Learning to use Pandas for Data Analysis

In [1]:
```
# Import the pandas package
import pandas as pd
```

In [2]:
```
## Read the customer table and assign it to 'customer' object
customer = pd.read_csv("customer.csv")
```

Exploring the data

Exploring using .head() and .tail()

In [3]:
```
# call the customer table
customer.head()
```

Out[3]:

	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update	ac
0	1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	5	True	2006-02-14	2006-02-15 09:57:20	
1	2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org	6	True	2006-02-14	2006-02-15 09:57:20	
2	3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org	7	True	2006-02-14	2006-02-15 09:57:20	
3	4	2	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org	8	True	2006-02-14	2006-02-15 09:57:20	
4	5	1	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org	9	True	2006-02-14	2006-02-15 09:57:20	

In [4]:
```
customer.tail()
```

Out[4]:

	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_up
594	595	1	TERRENCE	GUNDERSON	TERRENCE.GUNDERSON@sakilacustomer.org	601	True	2006-02-14	2006- 09:
595	596	1	ENRIQUE	FORSYTHE	ENRIQUE.FORSYTHE@sakilacustomer.org	602	True	2006-02-14	2006- 09:
596	597	1	FREDDIE	DUGGAN	FREDDIE.DUGGAN@sakilacustomer.org	603	True	2006-02-14	2006- 09:
597	598	1	WADE	DELVALLE	WADE.DELVALLE@sakilacustomer.org	604	True	2006-02-14	2006- 09:
598	599	2	AUSTIN	CINTRON	AUSTIN.CINTRON@sakilacustomer.org	605	True	2006-02-14	2006- 09:

In [5]:
```
# examine the tail
customer.sample(5)
```

Out[5]:

	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update	a
65	66	2	JANICE	WARD	JANICE.WARD@sakilacustomer.org	70	True	2006-02-14	2006-02-15 09:57:20	
480	481	1	HERMAN	DEVORE	HERMAN.DEVORE@sakilacustomer.org	486	True	2006-02-14	2006-02-15 09:57:20	
533	534	1	CHRISTIAN	JUNG	CHRISTIAN.JUNG@sakilacustomer.org	540	True	2006-02-14	2006-02-15 09:57:20	
234	235	1	JACKIE	LYNCH	JACKIE.LYNCH@sakilacustomer.org	239	True	2006-02-14	2006-02-15 09:57:20	
264	265	2	JENNIE	TERRY	JENNIE.TERRY@sakilacustomer.org	270	True	2006-02-14	2006-02-15 09:57:20	

In [6]:
```
# examine column names
customer.columns
```

Out[6]:
```
Index(['customer_id', 'store_id', 'first_name', 'last_name', 'email',
       'address_id', 'activebool', 'create_date', 'last_update', 'active'],
      dtype='object')
```

In [7]:
```
# examine the data types
customer.dtypes
```

Out[7]:
```
customer_id      int64
store_id         int64
first_name      object
last_name       object
email           object
address_id       int64
activebool        bool
create_date     object
last_update     object
active           int64
dtype: object
```

In [8]:
```
# import the payment table and repeat the above
payment = pd.read_csv("payment.csv")
payment.head()
```

Out[8]:

	payment_id	customer_id	staff_id	rental_id	amount	payment_date
0	16050	269	2	7	1.99	2007-01-24 21:40:19.996577
1	16051	269	1	98	0.99	2007-01-25 15:16:50.996577
2	16052	269	2	678	6.99	2007-01-28 21:44:14.996577
3	16053	269	2	703	0.99	2007-01-29 00:58:02.996577
4	16054	269	1	750	4.99	2007-01-29 08:10:06.996577

In [9]:
```
# descriptive stats on the customer and payment tables
customer.describe()
```

Out[9]:

	customer_id	store_id	address_id	active
count	599.000000	599.000000	599.000000	599.000000
mean	300.000000	1.455760	304.724541	0.974958
std	173.060683	0.498455	173.698609	0.156382
min	1.000000	1.000000	5.000000	0.000000
25%	150.500000	1.000000	154.500000	1.000000
50%	300.000000	1.000000	305.000000	1.000000
75%	449.500000	2.000000	454.500000	1.000000
max	599.000000	2.000000	605.000000	1.000000

Why are only certain columns described? Ans - Because the method by default provides descriptive statistics for numerical columns. It calculates statistics such as count, mean, standard deviation, minimum, quartiles, and maximum for numerical data.

