Cheatsheet

Keyword

predefined or reserved words used in programming that have special meanings to the compiler(the things that translates your code to the computer to understand)

Variables

--> used to store any type of info or value

Things to keep in mind when creating variable names:

1) No spaces allowed
--> test var **{not proper variable name}**

2) Can **only** include letters, numbers, and underscores
--> test.var! **{not proper variable name}**

3) have to start with a letter or underscore
--> 1var **{not proper variable name}**

Printing

print("Hello World")

Commenting

--> to make a comment use # and then on that line write anything afterward

this is a comment

Indexing

--> to directly access your elements of choice from a sequence of elements
--> index is the position of an element, ranges from 0 to length of list minus 1
--> use brackets, then inside brackets put start index (inclusive), then colon, and then put end index (exclusive)
EX:
var_name = [1, 2, 3, 4, 5]
var_name[0:2]
--> [1, 2]

Data Types

--> variable is a "named storage" for data
--> each data value stored has a type

Types of Data:

- **Integer**
  - numerical value *(**without** decimal value)*
  - ***ex:*** *-6*
- **Float**
  - numerical value *(**with** decimal value)*
  - ***ex:*** *3.14*
- **Boolean**
  - value of *True* or *False*
  - ***ex:*** True
- **String**
  - sequence of characters between quotation marks
  - **ex:** "data activism"
- **List**
  - sequence of values between brackets
  - **ex:** [1, 2, 65, -75, 252, 9, 7]

Comparison Operators

--> used to compare two values

Operators:

- **equals -->** ==
  - **ex:** x == y
- **not equal -->** !=
  - **ex:** x != y
- **greater than -->** >
  - **ex:** x > y
- **less than -->** <
  - **ex:** x < y
- **greater than or equal to -->** >=
  - **ex:** x >= y
- **less than or equal to -->** <=
  - **ex:** x<=y

Imports

--> must import specific libraries *(collection of files, programs, or functions that can be used in code)*

Names of Libraries to Import:

- pandas
- seaborn
- numpy
- matplotlib.pyplot

ex: import pandas as pd

How to Debug

--> fixing errors in your code

Common Type of Errors:

- **syntax error**
  - improper syntax or structuring of code
  - ***fixes:***
    - check the syntax of your code, i.e. make sure you have opening and closing parentheses
    - make sure you put a colon at the end of if/elif/else statements, for/while
- **logic error**
  - when program can run but does not do what it is expected to do
  - **fixes:**
    - check the calculations you make in your program
    - check that in your if/elif/else statements that you are comparing values using **==**
- **name error**
  - when an undefined variable name is used or improper syntax for string
  - **fixes:**
    - check if you spelled the variable name the same each time you reference it
      - check if the capitalization is the same
      - check if you misspelled the variable
      - make sure there are *ONLY* underscores, letters, or numbers (the number must be after an underscore or alphabet character) in a variable name
        - **ie:** *NO* periods, front/back slashes, exclamations, question marks, hyphens, asterisks, etc
    - check for any strings without opening and closing quotes; strings look like this **-->** **"**hello world**"**
    - check if you remember to import a specific library, i.e. pandas, maplotlib.pyplot, numpy, etc
    - check to make sure you ran previous DeepNote cells
- **indentation error**
  - improper indentation in code
  - ***fixes:***
    - check if you wrote lines of code for an if/elif/else statement or a for/while loop
    - check if you mistakenly indented something
    - check if you forgot to indent code inside if/elif/else statement or a for/while loop

Websites to help debug code, and learn programming

- Stackoverflow
- greeksforgreeks
- tutorialspoint
- YouTube
- Codecademy
- kaggle
- khanacademy

Correlation

--> statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate)

Types of Correlation:

- **positive**
  - **ex:** The longer your hair grows, the more shampoo you will need
- **negative**
  - **ex:** climb the mountain (increase in height) it gets colder (decrease in temperature)
- **zero**
  - no relationship between the two variables

--> python uses the function **corr()** to calculate correlation

Co-Variance

--> measures the direction of the relationship between two variables.

