

Week 6 lab

Adapted from slides in COGS108 FA22



AGENDA FOR TODAY



01

LOGISTICS

02

Sentiment Analysis

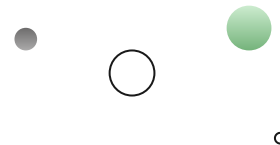


03

**DISCUSSION
LAB 5**



LOGISTICS



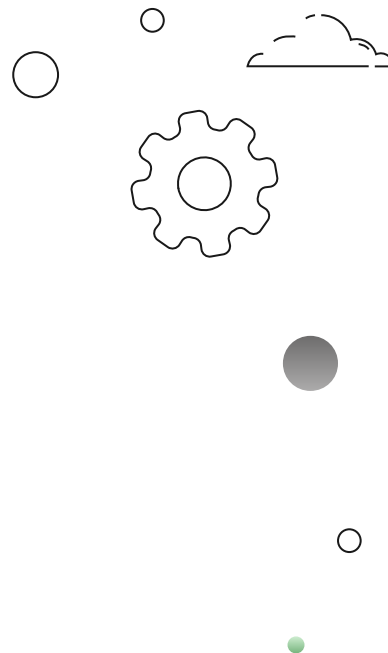
DUE DATES

- A2 is due Friday, May 10th
- Q6 is due Monday, May 13th
- D5 is due Monday, May 13th
- Project CheckPoint #1 coming up on May 17th
 - Understand the feedback received in the project proposal.
 - Use TA/Professor OH to discuss on the feedback and next steps.

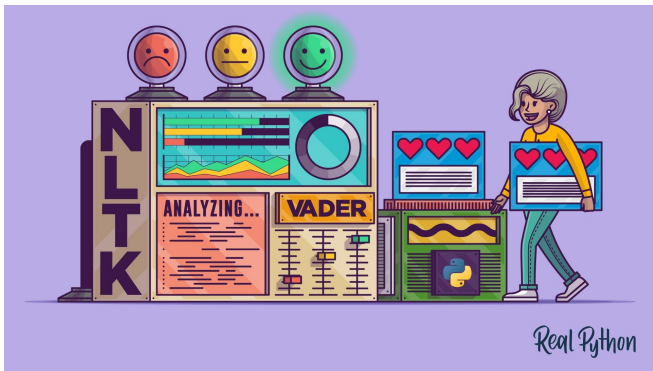
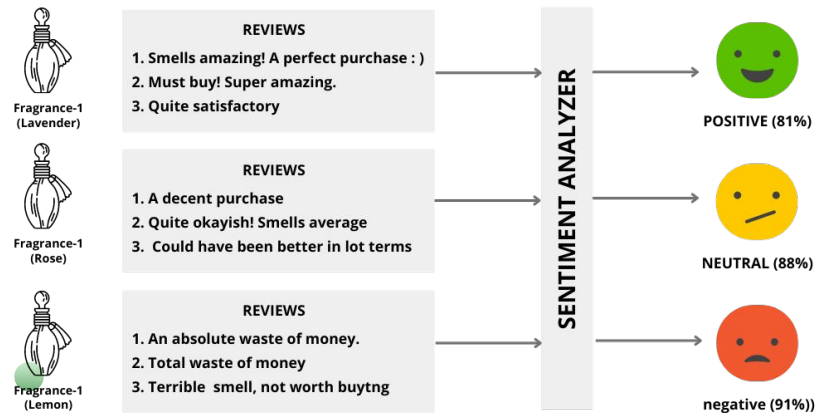




Sentiment Analysis



Sentiment Analysis





DISCUSSION LAB 5

Inference



Part I : Data & Wrangling

```
df['height'] = df['height'].apply(function)
```

The `apply()` method allows you to apply a function along one of the axis of the DataFrame, default 0, which is the index (row) axis.

Syntax

```
dataframe.apply(func, axis, raw, result_type, args, kwds)
```

Parameters

The `axis`, `raw`, `result_type`, and `args` parameters are keyword arguments.

Parameter	Value	Description
<i>func</i>		Required. A function to apply to the DataFrame.
axis	0 1 'index' 'columns'	Optional, Which axis to apply the function to. default 0.
raw	True False	Optional, default False. Set to true if the row/column should be passed as an ndarray object
result_type	'expand' 'reduce' 'broadcast' None	Optional, default None. Specifies how the result will be returned
args	<i>a tuple</i>	Optional, arguments to send into the function
kwds	<i>keyword arguments</i>	Optional, keyword arguments to send into the function

Part II : EDA

fig =
scatter_matrix(df[['column
_1', column_2']])

A scatter matrix is useful because it allows you to visualize the relationship between multiple variables in a dataset at once.

pandas.plotting.scatter_matrix

```
pandas.plotting.scatter_matrix(frame, alpha=0.5, figsize=None, ax=None,  
grid=False, diagonal='hist', marker='.', density_kws=None, hist_kws=None,  
range_padding=0.05, **kwargs)
```

[source]

Draw a matrix of scatter plots.

Parameters:

frame : DataFrame

alpha : float, optional

Amount of transparency applied.

figsize : (float,float), optional

A tuple (width, height) in inches.

ax : Matplotlib axis object, optional

grid : bool, optional

Setting this to True will show the grid.

diagonal : ('hist', 'kde')

Pick between 'kde' and 'hist' for either Kernel Density Estimation or Histogram plot in the diagonal.

marker : str, optional

Matplotlib marker type, default '.'.

density_kws : keywords

Keyword arguments to be passed to kernel density estimate plot.

hist_kws : keywords

Keyword arguments to be passed to hist function.

range_padding : float, default 0.05

Relative extension of axis range in x and y with respect to (x_max - x_min) or (y_max - y_min).