What happens if we try and call describe on categorical columns? Ans - If you try to call describe() on categorical columns, it will provide descriptive statistics specific to categorical data. It will include the count of non-null values, the number of unique categories, the most frequent category, and the frequency of the most frequent category.

Selecting Columns

A dataframe is a collection of `series` (columns), a `series` is a `numpy array` with an index

In [10]:
```
# select a column as a series
customer['first_name']
```

Out[10]:
```
0           MARY
1       PATRICIA
2          LINDA
3        BARBARA
4      ELIZABETH
         ...
594     TERRENCE
595      ENRIQUE
596      FREDDIE
597         WADE
598       AUSTIN
Name: first_name, Length: 599, dtype: object
```

In [11]:
```
# select a column as a dataframe
customer[['first_name']]
```

Out[11]:

	first_name
0	MARY
1	PATRICIA
2	LINDA
3	BARBARA
4	ELIZABETH
...	...
594	TERRENCE
595	ENRIQUE
596	FREDDIE
597	WADE
598	AUSTIN

599 rows × 1 columns

In [12]:
```
# select multiple columns
customer[['first_name', 'last_name']]
```

Out[12]:

	first_name	last_name
0	MARY	SMITH
1	PATRICIA	JOHNSON
2	LINDA	WILLIAMS
3	BARBARA	JONES
4	ELIZABETH	BROWN
...	...	...
594	TERRENCE	GUNDERSON
595	ENRIQUE	FORSYTHE
596	FREDDIE	DUGGAN
597	WADE	DELVALLE
598	AUSTIN	CINTRON

599 rows × 2 columns

In [13]:
```
# describe first and last name columns
customer[['first_name', 'last_name']].describe()
```

Out[13]:

	first_name	last_name
count	599	599
unique	591	599
top	JESSIE	SMITH
freq	2	1

Describe a categorical column

In [14]:
```
# look at unique values for store_id
customer['store_id'].unique()
```

Out[14]:
```
array([1, 2], dtype=int64)
```

In [15]:
```
# using value counts
customer['store_id'].value_counts()
```

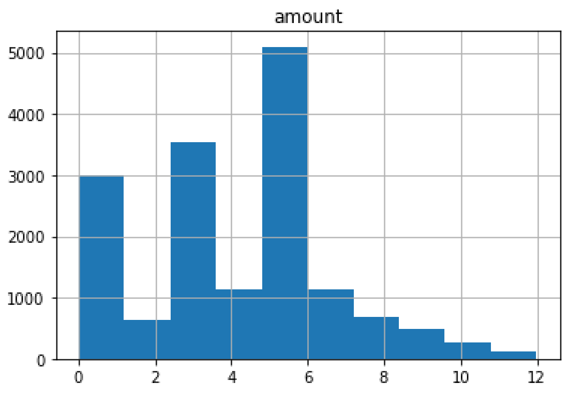
Out[15]:
```
1    326
2    273
Name: store_id, dtype: int64
```

Plotting

Lets find out how frequent different amounts are paid.

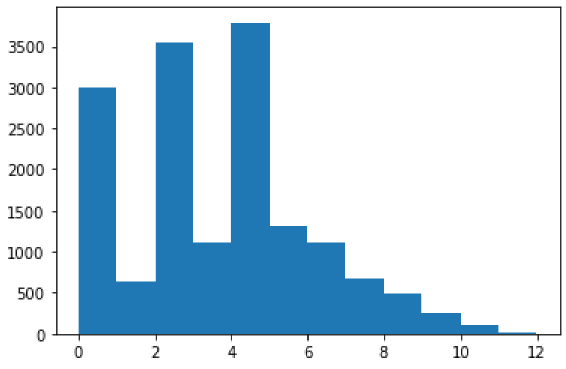
```
In [16]: # call the payment table
         payment = pd.read_csv('payment.csv')
```

```
In [17]: payment.hist(column = 'amount');
```

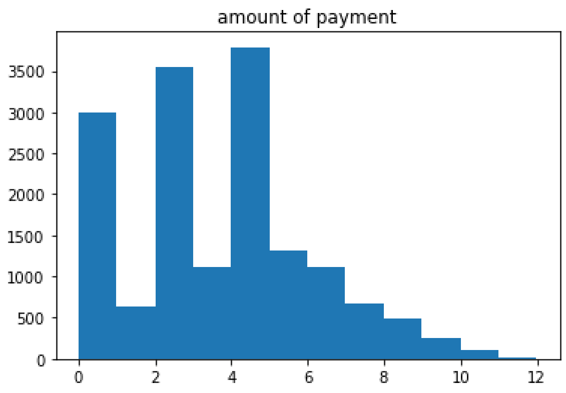


```
In [18]: # pandas method of histogram for amount in payment table
         payment['amount'].hist(grid = False, bins = 12)
```

```
Out[18]: <AxesSubplot:>
```



```
In [19]: # save the plot
         plot = payment['amount'].hist(grid = False, bins = 12)
         plot.set_title('amount of payment')
         plot.get_figure().savefig('output.pdf', format='pdf')
```



Sorting

```
In [20]: # sort by customers by name
         customer.sort_values(by = 'first_name').head()
```

Out[20]:

	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update	acti
374	375	2	AARON	SELBY	AARON.SELBY@sakilacustomer.org	380	True	2006-02-14	2006-02-15 09:57:20	
366	367	1	ADAM	GOOCH	ADAM.GOOCH@sakilacustomer.org	372	True	2006-02-14	2006-02-15 09:57:20	
524	525	2	ADRIAN	CLARY	ADRIAN.CLARY@sakilacustomer.org	531	True	2006-02-14	2006-02-15 09:57:20	
216	217	2	AGNES	BISHOP	AGNES.BISHOP@sakilacustomer.org	221	True	2006-02-14	2006-02-15 09:57:20	
388	389	1	ALAN	KAHN	ALAN.KAHN@sakilacustomer.org	394	True	2006-02-14	2006-02-15 09:57:20	

In [21]:

```python
# sort by store_id and address_id
customer.sort_values(by = ['store_id', 'address_id'], ascending = False).head()
```

Out[21]:

	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update	a
598	599	2	AUSTIN	CINTRON	AUSTIN.CINTRON@sakilacustomer.org	605	True	2006-02-14	2006-02-15 09:57:20	
592	593	2	RENE	MCALISTER	RENE.MCALISTER@sakilacustomer.org	599	True	2006-02-14	2006-02-15 09:57:20	
589	590	2	SETH	HANNON	SETH.HANNON@sakilacustomer.org	596	True	2006-02-14	2006-02-15 09:57:20	
583	584	2	SALVADOR	TEEL	SALVADOR.TEEL@sakilacustomer.org	590	True	2006-02-14	2006-02-15 09:57:20	
581	582	2	ANDY	VANHORN	ANDY.VANHORN@sakilacustomer.org	588	True	2006-02-14	2006-02-15 09:57:20	

This does not alter the values in the dataframe, in order to do so we must reassign or use a flag for inplace, Re-assigning is the preferred method

In [22]:

```python
# using inplace
customer.sort_values(by = 'first_name', inplace=True)
customer.head()
```

Out[22]:

	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update	acti
374	375	2	AARON	SELBY	AARON.SELBY@sakilacustomer.org	380	True	2006-02-14	2006-02-15 09:57:20	
366	367	1	ADAM	GOOCH	ADAM.GOOCH@sakilacustomer.org	372	True	2006-02-14	2006-02-15 09:57:20	
524	525	2	ADRIAN	CLARY	ADRIAN.CLARY@sakilacustomer.org	531	True	2006-02-14	2006-02-15 09:57:20	
216	217	2	AGNES	BISHOP	AGNES.BISHOP@sakilacustomer.org	221	True	2006-02-14	2006-02-15 09:57:20	
388	389	1	ALAN	KAHN	ALAN.KAHN@sakilacustomer.org	394	True	2006-02-14	2006-02-15 09:57:20	

In [23]:

```python
# reset by index
customer.sort_index(inplace=True)
customer.head()
```

Out[23]:

	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update	ac
0	1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	5	True	2006-02-14	2006-02-15 09:57:20	
1	2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org	6	True	2006-02-14	2006-02-15 09:57:20	
2	3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org	7	True	2006-02-14	2006-02-15 09:57:20	
3	4	2	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org	8	True	2006-02-14	2006-02-15 09:57:20	
4	5	1	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org	9	True	2006-02-14	2006-02-15 09:57:20	

In [24]:

```python
# using reassignment
customer = customer.sort_values(by='first_name')
```

In [25]:

```python
## Reset the index
customer.reset_index().head()
```

Out[25]:

	index	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update
0	374	375	2	AARON	SELBY	AARON.SELBY@sakilacustomer.org	380	True	2006-02-14	2006-02-15 09:57:20
1	366	367	1	ADAM	GOOCH	ADAM.GOOCH@sakilacustomer.org	372	True	2006-02-14	2006-02-15 09:57:20
2	524	525	2	ADRIAN	CLARY	ADRIAN.CLARY@sakilacustomer.org	531	True	2006-02-14	2006-02-15 09:57:20
3	216	217	2	AGNES	BISHOP	AGNES.BISHOP@sakilacustomer.org	221	True	2006-02-14	2006-02-15 09:57:20
4	388	389	1	ALAN	KAHN	ALAN.KAHN@sakilacustomer.org	394	True	2006-02-14	2006-02-15 09:57:20

This creates a new column, order to do so without we drop the previous index

In [26]:

```python
## Reset the index in place and drop previous index column
customer = customer.reset_index(drop=True)
customer.head()
```

Out[26]:

	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update	active
0	375	2	AARON	SELBY	AARON.SELBY@sakilacustomer.org	380	True	2006-02-14	2006-02-15 09:57:20	1
1	367	1	ADAM	GOOCH	ADAM.GOOCH@sakilacustomer.org	372	True	2006-02-14	2006-02-15 09:57:20	1
2	525	2	ADRIAN	CLARY	ADRIAN.CLARY@sakilacustomer.org	531	True	2006-02-14	2006-02-15 09:57:20	1
3	217	2	AGNES	BISHOP	AGNES.BISHOP@sakilacustomer.org	221	True	2006-02-14	2006-02-15 09:57:20	1
4	389	1	ALAN	KAHN	ALAN.KAHN@sakilacustomer.org	394	True	2006-02-14	2006-02-15 09:57:20	1

Filtering Rows

To look at subsets of the data, we will filter or group required sets.

In [27]:

```python
# Filter the table to just the store that we are interested in, store number 2
customer[customer['store_id'] == 2]
```

Out[27]:

	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update
0	375	2	AARON	SELBY	AARON.SELBY@sakilacustomer.org	380	True	2006-02-14	2006-02-15 09:57:20
2	525	2	ADRIAN	CLARY	ADRIAN.CLARY@sakilacustomer.org	531	True	2006-02-14	2006-02-15 09:57:20
3	217	2	AGNES	BISHOP	AGNES.BISHOP@sakilacustomer.org	221	True	2006-02-14	2006-02-15 09:57:20
6	568	2	ALBERTO	HENNING	ALBERTO.HENNING@sakilacustomer.org	574	True	2006-02-14	2006-02-15 09:57:20
7	454	2	ALEX	GRESHAM	ALEX.GRESHAM@sakilacustomer.org	459	True	2006-02-14	2006-02-15 09:57:20
...	...	...	...	...	...	...	...	...	...
593	359	2	WILLIE	MARKHAM	WILLIE.MARKHAM@sakilacustomer.org	364	True	2006-02-14	2006-02-15 09:57:20
594	219	2	WILLIE	HOWELL	WILLIE.HOWELL@sakilacustomer.org	223	True	2006-02-14	2006-02-15 09:57:20
595	212	2	WILMA	RICHARDS	WILMA.RICHARDS@sakilacustomer.org	216	True	2006-02-14	2006-02-15 09:57:20
596	190	2	YOLANDA	WEAVER	YOLANDA.WEAVER@sakilacustomer.org	194	True	2006-02-14	2006-02-15 09:57:20
597	174	2	YVONNE	WATKINS	YVONNE.WATKINS@sakilacustomer.org	178	True	2006-02-14	2006-02-15 09:57:20

273 rows × 10 columns

Explanation of whats going on in this operation

In [28]:

```
# Creating a Boolean mask to filter rows

store_id_filter = customer['store_id'] == 2
store_id_filter
```

Out[28]:

```
0       True
1      False
2       True
3       True
4      False
       ...
594     True
595     True
596     True
597     True
598    False
Name: store_id, Length: 599, dtype: bool
```

In [29]:

```
# Applying boolean mask to the dataframe
customer[store_id_filter]
```

Out[29]:

	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update
0	375	2	AARON	SELBY	AARON.SELBY@sakilacustomer.org	380	True	2006-02-14	2006-02-15 09:57:20
2	525	2	ADRIAN	CLARY	ADRIAN.CLARY@sakilacustomer.org	531	True	2006-02-14	2006-02-15 09:57:20
3	217	2	AGNES	BISHOP	AGNES.BISHOP@sakilacustomer.org	221	True	2006-02-14	2006-02-15 09:57:20
6	568	2	ALBERTO	HENNING	ALBERTO.HENNING@sakilacustomer.org	574	True	2006-02-14	2006-02-15 09:57:20
7	454	2	ALEX	GRESHAM	ALEX.GRESHAM@sakilacustomer.org	459	True	2006-02-14	2006-02-15 09:57:20
...	...	...	...	...	...	...	...	...	...
593	359	2	WILLIE	MARKHAM	WILLIE.MARKHAM@sakilacustomer.org	364	True	2006-02-14	2006-02-15 09:57:20
594	219	2	WILLIE	HOWELL	WILLIE.HOWELL@sakilacustomer.org	223	True	2006-02-14	2006-02-15 09:57:20
595	212	2	WILMA	RICHARDS	WILMA.RICHARDS@sakilacustomer.org	216	True	2006-02-14	2006-02-15 09:57:20
596	190	2	YOLANDA	WEAVER	YOLANDA.WEAVER@sakilacustomer.org	194	True	2006-02-14	2006-02-15 09:57:20
597	174	2	YVONNE	WATKINS	YVONNE.WATKINS@sakilacustomer.org	178	True	2006-02-14	2006-02-15 09:57:20

273 rows × 10 columns

In [30]:
```python
# create a boolean mask where first name is Terry
# experiment with case
first_name_filter = customer['first_name'].str.upper() == 'TERRY'
first_name_filter
```

Out[30]:
```
0      False
1      False
2      False
3      False
4      False
       ...  
594    False
595    False
596    False
597    False
598    False
Name: first_name, Length: 599, dtype: bool
```