- **Positive** covariance means that both variables tend to be high or low at the same time
- **Negative** covariance means that when one variable is high, the other tends to be low or vice versa
- **zero** covariance means that the two variables are independent of each other

--> python uses the function **covar()** to calculate covariance

modified Retail Food Environment Index (mrfei)

--> is a way of measuring the number of healthy and less healthy food retailers within a census tract using a single number

--> ranges from 0 to 100 and was calculated as the number of healthy food retailers divided by the sum of the number of healthy food retailers and less healthy food retailers and multiplied by 100

--> Lower scores indicate that census tracts contain a higher number of less healthy retailers than healthy retailers

census tract: are small, relatively permanent statistical subdivisions of a county

--> in the food deserts/swamps DeepNote activity we are trying to see if there is a correlation between mrfei value and median household income to see if lower income areas have less access to healthy food or grocery stores

Built-in functions

function: group of related statements that performs a specific task. Functions help break our program into smaller and modular chunks

built-in function: functions that have already been created as a part of the Python programming language

--> any person programming in python can use them

- To call a Python function at a place in your code, you simply need to name it, and pass arguments

Useful built-in functions:

- print()
- len()
- sum()
- range()
- max()
- min()

User-defined functions

user-defined function: a function that is created by a programmer to perform a specific task in their specific program

How to create a user-defined function:

```
def function_name(parameters):
        body of code for the function to perform
        return a variable or value
```

Ex:

```
# function to subtract variables passed into it
def diff_variable(var1, var2, var3):
        difference = var1 - var2 - var3
        return difference
var1 = 12
var2 = 4
var3 = 5
var4 = diff_variable(var1, var2, var3)
```

NOTE

--> before using any of these functions make sure to import matplotlib.pyplot

Syntax:

import matplotlib.pyplot **as** plt

plt.barh()

--> plots the provided data on a horizontal bar graph

Syntax:

plt.barh(data)

plt.bar()

--> plots the provided data on a typical bar graph

Syntax:

plt.bar()

plt.figure()

plt.figure()

plt.title()

--> to put a title for your graph use this function

Syntax:

plt.title()

plt.show()

--> to make sure your graphs are outputted use this function

Syntax:

plt.show()

Steps to Make Pie Chart

- make sure to import a library that can plot data, i.e **matplotlib.pyplot** has been imported
- have a sequence of x-values
  - your x-values could be stored in **DataFrame column** or **list**
- create a list to store the **labels** for your **x values**
- create a variable for **title** of the graph
- **Syntax:**
  - **import** matplotlib.pyplot **as** plt
  - plt.pie(x_vals, labels= x_labels)
  - plt.title(title_name)
  - plt.show()

Pandas Concepts

Creating a DataFrame

df = pd.DataFrame({"column1":[1,2], "column2":[3,4] })

Boolean Indexing

To index specific rows of a DataFrame for which the values within a certain column match a certain value, follow this format:

df[df["column_name"] == descriptor]

In this case descriptor is one of the values from the column you have selected

sum()

--> adds all values in each row of a column and returns the sum for each column

Syntax:

sum(df["column_name"])

len()

--> finds the length of sequence of elments

Syntax:

len(df["column_name"])

Column selection in DataFrame

df["column_name"]
df.loc[: , "column_name"]
df.iloc[:, column_index]

.size

--> finds the number of all the elements in the DataFrame; think of it like the area of the DataFrame

Syntax:

df.size

.shape

--> finds the dimensions of the DataFrame; will return the number of rows and columns

--> *(num_row, num_column)*

Syntax:

df.shape

.value_counts()

--> goes through each row in DataFrame and counts the number of unique values in a specific column; will need to use column selection in order to use this built-in function

Syntax:

df["column_name"].value_counts()

.fillna()

--> replaces all the data values that have no input with a value indicated by the programmer

Syntax:

df.fillna(value)

Changing an entire column of values

df["column_name"] = value

.dtypes

--> finds the data type for each column in DataFrame

Syntax:

df.dtype

.iterrows()

--> Iterates over DataFrame rows as (index, Series**-->** *one column or one row in a DataFrame*) pairs.

