BUSAN300 Break Revision

Week 2

Numpy

NumPy stands for Numerical Python, it is the fundamental package in Python for high-performance computing and data analysis. NumPy arrays are important because they enable vectorisation, that is, to apply a batch of operations to an entire array without writing loops.

Creating a NumPy Array

import numpy as np a = np.array([10, 20, 30, 40, 50]) print(a) print(type(a)) [10 20 30 40 50] <class 'numpy.ndarray'>	import numpy as np data = [10, 20, 30, 40, 50] array1 = np.array(data) print(array1.dtype) print(array1.shape) int64 (5,)	import numpy as np data = [10, 20.5, 30, 40, 50] array1 = np.array(data) print(array1) print(array.dtype) [10. 20.5 30. 40. 50.] float64
array = np.zeros((2,3)) [[0. 0. 0.] [0. 0. 0.]]	array = np.ones((2,3)) [[1. 1. 1.] [1. 1. 1.]]	array = np.eye(3) [[1. 0. 0.] [0. 1. 0.] [0. 0. 1.]]
array = np.arange(0, 10, 2) [0, 2, 4, 6, 8]	array = np.random.randint(0, 10, (3,3)) [[6 4 3] [1 5 6] [9 8 5]]	array = np.ones((2,3),dtype=np.int32) [[1 1 1] [1 1 1]]

Array type conversion

```
array = np.array([[2.5, 3.8, 1.5], [4.7, 2.9, 1.56]])
b = array.astype('int')
print(b)
[[2 3 1]
 [4 2 1]]
```

Arange() to create a 2D array

```
array = np.arange(20).reshape(4,5)
print(array)
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9, 9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
print(array[3,4])
19
```

Full function

```
np.full((2,2),3)            # Creates a 2x2 array with all elements to be '3'
array([[3, 3],[3, 3]])
```

Index Slicing

In index living, the comma separates from the row to the column. Hence everything before the column refers to the rows, and everything after the comma refers to the columns.

0	1	2	3	4	array[0:2,:]	array[2:1,:]
5	6	7	8	9		
10	11	12	13	14		

0	1	2	3	4	array[:2, 2:3]
5	6	7	8	9	
10	11	12	13	14	

Matrix Multiplication

p = [[1,0], [0, 1]]
q = [[1,2], [3, 4]]
p * q

1 * 1 + 0 * 3	1 * 2 + 0 * 4	=	1	2
0 * 1 + 1 * 3	0 * 2 + 1 * 4		3	4

Week 3

Pandas

Python Pandas are used to import datasets from databases, spreadsheets, CSV's, etc. Pandas are used to clean datasets, tidy datasets by reshaping their structure into a suitable format for analysis, aggregate data by calculating summary statistics, and ofcourse, data visualisation.

Install Pandas using pip install pandas if you are using an IDE other than Google Colab.

```
import pandas as pd
```

There are different types of Data Structure in Pandas:

Series	1 dimension	1D labeled homogeneous array with immutable size.
Data Frames	2 dimensions	General 2D labeled, size mutable tabular structure with potentially heterogeneous types columns.

Series are essentially a column, while a data frame is essentially a collection of series. For both Series and data frames, the same functions and methods can be used on both.

Creating a Pandas Dataframe

pandas.DataFrame(data, index, columns, dtype, copy)

- data: data takes various forms such as ndarray, series, map, lists, dict, dataframes
- index: For the row labels, this is optional
- columns: For the column labels, this is optional
- dtype: Data type of each column
- copy: This command is used for copying data, the default is False.

```
data = {'apples':[3,2,0,1], 'oranges':[0,3,7,2]}
df = pandas.DataFrame(data)
```

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

```
df = pandas.DataFrame(data, index=['a', 'b', 'c', 'd'])
```

	apples	oranges
a	3	0
b	2	3
c	0	7
d	1	2

Reading data from CSV

```
import pandas as pd
df = pd.read_csv('dataset.csv', index_col=0)
```

In Google Colab:

```
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/My Drive/path_to_your_csv/iris.csv')
```

Viewing data

```
df.head()                 # By default, it shows the first 5 rows, can modify the parameter
df.tail()                 # By default, it shows the last 5 rows, can modify the parameter
df.info()                 # Shows info about the dataframe, including the number of rows, data types
df.shape                  # Shows the number of (rows, columns), AKA dataframe dimensions
df.describe()             # Continuous numeric values are used to return summary statistics
df['gender'].describe()   # Categorical variable summary statistics: Count, unique, top, freq
df[df.index==1]           # Extracts a single row using [] index number
```

Using .loc[] and .iloc[]

Can be used to fetch rows using slicing. Loc stands for locations. .loc locates by name, while .iloc locates by the index number.

Example dataset:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
6	148	72	35	0	33.6
1	85	66	29	1	26.6

`df.loc[df['Insulin']==0].head()`
The above code will return the first five rows of data where the column 'Insulin' is equal to 0.
`df.iloc[2]`
The above code simply returns the row for the second [2] index.
`df.iloc[0:4, 2:5]`
The above code will return the row of index 0 to the row of index 3 but not all the columns. It returns the columns with index 2 to index 4.

Handling Duplicates

```
temp_df = my_df.append(my_df)                  # Create a copy of the raw data
temp_df = temp_df.drop_duplicates()            # Drops duplicate values
temp_df.drop_duplicates(inplace=True)          # Do and apply changes in the same dataframe
```

Data Cleaning

Missing Values

Missing values may be represented by NaN, Na, NULL, or simply just blank. There is NO single universally acceptable method to handle missing values. It is often left to the judgement of the analyst, to either replace or drop them all together.

Detecting, Removing and Replacing
isnull(), dropna(), fillna(), replace(), df.isnull().any(), df.isnull().sum()

Filling missing values

1. df['model_year'].fillna(df['model_year'].mean(), inplace=True) #Fills with the datasets mean
2. df['column_name'] = df['column_name'].replace('old_value', 'new_value') # Replaces values

Week 4

Matplotlib & Seaborn

import matplotlib.pyplot as plt	import seaborn as sns
It is used for basic graph plotting like line charts, bar graphs, etc, and acts productively with data arrays.	It is used for statistics visualisation and can perform complex visualisations with fewer commands, it is considerably more organised and functional than Matplotlib and treats the entire dataset as a unit

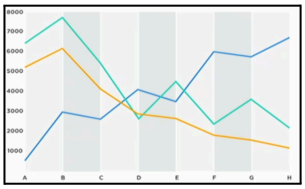
Seaborn Datasets

The Seaborn provides many datasets built-in that are stored in the Pandas data frame, making them easy to use with Seaborns plotting functions.

1. Tips Dataset
2. Iris Dataset
3. Penguins Dataset
4. Flights Dataset
5. Diamonds Dataset
6. Titanic Dataset
7. Exercise Dataset
8. MPG Dataset
9. Planets Dataset

Loading in a Seaborn Dataset: tips_df = sns.load_dataset("tips")

Line charts



A line chart is a graph that represents information as a series of data points connected by a straight line. Each data point is plotted and connected with a line or a curve. It is used to show a change overtime for numeric data and can be used to make predictions

Simple Plot

```
yield_apples = [0.895, 0.91, 0.919, 0.926, 0.929, 0.931]
years = [2010, 2011, 2012, 2013, 2014, 2015]
plt.plot(years, yield_apples)                    # Initialise the plot
plt.xlabel('Year')                               # Add a X-Axis label
plt.ylabel('Yield (tones per hectare)')          # Add a Y-Axis label
```

Multi-Plots

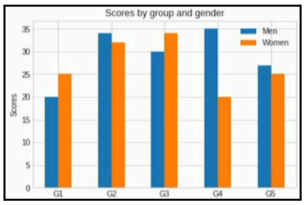
```
yield_oranges = [0.962, 0.941, 0.930, 0.923, 0.918, 0.908]
plt.plot(years, yield_apples)
plt.plot(years, yield_oranges)                  # Layer the plot under the first initialisation
plt.title('Crop Yields in Kanto')               # Add a main title
plt.legend(['Apples', 'Oranges'])               # Adding a legend (list in order of plot initialisation

plt.plot(years, yield_apples, marker='o')       # Adding a marker
plt.plot(years, yield_oranges, marker='x')      # Adding a marker
plt.figure(figsize=(12,6))                      # Sets size of figure 12 inches wide and 6 inches tall
```

Setting the style using set_style will change the style for ALL graphs in the file.

```
plt.figure(figsize=(12,6))
sns.set_style("whitegrid")        # Sets gridlines, is used WITH the matplotlib plot / "darkgrid"
```

Bar plots



A bar plot is used for nominal or ordinal categories. It can compare data amongst different categories, and is mainly ideal for more than three categories. It can also show large data changes over time.

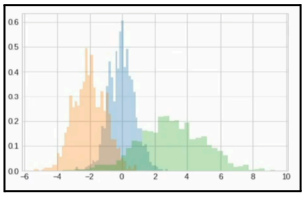
Stacked Bar Charts

```
years = range(2000, 2006)
apples = [0.35, 0.6, 0.9, 0.8, 0.65, 0.8]
oranges = [0.4, 0.8, 0.9, 0.7, 0.6, 0.8]
plt.bar(years, apples)                          # Plotting a Bar plot using matplotlib
plt.bar(years, oranges, bottom=apples)          # Creates a Stacked Bar chart
plt.xlabel('Year')
plt.ylabel('Yield (tones per hectare)')
plt.title('Crop Yields in Kanto')
```

Plotting Averages of Each Bar

```
sns.barplot(x='day', y='total_bill', data=tips_df, palette='viridis', hue='sex') # Uses Seaborn to plot
```

Histogram



Are used for continuous data, they display the frequency distribution (shape) of the data. They summarise large data sets graphically and compare multiple distributions.

Simple Histograms

```
flowers_df = sns.load_dataset("iris")
flowers_df .sepal_width                         # Extracting a single column
plt.title("Distribution of Sepal Width")
plt.hist(flowers_df.sepal_width, bins=5)        # Modifying bin size
```

```
plt.hist(flowers_df.sepal_width, bins=np.arange(2,5,0.25))        # Specify boundaries of each bin
plt.hist(flowers_df.sepal_width, bins=[1,3,4,4.5])                # Unequal bin sizes
```

Multiple Histograms

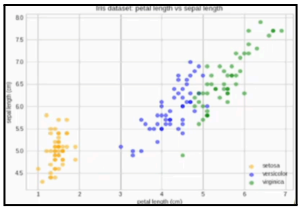
```
flowers_df.species.unique()       # Get the unique() values for a specified column
setosa = flowers_df[flowers_df.species=='setosa']          # Make subset of Setosa
versi = flowers_df[flowers_df.species=='versicolor']       # Make subset of Versicolor
virgi = flowers_df[flowers_df.species=='virginica']        # Make subset of Versicolor

plt.hist(setosa.sepal_width, bins=np.arange(2,5,0.25), alpha=0.4) # alpha controls transparency
plt.hist(versicolor.sepal_width, bins=np.arange(2,5,0.25), alpha=0.5)
plt.hist(virginica.sepal_width, bins=np.arange(2,5,0.25), alpha=0.6)
```

Stacked Histograms

```
plt.hist(
        [setosa.sepal_width, versi.sepal_width, virgi.sepal_width],
        bins = np.arange(2, 5, 0.25),
        stacked=True)
plt.legend(['Setosa', 'Versicolor', 'Virginica'])
```

Scatter plots

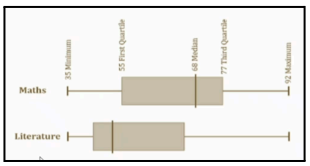


Scatter plots are used to visualise relations between two numeric variables. It is used to visualise correlation in a large dataset, and to predict behaviour of dependent variables based on the independent variable.

Simple Scatterplot

```
sns.scatterplot(x=flowers_df.sepal_length, y=flowers_df.sepal_width, hue=flowers_df.species, s=70)
# The color of the dots can be changed using hue=, while the size of the dots are modified using s=
```

Box plots

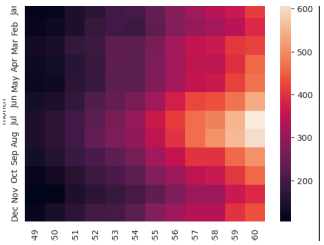


Also known as whisker plots are a statistical graph used on sets of numerical data, it shows the range, spread and central tendency. They are mostly used to compare data from different categories.

Simple Boxplots

```
plt.figure(figsize=(3,4))
sns.boxplot(x=flowers_df['species'], y=flowers_df['sepal_length'], palette='Blues')
```

Heat Maps



Heatmaps are used to see changes in behaviour or gradual changes in data using different colors to represent different values, we can use the pivot() method to create a heat map.

Simple Heat Map

```
flights_df = sns.load_dataset("flights").pivot(index="month", columns="year", values="passengers")
plt.title("Number of Passengers (1000s)")
sns.heatmap(flights_df, fmt="d", annot=True, cmap='Blues')
```

Week 5

Exploratory Data Analysis

Exploratory data analysis (EDA) is an important first step in the data analysis process to understand the dataset better, and guide subsequent modelling and analysis. It involves looking at and visualising data to understand its main features, finding variables that are relevant to our problem and understanding the relationships between variables.

EDA can help us identify hidden patterns and relationships between different data points, helping us to build our machine learning models. Insights obtained from EDA help you decide which features are most important for building models and how to prepare them to improve performance. EDA helps us choose the best modelling techniques and adjust them for better results.

Key Steps in EDA

1. Understand the Problem and the Data
2. Importing Libraries
3. Loading/Reading the dataset
4. Data inspection
5. Data cleaning
    a. Checking for Missing values
    b. Checking for duplicates
6. Analysing the dataset
    a. Univariate analysis — Inspecting ONE variable
    b. Bivariate analysis — Inspecting TWO variables and their relationship
    c. Multivariate analysis — Inspecting more than two variables and their relationships

1	Understand the Problem and the Data	● What is the business goal or research question? ● What are the variables in the data and what do they represent? ● What types of data (numerical, categorical, text, etc) do you have? ● Are there any known data quality issues or limitations? ● Are there domain-specific concerns or restrictions?
2	Importing Libraries	● numPy (np), pandas (pd), seaborn (sns), matplotlib (plt)
3	Loading/Reading the dataset	● df = sns.load_dataset() ● df = pd.read_csv()
4	Data Inspection	● df.head(), df.tail(), df.info(), df.describe(), df.shape() ● Check data types using df.dtypes

5	Data Cleaning	● Identify and handle missing values using methods like df.isnull().sum(● Find and address duplicates with df.duplicated.sum()
6	Univariate Analysis	● Analyse single variable at a time ● Use descriptive statistics with df.describe() for numerical data ● Create histograms, box-plots, and density plots
	Bivariate analysis	● Explore relationships between two variables ● Create scatter plots, pair plots
	Multivariate analysis	● Interactions between three or more variables in a dataset are simultaneously analysed and interpreted in multivariate analysis ● Use heatmaps, line charts

Example: Insurance.csv

```
insurance = pd.read_csv('/content/insurance.csv')
insurance.head()                          # View top 5 rows
insurance.sample(6)                       # Show 6 random rows
insurance.info()                          # Check for NULL values, data types, and total rows
insurance.columns                          # Shows us all column names
insurance.describe()                      # Shows us summary statistics for numeric columns
insurance.describe(include='O')           # Shows us summary statistics for categorical columns
insurance.region.unique()                 # Shows all unique values for a particular column
insurance.isnull().sum()                  # Shows a sum of all NULL values per column
insurance[insurance.duplicated()]         # Shows all the duplicated rows (observations)
insurance.drop_duplicates(keep='first', inplace=True) # Removes the second duplicate row

# Univariate Analysis for Numerical Variables
plt.figure(figsize=(10,6))
sns.distplot(insurance.charges, color='lightblue')
plt.title('Charges Distribution', size=18)
plt.xlabel('Charges', size=14)
plt.ylabel('Density', size=14)
plt.show()

# Univariate Analysis for Categorical Variables
plt.figure(figsize=(10,6))
sns.countplot(x='smoker', data=insurance, hue='smoker')
plt.title('Smoker Distribution', size=18)
plt.xlabel('Smoker', size=14)
plt.ylabel('Count', size=14)
plt.legend(['Smoker', 'Non Smoker'])
plt.show()
```

```
# Bivariate Analysis for Numerical Variables
plt.figure(figsize=(10,6))
sns.scatterplot(x='age', y='charges', color='b', data=insurance)
plt.title('Age vs Chargers', size=18)
plt.xlabel('Age', size=14)
plt.ylabel('Charges', size=14)
plt.show()

# Bivariate Analysis for Categorical Variables
plt.figure(figsize=(10,6))
sns.set_style('darkgrid')
sns.boxplot(x='smoker', y='charges', data=insurance)
plt.title('Smoker vs Charges', size=18)

# Multivariate Analysis for NumericalVariables
numeric_df = df.select_dtypes(include=np.number)
sns.heatmeap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

Filling Missing Values

```
df['age'].fillna(df['age'].median(), inplace=True)
df['embarked'].fillna(df['embarked'].mode()[0], inplace=True)
print(df.isnull().sum())
```

Grouping Values

```
grouped = df.groupby('pclass')['survived'].mean()
grouped.plot(kind='bar')
```

Week 6

Machine Learning

Machine learning is a subfield of AI that focuses on developing algorithms that enable computers to learn from data and make predictions or decisions without being explicitly programmed for each task. ML allows machines to improve their performance over time by identifying patterns and relationships within datasets.

Machine Learning Examples:

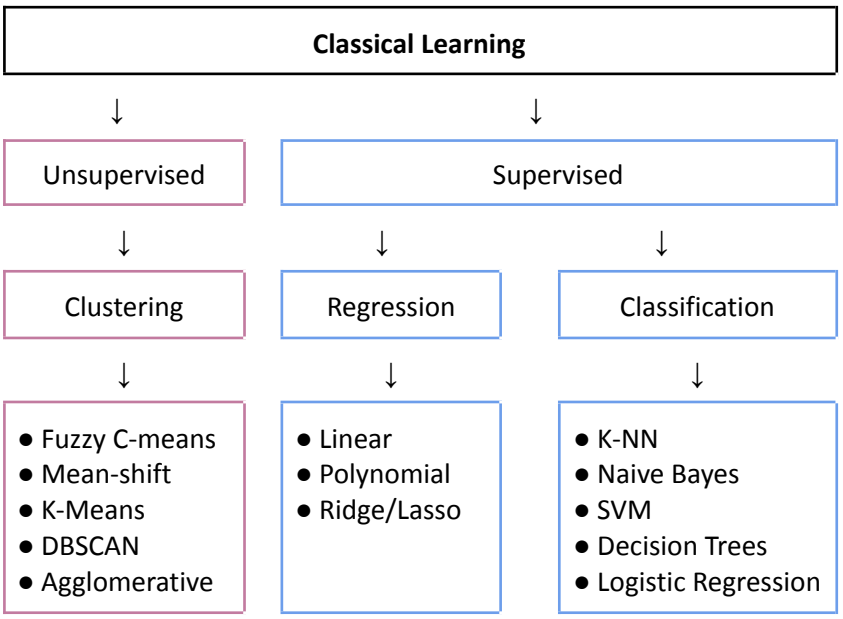
- banks look for trends in transaction data to detect outliers that may be fraudulent.
- Email inboxes use text to classify an email as spam or not, and adjust their classification rules based upon how we flag emails.
- Travel apps use live and historical data to predict traffic, travel times, and journey routes.
- Retail companies using data to recommend products

The tasks in the given examples above can be classified into at least one of the four broad groupings.

1. Finding trends in data
2. Classifying data into groups and categories
3. Making decisions and predictions
4. Learning to interact in an environment

Types of Machine Learning

1. Supervised ML Algorithms
2. Unsupervised ML Algorithms
3. Reinforcement ML Algorithms



Supervised Learning Algorithm

Supervised learning needs external supervision to learn, it is trained using the LABELED dataset. A SLA takes a known set of input data and known responses to the data and trains a model to generate reasonable predictions for the response to a new data. Once training and processing are done, the model is tested by providing sample test data to check whether it predicts the correct output. You should use SLA if you have known data for the output you are trying to predict.

ELI5: If you see a shape that has four equal sides, label it as "square". This is fed into the model for testing, it makes a prediction, and we verify whether it is correct or not.

Regression: If you have to predict a continuous variable, use Regression algorithms. That is, if there is a relationship between the input and output variables, it may predict continuous variables, such as weather forecasting, market trends, etc.

Classification: Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Fale, True-False, etc.

Unsupervised Learning Algorithm

In unsupervised learning algorithms, the machine does NOT need any external supervision to learn from the data, they are trained using unlabeled dataset and draws inferences from data sets consisting of input data without labeled answers. The model has NO predefined output and tries to find useful insights from a huge amount of data.

ELI5: Providing pictures of dogs and cats, the algorithm tries to find features, such as ears, tails, color, shape, etc. It will create a classification to provide an output.

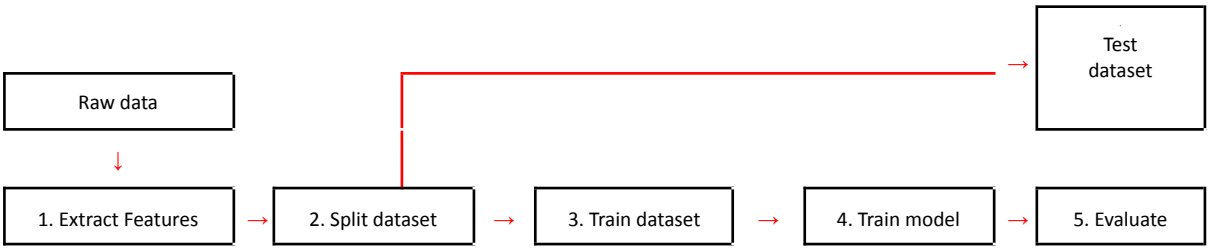
Clustering: Is a method of grouping objects into a cluster based on the most similarities between groups, or no similarities with the objects of another. For example: Clustering can help group customers with similar purchasing behaviours, which can be used for targeted marketing and product recommendations.

Supervised vs Unsupervised

Supervised: If you are training the model to make a prediction such as the future value of a continuous variable, such as temperature or a stock price, or a classification, such as identifying car makers from webcam video footage.

Unsupervised: If you need to explore your data and want to train a model to find a good internal representation, such as splitting data up into clusters. Such as splitting customers based on different features such as frequency, expensive shoppers, etc

Machine Learning Workflow



Machine Learning Limitations

- Garbage in = Garbage out. Without data preprocessing or data cleaning, our dataset may be skewed and produce meaningless predictions
- Biases due to training data. The performance of an ML system depends on the quality of input data used to train it. For example, if we collect data on public transport use from only high socioeconomic areas, the resulting input data may be biased due to a range of factors that may increase the likelihood of people from those areas using private transport versus public options.
- Extrapolation. We can only make reliable predictions about data which is in the same range as our training data. Extrapolation of the training data means we cannot be confident in our results
- Overfitting. Sometimes, ML models become overtrained, for example, continuously finetuning the model to become overtrained.
- Inability to explain answers. ML techniques will return an answer based on the input data and model parameters, even if that answer is wrong. Most systems cannot explain the logic used to arrive at that answer. This can make detecting and diagnosing problems difficult.

Regression

Regression is a method that allows you to estimate how the dependent variable changes as the independent variable(s) change. In this course, we will only focus on Linear Regression. Some examples of applications of Linear Regression include: How temperature affects ice cream sales, predicting the price of a house given house features, or predicting the impact of college scores on University admissions.

Least Squares regression equations

Least squares regression can calculate a predictive model. For example, with the simple linear equation of Y = a + bX, the least squares can be calculated as:

$$b = \frac{\Sigma(x - \bar{x}) * (y - \bar{y})}{\Sigma(x - \bar{x})^2}, a = Y - bX$$

Prediction models have limitations. For example, if we are using least squares regression to predict the test score (0-100) of a student based on their hours spent studying, and we want to cast a prediction for for 20 hours spent, using the regression equation y' = 30.18 + 6.49 * x their predicted score would be 160, which does not make sense for a score scale of 0 to 100.

Regression Example

Scikit-Learn

Scikit-Learn is a python package that provides machine-learning algorithms.

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()

import seaborn as sns
sns.relplot(x='Scores', y='Hours', data=stud_scores, height=3.8, aspect=1.8, kind)
sns.set_style('darkgrid')
```

