.0	0.0	0.0	0.004764
2	0.0	0.0	0.000000	0.007686	0.002	0.0	0.0	0.0	0.001606
3	0.0	0.0	0.000018	0.002708	0.014	0.0	0.0	0.0	0.005398
4	0.0	0.0	0.000000	0.001018	1.000	0.0	0.0	0.0	0.006010

Lets create a new dataframe with all our numeric columns

In [53]:
```
mdl_data = pd.concat([cat_col_end ,num_col_scl, df[flags], df[y]], axis = 1)
#here is the first time we use the axis function, this allows us to specify how we are c
ombining the dataframes. By rows (axis  = 0) or by columns(axis = 1)
```

In [54]:
```
mdl_data.head()
```

Out[54]:

	device_desktop	device_mobile	device_tablet	device_unknown	frtpct_20%	frtpct_40%	frtpct_60%	frtpct_6
0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	
1	0.0	1.0	0.0	0.0	1.0	0.0	0.0	
2	0.0	1.0	0.0	0.0	1.0	0.0	0.0	
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	1.0	0.0	0.0	

5 rows × 64 columns

In [55]:
```
mdl_data.columns
```

Out[55]:
```
Index(['device_desktop', 'device_mobile', 'device_tablet', 'device_unknown',
       'frtpct_20%', 'frtpct_40%', 'frtpct_60%', 'frtpct_60+%',
       'ordtype_EMAIL ORDER     ', 'ordtype_FAX ORDER       ',
       'ordtype_INTERNET        ', 'ordtype_MAIL ORDER      ',
       'ordtype_PHONE ORDER     ', 'ordtype_VENDOR SHIPMENTS',
       'ordtype_WALK IN ORDER   ', 'ordtype_WEB PHONE#      ',
       'timeblock_1159AM', 'timeblock_1159PM', 'timeblock_3AM',
       'timeblock_3PM', 'timeblock_6AM', 'timeblock_6PM', 'timeblock_9AM',
       'timeblock_9PM', 'ipcountry_CN', 'ipcountry_MX', 'ipcountry_Other',
       'ipcountry_US', 'numempl_A', 'numempl_B', 'numempl_C', 'numempl_D',
       'numempl_E', 'numempl_F', 'numempl_G', 'numempl_H', 'numempl_I',
       'numempl_X', 'taxflg_notax', 'taxflg_taxable', 'paymenttype_COD',
       'paymenttype_CWO', 'paymenttype_CreditCard', 'paymenttype_ECheck',
       'paymenttype_Net30', 'addtype_commercial', 'addtype_residential',
       'addtype_unknown', 'ltdsales', 'ltdord', 'carttime', 'invcamt',
       'ipdistance', 'customerdays', 'pmtscleared', 'shiptopmts', 'subgrouplr',
       'fbitemflg', 'dunsflg', 'quoteflg', 'btstmatch', 'fraudip', 'prevfraud',
       'fraud'],
      dtype='object')
```

Cleaned Data Save

In [56]:
```
mdl_data.to_csv("Data\Cleaned Fraud Data.csv", index = False)
```

Linear & Logistic Regression

https://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html (https://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html)

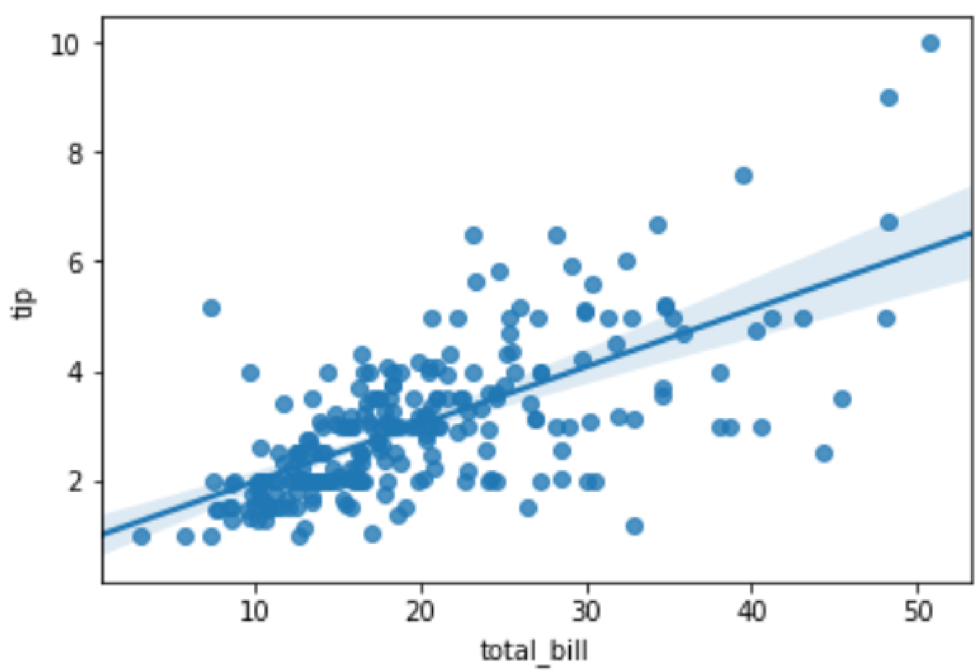
https://mccormickml.com/2014/03/04/gradient-descent-derivation/ (https://mccormickml.com/2014/03/04/gradient-descent-derivation/)

Linear Regression Math

Linear regression is some basic mathematics we have learned in high-school. It involves linear algebra & calculus. From a general perspective we have a value we want to predict from historical data. To do so we will take some random weights multiplied by our data points + intercept. y = mx + b. The summation sign lets us know we do the prediction for each row. You would read the below equation as "Y(hat) = the summation of weights times data points + bias"

Formula

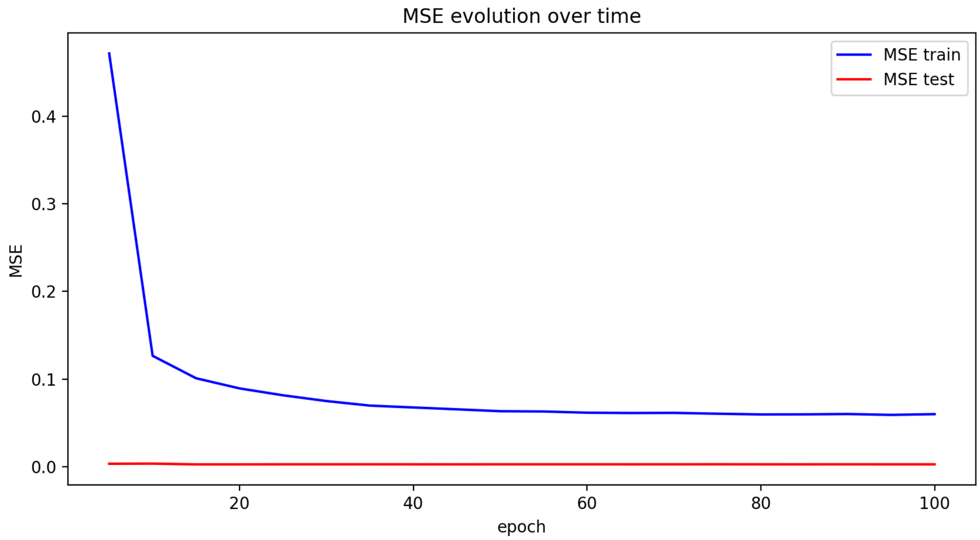
$$\hat{y}_j = \sum_{i=1}^{n} \theta \cdot x_{ji} + b,$$



Cost Function: MSE

Once we have a set of predicted values we can measure the error using the "Mean Squared Error Formula". The equation is saying subtract your predicted value from your original value and square it(to get rid of negative differences). Once you have that you can take the average of the summations of the errors. This is your error for the first iteration.