Syntax:

df.iterrows()

dictionary

--> type of data that is used to store data values in **key:value** pairs

--> They are similar to DataFrames

--> *values* can be any data type, **i.e.** they can be a list, int, float, String, etc

Syntax:

dict_name = {key1: value2, key2: value2}

REMINDER

--> if DeepNote says variable undefined make sure to **rerun** notebook

Get a dataframe with only specific values: Selecting data

Create: Create a data set that only contains values that are equal to 'Risk of Recidivism' and save it to a variable named "compas_ds"

[7]

```
compas_ds= compas_ds[compas_ds["DisplayText"] == 'Risk of Recidivism']
compas_ds
```

See the count of each value in a column

Syntax

Syntax:

dataframe**["column_name"].value_counts()**

name of dataframe

name of the column you want to analyze

Series: a single column in DataFrame

column_name
origingal_value
origingal_value
origingal_value

Produces a **Series** of unique data values and the number of instances of those unique data values

Example Code

Create: Use value_counts() on the "category" column of the "first" Data

```
# type code here
firsts["category"].value_counts()
```

```
Arts & Entertainment    107
Education & Science      87
Politics                 82
Military                 73
Social & Jobs            57
Sports                   38
Religion                 21
Law                      14
Name: category, dtype: int64
```

See the percentage of each value in a column

Example Code

Create: Use the dataframe named "compas_ds" and the value_counts(normalize=True) function to get the percentage of people in each ethnic group from the "Ethnic_Code_Text" column

```
# Count percentage of different ethnic groups


compas_ds["Ethnic_Code_Text"].value_counts(normalize=True) * 100
```

```
African-American    44.434577
Caucasian           35.824946
Hispanic            14.377344
Other                4.262877
Asian                0.532860
Native American      0.360174
Arab                 0.123347
African-Am           0.083876
Name: Ethnic_Code_Text, dtype: float64
```

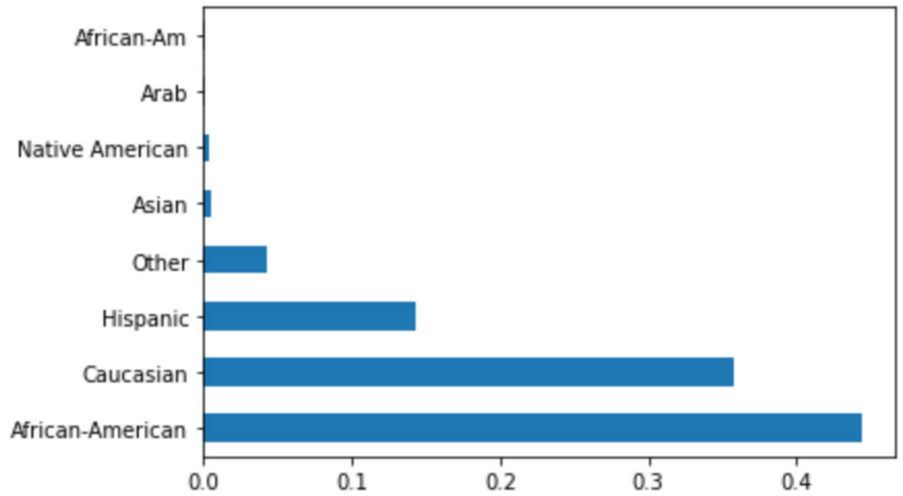
Convert your value counts series into a bar chart

Example Code

Create: Visualize the percentage of each ethnic groups from the "Ethnic_Code_Text" column

```
# Visualize percentage of different ethnic groups
compas_ds["Ethnic_Code_Text"].value_counts(normalize=True).plot(kind='barh')
```

```
<AxesSubplot:>
```



See the mean of a column

Mean Syntax

Syntax:

```
dataframe["column_name"].mean()
```

- name of dataframe
- name of the column you want to analyze

column_name
origingal_value
origingal_value
origingal_value

Finds the mean of all the data values in a specific column

The column you want to find the mean of must have data values that are integers

Example Code

Use:

```
firsts.year.mean()
```

```
1951.0313152400836
```

See a high-level summary of the attributes of the given column

Syntax

Syntax:

```
dataframe["column_name"].describe()
```

- name of dataframe
- name of the column you want to analyze

column_name
origingal_value
origingal_value
origingal_value

Produces a Series of a high-level summary of the attributes of the given column;

If the data in the column are integers then the describe function will produce a Series with a summary of statistical information

If the data in the column are Strings then the describe function will produce a series with a summary of the String values

Example Code

```
firsts.year.describe()
firsts["year"].describe()
```

```
count     479.000000
mean     1951.031315
std        54.519747
min      1738.000000
25%      1930.000000
50%      1965.000000
75%      1990.500000
max      2019.000000
Name: year, dtype: float64
```

Syntax for Bar Chart with Matplotlib:

```
#Import the matplotlib library
from matplotlib import pyplot as plt
# Figure Size
fig = plt.figure(figsize =(10, 7))
 # Horizontal Bar Plot
plt.bar(["bar_name1","bar_name2"],[variable1, variable2])
```

label names that will appear under each bar on the x axis

values that will be represented on the y axis and correspond to the label names

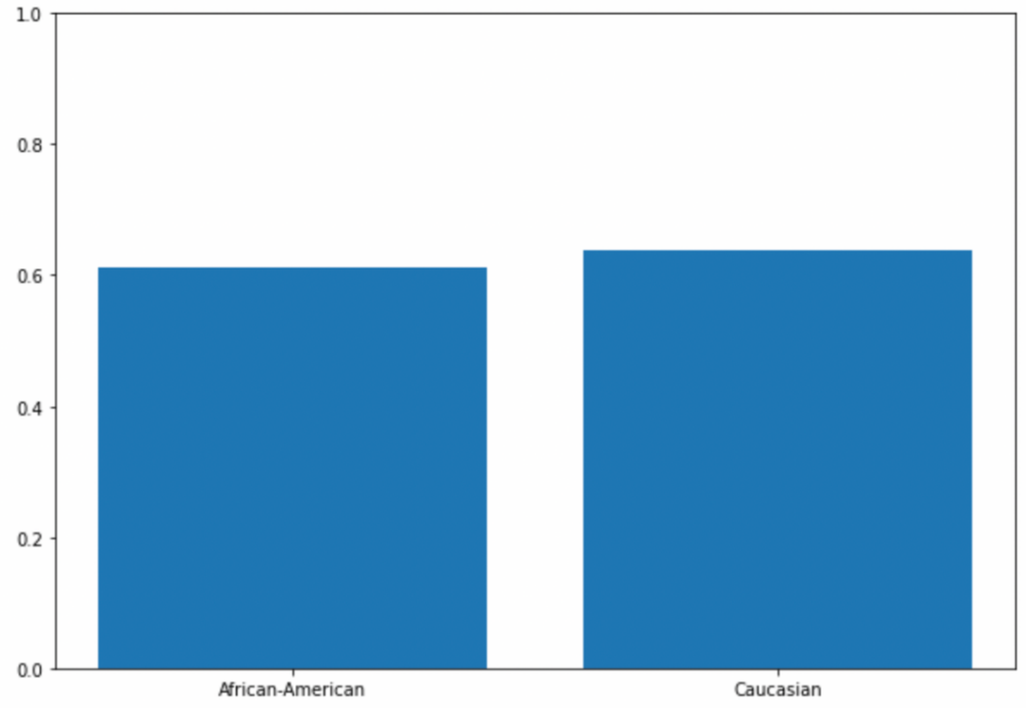
```
plt.title("title_name")
plt.xlabel('x axis label')
plt.ylabel('y-axis label')

# Show Plot
plt.show()
```

