DATA 301

A comprehensive cheat sheet covering essential data analysis tools and techniques, including data representation, Excel, SQL, Python, R, visualization, open data, APIs, and geospatial concepts.

Data Representation & Excel Basics

Data Representation

Binary:	Base-2 numeral system (0, 1). Fundamental for digital data.
Sizes:	MB (Megabyte), GB (Gigabyte), TB (Terabyte). 1 TB = 1024 GB, 1 GB = 1024 MB.
Data Types:	Strings (text), Dates (YYYY-MM-DD), Floats (decimal numbers).
File Encodings:	UTF-8 (Unicode Transformation Format - 8-bit) is the most common. ASCII, UTF-16 are others.
Character Encoding	A character encoding tells the computer how to interpret raw zeroes and ones into actual characters. It determines what code belongs to which character. Common encodings: ASCII, UTF-8, UTF-16
Endianness	Describes the order in which bytes of a multi-byte data type (like integers) are stored in computer memory. Big-Endian: Most significant byte first. Little-Endian: Least significant byte first.

Essential Excel Formulas

`IF(condition, value_if_true, value_if_false)`	Returns one value if a condition is TRUE and another value if it's FALSE. **Example:** `=IF(A1>10, "High", "Low")`
`COUNTIF(range, criteria)`	Counts the number of cells within a range that meet the given criteria. **Example:** `=COUNTIF(B1:B10, ">50")`
`SUM(number1, [number2], ...)`	Adds all the numbers in a range of cells. **Example:** `=SUM(C1:C10)`
`MAX(number1, [number2], ...)`	Returns the largest value in a set of numbers. **Example:** `=MAX(D1:D10)`
`CONCATENATE(text1, [text2], ...)`	Joins several text strings into one text string. Use `&` operator as shortcut. **Example:** `=CONCATENATE(A1, " ", B1)` or `=A1 & " " & B1`
`AVERAGE(number1, [number2], ...)`	Calculates the average (arithmetic mean) of the numbers in a range. **Example:** `=AVERAGE(E1:E10)`

Excel - Pivot Tables

Pivot tables are used to summarize and analyze data. They allow you to rearrange and group data in different ways to see patterns and trends.

Key Components:

- **Rows:** Categorical fields displayed as rows.
- **Columns:** Categorical fields displayed as columns.
- **Values:** Numerical fields that are aggregated (summed, averaged, etc.).
- **Filters:** Used to narrow down the data being displayed.

Creating a Pivot Table:

1. Select your data range.
2. Go to `Insert > PivotTable`.
3. Choose where to place the pivot table (new worksheet or existing location).
4. Drag and drop fields into the Rows, Columns, Values, and Filters areas.

Example Scenario:

Imagine you have sales data with columns: `Date`, `Region`, `Product`, `Sales`. You can create a pivot table to:

- Show total sales by region.
- Show average sales by product over time.
- Filter the data to show sales for a specific month.

SQL Fundamentals

Basic SQL Commands

`CREATE TABLE table_name (column1 datatype, column2 datatype, ...);`	Creates a new table in the database. Define column names and their data types. **Example:** `CREATE TABLE Employees (ID INT, Name VARCHAR(255), Salary DECIMAL(10, 2));`
`INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);`	Inserts a new row into a table. **Example:** `INSERT INTO Employees (ID, Name, Salary) VALUES (1, 'John Doe', 60000.00);`
`UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;`	Modifies existing data in a table. Use `WHERE` clause to specify which rows to update. **Example:** `UPDATE Employees SET Salary = 65000.00 WHERE ID = 1;`
`DELETE FROM table_name WHERE condition;`	Deletes rows from a table. Always use a `WHERE` clause to avoid deleting all rows. **Example:** `DELETE FROM Employees WHERE ID = 1;`
`SELECT column1, column2, ... FROM table_name WHERE condition;`	Retrieves data from one or more tables.

Filtering and Aggregations

`WHERE condition`	Filters rows based on a specified condition. **Example:** `SELECT * FROM Employees WHERE Salary > 50000;`
`GROUP BY column1, column2, ...`	Groups rows that have the same values in specified columns. Often used with aggregate functions. **Example:** `SELECT Department, AVG(Salary) FROM Employees GROUP BY Department;`
`ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...`	Sorts the result-set based on one or more columns. `ASC` for ascending, `DESC` for descending. **Example:** `SELECT * FROM Employees ORDER BY Salary DESC;`
`LIMIT number`	Limits the number of rows returned. **Example:** `SELECT * FROM Employees ORDER BY Salary DESC LIMIT 10;`
`HAVING condition`	Filters results after grouping, used with `GROUP BY`. **Example:** `SELECT Department, AVG(Salary) FROM Employees GROUP BY Department HAVING AVG(Salary) > 55000;`

Joins

JOIN: Returns rows when there is a match in both tables based on the join condition.

```
SELECT * FROM TableA JOIN TableB ON TableA.Column = TableB.Column;
```

LEFT JOIN (or LEFT OUTER JOIN): Returns all rows from the left table (TableA), and the matched rows from the right table (TableB). If there is no match in TableB, it returns NULL values for columns from TableB.

```
SELECT * FROM TableA LEFT JOIN TableB ON TableA.Column = TableB.Column;
```