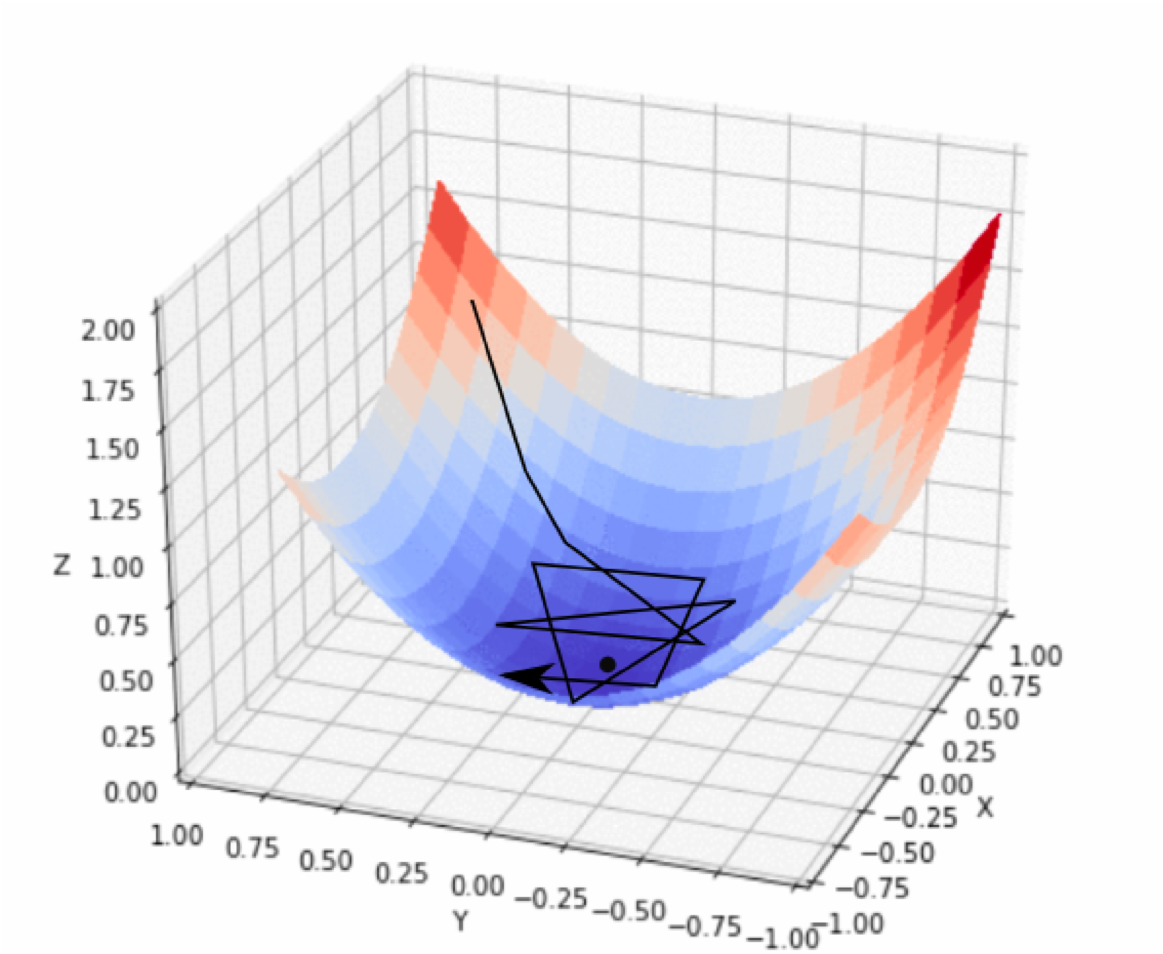
$$f(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\left[\boldsymbol{\theta}\cdot\mathbf{x}_i' - y_i\right]^2$$



Gradient Descent

This is where calculus comes in, our goal is to minimize the error , and we know that to minimize a function we must take the derivative of it. The below equation is just that. It is the derivative of MSE multiplied by our data. When we do this we get our first gradient. This will tell us in which direction we need to update our weights to get a better result.

$$\nabla_{LOSS_{MSE}}(\boldsymbol{\theta}) = \frac{2}{n}\sum_{i=1}^{n}\left[\boldsymbol{\theta}\cdot\mathbf{x}_i' - y_i\right]\cdot\mathbf{x}'$$



Learning Rate

The learning rate rate is introduced because out gradient can be a very large number, and if we take it at its face value, we might bounce back and forth in our prediction error. The 2/n in the above equation is replaced by the learning rate, which is user defined. (0.10) is the typical learning rate to start with.

$$\eta$$

Weights Update

Now that we have our gradient and learning rate, we can update our weights. We take the old weight and subtract it from out (learning rate * gradient). We take this new weight and loop the process above. During each iteration(epoch) we keep track of MSE, when the MSE does not change dramatically, we know we have converged. (Convergence in an algorithm means additional loops will not help use lower the MSE). Congratulations you have created your first linear model.

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot \nabla_{LOSS_{MSE}}(\boldsymbol{\theta})$$

Logistic Regression Math

Formula

Logistic regression follows the same formula as above, but there are slight changes to the results and cost function.

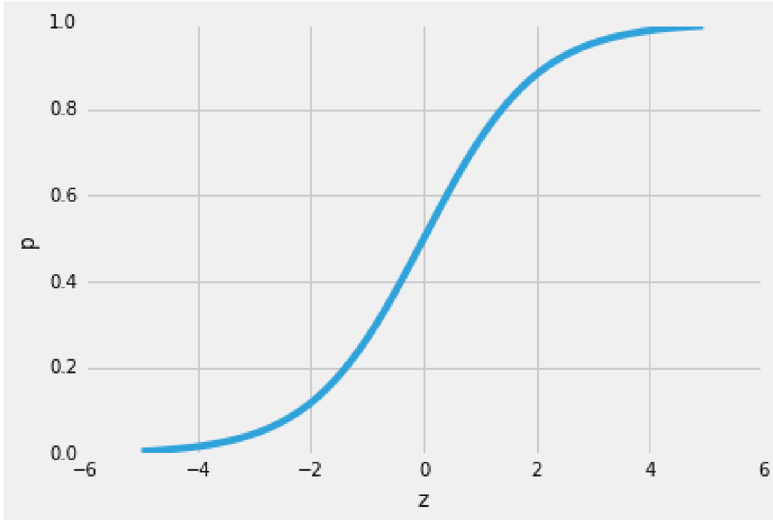
$$\hat{y}_j = \sum_{i=1}^{n} \theta \cdot x_{ji} + b,$$

Sigmoid Activation

Below is the sigmoid activation function. We will take out error numbers for each row and push it through the sigmoid function, this will give us a number between 0 and 1. Once we have that we can create a simple decision boundary that says 1 of probability > .5 else 0

$$S(z) = \frac{1}{1 + e^{-z}}$$

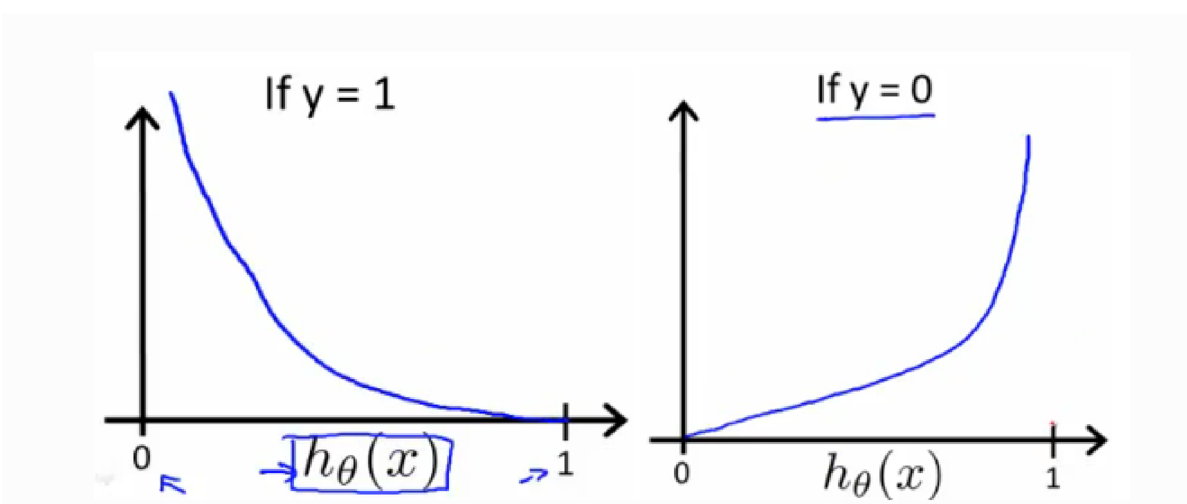
$$p \geq 0.5, class = 1, p < 0.5, class = 0$$



Cost Function: Cross Entropy

We can not use MSE for logistic evaluations, we need to use the cross entropy function to measure error.