Ex.

```
from matplotlib import pyplot as plt
# Figure Size
fig = plt.figure(figsize =(10, 7))
# Horizontal Bar Plot
plt.bar(["African-American","Caucasian"],[aa_accuracy, white_accuracy])
plt.ylim(top=1)
```

```
(0.0, 1.0)
```



Create a Bar Chart Using a Data Set

Creating a bar chart to show the number of achievements for each category

In this tutorial, you'll learn about bar charts and how to use them to create a bar chart to represent how many achievements are made in each category.

Here is what our finished result should look like:

[15

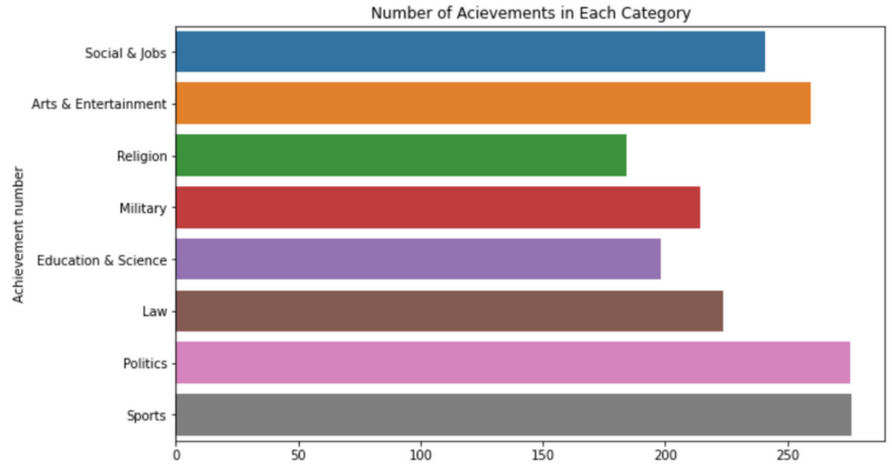
```
# Set the width and height of the figure
plt.figure(figsize=(10,6))

# Add title
plt.title("Number of Acievements in Each Category")

# Select the x and Y axes
sns.barplot(data=firsts, x=firsts.index, y=firsts['category'],ci=None)

# Add label for vertical axis
plt.ylabel("Achievement number")
```

```
Text(0, 0.5, 'Achievement number')
```



Create a frequency table of 2 factors

crosstab(shootings_ds['Victim_Race'],shootings_ds['District'])

shootings_ds → pd.crosstab()

Incident_Num	Victim_Race	District
I152029808-00	White	Brighton
I152034171-00	Black	Dorchester
I152036137-00	White	Brighton

shootings_ds displays our data

Victim_Race	Dorchester	Brighton
Asian	0	0
Black	1	0
White	0	2

crosstab() calculates the relationship between two provided columns

[16]

```
#For example, let's examine the connection between dsitricts and the race of the victim
pd.crosstab(shootings_ds['Victim_Race'],shootings_ds['District'])
```

Visualize

	Brighton int64	Charlestown int…	Dorchester int64	Downtown int64	East Boston int…	Hyde Park int64	Jamaica P
Asian	0	0	0	1	0	0	
Black or …	11	4	234	13	13	65	
Unknown	2	1	8	1	5	3	
White	7	6	39	3	12	14	

Group data according to categories

(6.1) Using the groupby and size functions

The `.groupby()` functions allows us to group the data according to the categories and apply a function to the categories. It also helps to aggregate data efficiently.

