# Data Visualization Micro Course

%matplotlib inline - magic command to display matplotlib plots within the notebook.

- import seaborn as sns
  ↓ alias for seaborn

- pd.read_csv - pandas fun$^n$ that reads data from a CSV file & converts it into a dataframe

- lines starting with `#` in python are called Comments & are ignored by the computer when the code is run.

- index_col="Date" - when we load the dataset, we want each entry in the first column to denote a different row. To do this, we set the value of index_col to the name of the first column.

- parse_dates=True - This tells the notebook to understand the each row label as a date (as opposed to a number or other text with a different meaning).

- df.head() - Prints the first 5 rows of the dataframe
  df.head(n) - Prints the first n rows of the dataframe.

## Set the width & height of the figure:-

```
plt.figure(figsize=(w,h))
```
w → weight
h → height

## Line chart showing the data over a period of time:-

```
sns.lineplot(data=df)
```

- list(df.columns) - displays a list of all the columns of a dataframe.

- plt.title("title") - sets the title for the plot.
- sns.lineplot(data=df[col], label=col) - displays l... chart containing data from only df[col] with leg... 'col'.
- plt.xlabel() - set the label for x-axis
- plt.ylabel() - set the label for y-axis
- sns.barplot - This tells the notebook that we want to create a bar chart.
- x=flight_data.index - This determines what to use on the horizontal axis. In this case, we've selected the column that indexes the rows.
- y= flight_data['NK'] - This sets the col. in the data that will be used to determine height of each bar.

Note:- You must select indexing col with flight_data.index & it's not possible to use flight_data['Month'] (which will return an error). This is because when we loaded the dataset, the "Month" col. was used to index the rows. we always have to use this special notation to select the indexing column.

- sns.heatmap - This tells the notebook that we want to create a heatmap.
- data=flight_data - This tells the notebook to use all entries in flight_data to create heatmap
- annot=True - This ensures that value appears for each cell on the chart.

## Scatter plots :-

- sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'])

- sns.regplot(x=insurance_data['bmi'], y=insurance_data['charges'])
  - Adds regression line (line that best fits data)

## Colour-Coded scatter plots :-

- sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'], hue=insurance_data['smoker'])

- sns.lmplot(x='bmi', y='charges', hue='smoker', data=insurance_data)
  - can be used to fit regression line for each class in the data.

- sns.swarmplot(x=insurance_data['smoker'], y=insurance_data['charges'])
  - ↳ used to adapt the design of scatter plot to feature a categorical variable (like "smoker") one one of the main axes, we refer to this type of plot as "categorical scatter plot" & we build it with sns.swarmplot command.

# Histograms & Density Plots :-

- sns.distplot(a=iris_data['Petal length (cm)'], kde=False)
  - ↳ creates a histogram of petal lengths of iris flowers

- sns.kdeplot(data=iris_dataset['Petal length (cm)'], shade=True)
  - ↳ Creates a kde plot which is like a smoothened histogram.
  - ↓ colors the area under the curve.

- sns.jointplot(x=iris_data['Petal length (cm)'], y=iris_data['Sepal width (cm)'], kind='kde')
  - ↳ creates a 2d KDE plot where color coding shows us how likely we are to see diff. combinations of Sepal width & Petal length, where darker parts of the figure are more likely

- sns.set_style(theme) - used to set custom style theme takes on 5 values → 1) darkgrid 2) whitegrid 3) dark 4) white 5) ticks.

Choosing plot types & Custom Styles:- Since, it's not always easy to decide how to best tell the story behind your data, we've broken the chart types into three broad Categories to help with this.

-Trends:- A trend is defined as a pattern of change

- sns.lineplot :- Line charts are best to show trends over a period of time & multiple lines can be used to show trends in more than one group.

-Relationship:- There are many different chart types that you can use to understand relationships b/w various variables in your data

- sns.barplot - Bar charts are useful for Comparing quantities Corr. to diff. groups.
- sns.heatmap - Heat maps can be used to find Color-Coded patterns in tables of numbers.
- sns.scatterplot - Scatterplots show the relationship b/w 2 Continuous variables; if Color-Coded, we can also show relationship with a 3rd Categorical Variable
- sns.regplot - Including a regression line in the scatter plot makes it easier to see any linear relationship b/w 2 variables
- sns.lmplot - This Command is useful to draw multiple regression lines, if the scatter plot Contains multiple Color-Coded groups.

- sns.swarmplot - Categorical scatter plots show the relation btw a Continuous variable & a Categorical variable.

-Distribution :- We visualize the distributions to show the possible values that we can expect to see in a variable, along with how likely they are.

- sns.distplot - Histograms show the distribution of a single numerical variable.
- sns.kdeplot - KDE plots (or 2D KDE plots) show an estimated, smooth distribution of a single numerical variable (or two numerical variables).
- sns.jointplot - This command is useful for simultaneously displaying a 2D KDE plot with the Corr. KDE plots for each individual variable.

## One-Hot Encoding (OHE):-

- Categorical data is data that takes only a limited no. of values
- One Hot encoding is a popular method to encode Categorical variables before feeding it to the ML model.
- One Hot encoding works very well unless your Categorical variable takes on a large no. of values (>15 values)
- OHE creates new (binary) columns, indicating the presence of each possible value from the ~~given~~ original data.
- Pandas assigns a data type to each column (or) Series depending on the type of values contained in the columns (or) Series.
- 'Object' dtype indicates col. has text.

- Pandas offers get_dummies to OHE categorical variables.

Ex: OHE_train_predictors = pd.get_dummies(train_predictors)

- Another approach is to drop all the categorical variables.
- OHE usually helps, but it varies on a case by case basis.

## Applying OHE to Multiple Files:-

- Sklearn is sensitive to ordering of columns, so if the training dataset & test dataset get misaligned your results will be Nonsense. This could happen if a categorical had a diff # values in the training data vs the test data.
- Ensure test data is encoded in the same manner as training data, with the align command

Ex:-

```
OHE_train = pd.get_dummies(train)
OHE_test = pd.get_dummies(test)
f_train, f_test = OHE_train.align(OHE_test, join='left', axis=1)
```

performs left join in SQL