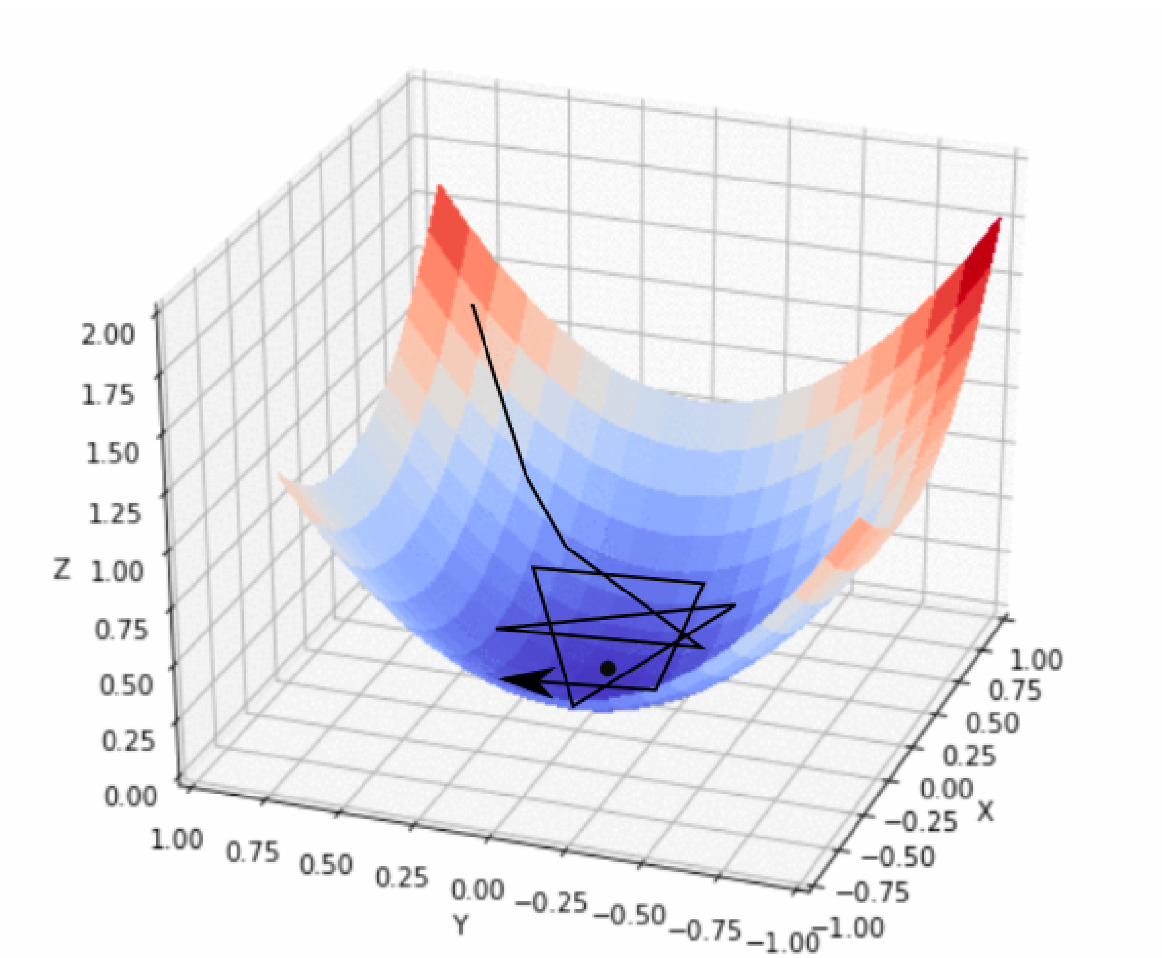
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[(y^i log(s(z))) + ((1-y^i)log(1-s(z)))]$$



Gradient Descent

Taking the first derivative of the cross entropy gets us to this gradient function which will tell us in which direction we need to move.

$$\nabla_{LOSS_{CrossEnt}}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left[\boldsymbol{sig}(\boldsymbol{\theta}\cdot\mathbf{x}_i') - y_i\right]\cdot\mathbf{x}'$$



Learning Rate

$$\eta$$

Weights

Now that we have our gradient and learning rate, we can update our weights. We take the old weight and subtract it from our (learning rate * gradient). We take this new weight and loop the process above. During each iteration(epoch) we keep track of Loss, when the loss does not change dramatically, we know we have converged. Congratulations you have created your first logistic model.

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot \nabla_{LOSS_{crossent}}(\boldsymbol{\theta})$$

In [1]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
from sklearn.model_selection import train_test_split
%matplotlib inline
```

In [2]:
```
df = pd.read_csv('Data/Cleaned Fraud Data.csv')
mdl_data = df[df.columns[0:len(df.columns)-1]]
y = df['fraud']
```

In [3]:
```
y.value_counts()
```

Out[3]:
```
0    100001
1     12580
Name: fraud, dtype: int64
```

New we can start predicting whether an order is fraud or nor. Remember the fraud column contains ones and zeros. Our job is to predict the same ones and zeros.We will use ROC curve and accuracy scores to see how accurate we were.

In [7]:
```
mdl = LogisticRegression(n_jobs = -1)
mdl.fit(mdl_data,y)
mdl.score(mdl_data, y)  #The score method shows us the accuracy score.
```

```
C:\miniconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Def
ault solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warnin
g.
  FutureWarning)
C:\miniconda3\lib\site-packages\sklearn\linear_model\logistic.py:1300: UserWarning: 'n_j
obs' > 1 does not have any effect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 8.
  " = {}.".format(effective_n_jobs(self.n_jobs)))
```

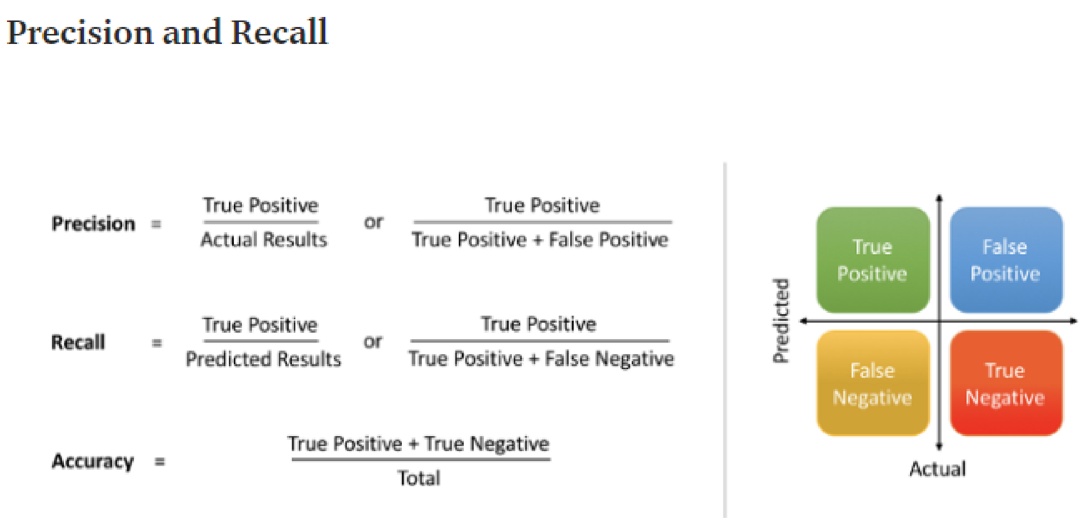
Out[7]:
```
0.9672591289826881
```

In [8]:
```
prd = mdl.predict(mdl_data)
```

In [9]:
```
pd.Series(prd).value_counts()
```

Out[9]:
```
0    101155
1     11426
dtype: int64
```

In classification we want to focus on how good the precision and recall metrics are. The f1 score combines the two metrics into a single score. We can see that for predicting not fraud out f1 score is 98%, but for predicting fraud our f1 score is only 81%.



Both precision and recall are trying to measure how accurately we predicted the positive results. In our case this is very important because over 90% of our data is negative, if we just predict everything negative then our accuracy will be 90%, which may sound great but is not helpful for what we are trying to do at all. We want to be able to predict the positive results accurately.

- Precision tells use of all the ones we predicted positive how many were actually positive.
- Recall tells us of all the ones that were actually positive how many did we get accurately.

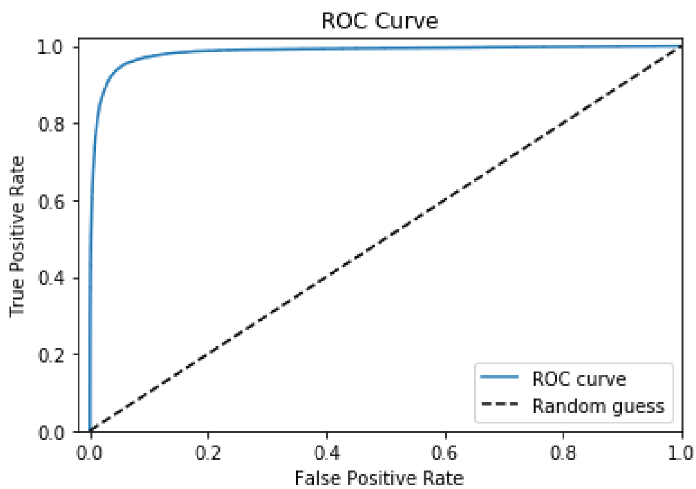
In [10]:
```
print(classification_report(y,prd))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98    100001
           1       0.89      0.81      0.85     12580