RIGHT JOIN (or RIGHT OUTER JOIN): Returns all rows from the right table (TableB), and the matched rows from the left table (TableA). If there is no match in TableA, it returns NULL values for columns from TableA.

```
SELECT * FROM TableA RIGHT JOIN TableB ON TableA.Column = TableB.Column;
```

FULL OUTER JOIN: Returns all rows when there is a match in one of the tables. If there are rows in TableA that do not match TableB, or rows in TableB that do not match TableA, the FULL OUTER JOIN will include these rows in the result set. Columns that do not have matching values will contain NULL.

```
SELECT * FROM TableA FULL OUTER JOIN TableB ON TableA.Column = TableB.Column;
```

INNER JOIN: Same as JOIN. Returns rows only when there's a match in both tables.

```
SELECT * FROM TableA INNER JOIN TableB ON TableA.Column = TableB.Column;
```

Python and R Programming

Python Fundamentals

Input Handling: Using `try/except` blocks to handle potential errors when taking user input.

```
try:
  age =
int(input("Enter your
age: "))
except ValueError:
  print("Invalid
input. Please enter a
number.")
```

Loops: `for` and `while` loops for iterating over data structures.

```
for i in range(5):
  print(i)

while True:
  response =
input("Type 'exit' to
quit: ")
  if response ==
'exit':
    break
```

List/Dictionary Logic: Creating, accessing, and manipulating lists and dictionaries.

```
my_list = [1, 2, 3]
my_dict = {'a': 1,
'b': 2}
print(my_list[0])  #
Accessing list element
print(my_dict['a']) #
Accessing dictionary
value
```

File Reading & String Parsing: Reading data from files and manipulating strings.

```
with open('data.txt',
'r') as f:
  content = f.read()
words =
content.split()
```

Function Definitions: Defining reusable blocks of code.

```
def greet(name):
  print(f"Hello,
{name}!")

greet("World")
```

R Programming Fundamentals

Vectors: Creating and manipulating vectors.

```
my_vector <- c(1, 2, 3,
4, 5)
print(my_vector[1]) #
Accessing vector element
```

Subsetting: Using `subset()` and indexing to filter data.

```
subset_data <-
subset(data, column > 10)
print(data[data$column >
10, ]) # Indexing
```

Data Frames: Working with data frames.

```
df <- data.frame(Name =
c("John", "Jane"), Age =
c(30, 25))
print(df$Name) #
Accessing column
```

Statistical Tests: Performing t-tests and other statistical analyses.

```
t.test(data$column1,
data$column2)
```

Summary Statistics: Calculating descriptive statistics.

```
summary(data$column)
```

Data Processing - Python & R

Python - Sum, Max, Filter: Simple data processing operations.

```
data = [1, 2, 3, 4, 5]
sum_data = sum(data)
max_data = max(data)
filtered_data =
list(filter(lambda x: x
> 2, data))
```

R - Filtering & Summary Stats: Data manipulation and summary statistics.

```
data <- c(1, 2, 3, 4,
5)
filtered_data <-
data[data > 2]
summary_data <-
summary(data)
```

Visualization, Open Data, and Geospatial

Data Visualization Tools

Python: Using `matplotlib` and `seaborn` for creating visualizations.

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

sns.histplot(data['column'])
plt.show()
```

R: Using `ggplot2` for creating visualizations.

```
library(ggplot2)

ggplot(data, aes(x = column)) +
geom_histogram()
```

Tableau: A data visualization tool for creating interactive dashboards and reports. Dimensions (blue pills) are categorical, and Measures (green pills) are numerical.

Drag dimensions and measures to rows, columns, and marks cards to create visualizations.

Open Data & APIs

CSV/JSON Data Loading (Python): Loading data from CSV and JSON files.

```
import pandas as pd

data_csv =
pd.read_csv('data.csv'
)
data_json =
pd.read_json('data.jso
n')
```

Accessing APIs (Python): Accessing data from APIs, e.g., Google Maps.

```
import requests

response =
requests.get('https://
maps.googleapis.com/..
.')
data = response.json()
```

CSV/JSON Data Loading (R): Loading data from CSV and JSON files.

```
data_csv <-
read.csv('data.csv')
library(jsonlite)
data_json <-
fromJSON('data.json')
```

Accessing APIs (R): Accessing data from APIs

```
library(httr)
response <-
GET('https://api.examp
le.com/data')
data <-
content(response,
"parsed")
```

Geospatial (GIS) Fundamentals

Coordinate Data:
Latitude and Longitude are used to specify locations on Earth. Latitude ranges from -90 to +90 (degrees North/South), and Longitude ranges from -180 to +180 (degrees East/West).

Geospatial File Types:

- **CSV with lat/lon:** Simple text file where columns represent latitude and longitude coordinates.
- **KML (Keyhole Markup Language):** XML-based file format for representing geographic data in Google Earth, Google Maps, and other GIS software.

Simple Mapping (Python):
Using libraries like `folium` to create interactive maps.

```
import folium

m = folium.Map(location=[40.7128,
-74.0060], zoom_start=10)
m.save('map.html')
```

Simple Mapping (R):
Using libraries like `leaflet` to create interactive maps.

```
library(leaflet)

leaflet() %>%
  addTiles() %>%
  setView(lng = -74.0060, lat = 40.7128,
zoom = 10)
```