In [31]:
```python
# apply both filters
customer[store_id_filter & first_name_filter]
```

Out[31]:

	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update	act
544	355	2	TERRY	GRISSOM	TERRY.GRISSOM@sakilacustomer.org	360	True	2006-02-14	2006-02-15 09:57:20	

Aggregation

In [32]:
```python
# total amount per customer
payment.groupby('customer_id').sum().head()
```

Out[32]:

	payment_id	staff_id	rental_id	amount
customer_id				
1	760358	47	241137	118.68
2	690998	39	257716	128.73
3	643639	38	206151	135.74
4	506081	32	197174	81.78
5	934784	59	300857	144.62

```
In [33]: payment[['customer_id', 'amount']].groupby('customer_id').sum()
```

Out[33]:

customer_id	amount
1	118.68
2	128.73
3	135.74
4	81.78
5	144.62
...	...
595	117.70
596	96.72
597	99.75
598	83.78
599	83.81

599 rows × 1 columns

```
In [34]: # agg with renaming the column
         payment[['customer_id', 'amount']].groupby('customer_id').agg(total_amount = ('amount', 'sum'))
```

Out[34]:

customer_id	total_amount
1	118.68
2	128.73
3	135.74
4	81.78
5	144.62
...	...
595	117.70
596	96.72
597	99.75
598	83.78
599	83.81

599 rows × 1 columns

Sort customers by descending total amount

```
In [35]: # do again with renaming to total_amount
         payment[['customer_id', 'amount']].groupby('customer_id'
                                                   ).agg(total_amount=('amount','sum')
                                                        ).sort_values(by='total_amount', ascending=False)
```

Out[35]:

	total_amount
customer_id	
526	221.55
148	216.54
144	195.58
137	194.61
178	194.61
...	...
97	58.82
395	57.81
318	52.88
281	50.86
248	50.85

599 rows × 1 columns

In [36]:
```
# Find the staff member with the highest average sale
payment[['staff_id', 'amount']].groupby('staff_id'
                                        ).agg(avg_sale = ('amount','mean')
                                             ).sort_values(by='avg_sale', ascending=False)
```

Out[36]:

	avg_sale
staff_id	
2	4.245125
1	4.156568

In [37]:
```
avg_sales_per_staff = payment[['staff_id', 'amount']].groupby('staff_id'
                                                              ).agg(avg_sale = ('amount','mean')
                                                                   ).sort_values(by='avg_sale', ascending=False)
```

In [38]:
```
# Save aggregation to csv
# do again with index=False
avg_sales_per_staff.to_csv('avg_sales_per_staff.csv', index=False)
```

In [39]:
```
# Save to Excel
avg_sales_per_staff.to_excel('customer.xlsx', sheet_name='payment_details')
```

Joins

We will use the merge function for this

In [40]:
```
## Merge the DataFrames using the .merge() method
pd.merge(left = customer,
         right = payment,
         how = 'left',
         left_on = 'customer_id',
         right_on = 'customer_id'
        ).head()
```

Out[40]:

	customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update	active
0	375	2	AARON	SELBY	AARON.SELBY@sakilacustomer.org	380	True	2006-02-14	2006-02-15 09:57:20	1
1	375	2	AARON	SELBY	AARON.SELBY@sakilacustomer.org	380	True	2006-02-14	2006-02-15 09:57:20	1
2	375	2	AARON	SELBY	AARON.SELBY@sakilacustomer.org	380	True	2006-02-14	2006-02-15 09:57:20	1
3	375	2	AARON	SELBY	AARON.SELBY@sakilacustomer.org	380	True	2006-02-14	2006-02-15 09:57:20	1
4	375	2	AARON	SELBY	AARON.SELBY@sakilacustomer.org	380	True	2006-02-14	2006-02-15 09:57:20	1