   micro avg       0.97      0.97      0.97    112581
   macro avg       0.93      0.90      0.91    112581
weighted avg       0.97      0.97      0.97    112581
```

In [12]:
```
#Notice here the roc_curve returns 3 values and we are assigning them to the fpr, tpr, a
nd threshold  variables.
#tpr = true positive rate
#fpr = false positive rate
fpr, tpr, thresholds = roc_curve(y,  mdl.predict_proba(mdl_data)[:,1])
# create plot
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random guess')  #The k-- tell the plot to make a
 dotted line.
_ = plt.xlabel('False Positive Rate')
_ = plt.ylabel('True Positive Rate')
_ = plt.title('ROC Curve')
_ = plt.xlim([-0.02, 1])
_ = plt.ylim([0, 1.02])
_ = plt.legend(loc="lower right")
```



In [32]:
```
mat = confusion_matrix(y,prd)
mat
```

Out[32]:
```
array([[98738,  1263],
       [ 2419, 10161]], dtype=int64)
```

In [21]:
```
mat.sum(axis = 1)
```

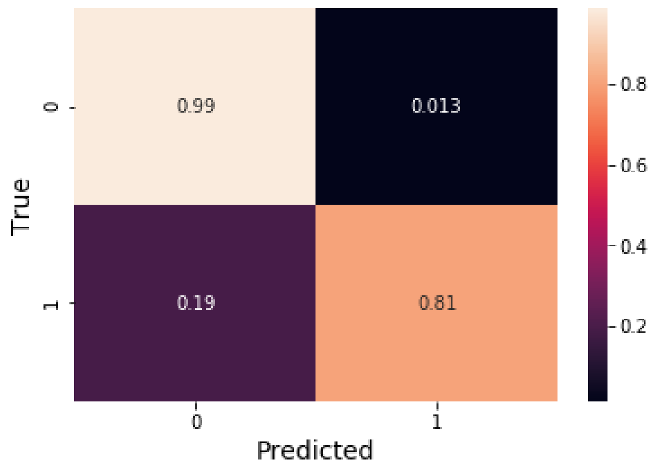
Out[21]:
```
array([100001,  12580], dtype=int64)
```

In [33]:
```
mat = mat/mat.sum(axis = 1)[:, np.newaxis]
mat
```

Out[33]:
```
array([[0.98737013, 0.01262987],
       [0.19228935, 0.80771065]])
```

In [34]:
```
sn.heatmap(mat, annot=True)
plt.xlabel("Predicted", fontsize=14)
plt.ylabel("True", fontsize=14)
```

Out[34]:
```
Text(33.0, 0.5, 'True')
```



The above was a sample of how we can predict using scikit-learn. We did not cover one thing, which was unseen data. Normally we have a set of data we create and algorithm and, then apply it to new data as it comes in. In this process we are hoping our algorithm is not overfitted. To avoid this issue, we can artificially create unseen data, by taking a small percentage (33%) of the stratified (the split hold the same percentage of zeros and ones) dataset and leaving it to test later.

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(mdl_data, y, test_size=0.33, random_
state=42, stratify  = y)
```

In [14]:
```
mdl2 = LogisticRegression(n_jobs = -1)
mdl2.fit(X_train,y_train)
mdl2.score(X_train,y_train)
```

```
C:\miniconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Def
ault solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warnin
g.
  FutureWarning)
C:\miniconda3\lib\site-packages\sklearn\linear_model\logistic.py:1300: UserWarning: 'n_j
obs' > 1 does not have any effect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 8.
  " = {}.".format(effective_n_jobs(self.n_jobs)))
```

Out[14]:
```
0.966869506423259
```

In [15]:
```
mdl2.score(X_test,y_test)
```

Out[15]:
```
0.966785099052541
```

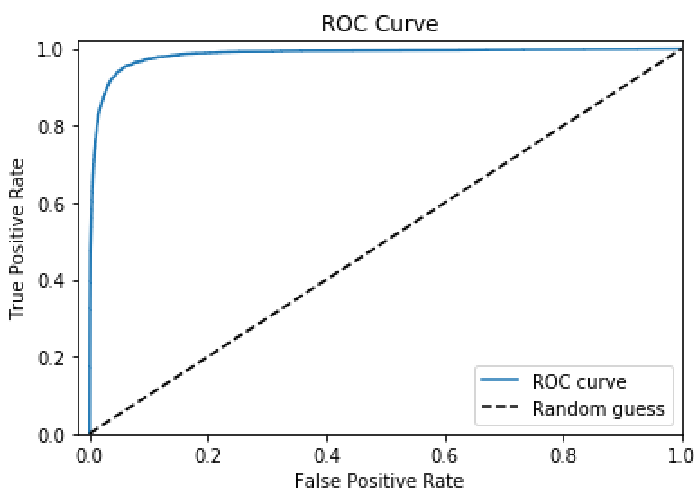
In [16]:
```
prd= mdl2.predict(X_test)
prd_pro = mdl2.predict_proba(X_test)[:,1]
print(classification_report(y_test,prd))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98     33001
           1       0.89      0.80      0.84      4151

   micro avg       0.97      0.97      0.97     37152
   macro avg       0.93      0.90      0.91     37152
weighted avg       0.97      0.97      0.97     37152
```

In [17]:
```
fpr, tpr, thresholds = roc_curve(y_test,  prd_pro)
# create plot
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random guess')
_ = plt.xlabel('False Positive Rate')
_ = plt.ylabel('True Positive Rate')
_ = plt.title('ROC Curve')
_ = plt.xlim([-0.02, 1])
_ = plt.ylim([0, 1.02])
_ = plt.legend(loc="lower right")
```



In [18]:
```
mat = confusion_matrix(y_test,prd)
mat
```

Out[18]:
```
array([[32577,   424],
       [  810,  3341]], dtype=int64)
```

In [19]:
```
mat.sum(axis = 1)
```

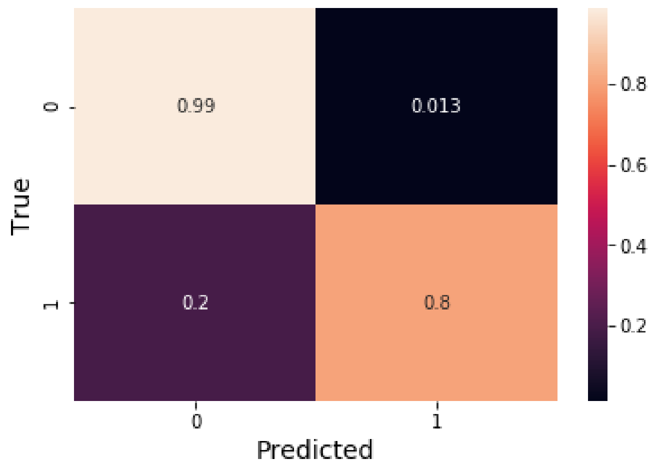
Out[19]:
```
array([33001,  4151], dtype=int64)
```

In [20]:
```
mat = mat/mat.sum(axis = 1)[:, np.newaxis]
mat
```

Out[20]:
```
array([[0.9871519, 0.0128481],
       [0.1951337, 0.8048663]])
```

In [21]:
```
sn.heatmap(mat, annot=True)
plt.xlabel("Predicted", fontsize=14)
plt.ylabel("True", fontsize=14)
```

