Practice 1

Python for ML

Machine Learning – LECD, LEEC
Informatics Engineering Department

Goals:

- Get familiar with Python for ML

Requirements:

- Working Python platform

Support materials:

- Class slides

Part I - Setup

1. First, you should have a Python setup that can be used throughout the semester. Several options exist:
    - Google colab: https://colab.research.google.com
    - Jupyter notebook (either online or locally): https://jupyter.org
    - Local setup with IDE, e.g., Visual Studio Code: https://code.visualstudio.com/download

    If you are setting up a local environment, use Python 3.XX (the latest version is advised) and install the following libraries: scikit-learn, pandas, numpy, scipy, matplotlib and seaborn. Do not do this during class time, if your setup is not ready use google colab and set it up later at home.

Part II – Voting Intentions in the 1988 Chilean Plebiscite

1. Toy dataset: https://bit.ly/CDA_chile
    a. A simple dataset to learn the very basics of data analytics.
    b. 2700 lines, 8 columns
    c. Survey between April and May 1988, missing data
    d. Columns:
        i. **Region**: C, Central; M, Metropolitan Santiago area; N, North; S, South; SA, city of Santiago.
        ii. **Population**: Population size of respondent's community.
        iii. **Sex**: F, female; M, male.
        iv. **Age**: in years.
        v. **Education**: P, Primary; PS, Post-secondary; S, Secondary.
        vi. **Income**: Monthly income, in Pesos.
        vii. **Statusquo**: Scale of support for the status quo.
        viii. **Vote**: A, will abstain; N, will vote no (against Pinochet); U, undecided; Y, will vote yes (for Pinochet).

2. Load and explore the data

Exploratory Data Analysis (EDA) is an approach in statistics and data analysis that focuses on exploring and understanding data sets to summarize their main characteristics. The primary goal of EDA is to gain insights and a deeper understanding of the data's underlying structure and patterns before making any assumptions or testing hypotheses.

We will make use of the **pandas** library to aid in the data manipulation. We will guide you through the main functionalities of the library, but it is important that you explore it beyond the limits of the proposed exercises. This is a key library in the field of machine learning and your ability to use it will make a difference when you approach any data science challenge.

2.1. Import pandas and load the dataset into a dataframe

```
import pandas as pd
url = 'https://bit.ly/CDA_chile'
df = pd.read_csv(url)
```

A dataframe is a two-dimensional data structure that holds data like a two-dimension array or a table with rows and columns. There is an index that works like a virtual column and contains a unique identifier for each row.

2.2. Explore the datafrane structure.

Use `df.info()` to check some information about the dataframe, `df.head()` and `df.tail()` to view the top and bottom rows of the frame respectively. `df.describe()` will give you a descriptive statistical analysis of the numeric columns.

2.3. Accessing data

`df.columns` contain a list of the columns of the dataset, and `df.index` the list of the identifiers of the rows. Learning how to access specific columns, rows or slices is fundamental to handle your dataframe. The rect parenthesis is used for data access and perform a different operation depending on the parameters passed. See:

For a DataFrame, passing a single label selects a column (equivalent to `df.COLNAME`):

```
age = df['age']
# is the same as
age = df.age
```

But passing a slice : selects matching rows:

```
first_10_rows = df[0:10]
```

The `.loc[]` and `.iloc[]` methods are used to access a group of rows and columns by label(s) or a boolean array. `.loc[]` is label-based, which means that you have to specify the name of the rows and columns that you need to filter out. On the other hand, `.iloc[]` is integer index-based, so you have to specify rows and columns by their integer index.

2.3.1. Selecting Columns with `.loc[]`

The `.loc[]` method can be used to select columns by their names. Select the 'age' and 'income' columns from the dataframe.

```
age_income = df.loc[:, ['age', 'income']]
print(age_income)
```

2.3.2. Filtering Rows with `.loc[]`

You can use conditions to filter rows that meet certain criteria with `.loc[]`. Select all rows where the respondent is from the Metropolitan Santiago area ('M') and their age is over 30.

```
# Select rows where 'region' is 'M' and 'age' is greater than 30
metro_over_30 = df.loc[(df['region'] == 'M') & (df['age'] > 30)]
print(metro_over_30)
```

2.3.3. Combined Row and Column Selection with `.loc[]`

`.loc`[] can also be used to select specific rows and columns. Select the 'statusquo' and 'vote' columns for all females in the dataframe.

```
# Select 'statusquo' and 'vote' columns for all females
female_status_vote = df.loc[df['sex'] == 'F', ['statusquo', 'vote']]
print(female_status_vote)
```

2.3.4. Selecting Rows with `.loc[]`

The `.loc[]` method is used to select rows by their integer location. Select the first 50 rows of the dataframe.

```
# Select the first 50 rows
first_50 = df.iloc[:50]
print(first_50)
```

2.3.5. Selecting Rows and Columns with `.loc[]`

With `.loc[]`, you can select specific rows and columns by using their integer locations. Select the first 3 columns of the first 10 rows.

```
# Select the first 3 columns of the first 10 rows
first_10_rows_first_3_cols = df.iloc[:10, :3]
print(first_10_rows_first_3_cols)
```

2.3.6. Complex Indexing with `.loc[]`

You can use a list of integer locations to select specific rows and columns. Select the 100th, 200th, and 300th rows, but only the 'education' and 'income' columns. Remember that in Python, indexing starts at 0, so the 100th row is indexed as 99, the 200th as 199, and so on.

```
# Select 'education' and 'income' for the 100th, 200th, and 300th
rows
selected_rows = df.iloc[[99, 199, 299],
df.columns.get_loc('education'):df.columns.get_loc('income')+1]
print(selected_rows)
```

2.4 Data aggregation

Let's explore various aggregation functions you can use to summarize grouped data.

2.4.1. Simple Grouping and Aggregation
Use the *groupby* method to perform a simple aggregation. Calculate the total 'population' for each 'region'.

```
# Group by 'region' and sum the 'population'
total_population_by_region =
df.groupby('region')['population'].sum()
print(total_population_by_region)
```

2.4.2. Multiple Aggregations on a Single Column
You can apply multiple aggregation functions to a single column after grouping. For each 'education' level, calculate the mean, median, and standard deviation of 'age'.

```
# Group by 'education' and apply multiple aggregation functions to
'age'
age_stats_by_education =
df.groupby('education')['age'].agg(['mean', 'median', 'std'])
print(age_stats_by_education)
```

2.4.3. Different Aggregations for Different Columns
Apply different aggregation functions to different columns after grouping. For each 'sex', find the average 'income' and the maximum 'age'.

```
# Group by 'sex' and apply different aggregations to 'income' and
'age'
income_age_stats_by_sex = df.groupby('sex').agg({'income': 'mean',
'age': 'max'})
print(income_age_stats_by_sex)
```

2.4.4. Grouping by Multiple Columns with Multiple Aggregations
Group by more than one column and perform multiple aggregations. Find the average 'income' and the count of respondents for each combination of 'region' and 'education'.

```
# Group by 'region' and 'education' and calculate the average
'income' and count the respondents
region_education_group = df.groupby(['region',
'education']).agg({'income': 'mean', 'region': 'count'})
region_education_group.rename(columns={'region': 'Count'},
inplace=True)
print(region_education_group)
```

2.4.5. Using Custom Aggregation Functions

You can also use your own custom functions for aggregation. Calculate the range (difference between max and min) of ‘income’ for each ‘region.

```
# Define a custom aggregation function
def income_range(x):
    return x.max() - x.min()

# Group by ‘region’ and apply the custom function to ‘income’
income_range_by_statusquo =
df.groupby(‘region’)[‘income’].agg(income_range)
print(income_range_by_region)
```

2.4.6. Aggregating with Named Aggregations

Use named aggregations to create a DataFrame with new column names directly. For each ‘vote’, calculate the median ‘age’ and the average ‘income’, naming the resulting columns ‘Median Age' and ‘Average Income'.

```
# Group by ‘vote’ and perform named aggregations
vote_stats = df.groupby(‘vote’).agg(
    Median_Age=(‘age’, 'median'),
    Average_Income=(‘income’, 'mean')
)
print(vote_stats)
```

2.4.7. Resetting Index After Grouping

After grouping and aggregating, you may want to reset the index to have a flat DataFrame. Calculate the mean ‘income’ by ‘region’ and reset the index of the resulting DataFrame.

```
# Group by ‘region’, calculate mean ‘income’, and reset the index
mean_income_by_region =
df.groupby(‘region’)[‘income’].mean().reset_index()
print(mean_income_by_region)
```

2.5 Data Visualization

Data visualization is a crucial part of data analysis as it helps to understand trends, patterns, and outliers in the data. Seaborn is a Python data visualization library based on matplotlib that provides a high-level interface for drawing attractive statistical graphics.