****kwargs**

Keyword arguments to be passed to scatter function.

Returns:

numpy.ndarray

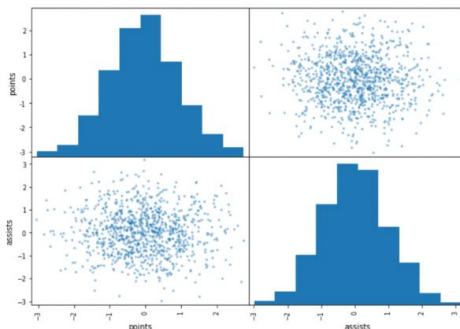
A matrix of scatter plots.

```
import pandas as pd  
import numpy as np  
  
#make this example reproducible  
np.random.seed(0)  
  
#create DataFrame  
df = pd.DataFrame({'points': np.random.randn(1000),  
                   'assists': np.random.randn(1000),  
                   'rebounds': np.random.randn(1000)})
```

```
#view first five rows of DataFrame  
df.head()
```

	points	assists	rebounds
0	1.764052	0.555963	-1.532921
1	0.400157	0.892474	-1.711970
2	0.978738	-0.422315	0.046135
3	2.240893	0.104714	-0.958374
4	1.867558	0.228053	-0.008812

```
pd.plotting.scatter_matrix(df.iloc[:, 0:2])
```



Part III : ttest_ind

`t_val, p_val = ttest_ind(df1, df_2)`

`ttest_ind` used to check whether the unknown population means of given pair of groups are equal.

It allows one to test the null hypothesis that the means of two groups are equal

`scipy.stats.ttest_ind`

```
scipy.stats.ttest_ind(a, b, axis=0, equal_var=True, nan_policy='propagate',  
permutations=None, random_state=None, alternative='two-sided', trim=0, *, keepdims=False)  
[source]
```

Calculate the T-test for the means of *two independent* samples of scores.

This is a test for the null hypothesis that 2 independent samples have identical average (expected) values. This test assumes that the populations have identical variances by default.

Parameters: `a, b` : *array_like*

The arrays must have the same shape, except in the dimension corresponding to `axis` (the first, by default).

`axis` : *int or None, default: 0*

If an `int`, the axis of the input along which to compute the statistic. The statistic of each axis-slice (e.g. row) of the input will appear in a corresponding element of the output. If `None`, the input will be raveled before computing the statistic.

`equal_var` : *bool, optional*

If `True` (default), perform a standard independent 2 sample test that assumes equal population variances [1]. If `False`, perform Welch's t-test, which does not assume equal population variance [2].

1 New in version 0.11.0.

`nan_policy` : *{'propagate', 'omit', 'raise'}*

Defines how to handle input NaNs.

- `propagate`: if a NaN is present in the axis slice (e.g. row) along which the statistic is computed, the corresponding entry of the output will be NaN.
- `omit`: NaNs will be omitted when performing the calculation. If insufficient data remains in the axis slice along which the statistic is computed, the corresponding entry of the output will be NaN.
- `raise`: if a NaN is present, a `ValueError` will be raised.

Part III : patsy.dmatrices

```
outcome_1, predictors_1 =  
patsy.dmatrices('y ~ x', df_sub)
```

Patsy uses R-style formulas to define the model and takes care of transforming the data for you.

We can pass the relation among variables as strings.

For Ex.

'y~x'

Or

'y~ x1 + x2 + b*x3'

4. Patsy

Patsy[↗] is a neat API to transform your data into experimentation model form. For regression and classification problems, you often want your data in the $\begin{bmatrix} \mathbf{x} & \mathbf{y} \end{bmatrix}$ form where \mathbf{x} is a matrix (independent variable) and \mathbf{y} is a column vector (dependent variable). In regression, let's say you have x_1, x_2 as your independent variable and y as your dependent variable. You might want to express the possible models as follows.

- $y = b_0 + x_1 + x_2$
- $y = b_0 + x_1 + x_2 + x_1x_2$
- $y = b_0 + x_1 + x_2 + x_1^2 + x_2^2$
- $y = b_0 + x_1 + x_2 + x_1^2 + x_2^2 + x_1x_2$

```
[3]: from patsy import dmatrices  
  
formula = 'y ~ height + weight + I(height**2) + I(weight**2) + height:weight'  
y, X = dmatrices(formula, df, return_type='dataframe')  
  
X
```

```
[3]:
```

	Intercept	height	weight	I(height ** 2)	I(weight ** 2)	height:weight
0	1.0	10.0	88.0	100.0	7744.0	880.0
1	1.0	20.0	99.0	400.0	9801.0	1980.0
2	1.0	30.0	125.0	900.0	15625.0	3750.0
3	1.0	40.0	155.0	1600.0	24025.0	6200.0
4	1.0	50.0	120.0	2500.0	14400.0	6000.0

Part III : statsmodels.regression.linear_model.OLS

```
mod_1 = sm.OLS(outcome_1, predictors_1)
res_1 = mod_1.fit()
```

Introduction :

A linear regression model establishes the relation between a dependent variable(**y**) and at least one independent variable(**x**) as :

$$\hat{y} = b_1x + b_0$$

In *OLS* method, we have to choose the values of b_1 and b_0 such that, the total sum of squares of the difference between the calculated and observed values of y , is minimised.

Formula for OLS:

$$S = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_{i=1}^n (y_i - b_1x_i - b_0)^2 = \sum_{i=1}^n (\epsilon_i)^2 = \min$$

Where,

\hat{y}_i = predicted value for the i th observation

y_i = actual value for the i th observation

ϵ_i = error/residual for the i th observation

n = total number of observations

To get the values of b_0 and b_1 which minimise S , we can take a partial derivative for each coefficient and equate it to zero.