Out[21]: `Text(33.0, 0.5, 'True')`



Decision Trees & Random Forest

Math

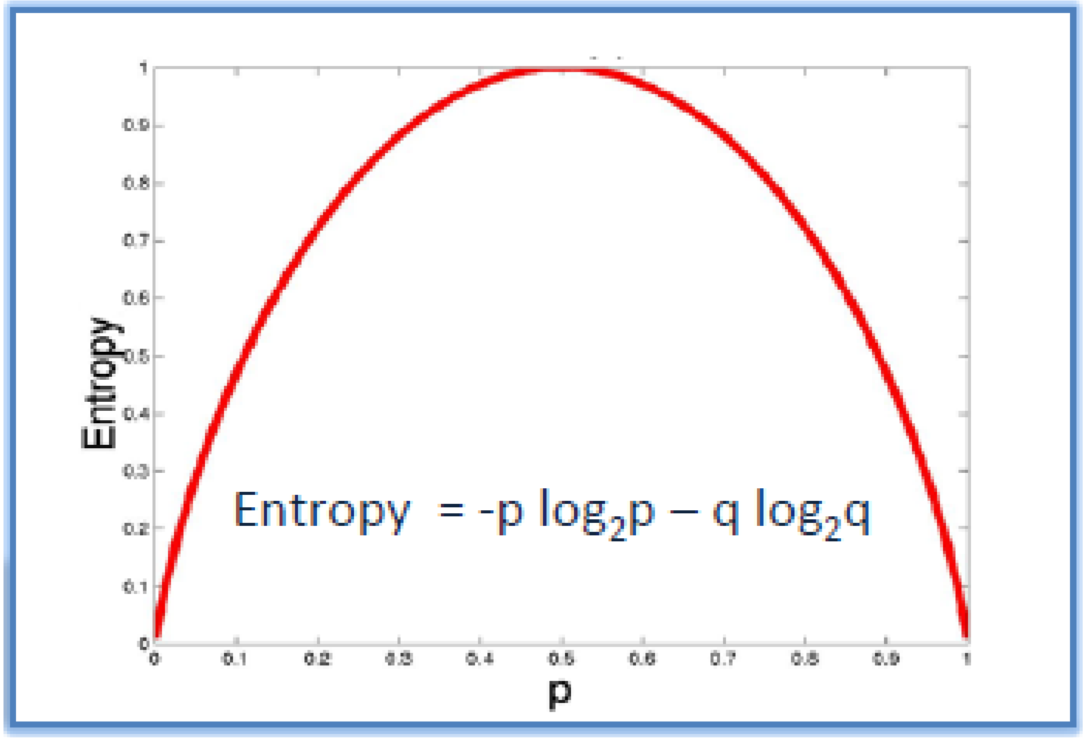
https://www.saedsayad.com/decision_tree.htm (https://www.saedsayad.com/decision_tree.htm)

ID3 algorithm using entropy & and information gain. Decision tree work fundamentally different than regression problems, they are more intuitive to understand. Our goal during each depth of a tree is to figure out which columns do we want to split on. Ideally we would want a split that segregates our data well. To do the split we focus on on using entropy and information gain.

Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Entropy

Entropy tries to measure how to split a column is within a dataset. For a two class dataset if the columns contain equal number of classes, the the entropy would be 1. If the split divides the class into one set, then entropy would be 0. We want to find entropy of all the columns and measure the information gain we get from splitting through that columns. Information gain will be the entropy of the raw target variable - entropy of one of the columns.



$$\text{Entropy} = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

Below we have calculated the entropy of the target dataset. We have 5 No's and 9 yes's. Please note in CS usually when you see log it is defined and log base 2. Do not confuse this with natural log or log base 10.

In [1]:
```
from math import log2
golfent = (-5/14 * log2(5/14)) + (-9/14 * log2(9/14))
golfent
```

Out[1]: 0.9402859586706311

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)

= Entropy (0.36, 0.64)

= - $(0.36 \log_2 0.36)$ - $(0.64 \log_2 0.64)$

= 0.94

To calculate the entropy of another columns that has more than 2 classes we need to calculate the entropy of each category first. Example below

In [2]:
```
sunny = (-(3/5) * log2(3/5)) + (-(2/5) * log2(2/5))
overcast = -(-(4/4) * log2(4/4))
rainy = (-(2/5) * log2(2/5)) + (-(3/5) * log2(3/5))
```

Then we combine the three categories entopy into a singple numnber using the formulate below

In [3]:
```
ent = (5/14 * sunny)  + (4/14 * overcast) + 5/14 * rainy
ent
```

Out[3]:
```
0.6935361388961918
```

$$E(T, X) = \sum_{c \in X} P(c)E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)

= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971

= 0.693

Infomation Gain

We then calculate the information gain we get from the original target variable. The variable that has the highest gain will be used for the split. Notice for a variable to have the highest gain it needs the lowest entropy, which implies the columns that is most homogeneous.

In [4]:
```
gainIoutlook = golfent - ent
gainIoutlook
```

Out[4]: 0.24674981977443933

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

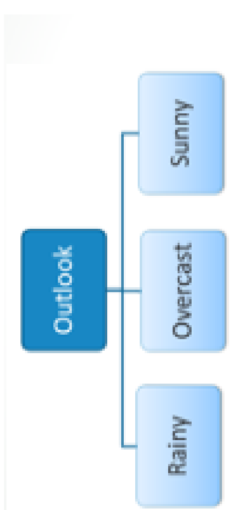
		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

$$Gain(T,X) = Entropy(T) - Entropy(T,X)$$

G(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

= 0.940 – 0.693 = 0.247



Outlook	Temp	Humidity	Windy	Play Golf
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Sunny	Mild	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

Overcast	Hot	High	FALSE	Yes
Overcast	Cool	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes

Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes

Now we iterate though this process until out data is completely homogeneous. Luckily we do not have to do this because, some creative programers and data scientists have crated objects in the scikit learn library that already do this process!

In [1]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn
from sklearn.model_selection import train_test_split
%matplotlib inline
```

In [2]:
```
df = pd.read_csv('Data\Cleaned Fraud Data.csv')
mdl_data = df[df.columns[0:len(df.columns)-1]]
y = df['fraud']
```

In [3]:
```
X_train, X_test, y_train, y_test = train_test_split(mdl_data, y, test_size=0.33, random_
state=42, stratify  = y)
```

Decision Trees

In [4]:
```
mdl2 =  DecisionTreeClassifier(max_depth = 4, criterion  = 'entropy')
mdl2.fit(X_train,y_train)
mdl2.score(X_train,y_train)
```

Out[4]:
```
0.9638733113258826
```

In [5]:
```
mdl2.score(X_test,y_test)
```

Out[5]:
```
0.9614825581395349
```

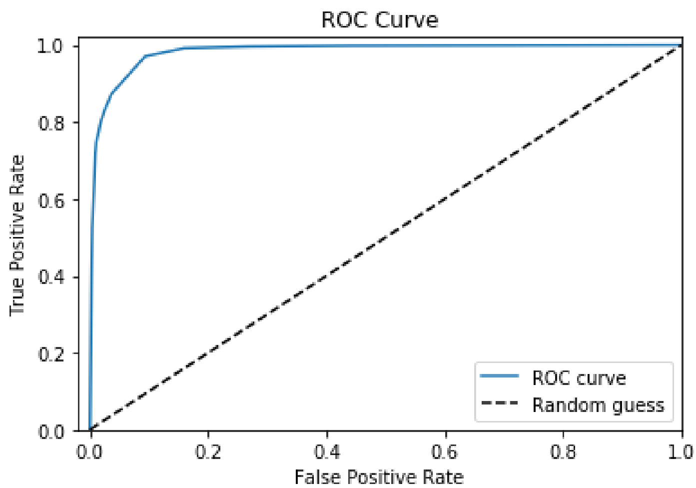
In [6]:
```
prd= mdl2.predict(X_test)
prd_pro = mdl2.predict_proba(X_test)[:,1]
print(classification_report(y_test,prd))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98     33001
           1       0.85      0.80      0.82      4151