Splitting the Data

Splitting the dataset is crucial in determining the accuracy of a model. If we were to train the model with the raw dataset and predict the response for the same dataset, the model would suffer flaws like overfitting, thus compromising its accuracy. For this reason, we have to split the data into training and testing sets. Scikit-learn provides a train_test_split function for splitting datasets into training and testing subsets. We will split the dataset in the ratio of 70:30, where 70% will be the training set, and 30% will be for the testing set.

```
X = stud_scores.iloc[:,:-1].values
y = stud_scores.iloc[:,1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1)
```

Fitting the Model

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

y = mx + c

m	m = regressor.coef_
c	c = regressor.intercept_

Prediction

```
y_pred = regressor.predict(X_test)
y_pred
```

Comparing the test values and Predicted values

```
comparison_df = pd.DataFrame({"Actual":y_test, "Predicted":y_pred})
comparison_df
```

Fit Error

```
import sklearn.metrics import mean_squared_error
print("Mean Squared Error", mean_squared_error(y_test, y_pred))
```

Root Mean Squared Error (RMSE)

```
print("RMSE", np.sqrt(mean_squared_error(y_test, y_pred)))
```

Week 7

Classification Algorithm: Logistic Regression

F1 Score

Imagine you have a basket of fruits and are trying to find the green apples. The F1 score checks how good you are at picking the right fruits by evaluating the "precision" and "recall". Precision is how often we are right, while recall is how many of the green apples we actually find. 🍏

Precision = True +ve's / True +ve's + False +ve's
Recall = True +ve's / True +ve's + False -ve's

$$harmonic\ mean\ = \frac{n}{\frac{1}{x1}+\frac{1}{x2}+\frac{1}{x3}+\ldots+\frac{1}{xn}}$$

Harmonic mean increases as values increase, decreases as values decrease, directly proportional.

- For a balanced dataset, we should go for an accuracy score, otherwise a precision/recall/F1 Score.
- If you intend that FP should be very small, go for the precision score
- If you intend that FN should be very small, go for the recall score
- If you intend that both FP and FN should be very small, go for the F1-Score

Cancer Diagnosis

	Actual	Predicted
False Positives	Cancer not detected	Cancer detected
False Negatives	Cancer detected	Cancer not detected

False negatives are more dangerous because as a person thinks they are not diagnosed with cancer, they will not take any corrective actions, hence we should use the recall score.

Spam Emails

	Actual	Predicted
False Positives	Not spam	Spam
False Negatives	Spam	Not spam

False positives are more dangerous because a person may miss an important email and will not take action, hence we should use the precision score.

The Titanic

	Actual	Predicted
False Positives	Not survived	Survived
False Negatives	Survived	Not survived

Both FP and FN are dangerous. Marking someone as alive when they have died, or marking someone as dead when they are alive, hence we should use the F1 Score

In Python

```
#from sklearn metrics import accuracy_score,f1_score, precision_score,recall_score,confusion_matrix
#from sklearn import metrics

#accuracy_score(y_test, y_pred)
#f1_score(y_test, y_pred)
#precision_score(y_test, y_pred)
#recall_score(y_test, y_pred)

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(xtrain, ytrain)
y_pred = classifier.predict(xtest)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ytest, y_pred)

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Week 8

K - Nearest Neighbour (KNN)

Is a supervised machine learning distance-based algorithm that can be used for classification and regression problems. It makes predictions by finding the 'k' closest data points in the training set to a new data point and then basing its prediction on those neighbours.

Example Age vs Loan: To predict Brianna's default status (Yes or No)

Customer	Age	Loan	Default
Aulia	21	40000	N
Ankita	21	40000	N
Mellisa	20	60000	Y
Brianna	21	80000	?

1. **Calculate Euclidean Distance**
   $d = sqrt(x1 - y1)^2 * (x2 - y2)^2$
   $sqrt(20 - 21)^2 + (60000 - 80000)^2$

2. **Calculate for K**
   So if K=3, there is one default = Y, two default = N

KNN is a lazy learner as it parses through all data points for each classification, hence it is only useful for smaller datasets, otherwise computational power, time and expense can burden exponentially. Additionally, KNN fails for variables with different scales, so feature scaling (standardisation and normalisation) are required before applying KNN.

In Python

```
# Split dataset into training and test set
knn = KneighboursClassifier(n_neighbours=7)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy_score(y_test, y_pred)
```