Cheat Sheet

Data Frames

```
df = pd.DataFrame([[...],..., [...]],
                  index = [0,1,2,...,],
                  columns=["var1","var2","var3",])
# Other way to make a DataFrame
df2 = pd.DataFrame({"var1":[...],
                    "var2":[...],
                    "var3":[...]},
                  index = [0,1,2,...,],
                  columns=["var1","var2","var3",])
```

Import Data

```
import pandas as pd

# Csv file
fb = pd.DataFrame.from_csv("../data/facebook.csv")
fb = pd.read_csv(r'../data/facebook.csv')

#Excel file
fb = pd.DataFrame.from_excel("../data/facebook.csv", sheet_name = "Sheet1" )

#Text file
filename = "filename.txt"
fb = pd.DataFrame.read_csv(filename)
```

Take a look inside the DataFrame

(types, columns, index, values,shape)

```
df.head()
df.header()
df.tail()

#looking each variable and its type, index, columns, values

df.types
df.index
df.columns
df.values
de.shape
```

Statistical summary of each variable (columns)

```
df.describe()
```

Selecting on a DataFrame

by position

```
# ILOC = Index location
fb.iloc[1:,:-1] # Select by the second row, but the columns should be all of it, except the last of them.

fb.iloc[:-1,[0,3]] # Select all rows except the last, only inside the first and the fourth columns.

# Row's number
fb [2:4] # Index with single column rows 2 to 3
```

by label

```
# LOC = Location
fb.loc['2015-01-02' , 'Close']
fb.loc['2015-01-02':'2015-01-01', 'Close']
fb.loc[ : , ["Open",'Close']]

# Column's name
fb[["Open","Close"]]
```

by label/position

```
df.ix[2] # Select a single row of a subset of rows
df.ix[:,"var_name1"] # Select a single column of a subset of columns
df.ix[1,"var_name1"] # Select the second row in the column called "var_name1"
```

Boolean indexing

```
# Idea General
df2 = df[filtro]

# Basics
df2 = df[df.var_1>0] # Filtering df, getting just values of the column "var_1" greater than 0
df[(df["Return"] < -.1) | (df["Return"] > .2)] # Getting a subset of data where just the returns are less than 10% or greater than 20%
df[df["Population"]>120000000] # Getting just values where observations are greater than 120mm

# Advanced
df[df["column1"].isin([ value1, value2, value3])] # Obtain just the observation, where values inside the serie are in column "column1"
```

Sort Values

```
df.sort_values(name_of_x_column, ascending = False)
```

Shift observations inside a column

```
prices = fb["Open"].shift(1) # Moving upward the rows
prices = fb["Open"].shift(-1) # Moving downward the rows
```

Assigments

```
import numpy as np

df.loc[ : ,"column1"] = np.nan # Assign a NaN in the column "column1"
df.loc[ : ,["column1", "column2"]] = np.array([5]*len(df)) # Assign 5's in all rows of the columns "column1" and "column2"
```

Renaming Columns

```
df.rename(columns = {"oldcol1_name":"newcol1_name"}, inplace=True)

#Other way to do the same

df = df.rename(columns = {"oldcol1_name":"newcol1_name"})
```

Label categorical variables

```
etiquetas = {"col1":{1:"valor_1",
                     2:"valor_2"}} # "1" is the first category inside the column "col1", taking the value "valor_1" as the limited value.
df["col1"].map(etiquetas["col1"]) #.map(), function whos mapping values gived, returning asignned values. (different from .replace() )
```

Empty Data Frame

```
df_vacio = pd.DataFrame(columns = [...])

df_vacio = pd.DataFrame(index = fb.shape[0], columns = [...])

df_vacio = df_vacio.append({"Nombre":"Facebook",
...,
}) #Each key is a new row
```

Retrieving Series / DataFrame Information

Summary

```
df.sum() # Sum of values
df.cumsum() # Cummulative sum of values
df.min()/df.max() # Minimum/Maximum values
df.idmin()/df.idmax() # Minimum / Maximum index value
df.describe() # Summary statistics
df.mean() #Mean of values
df.median() #Median of values
df["var1"].pct_change() # Obtain the percentage change between the observations.
df.std() # Standard Deviation
```

Generate new variables (columns)

```
fb['Price3'] = fb['Close'].shift(-1)
fb["Return"] = fb['PriceDiff']/fb["Close"]


# Generating Direction Variable
fb['Direction'] = [1 if fb.loc[ei,"PriceDiff"]>0 else -1
                   for ei in fb.index]
```

Data Munging

Obtain and drop NA values

```
# Obtain
fb = fb.isnull.sum()

# Drop NA
fb = fb.fillna(method="ffill")
fb = fb.dropna()
# or
fb = fb.dropna(how="any", inplace=True)

# Save the data
fb.to_csv("data/Facebook.csv")
```

Drop missings or outliers

```
# Modo corto con varias variables

categories = ["col0","col2","col4"]
for variables in categorises:
    df[variables]=pd.Categorical(df[variables])
```

Droping columns or rows

Columns

```
df.drop("var_want_delete", axis=1, inplace=True)
```

Rows

```
df.drop(value_want_row_drop, axis=0, inplace=True)
```

Cluster Data

Gruopby | Pandas | Python

```
import pandas as pd
data = pd.read_csv("___.csv")
df = pd.DataFrame(data, index = ___ , columns:_____)

print("Maxima duración de las películas {}".format(df["duration"].max()))

minutes = df["duration"].max()
print("Horas:", minutes/60)

print("La película que más dura es: {}".format(df.get_value(df["duration"].idxmax(),"movie_title"))) #Return the value in column "movie title"
by giving the accurate index.

print("Comparación entre géneros por likes")
df1 = df.groupby(["genres"])[["movie_facebook_likes"]].sum() # Applying the sum() function for each genres by adding the likes of them.
print(df1)

---

# If you want to see how .groupby() splitting a new kind of data frames, use this:
for genres, genres_df in df:
    print(genres)
    print(genres_df)
#It appears the showing of the dataframes of each genres in the original data

#If you want to get just one of the groups generated by using the function groupby() :
df.get_group("terror")

---
```

```
likes = df1
likes["more_facebook_likes"].idmax()
print("con", likes["more_facebook_likes"].max(),"likes")

df2 = df.groupby(["director_name"])[["budget"]].sum()
print(df2)
df2 = df.groupby(["director_name"])[["budget"]].mean()

print("¿Cuál es el género que gasta más dinero?")
df3 = df.groupby(["genre"])[["budget"]].sum()
print("El género es: ", df3["budget"].idmax(), "con: $", df3["budget"].max)
```