   micro avg       0.96      0.96      0.96     37152
   macro avg       0.91      0.89      0.90     37152
weighted avg       0.96      0.96      0.96     37152
```

In [7]:
```
fpr, tpr, thresholds = roc_curve(y_test,  prd_pro)
# create plot
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random guess')
_ = plt.xlabel('False Positive Rate')
_ = plt.ylabel('True Positive Rate')
_ = plt.title('ROC Curve')
_ = plt.xlim([-0.02, 1])
_ = plt.ylim([0, 1.02])
_ = plt.legend(loc="lower right")
```



In [8]:
```
mat = confusion_matrix(y_test,prd)
mat
```

Out[8]:
```
array([[32394,   607],
       [  824,  3327]], dtype=int64)
```

In [9]:
```
mat.sum(axis = 1)
```

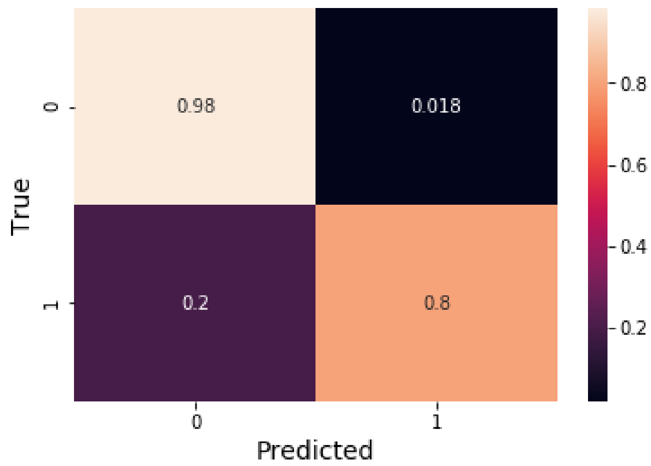
Out[9]:
```
array([33001,  4151], dtype=int64)
```

In [10]:
```
mat = mat/mat.sum(axis = 1)[:, np.newaxis]
mat
```

Out[10]:
```
array([[0.98160662, 0.01839338],
       [0.19850638, 0.80149362]])
```

In [11]:
```python
sn.heatmap(mat, annot=True)
plt.xlabel("Predicted", fontsize=14)
plt.ylabel("True", fontsize=14)
```

Out[11]:
```
Text(33.0, 0.5, 'True')
```

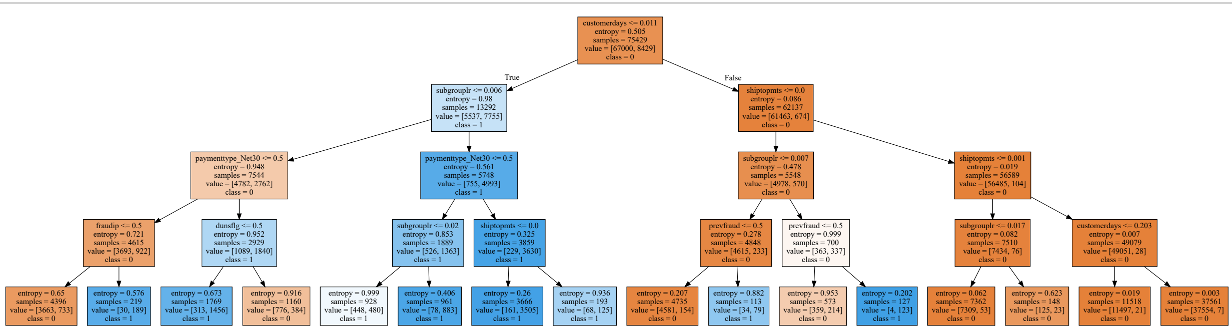


In [25]:
```python
#!pip install graphviz "C:\Uline-Python-master\packages\graphviz-0.11.1.zip"
!conda install graphviz
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.
```

In [12]:
```python
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import tree
from graphviz import Source
from IPython.display import SVG
from IPython.display import Image
labels = mdl_data.columns
graph = Source(tree.export_graphviz(mdl2, out_file=None
   , feature_names=labels, class_names=['0', '1']
   , filled = True))
display(SVG(graph.pipe(format='svg')))
```



In [13]:
```
from ipywidgets import interactive
# class labels
labels = mdl_data.columns
def plot_tree(crit, split, depth, min_split, min_leaf=0.2):
    estimator = DecisionTreeClassifier(random_state = 0
          , criterion = crit
          , splitter = split
          , max_depth = depth
          , min_samples_split=min_split
          , min_samples_leaf=min_leaf)
    estimator.fit(X_train, y_train)
    graph = Source(tree.export_graphviz(estimator
          , out_file=None
          , feature_names=labels
          , class_names=['0', '1', '2']
          , filled = True))

    display(SVG(graph.pipe(format='svg')))
    return estimator


inter=interactive(plot_tree
   , crit = ["gini", "entropy"]
   , split = ["best", "random"]
   , depth=[1,2,3,4]
   , min_split=(0.01,1)
   , min_leaf=(0.01,0.5))
display(inter)
```

Random Forest

A random forest is and ensemble if decision tree, meaning it creates n(user defined) decision tree. When predicting it will run the dataset through all the models, and then select the most predicted class. A random forest will select random samples & features (with replacement) from out data set to create trees.

In [14]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [15]:
```
mdl3 =  RandomForestClassifier(max_depth = 25,n_estimators=200, criterion  = 'entropy')
mdl3.fit(X_train,y_train)
mdl3.score(X_train,y_train)
```

Out[15]: 0.9998674249956914

In [16]:
```
mdl3.score(X_test,y_test)
```

Out[16]: 0.9803779069767442

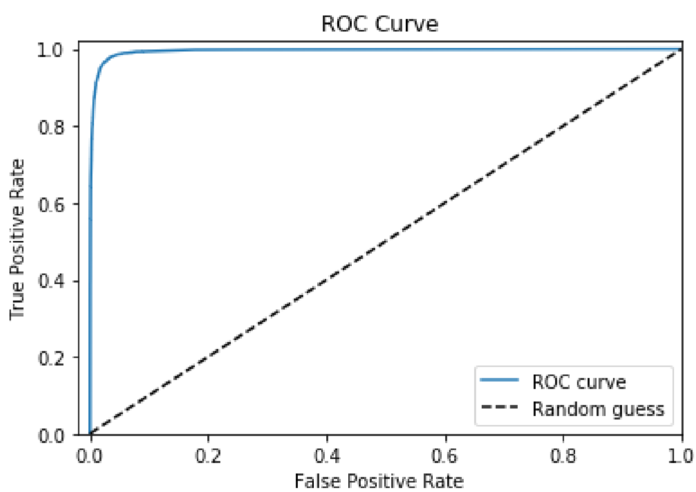
In [17]:
```
prd= mdl3.predict(X_test)
prd_pro = mdl3.predict_proba(X_test)[:,1]
print(classification_report(y_test,prd))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     33001
           1       0.92      0.90      0.91      4151

   micro avg       0.98      0.98      0.98     37152
   macro avg       0.95      0.95      0.95     37152
weighted avg       0.98      0.98      0.98     37152
```

In [18]:
```
fpr, tpr, thresholds = roc_curve(y_test,  prd_pro)
# create plot
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random guess')
_ = plt.xlabel('False Positive Rate')
_ = plt.ylabel('True Positive Rate')
_ = plt.title('ROC Curve')
_ = plt.xlim([-0.02, 1])
_ = plt.ylim([0, 1.02])
_ = plt.legend(loc="lower right")
```



In [19]:
```
mat = confusion_matrix(y_test,prd)
mat
```

Out[19]:
```
array([[32682,   319],
       [  410,  3741]], dtype=int64)
```

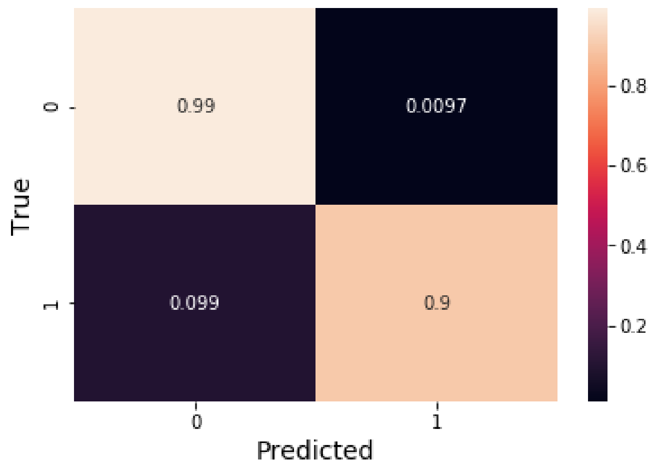
In [20]:
```
mat = mat/mat.sum(axis = 1)[:, np.newaxis]
mat
```

Out[20]:
```
array([[0.99033363, 0.00966637],
       [0.09877138, 0.90122862]])
```

In [21]:

```
sn.heatmap(mat, annot=True)
plt.xlabel("Predicted", fontsize=14)
plt.ylabel("True", fontsize=14)
```

Out[21]: Text(33.0, 0.5, 'True')



In [24]:

```
from ipywidgets import interactive
# class labels
labels = mdl_data.columns
def plot_tree(crit, depth, estm):
    estimator = RandomForestClassifier(random_state = 0
          , criterion = crit
          , max_depth = depth
          , n_estimators = estm)
    estimator.fit(X_train, y_train)
    prd= estimator.predict(X_test)
    prd_pro = estimator.predict_proba(X_test)[:,1]
    print(classification_report(y_test,prd))
    return estimator


inter=interactive(plot_tree
   , crit = ["gini", "entropy"]
   , depth=[3,6,9,12,15,18,21]
   ,estm=(10,70))
display(inter)
```

Data Structures

Lists

1.Create a List with values 1 through 10 and name it lst1

In [91]:
```
lst1 = [1,2,3,4,5,6,7,8,9,10]
```

2.Display the 3rd entry in lst1

In [146]:
```
lst2[2]
```

Out[146]: 8

3.Delete the 3rd entry in lst1

In [92]:
```
del lst1[2]
```

4.Create a 2nd list with values 10 through 1 and name it lst2

In [93]:
```
lst2 = [10,9,8,7,6,5,4,3,2,1]
```

5.Add lst1 and lst2 together

In [94]:
```
lst1 + lst2
```

Out[94]: [1, 2, 4, 5, 6, 7, 8, 9, 10, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

6.Append 11 to lst1

In [97]:
```
lst1.append(11)
```

7.Print lst1 in reverse order

In [105]:
```
lst1.reverse()
lst1
```

Out[105]: [11, 10, 9, 8, 7, 6, 5, 4, 2, 1]

Dictionaries

1.Create a dictionary named colors with key values

- r for red
- b for blue
- y for yellow
- g for green

In [140]:
```
colors = { 'r' : 'red',
       'b' : 'blue',
       'y' : 'yellow',
       'g' : 'green'
    }
```

2.Print Colors

In [141]:
```
colors
```

Out[141]:
```
{'r': 'red', 'b': 'blue', 'y': 'yellow', 'g': 'green'}
```

3.Change the value of key g to grey

In [142]:
```
colors['g'] = 'grey'
```

In [143]:
```
colors
```

Out[143]:
```
{'r': 'red', 'b': 'blue', 'y': 'yellow', 'g': 'grey'}
```

4. Delete the value of key g

In [144]:
```
del colors['g']
```

In [145]:
```
colors
```

Out[145]:
```
{'r': 'red', 'b': 'blue', 'y': 'yellow'}
```

Tuples

1.Create a tuple tup with lst1 and colors

In [136]:
```
tup = (lst1, colors)
```

In [137]:
```
tup
```

Out[137]:
```
([11, 10, 9, 8, 7, 6, 5, 4, 2, 1], {'r': 'red', 'b': 'blue', 'y': 'yellow'})
```

2.Attempt to delete the first index in the tuple

In [138]:
```
del tup[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-138-28eafd173795> in <module>
----> 1 del tup[0]