Import Seaborn along with matplotlib:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Set a theme for Seaborn plots
sns.set_theme()
```

2.5.1. Scatter Plot

Scatter plots are used to observe relationships between variables. Create a scatter plot to visualize the relationship between ‘age’ and ‘income’.

```
# Scatter plot of ‘age’ vs ‘income’
plt.figure(figsize=(10, 6))
sns.scatterplot(x=‘age’, y=‘income’, data=df)
plt.title('Age vs Income')
plt.show()
```

2.5.2. Pairplot

Pairplots are used to plot pairwise relationships in a dataset. Generate pairplots for the ‘age’, ‘income’, and ‘statusquo’ columns.

```
# Pairplot for ‘age’, ‘income’, and ‘statusquo’
sns.pairplot(df[[‘age’, ‘income’, ‘statusquo’]])
plt.show()
```

2.5.3. Histogram

Histograms are used to plot the distribution of a numeric variable. Create a histogram to see the distribution of ‘age’ in the dataset.

```
# Histogram of ‘age’
plt.figure(figsize=(10, 6))
sns.histplot(df[‘age’], bins=30, kde=False)
plt.title('Age Distribution')
plt.show()
```

2.5.4. Boxplot

Boxplots are used to show the distribution of categorical data and to spot any outliers. Draw a boxplot to compare the ‘income’ distributions across different ‘education’ levels.

```
# Boxplot of ‘income’ for each ‘education’ level
plt.figure(figsize=(10, 6))
sns.boxplot(x=‘education’, y=‘income’, data=df)
plt.title('Income Distribution by Education Level')
plt.show()
```

2.5.5. Cumulative Distribution Function (CDF)

CDFs are used to determine the probability that a variable takes a value less than or equal to a certain value. Plot the CDF for ‘age’ in the dataset.

```
# CDF of ‘age’
plt.figure(figsize=(10, 6))
sns.histplot(df[‘age’], bins=30, cumulative=True, stat='density',
kde=True)
plt.title('Cumulative Distribution Function for Age')
plt.show()
```

2.5.6. Heatmap

Heatmaps are used to visualize matrix-like data and can be particularly useful for displaying correlation matrices. Create a heatmap to show the correlation matrix of the numerical variables in the dataframe.

```
# Correlation matrix heatmap
plt.figure(figsize=(10, 6))
correlation_matrix = df.loc[:, ('population', 'age', 'income',
'statusquo')].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```

2.7 Class Quiz

Go to the link that was made available in class and answer the questions based on your analysis of the data.

2.8 Alternative ways of accessing datasets

2.8.1 Libraries

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()
dataF = pd.DataFrame(iris["data"], columns=iris["feature_names"])
dataF['species']=iris["target"]
```

2.8.2 Google Drive and Colab

```
from google.colab import drive
drive.mount('/content/drive')

data = pd.read_csv('/content/drive/My Drive/.../iris.csv')
```

2.9 Additional training

More details on python for exploratory data analysis:
https://www.kaggle.com/ekami66/detailed-exploratory-data-analysis-with-python

Part III – Other examples

1. Experiment with different datasets:
    a. Iris dataset, small famous classic ML dataset
    https://archive.ics.uci.edu/dataset/53/iris or https://bit.ly/ML_iris

b. Haberman Cancer Survival dataset from Kaggle
https://www.kaggle.com/gilsousa/habermans-survival-data-set

c. The number of awards earned by students at one high school. Predictors of the number of awards earned include the type of program in which the student was enrolled (e.g., vocational, general or academic) and the score on their final exam in math. This data set is used to show an example of the Poisson Regression. The predicted variable is the number of awards and the predictors are the program type and the Maths score. https://bit.ly/CDA_awards

Variable name	Variable	Data type
id	Participant number	
Num_awards	Number of awards	Scale
prog	Program type	Nominal
math	Maths Score	Scale

d. https://www.sheffield.ac.uk/mash/statistics/datasets

e. https://archive.ics.uci.edu/ml/index.php

f. https://www.kaggle.com/robstepanyan/murder-rates-by-states

g. https://research.google/tools/datasets/