In this case we have been exploring the sex of the victims so we can try and see the sex of the victims with another intersection.

The `.size()` function allows us to get a integer value as it returns an int representing the number of elements in this object.

By combining these two functions and the `.unstack()` function we can graph **intersectional** data points, as these functions allows us to explore multiple facets of data.

[34]

```
shootings_ds_graph = shootings_ds.groupby(['Victim_Gender','Multi_Victim']).size()
shootings_ds_graph
```

```
Victim_Gender  Multi_Victim
Female         f                99
               t                72
Male           f              1073
               t               407
dtype: int64
```

Create a Line Chart Using a Data Set

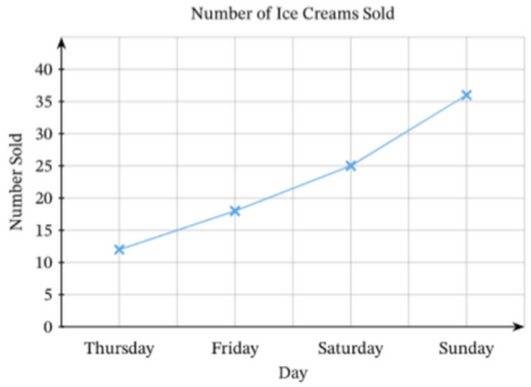
Line plot

Syntax:

```
sns.lineplot(data= dataframe, x="column_name", y="column_name")
```

lineplot function in seaborn library that creates a line graph

you have to specify which dataframe you will be using



outputs a line graph with specified x and y columns

Example Code

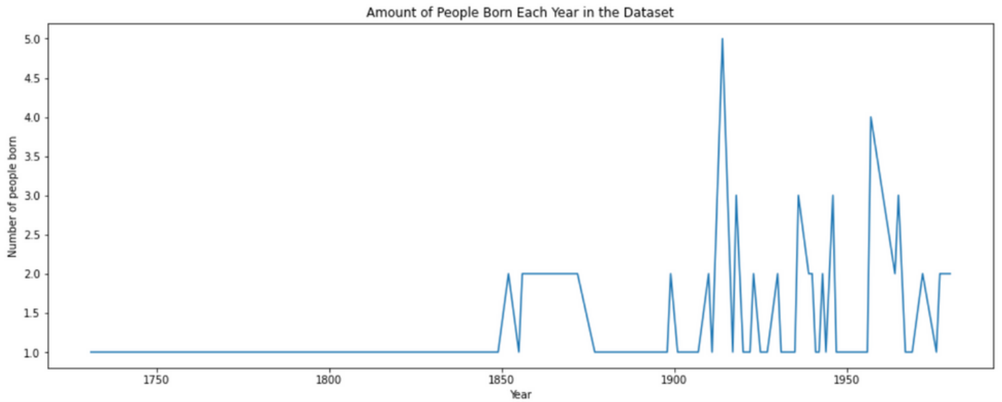
```
# Set the width and height of the figure
# width --> 16 inches
# height --> 6 inches
plt.figure(figsize=(16,6))

# set the title of the graph
plt.title("Amount of People Born Each Year in the Dataset")

# set the x and y axis labels of the graph
plt.xlabel("Year")
plt.ylabel("Number of people born")

# Line chart showing how the year of birth for people in the data set changed over time
sns.lineplot(data=birth_count_df, x="index", y="birth")

# show plot
plt.show()
```



Create a histogram that shows the distribution of values

Example Code

```
# Histogram of decile scores grouped by ethinc group
smaller_ds.groupby('Ethnic_Code_Text')["DecileScore"].plot(kind='hist', alpha=0.5
, legend=True, figsize=(14,6))
```

```
Ethnic_Code_Text
African-American    AxesSubplot(0.125,0.125;0.775x0.755)
Caucasian           AxesSubplot(0.125,0.125;0.775x0.755)
Name: DecileScore, dtype: object
```