TypeError: 'tuple' object doesn't support item deletion
```

3. Delete The entire Tuples

In [139]:
```
del tup
```

Functions

1.Create a function called sum2 that adds two variables together and returns their value

In [128]:
```
def sum2(a, b):
    return a+b

sum2(5,7)
```

Out[128]:
```
12
```

2.Create a function called prt that print a string you pass.

In [130]:
```
def prt(a):
    print(a)

prt ("Hello world")
```

```
Hello world
```

Loops

1.Write a for loop that will iterate over lst1

In [131]:
```
for l in lst1:
    print(l)
```

```
11
10
9
8
7
6
5
4
2
1
```

2.Write a if statement that will print found if it finds the 3rd element from lst1

In [132]:
```
for l in lst1:
    if l == lst1[2]:
        print("Found "  + str(l))
```

```
Found 9
```

3.Create a variable counter with value 1

- Write a while loop that print the value of counter,
- Within the loop add 1 to counter during each iteration. Stop the loop when it reaches 10.

In [135]:
```
counter = 1
while counter < 11:
    print(counter)
    counter += 1
```

```
1
2
3
4
5
6
7
8
9
10
```

Data Frames

In [31]:
```
import pypyodbc
```

1.Read data from iscy.sql query as assign it to variable df.

- File is located in “2.Reading.Writing Data & EDA\Data”

In [42]:
```
cnxn = pypyodbc.connect(r"Driver={ODBC Driver 11 for SQL Server};"
                        "Server=WK-SQL2;"
                        "Database=UlineReporting;"
                        "Trusted_Connection=yes"
                       )

query = open(r'C:\Uline-Python-master\2.Reading.Writing Data & EDA\Data\iscy.sql', 'r')
df = pd.read_sql(query.read(), cnxn)
cnxn.close()
df.head()
```

Out[42]:

	item#	itemdesc	group#	subgrp#	ytd_ext_cost	ytd_ext_price	ytd_qty
0	S-11777	1/2" 24X125 FOAM 2/BD	18.0	855.0	68067.69	196112.0	1473.0
1	S-20562W	WHITE 16 OZ ULINE RIPPLE CUP	320.0	594.0	1711.00	4823.0	58.0
2	S-8546	10X7X2 GOLD STATIONERY BOX 50/CT	97.0	5605.0	2591.69	5597.0	75.0
3	S-17065G	3X5 GRN IND DIRECT THERMAL LBL	125.0	7110.0	2038.25	9690.0	162.0
4	H-6299	6' SAFETY RAILING - ALUMINUM	350.0	791.0	7085.45	31190.0	188.0

2. Describe the data frame.

In [43]:
```
df.describe()
```

Out[43]:

	group#	subgrp#	ytd_ext_cost	ytd_ext_price	ytd_qty
count	32501.000000	32501.000000	3.250100e+04	3.250100e+04	3.250100e+04
mean	208.150303	3150.044891	3.040781e+04	7.323626e+04	2.755389e+04
std	158.475750	2675.494891	8.555564e+04	1.905085e+05	2.217061e+05
min	2.000000	2.000000	-1.636800e+03	-2.575600e+03	-1.356000e+03
25%	80.000000	955.000000	2.477000e+03	6.802500e+03	1.830000e+02
50%	177.000000	2239.000000	8.912940e+03	2.451280e+04	6.810000e+02
75%	325.000000	5554.000000	2.786231e+04	7.271000e+04	3.345000e+03
max	565.000000	8991.000000	4.991848e+06	1.177572e+07	1.207700e+07

3.Display the first 10 rows.

In [44]:
```
df.head(10)
```

Out[44]:

	item#	itemdesc	group#	subgrp#	ytd_ext_cost	ytd_ext_price	ytd_qty
0	S-11777	1/2" 24X125 FOAM 2/BD	18.0	855.0	68067.69	196112.0	1473.0
1	S-20562W	WHITE 16 OZ ULINE RIPPLE CUP	320.0	594.0	1711.00	4823.0	58.0
2	S-8546	10X7X2 GOLD STATIONERY BOX 50/CT	97.0	5605.0	2591.69	5597.0	75.0
3	S-17065G	3X5 GRN IND DIRECT THERMAL LBL	125.0	7110.0	2038.25	9690.0	162.0
4	H-6299	6' SAFETY RAILING - ALUMINUM	350.0	791.0	7085.45	31190.0	188.0
5	S-10647Y	3X4 YELLOW ORGANZA BAG 100/BD	218.0	5517.0	267.67	1548.0	77.0
6	S-17207	17X22 JUMBO WHITE ENV 100/CT	32.0	1562.0	1529.06	3820.0	68.0
7	S-11896	3M908 1/2X36 ADHESIVE TRANS TAPE	238.0	6002.0	5630.05	11423.5	1126.0
8	H-3631	48X36" ASSEMBLY TABLE ADD-ON	537.0	37.0	1823.06	6985.0	29.0
9	S-17832	8X14 CLOTH PARTS BAG 100/BD	209.0	202.0	2728.30	14553.0	23700.0

4.Display the last 10 rows.

In [45]:
```
df.tail(10)
```

Out[45]:

	item#	itemdesc	group#	subgrp#	ytd_ext_cost	ytd_ext_price	ytd_qty
32491	S-15856	SIMPLE GREEN EXTREME 5GAL PAIL	122.0	6974.0	24881.02	53756.30	556.0
32492	S-19767	3M8512 PARTICULATE RESPIRATOR	230.0	1025.0	1851.52	3548.00	46.0
32493	S-17851	ULINE PLIER STAPLES - 5/8"	51.0	2557.0	43993.86	124133.91	14382.0
32494	S-19389	WYPALL X90 JUMBO ROLL WIPER	206.0	6961.0	32356.93	69064.50	747.0
32495	S-21545C	VIRTUA CCS SAFETY GLASSES-CLEAR	208.0	1730.0	40621.61	69324.10	7663.0
32496	S-11505BLU	13X17.5 BLUE GLAMOUR BUBBLE MLR	188.0	1566.0	16126.81	41496.61	346.0
32497	S-13561	17X1/4" TRASH CAN BANDS 400/CT	118.0	3755.0	36313.41	73274.15	1029.0
32498	H-5767	LOUVERED PANEL FOR DLX WORKSTAT	539.0	2763.0	22640.29	78302.89	3585.0
32499	S-10648TRQ	4X6 TURQUOISE ORGANZA BAG 100/BD	218.0	5517.0	1659.51	11129.70	386.0
32500	S-2997	1 1/4X025 HT STEEL STRAP 20CL/SK	59.0	2904.0	67528.20	164868.00	1757.0

5.Which items had the most sales dollars?

```
In [46]: df[['item#','ytd_ext_price']].sort_values(by = ['ytd_ext_price'] , ascending = False).head(10)
```

Out[46]:

	item#	ytd_ext_price
20273	S-423	11775719.89
20278	S-2190	9713266.24
20303	S-445	5534045.30
20259	H-1043	4995684.34
20192	S-3212	4292688.76
20150	S-4125	4277128.35
16336	S-3927P	4276719.00
19702	S-3931P	4191295.06
17271	S-6802	3991820.41
21084	S-3193	3665456.50

6.Which item had the least sales dollars?

```
In [148]: df[['item#','ytd_ext_price']].sort_values(by = ['ytd_ext_price'] , ascending = True).head(10)
```

Out[148]:

	item#	ytd_ext_price
25463	S-21667TREE	-2575.60
7463	S-12565	-821.00
19708	H-5335	-700.00
17305	H-4234	-627.00
8653	S-13212B	-528.00
4434	S-19143GOLD	-374.66
31689	H-6332B	-330.00
8301	S-19143R	-148.00
2229	H-3616BL-SHF	-143.49
21160	H-2080HANDLE	-100.00

7.List all the columns in the dataframe.

```
In [48]: df.columns
```

```
Out[48]: Index(['item#', 'itemdesc', 'group#', 'subgrp#', 'ytd_ext_cost',
                'ytd_ext_price', 'ytd_qty'],
               dtype='object')
```

8.Create a new dataframe df3 with the item# and ytd_ex_price columns.

In [49]:
```
df3 = df[['item#','ytd_ext_price']]
```

Out[49]:

	item#	ytd_ext_price
0	S-11777	196112.0
4	H-6299	31190.0
7	S-11896	11423.5
9	S-17832	14553.0
10	H-2099Y	72649.0

9.From df3 select items that has ytd_ext_price greater than 10000?

In [ ]:
```
df3[df3['ytd_ext_price'] > 10000].head()
```

10.From df display records for items S-423 and S-4125?

In [50]:
```
df[df['item#'].isin(['S-423','S-4125'])]
```

Out[50]:

	item#	itemdesc	group#	subgrp#	ytd_ext_cost	ytd_ext_price	ytd_qty
20150	S-4125	12X12X12 CUBE BOX 25/500	9.0	402.0	2437974.21	4277128.35	5190157.0
20273	S-423	TAPE 2X110 CLR 2MIL 36RL/CS	168.0	3059.0	4522842.37	11775719.89	7214040.0

11.From df sum up ytd_ext_price

In [51]:
```
sum(df['ytd_ext_price'])
```

Out[51]: 2380251702.2400103

12.Aggregate ytd_ext_price by group

- Flatten the columns
- Save the variable as df2
- Save df2 as groupsales.csv

In [58]:
```
d = { 'ytd_ext_price' : ['sum']}
df2 = df.groupby(['group#']).agg(d)
df2.columns  = ['_'.join(col) for col in df2.columns.values]
df2.to_csv('groupsaels.csv')
df2.head()
```

Out[58]:

	ytd_ext_price_sum
group#	
2.0	5430014.80
3.0	42100945.88
4.0	9557458.05
5.0	34128555.98
6.0	1718918.58

13.From df2 display the groups that had the most sales dollars

In [60]:
```
df2.sort_values(by = 'ytd_ext_price_sum', ascending = False).head()
```

Out[60]:

	ytd_ext_price_sum
group#	
9.0	2.550686e+08
168.0	5.277518e+07
60.0	4.819270e+07
3.0	4.210095e+07
125.0	4.166166e+07

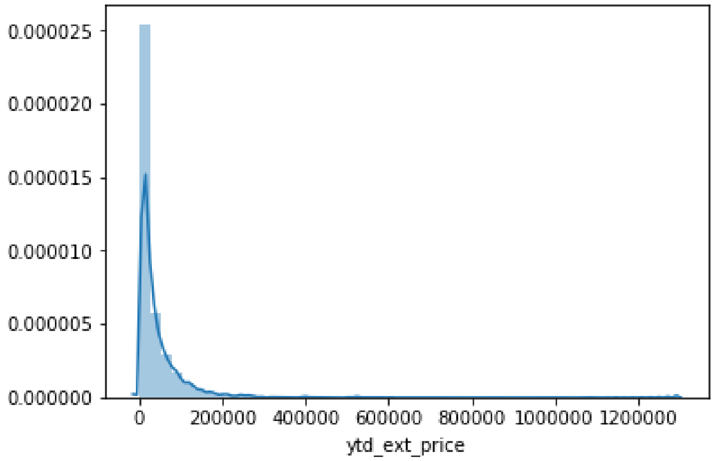
EDA

In [61]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

1.Graph distribution of the first 10000 records of YTD_EXT_PRICE

In [85]:
```
sns.distplot(df['ytd_ext_price'].head(5000))
```

Out[85]: <matplotlib.axes._subplots.AxesSubplot at 0x28851cc6940>

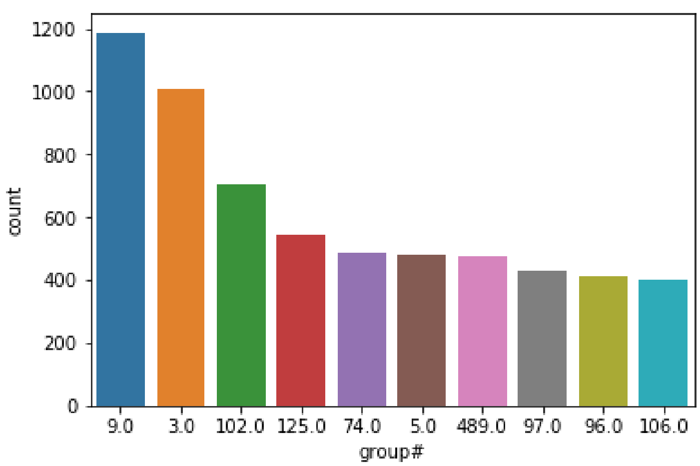


2.Create a new df called totItems that counts total items by group

- Create a bar graph of the total number of items in each group for the top 10 groups

In [156]:
```
a = { 'item#' : ['count']}
totItems = df.groupby(['group#']).agg(a).sort_values(by = ('item#','count'), ascending =
False).head(10)
sns.barplot(x = totItems.index , y = totItems['item#']['count'] , order = totItems.index
)
```

Out[156]: <matplotlib.axes._subplots.AxesSubplot at 0x28853357b00>

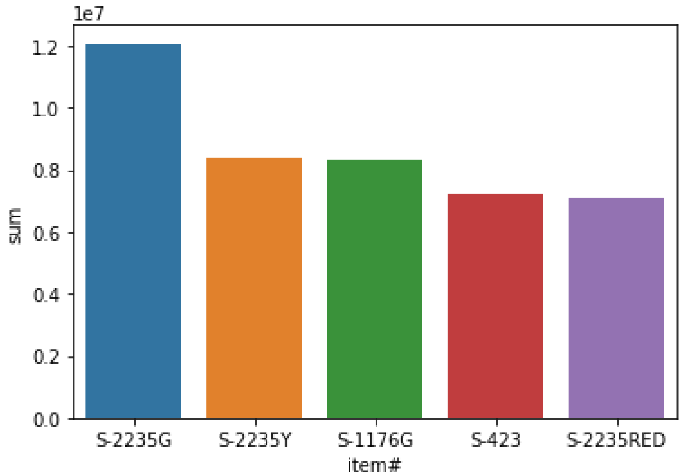


3.Create a new df called topItems that sums ytd_qty by item

- Create a bargraph top 5 sold items: Hint Sort the newly created df

In [75]:
```
a = { 'ytd_qty' : ['sum']}
topItems = df.groupby(['item#']).agg(a).sort_values(by = ('ytd_qty','sum'), ascending =
False).head()
sns.barplot(x = topItems.index, y= topItems['ytd_qty']['sum'])
```

Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x2885151e160>

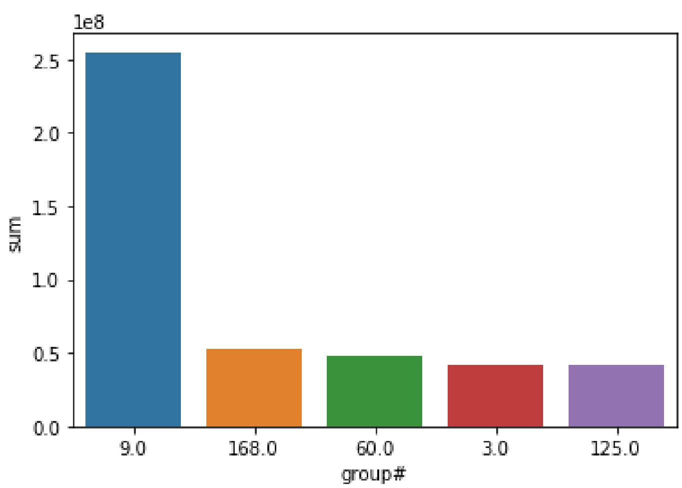


4.Create a new df called topSales that sums ytd_qty by item

- Create a bargraph of the top 5 groups with most sales Hint Sort the newly created df
- Set the order to the index of topSales

In [79]:
```
a = { 'ytd_ext_price' : ['sum']}
topSales = df.groupby(['group#']).agg(a).sort_values(by = ('ytd_ext_price','sum'), ascen
ding = False).head()
sns.barplot(x = topSales.index, y= topSales['ytd_ext_price']['sum'], order = topSales.in
dex)
```

Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x288518be1d0>



Data Modeling

In [ ]: