PYTHON BASICS
Cheat Sheet

Data Structres

Numbers

```
# Integer (int)
>>> x = 2
# Float
>>> x = 2.3
# Complex
>>> x = 2j + 1
```

Operations

```
# Exponent
>>> 2**3
8

# Modulus / Remainder
>>> 22%8
6

# Integer Division
>>> 22//8
2

# Division
>>> 22/8
2.75

# Multiplication
>>> 3*3
9

# Subtraction
>>> 5-2
3

# Addition
>>> 2+2
4
```

Boolean

```
>>> a = 2
>>> b = 7

>>> a == b
False

>>> a != b
True

>>> a > b
False

>>> a >= b
False

>>> a < b
True

>>> a <= b
True

>>> (a > 1) & (b < 10)
True

>>> (a > 3) | (b > 5)
True

>>> a is b
False

>>> a is not b
True
```

Strings

```
>>> txt_1 = 'Hello World!'
'Hello World!'

>>> txt_2 = "HELLO WORLD!"
'HELLO WORLD!'

>>> long_txt = """
               Hello World,
               Welcome DSMLBC!
               """
'Hello World, \n Welcome DSMLBC!'
```

Indexing & Slicing

```
>>> txt_1[0]
'H'

>>> txt_1[-1]
'!'

>>> txt_1[1:4]
'ell'

>>> txt_1[:5]
'Hello'
```

String Methods

```
>>> len(txt_1)
12

>>> txt_1.upper()
'HELLO WORLD!'

>>> txt_2.lower()
'hello world!'

>>> txt_1.replace("World", "Era")
'Hello Era!'

>>> txt_1.split()
['Hello', 'World!']

>>> ' Hello World! '.strip()
'Hello World!'

>>> 'hello world!'.capitalize()
'Hello world!'
```

Dictionaries

```
>>> dict_1 = {"REG" : "Regression",
              "LOG" : "Logistic Regression",
              "CART": "Classification and Reg"}

>>> dict_2 = {"REG": ["RMSE", 10]
              "LOG": ["MSE", 20]
              "CART": ["SSE", 30]}
```

Key - Value Methods

```
>>> dict_1.keys()
dict_keys(['REG', 'LOG', 'CART'])

>>> dict_1.values()
dict_values(['Regression', 'Logistic Regression',
             'Classification and Reg'])

>>> dict_1.items()
dict_items([('REG', 'Regression'),
            ('LOG', 'Logistic Regression'),
            ('CART', 'Classification and Reg')])

>>> dict_1["REG"]
'Regression'

>>> dict_2["CART"][1]
30
```

Dictionary Methods

```
>>> dict_1 = {"REG": "Regression",
              "LOG": "Logistic Regression",
              "CART": "Classification and Reg"}

>>> dict_2 = {"REG": ["RMSE", 10]
              "LOG": ["MSE", 20]
              "CART":["SSE", 30]}
```

Lists

```
>>> list_1 = [1, 2, 3, "a", "b"]

>>> list_2 = [True, [1, 2, 3]]

>>> numbers = [4, 2, 1, 3]
```

Indexing & Slicing

```
>>> list_1[3]
'a'

>>>list_2[-1]
[1, 2, 3]

>>>list_1[0:4]
[2, 3, 'a']
```

List Methods

```
>>> list_1 + list_2
[1, 2, 3, 'a', 'b', True, [1, 2, 3]]

>>> list_1.append('c')
[1, 2, 3, 'a', 'b', 'c']

>>> list_2.remove('True')
[[1, 2, 3]]

>>> len(list_1)
5

>>> numbers.sort()
[1, 2, 3, 4]

>>> list_2.insert(1, False)
[True, False, [1, 2, 3]]

>>> numbers.pop()
[4, 2, 1]
```

Tuples

```
>>> tuple = ("john", "mark", 1, 2)

>>> tuple[0]
'john'

>>> tuple[1:3]
       ('mark', 1)

>> len(tuple)
4
```

Set

```
>>> set_1 = {1, 2, 2, 3, 3, 3}
{1, 2, 3}

>>> set_2 = set([3, 4, 4, 5, 5, 5, 6])
{3, 4, 5, 6}
```

Set Methods

```
>>> set_1.difference(set_2)
{1, 2}

>>> set_2.difference(set_1)
{4, 5, 6}

>>> set_1.symmetric_difference(set_2)
{1, 2, 4, 5, 6}

>>> set_1.intersection(set_2)
{3}

>>> set_1.union(set_2)
{1, 2, 3, 4, 5, 6}

>>> set_1.isdisjoint(set_2)
False

>>> set_1.issubset(set_2)
False

>>> set_1.issuperset(set_2)
False
```

Comprehensions

List Comprehensions

```
# Syntax
list = [expression for item in iterable if condition]
>>> squares = [x**2 for x in range(1, 11)]
>>> print(squares)
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

>>> evens = [f"{x}: even" for x in range(1,10) if x % 2 == 0]
>>> print(evens)
['2: even', '4: even', '6: even', '8: even']

>>> even_odd = [f"{x}: even" if x%2 == 0 else f"{x}: odd"
                for x in range(1,5)]

>>> print(even_odd)
['1: odd', '2: even', '3: odd', '4: even']
```

Dictionary Comprehensions

```
# Syntax
dict = {key_exp: value_exp for item in iterable if condition}

>>> dictionary = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

>>> {k: v ** 2 for (k, v) in dictionary.items()}
{'a': 1, 'b': 4, 'c': 9, 'd': 16}

>>> {k.upper(): v for (k, v) in dictionary.items()}
{'A': 1, 'B': 2, 'C': 3, 'D': 4}

>>> {k.upper(): v*2 for (k, v) in dictionary.items()}
{'A': 2, 'B': 4, 'C': 6, 'D': 8}
```

Loops

For Loop

```
# Syntax
for <variable> in <list>:
  <code block>

>>> students = ["John", "Mark", "Venessa"]

>>> for student in students:
...     print(student)
John
Mark
Venessa

>>> for index, student in enumerate(students):
...     print(index, student)

0 John
1 Mark
2 Venessa
```

While Loop

```
# Syntax
while <condition>:
  <code to execute while the condition is true>

>>> i = 1
>>> while i < 5:
...     print(i)
...     if i==3:
...         break
...     i+=1
1
2
3

>>> while i < 5:
...     i+=1
...     if i==3:
...         continue
...     print(i)
2
4
5
```

Conditions

```
>>> number = 3.14
>>> if number > 0:
...     print(f"{number} is positive.")
... elif number < 0:
...     print(f"{number} is negative.")
... else:
...     print(f"{number} is zero!")
3.14 is positive.
```

Functions

```
# Function_1
>>> def say_hi(name):
...     print(f'Merhaba {name}')

# Calling function
>>> say_hi('Miuul')
Merhaba Miuul

# Function_2
>>> def summer(num_1, num_2):
...     """
...     Sum of two numbers
...     Args:
...         num_1: int, float
...         num_22: int, float
...     Returns:
...         int, float
...     """
...     Return num_1 + num_2

# Calling function
>>> summer(3, 4)
7

# Function_3
>>> def find_volume(length=1, width=1, depth=1):
...     print(f'Length = {length}')
...     print(f'Width = {width}')
...     print(f'Depth = {depth}')
...     volume = length * width * depth

# Calling function
>>> find_volume(1, 2, 3)
Length = 1
Width = 2
Depth = 3
6

>>> find_volume(2, depth=3, width=4)
Length = 2
Width = 4
Depth = 3
24
...     return volume
```

Local & Global Variables

```
>>> list_store = [1, 2] # Global variable

>>> def add_element(a, b):
... c=a*b
... list_store.append(c)
... print(list_store)

>>> add_element(1, 9)
[1, 2, 9]
```

Built-in Functions

```
# lambda
>>> summer = lambda a, b: a + b
>>> summer(3, 5)
8

# map
>>> salaries = [1000, 2000, 3000, 4000, 5000]
>>> list(map(lambda x: x * 20 / 100 + x, salaries))
[1200.0, 2400.0, 3600.0, 4800.0, 6000.0]

# filter
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(filter(lambda x: x % 2 == 0, numbers))
[2, 4, 6, 8, 10]

# reduce
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> reduce(lambda a, b: a + b, numbers)
55

# zip
>>> students = ["John", "Mark", "Venessa"]
>>> departments = ["mathematics", "statistics", "physics"]
>>> list(zip(students, departments))
[('John', 'mathematics'), ('Mark', 'statistics'), ('Venessa', 'physics')]
```

NumPy

Cheat Sheet

```
import numpy as np
```

NumPy Arrays

Creating NumPy Arrays

```
# One dimensional array
>>> np.array([1, 2, 3])
array([1, 2, 3])

# Two dimensional array
>>> np.array([(1,2,3),(4,5,6)])
array([[1, 2, 3],
       [4, 5, 6]])

# Array of zeros with 3 elements
>>> np.zeros(3)
array([0., 0., 0.])

# 2x2 array filled with ones
>>> np.ones((2,2))
array([[1., 1.],
       [1., 1.]])

# 3x3 identity matrix
>>> np.eye(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])

# 2x3 array filled with 4s
>>> np.full((2,3),4)
array([[4, 4, 4],
       [4, 4, 4]])

# 1D array of length 3 with random integers between 0 and 10
>>> np.random.randint(0, 10, size=3)
array([8, 2, 7])

# 2x3 array of random numbers between 0 and 1
>>> np.random.rand(2, 3)
array([[0.60738971, 0.84209229, 0.64491569],
       [0.50990228, 0.25552633, 0.56484642]])

# 2x3 array of random numbers from a normal distribution with mean 0
and standard deviation 1
>>> np.random.normal(0, 1, (2, 3))
array([[ 0.36527849, -2.48435406,  0.77739812],
       [ 0.07923544, -0.30833118,  0.32393125]])

# 1D array with values from 0 to 10 with a step of 3
>>> np.arange(0,10,3)
array([0, 3, 6, 9])

# 1D array with 3 evenly spaced values between 0 and 10
>>> np.linspace(0,10,3)
array([ 0.,  5., 10.])
```

Attributes of Arrays

```
>>> a = np.array([(1,2,3),(4,5,6)])

>>> a.ndim
2

>>> a.shape
(2, 3)

>>> a.size
6

>>> a.dtype
dtype('int32')

>>> a.astype(float)
array([[1., 2., 3.],
       [4., 5., 6.]])
```

Data Manipulation

Reshaping

```
>>> a = np.array([(1,2,3),(4,5,6)])
array([[1, 2, 3],
       [4, 5, 6]])

>>> a.flatten()
array([1, 2, 3, 4, 5, 6])

>>> a.resize((6,1))
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6]])

>>> a.T
array([[1, 4],
       [2, 5],
       [3, 6]])

>>> b = np.random.randint(1, 10, size=9)
array([4, 7, 1, 8, 2, 3, 3, 7, 8])

>>> b.reshape(3, 3)
array([[4, 7, 1],
       [8, 2, 3],
       [3, 7, 8]])
```

Combining

```
>>> a = np.array([1, 2, 3, 4, 5])
>>> b = np.array([6, 7, 8, 9,10])

# Concatenating arrays horizontally (along
columns)
>>> np.hstack((a, b))
array([ 1,  2,  3,  4,  5,  6,  7,  8, 9,10])

# Concatenating arrays vertically (along rows)
>>> np.vstack((a, b))
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])

>>> np.concatenate((a, b))
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9,
10])

>>> np.stack((a, b))
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

Indexing / Fancy Indexing

```
>>> a = np.array([1, 2, 3, 4, 5])
array([1, 2, 3, 4, 5])

# Access the first element of the array
>>> a[0]
1

>>> indices = np.array([0, 2, 4])
# Access elements at indices 0, 2, and 4
>>> a[indices]
array([1, 3, 5])

>>> b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

# Access the element at row 1, column 2 of the array
>>> b[1, 2]
6
```

Slicing

```
# Slice the array from index 2 to index 4 (exclusive)
>>> a[2:5]
array([3, 4, 5])

# Slice the array from index 1 to the end
>>> a[1:]
array([2, 3, 4, 5])

# Slice the array from the beginning to index 2(exclusive)
>>> a[:3]
array([1, 2, 3])

# Slice the array in reverse order
>>> a[::-1]
array([5, 4, 3, 2, 1])

# Slice the first row
>>> b[0, :]
array([1, 2, 3])

# Slice the second column
>>> b[:, 1]
array([2, 5, 8])

# Slice the sub-array starting from row 1 and column 0 up
to row end and column 1 (exclusive)
>>> b[1:, :2]
array([[4, 5],
       [7, 8]])
```

Subsetting

```
>>> a[a > 2]
array([3, 4, 5])

>>> a[(a > 1) & (a < 4)]
array([2, 3])

>>> a[a != 3]
array([1, 2, 4, 5])

>>> np.where(a > 2, a, 0)
array([0, 0, 3, 4, 5])
```

Numerical Operations

Vector Operations

```
>>> a = np.array([1, 2, 3, 4, 5])
>>> b = np.array([6, 7, 8, 9, 10])

>>> a+b
array([ 7,  9, 11, 13, 15])

>>> a-b
array([-5, -5, -5, -5, -5])

>>> b/a
array([6.        , 3.5       , 2.66666667, 2.25      ,2.        ])

>>> b//a
array([6, 3, 2, 2, 2], dtype=int32)

>>> a*b
array([ 6, 14, 24, 36, 50])

>>> a**b
array([     1,     128,    6561,  262144, 9765625], dtype=int32)

>>> np.exp(a)
array([  2.71828183,   7.3890561 ,  20.08553692,  54.59815003, 148.4131591 ])

>>> np.log1p(b)
array([1.94591015, 2.07944154, 2.19722458, 2.30258509, 2.39789527])

>>> np.sin(a)
array([[-0.95892427,  0.84147098],
       [ 0.84147098,  0.14112001]])

>>> np.sqrt(b)
array([3.46410162, 3.16227766])

>>> np.ceil(np.array([1.2, 2.3, 3.4, 4.5, 5.6, 7.6]))
array([2., 3., 4., 5., 6., 8.])

>>> np.floor(np.array([1.2, 2.3, 3.4, 4.5, 5.6, 7.6]))
array([1., 2., 3., 4., 5., 7.])

>>> np.round(np.array([1.2, 2.3, 3.4, 4.5, 5.6, 7.6]))
array([1., 2., 3., 4., 6., 8.])

>>> np.abs(np.array([-1, 2, -3, -4, 5, 6, -7]))
array([1, 2, 3, 4, 5, 6, 7])
```

Statistics

```
>>> a = np.array([1, 2, 3, 4, 5])

>>> np.mean(a)
3.0

>>> a.sum()
15

>>> a.max()
5

>>> a.min()
1

>>> np.var(a)
2.0

>>> np.std(a)
1.4142135623730951

>>> np.corrcoef(a)
1.0
```

Linear Algebra

```
#  Solve the linear system
```

$$\begin{cases} 5x_0 + x_1 = 12 \\ x_0 + 3x_1 = 10 \end{cases}$$

```
>>> a = np.array([[5, 1], [1, 3]])
>>> b = np.array([12, 10])

>>> np.linalg.solve(a, b)
array([1.85714286, 2.71428571])

#  Solve the linear system
```

$$\begin{cases} x_0 + 2x_1 + 3x_2 = 10 \\ 4x_0 + 5x_1 + 6x_2 = 11 \\ 7x_0 + 8x_1 + 9x_2 = 12 \end{cases}$$

```
>>> a = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])
>>> b = np.array([10, 11, 12 ])

>>> np.linalg.solve(a, b)
arrayarray([-25.33333333,  41.66666667, -16.])
```

PANDAS Cheat Sheet

```
import pandas as pd
```

Data Structures

Series

```
# Create a series
>>> s = pd.Series([1, 2, 3], index=['A' ,'B' ,'C'])
```

Index	Data
A	1
B	2
C	3

```
>>> s.index
Index(['A', 'B', 'C'], dtype='object')

>>> s.dtype
dtype('int64')

>>> s.size
3

>>> s.ndim
1

>>> s.values
array([1, 2, 3], dtype=int64)
```

DataFrame

```
# Create a DataFrame
>>> df = pd.DataFrame({'ID': [1, 2, 3],
                       'Name': ['Alex', 'Brian', 'David'],
                       'Profession': ['DA', 'DE', 'DS']},
                        index = ['1 (One)', '2 (Two)', '3 (Three)'])
```

	ID	Name	Profession
1 (One)	1	Alex	DA
2 (Two)	2	Brian	DE
3 (Three)	3	David	DS

```
# Get the shape of the DataFrame
>>> df.shape
(3, 3)

# Get the column names of the DataFrame
>>> df.columns
Index(['ID', 'Name', 'Profession'], dtype='object')

# Get the index information of the DataFrame
>>> df.index
Index(['1 (One)', '2 (Two)', '3 (Three)'], dtype='object')

# Get the data types of the DataFrame columns
>>> df.dtypes
ID             int64
Name          object
Profession    object
dtype: object

# Check if there are any missing values in the DataFrame
>>> df.isnull().values.any()
False

# Get the count of missing values in each column
>>> df.isnull().sum()
ID            0
Name          0
Profession    0
dtype: int64

# Get the count of non-null values in each column
>>> df.count()
ID            3
Name          3
Profession    3
dtype: int64

# Generate descriptive statistics of the DataFrame (transposed)
>>> df.describe().T
```

Importing & Exporting

```
# Read a CSV file and create a DataFrame
>>> pd.read_csv('filename.csv')

# Save the DataFrame to a CSV file
df.to_csv(filename.csv)

# Read an Excel file and create a DataFrame
>>> pd.read_excel('filename.xlsx')

# Save the DataFrame to an Excel file
>>> df.to_excel('filename.xlsx')
```

Data Manipulation

Selection

Selecting Rows

```
# Select a specific row by its label
>>> df.loc['1 (One)']:

# Select a specific row by its index
>>> df.loc[1]
```

Selecting Columns

```
# Select a single column by its name
>>> df['ID']

# Select multiple columns by their name
>>> df[['ID', 'Profession']]

# Select a single column using label-based indexing
>>> df.loc[:, 'Name']

# Select a single column using integer-based indexing
>>> df.iloc[:, 1]
```

Selecting Rows and Columns

```
# Select a specific cell by its label-based row and column indices
>>> df.loc['2 (Two)', 'Name']

# Select a specific cell by its integer-based row and column indices
>>> df.iloc[1, 1]

# Select specific rows and columns using label-based indexing
>>> df.loc['1 (One)', ['ID', 'Profession']]

# Select specific rows and columns using integer-based indexing
>>> df.iloc[1, ['ID', 'Profession']]
```

Conditional Selection

```
# Select rows based on a condition
>>> df[df['ID'] > 1]

# Select rows based on multiple conditions using logical operators
>>> df[(df['ID'] > 2) & (df['Profession'] == 'DS')]

# Select rows where a column value is present in a given list
>>> df.loc[df['Name'].isin(['Alex', 'David'])]
```

Deleting & Adding

```
# Drop rows with any missing values
>>> df.dropna()

# Drop columns with any missing values
>>> df.dropna(axis=1)

# Drop columns with fewer than n non-null values
>>> df.dropna(axis=1, thresh=n)

# Fill missing values with a specified value
df.fillna(value)
```

Filtering & Sorting

```
# Filter columns using regular expression matching
>>> df.filter(regex=regex)


# Sort DataFrame by values in a specific column in descending order
>>> df.sort_values('ID', ascending=False)
```

Editting

```
# Perform quantile-based discretization of values in 'col' into 3
bins with custom labels
>>> pd.qcut(df['col'], 3, labels=qcut_labels)

# Perform value-based discretization of values in 'col' into custom
bins with labels
>>> pd.cut(df['col'], bins=cut_bins, labels=cut_labels)

# Create a pivot table using 'col1' as index, 'col2' as columns, and
'col3' as values
>>> df.pivot(index='col1', columns='col2', values='col3')

# Reset the index of the DataFrame
>>> df.reset_index()

# Rename specific columns of the DataFrame
>>> df.rename(columns = rename_list)
```

Grouping & Aggregation

```
# Perform grouping operation on 'col' and obtain a GroupBy object
>>> df.groupby('col')


# Perform grouping operation on multiple columns ('col1' and 'col2')
and calculate the mean of 'col3'
>>> df.groupby(['col1', 'col2']).agg({"col3": "mean"})

# Perform grouping operation on multiple columns ('col1' and 'col2')
and calculate the mean of 'col3' and count of 'col4'
>>> df.groupby(['col1', 'col2']).agg({'col3': 'mean', 'col4':
'count'})


# Grouping the DataFrame by the first level of the index
>>> df.groupby(level=0)


# An example
>>> df = pd.DataFrame({
    'Name': ['John', 'Alice', 'John', 'Alice', 'Bob'],
    'City': ['New York', 'Paris', 'London', 'Paris', 'London'],
    'Age': [30, 25, 35, 28, 40],
    'Salary': [50000, 60000, 55000, 45000, 70000]})


>>> df.groupby('City').agg({'Salary': 'mean', 'Age': 'max'})
```

	Name	City	Age	Salary
0	John	New York	30	50000
1	Alice	Paris	25	60000
2	John	London	35	55000
3	Alice	Paris	28	45000
4	Bob	London	40	70000

→

City	Salary	Age
London	62500.0	40
New York	50000.0	30
Paris	52500.0	28

Joining

```
# Merge operation
>>> df1 = pd.DataFrame({
        'key': ['A', 'B', 'C', 'D'],
        'value1': [1, 2, 3, 4]})

>>>> df2 = pd.DataFrame({
        'key': ['B', 'D', 'E', 'F'],
        'value2': [5, 6, 7, 8]})

>>> pd.merge(df1, df2, on='key', how='inner')
```

	Key	Value1
0	A	1
1	B	2
2	C	3
3	D	4

+

	Key	Value2
0	B	5
1	D	6
2	E	7
3	F	8

=

	Key	Value1	Value2
0	B	2	5
1	D	4	6

```
>>> df4 = pd.DataFrame({
        'key': ['B', 'D', 'E', 'F'],
        'value4': ['orange', 'grape', 'kiwi', 'lemon']})

>>> df3.set_index('key').join(df4.set_index('key'), how='inner')
```

	Key	Value3
0	A	apple
1	B	bannana
2	C	cherry
3	D	date

+

	Key	Value2
0	B	orange
1	D	grape
2	E	kiwi
3	F	lemon

=

Key	Value1	Value2
B	banana	orange
D	date	grape

```
# Example for concatenate operation
>>> df5 = pd.DataFrame({
        'A': ['A0', 'A1', 'A2'],
        'B': ['B0', 'B1', 'B2']})

>>> df6 = pd.DataFrame({
       'A': ['A3', 'A4', 'A5'],
       'B': ['B3', 'B4', 'B5']})

>>> pd.concat([df5, df6], axis=0)
```

	A	B
0	A0	B0
1	A1	B1
2	A2	B2

+

	A	B
0	A3	B3
1	A4	B4
2	A5	B5

=

	A	B
0	A0	B0
1	A1	B1
2	A2	B2
0	A3	B3
1	A4	B4
2	A5	B5

Statistics

```
# Generate descriptive statistics of the DataFrame
>>> df.describe()

# Calculate the mean of each column in the DataFrame
>>> df.mean()

# Calculate the correlation matrix of the DataFrame
>>> df.corr()

# Count the number of non-null values in each column of the DataFrame
>>> df.count()

# Find the maximum value in each column of the DataFrame
>>> df.max()

# Find the minimum value in each column of the DataFrame
>>> df.min()

# Calculate the median of each column in the DataFrame
>>> df.median()

# Calculate the standard deviation of each column in the DataFrame
>>> df.std()
```

SCIKIT-LEARN Cheat Sheet

Preprocessing

Splitting data into train and test sets

```
from sklearn.model_selection import train_test_split
X = df[["Independent Variables"]]
y = df[["Target Variable"]]
X_train, X_test, y_train, y_test = train_test_split(X, y, test size=0.2)
```

Handling Missing Values

```
from sklearn.impute import SimpleImputer, KNNImputer

# Dropping Rows or Columns
df.dropna(axis=0)
df.dropna(axis=1)

# Imputation
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.fit_transform(X_test)

# K-Nearest Neighbors (KNN) Imputation
knn_imputer = KNNImputer()
X_train_knn_imputed = knn_imputer.fit_transform(X_train)
X_test_knn_imputed = knn_imputer.fit_transform(X_test)
```

Handling Outliers

```
def suppress_outliers_iqr(df, col_name, q1=0.25, q3=0.75, multiplier=1.5):
    quartile1 = df[col_name].quantile(q1)
    quartile3 = df[col_name].quantile(q3)
    interquartile_range = quartile3 - quartile1
    lower_bound = quartile1 - multiplier * interquartile_range
    upper_bound = quartile3 + multiplier * interquartile_range

    # Suppress outliers by replacing them with the lower/upper bounds
    df.loc[df[col_name] < lower_bound, col_name] = lower_bound
    df.loc[df[col_name] > upper_bound, col_name] = upper_bound

# Example usage
suppress_outliers_iqr(dataframe, 'A')
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Numeric feature scaling (StandardScaler)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


# Numeric feature scaling (MinMaxScaler)
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Feature Encoding

```
from sklearn.preprocessing import OneHotEncoder

# Categorical feature encoding (One-Hot Encoder)
encoder = OneHotEncoder()
X_train_encoded = encoder.fit_transform(X_train)
X_test_encoded = encoder.transform(X_test)
```

Supervised Learning Algorithms

```
# Linear Regression
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)

# Logistic Regression
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)


# K-Nearest Neighbors (KNN)
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(X_train, y_train)


# CART
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)


# Random Forests
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train, y_train)


# GBM
from sklearn.ensemble import GradientBoostingClassifier
model = GradientBoostingClassifier()
model.fit(X_train, y_train)


# XGBoost
!pip install xgboost
from xgboost import XGBClassifier
model = XGBClassifier
model.fit(X_train, y_train)


# LightGBM
!pip install lightgbm
from lightgbm import LGBMClassifier
model = LGBMClassifier()
model.fit(X_train, y_train)


# Catboost
!pip install catboost
from catboost import CatBoostClassifier
model = CatBoostClassifie
model.fit(X_train, y_train)
```

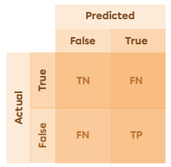
Unsupervised Learning Algorithms

```
# K-Means
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
model.fit(df)

# Hierarchical Clustering
from sklearn.cluster import AgglomerativeClustering
model = AgglomerativeClustering(n_clusters=5),
model.fit(df)

# PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(df)
```

Model Evaluation

Classification metrics

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score, classification_report
confusion_matrix

y_pred = model.predict(X_test)

# Confusion Matrix
confusion_matrix(y_test, y_pred)
```




```
# Accuracy = ( TP + TN ) / ( TP + TN + FP + FN )
accuracy_score(y_test, y_pred)

# Precision = TP / ( TP + FP )
 precision_score(y_test, y_pred)

# Recall = TP / ( TP + FN )
recall_score(y_test, y_pred)

# F1 Score = TN / ( TN + FP )
f1_score(y_test, y_pred)

# AUC
roc_auc_score(y_test, y_pred)

# Classification Report
classification_report(y_test, y_pred)
```

Regression metrics

```
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score

y_pred = model.predict(X_test)

# MAE
mean_absolute_error(y_test, y_pred)

# MSE
mean_squared_error(y_test, y_pred)

# RMSE
np.sqrt(mean_squared_error(y, y_pred))

# R-Square
r2_score(y_test, y_pred)

# Cross-Validation
cross_val_score(model, X, y, cv=5)
```

Clustering metrics

```
from sklearn.metrics import adjusted_rand_score,
homogeneity_score, v_measure_score


# Adjusted Rand Index
adjusted_rand_score(y_true, y_pred)

# Homogeneity
homogeneity_score(y_true, y_pred)

# V-measure
metrics.v_measure_score(y_true, y_pred))
```

Model Tuning

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

# Logistic Regression parameter optimization
lr_param_grid = {'C': [0.1, 1, 10], 'penalty': ['l1', 'l2']}
lr_grid_search = GridSearchCV(LogisticRegression(), lr_param_grid, cv=5)
lr_grid_search.fit(X_train, y_train)
lr_grid_search.best_params_    # Best parameters for Logistic Regression
lr_grid_search.best_score_     # Best score for Logistic Regression

# Decision Tree parameter optimization
dt_param_grid = {'max_depth': [None, 5, 10], 'min_samples_split': [2, 5, 10]}
dt_grid_search = GridSearchCV(DecisionTreeClassifier(), dt_param_grid, cv=5)
dt_grid_search.fit(X_train, y_train)
dt_grid_search.best_params_     # Best parameters for Decision Tree
dt_grid_search.best_score_      # Best score for Decision Tree

# Random Forest parameter optimization
rf_param_grid = {'n_estimators': [100, 200, 300], 'max_depth': [None, 5, 10]}
rf_random_search = RandomizedSearchCV(RandomForestClassifier(), rf_param_grid,
n_iter=5, cv=5)
rf_random_search.fit(X_train, y_train)
rf_random_search.best_params_    # Best parameters for Random Forest
rf_random_search.best_score_     # Best score for Random Forest

# K-Nearest Neighbors parameter optimization
knn_param_grid = {'n_neighbors': [3, 5, 7], 'weights': ['uniform','distance']}
knn_random_search = RandomizedSearchCV(KNeighborsClassifier(), knn_param_grid,
n_iter=5, cv=5)
knn_random_search.fit(X_train, y_train)
knn_random_search.best_params_   # Best parameters for K-Nearest Neighbors
knn_random_search.best_score_    # Best score for K-Nearest Neighbors

# GBM parameter optimization
gbm_param_grid = {'n_estimators': [100, 200, 300], 'max_depth': [None, 5, 10]}
gbm_grid_search = GridSearchCV(GradientBoostingClassifier(), gbm_param_grid,cv=5)
gbm_grid_search.fit(X_train, y_train)
gbm_grid_search.best_params_    # Best parameters for GBM
gbm_grid_search.best_score_     # Best score for GBM

# LightGBM parameter optimization
lgb_param_grid = {'n_estimators': [100, 200, 300], 'max_depth': [None, 5, 10]}
lgb_grid_search = GridSearchCV(lgb.LGBMClassifier(), lgb_param_grid, cv=5)
lgb_grid_search.fit(X_train, y_train)
lgb_grid_search.best_params_   # Best parameters for LightGBM
lgb_grid_search.best_score_    # Best score for LightGBM

# XGBoost parameter optimization
xgb_param_grid = {'n_estimators': [100, 200, 300], 'max_depth': [None, 5, 10]}
xgb_grid_search = GridSearchCV(xgb.XGBClassifier(), xgb_param_grid, cv=5)
xgb_grid_search.fit(X_train, y_train)
xgb_grid_search.best_params_  # Best parameters for XGBoost
xgb_grid_search.best_score_   # Best score for XGBoost


# CatBoost parameter optimization
cat_param_grid = {'iterations': [100, 200, 300], 'depth': [4, 6, 8]}
cat_random_search = RandomizedSearcahCV(CatBoostClassifier(), cat_param_grid,
n_iter=5, cv=5)
cat_random_search.fit(X_train, y_train)
cat_random_search.best_params_ # Best parameters for CatBoost
cat_random_search.best_score_  # Best score for CatBoost

# K-means parameter optimization
kmeans_param_grid = {'n_clusters': [3, 5, 7], 'init': ['k-means++', 'random']}
kmeans_grid_search = GridSearchCV(KMeans(), kmeans_param_grid, cv=5)
kmeans_grid_search.fit(X_train)
kmeans_grid_search.best_params_ # Best parameters for K-Means
kmeans_grid_search.best_score_  # Best score for K-Means

# PCA parameter optimization
pca_param_grid = {'n_components': [2, 5, 10]}
pca_grid_search = GridSearchCV(PCA(), pca_param_grid, cv=5)
pca_grid_search.fit(X_train)
pca_grid_search.best_params_ # Best parameters for PCA
pca_grid_search.best_score_  # Best score for CatBoost
```

Data Visualization

Cheat Sheet

Importing Libraries

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

Basic Line Plot

```
plt.plot(x, y, linestyle = '-', color = 'b')
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title')
plt.show()
```

Example

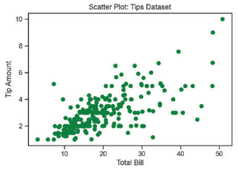
```
# Sample dataset
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May']
revenue = [10000, 15000, 12000, 18000, 20000]

# Create the line plot
plt.plot(months, revenue, marker='o', linestyle='-', color='b')
plt.xlabel('Months')
plt.ylabel('Revenue ($)')
plt.title('Monthly Revenue')
plt.grid(True)
plt.show()
```

Scatter Plot

```
plt.scatter(x, y, c='color')
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title')
plt.show()
```

Example

```
# Load the tips dataset
df = sns.load_dataset("tips")

# Extract the total bill and tip amounts
total_bill = df["total_bill"].values
tip_amount = df["tip"].values

# Create a scatter plot
plt.scatter(total_bill, tip_amount, c='green')
plt.xlabel('Total Bill')
plt.ylabel('Tip Amount')
plt.title('Scatter Plot: Tips Dataset')
plt.show()
```

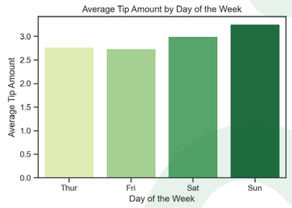


Bar Plot

```
plt.bar(x, y, color = 'b', width = 0.5, height = 0.3)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title')
plt.show()

# Horizontal bar
plt.barh(x, y)
plt.show()
```

Example

```
# Load the 'tip' dataset from seaborn
tips = sns.load_dataset('tips')

# Calculate the average tip amount for each day of the week
avg_tip_by_day = tips.groupby('day')['tip'].mean()

# Create the bar plot
sns.barplot(x=avg_tip_by_day.index, y=avg_tip_by_day.values)
plt.xlabel('Day of the Week')
plt.ylabel('Average Tip Amount')
plt.title('Average Tip Amount by Day of the Week')
plt.show()
```

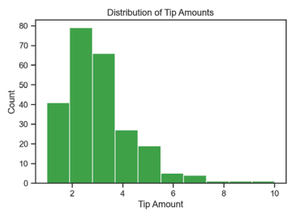


Histogram

```
plt.hist(data, bins=10)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title')
plt.show()
```

Example

```
# Load the 'tip' dataset from seaborn
tips = sns.load_dataset('tips')

# Create the histogram
sns.histplot(data=tips, x='tip', bins=10, color='green')
plt.xlabel('Tip Amount')
plt.ylabel('Count')
plt.title('Distribution of Tip Amounts')
plt.show()
```

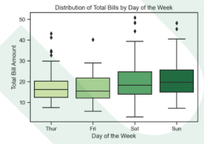


Box Plot

```
sns.boxplot(data, x, y, color)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title')
plt.show()
```

Example

```
# Load the 'tip' dataset from seaborn
tips = sns.load_dataset('tips')

# Create the box plot with a different color
sns.boxplot(data=tips, x='day', y='total_bill',
color='green')
plt.xlabel('Day of the Week')
plt.ylabel('Total Bill Amount')
plt.title('Distribution of Total Bills by Day of the Week')
plt.show()
```

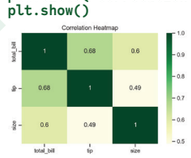


Heatmap

```
sns.heatmap(data, annot=True)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title')
plt.show()
```

Example

```
# Load the 'tip' dataset from seaborn
tips = sns.load_dataset('tips')

# Calculate the correlation matrix
correlation_matrix = tips.corr()

# Create the heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='YlGn')
plt.title('Correlation Heatmap')
plt.show()
```

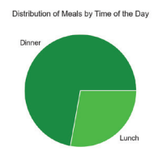


Pie Chart

```
plt.pie(y, labels=labels, colors=[colors])
plt.title('Title')
plt.show()
```

Example

```
# Load the 'tip' dataset from seaborn
tips = sns.load_dataset('tips')

# Calculate the count of meals for each time of the day
meal_counts = tips['time'].value_counts()

# Specify custom labels and colors
labels = meal_counts.index
colors = ['darkgreen', 'aquamarine']

# Create the pie chart
plt.pie(meal_counts, labels=labels, colors=colors)
plt.title('Distribution of Meals by Time of the Day')
plt.show()
```

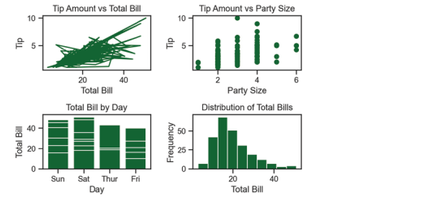


Multiple Subplots

```
fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(x1, y1)
axs[0, 1].scatter(x2, y2)
axs[1, 0].bar(x3, height3)
axs[1, 1].hist(data4, bins=10)
plt.show()
```

Example

```
# Load the 'tips' dataset from seaborn
tips = sns.load_dataset('tips')

# Prepare the data for plotting
x1 = tips['total_bill']
y1 = tips['tip']
x2 = tips['size']
y2 = tips['tip']
x3 = tips['day']
height3 = tips['total_bill']
data4 = tips['total_bill']

# Create the subplots
fig, axs = plt.subplots(2, 2)

# Plot 1: Line plot
axs[0, 0].plot(x1, y1)
axs[0, 0].set_xlabel('Total Bill')
axs[0, 0].set_ylabel('Tip')
axs[0, 0].set_title('Tip Amount vs Total Bill')

# Plot 2: Scatter plot
axs[0, 1].scatter(x2, y2)
axs[0, 1].set_xlabel('Party Size')
axs[0, 1].set_ylabel('Tip')
axs[0, 1].set_title('Tip Amount vs Party Size')

# Plot 3: Bar plot
axs[1, 0].bar(x3, height3)
axs[1, 0].set_xlabel('Day')
axs[1, 0].set_ylabel('Total Bill')
axs[1, 0].set_title('Total Bill by Day')

# Plot 4: Histogram
axs[1, 1].hist(data4, bins=10)
axs[1, 1].set_xlabel('Total Bill')
axs[1, 1].set_ylabel('Frequency')
axs[1, 1].set_title('Distribution of Total Bills')

# Adjust the layout and spacing
fig.tight_layout()

# Display the plot
plt.show()
```

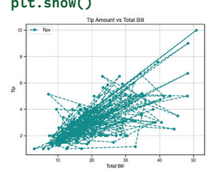


Customizing Plots

```
plt.figure(figsize=(8, 6))
plt.plot(x, y, color='red', linestyle='--', linewidth=2,
marker='o', markersize=6)
plt.xlabel('X-axis label', fontsize=12)
plt.ylabel('Y-axis label', fontsize=12)
plt.title('Title', fontsize=14)
plt.legend(['Legend'])
plt.grid(True)
plt.show()
```

Example

```
# Load the 'tip' dataset from seaborn
tips = sns.load_dataset('tips')

# Prepare the data for plotting
x = tips['total_bill']
y = tips['tip']

# Create the plot
plt.figure(figsize=(8, 6))
plt.plot(x, y, color='red', linestyle='--', linewidth=2,
marker='o', markersize=6)
plt.xlabel('Total Bill', fontsize=12)
plt.ylabel('Tip', fontsize=12)
plt.title('Tip Amount vs Total Bill', fontsize=14)
plt.legend(['Tips'])
plt.grid(True)
plt.show()
```



MS SQL Cheat Sheet

Basic SQL Commands

```
-- Select all columns from a table
SELECT * FROM table_name;

-- Select specific columns from a table
SELECT column1, column2 FROM table_name;

-- Filter rows with a condition
SELECT * FROM table_name WHERE condition;

-- Insert a new row into a table
INSERT INTO table_name (column1, column2)
VALUES (value1, value2);

-- Update existing rows in a table
UPDATE table_name SET column1 = value1,
column2 = value2 WHERE condition;

-- Delete rows from a table
DELETE FROM table_name WHERE condition;
```

Aggregation Functions

```
-- Calculate the sum of a column
SELECT SUM(column_name) FROM table_name;

-- Calculate the average of a column
SELECT AVG(column_name) FROM table_name;

-- Get the maximum value from a column
SELECT MAX(column_name) FROM table_name;

-- Get the minimum value from a column
SELECT MIN(column_name) FROM table_name;

-- Count the number of rows in a table
SELECT COUNT(*) FROM table_name;
```

Sorting and Grouping

```
-- Order rows in ascending order
SELECT * FROM table_name ORDER BY column_name
ASC;

-- Order rows in descending order
SELECT * FROM table_name ORDER BY column_name
DESC;

-- Group rows based on a column
SELECT column_name, COUNT (*) FROM table_name
GROUP BY column_name;
```

Joins

```
-- Inner Join
SELECT * FROM table1 INNER JOIN table2
ON table1.column_name = table2.column_name;

-- Left Join
SELECT * FROM table1 LEFT JOIN table2
ON table1.column_name = table2.column_name;

-- Right Join
SELECT * FROM table1 RIGHT JOIN table2
ON table1.column_name = table2.column_name;

-- Full Outer Join
SELECT * FROM table1 FULL OUTER JOIN table2
ON table1.column_name = table2.column_name;
```

Filtering

```
-- Using the LIKE operator for pattern matching
SELECT * FROM table_name WHERE column_name
LIKE 'pattern%';

-- Using the IN operator to filter by multiple values
SELECT * FROM table_name WHERE column_name
IN (value1, value2, value3);

-- Using the BETWEEN operator to filter within a range
SELECT * FROM table_name WHERE column_name
BETWEEN value1 AND value2;

-- Using the IS NULL operator to filter NULL values
SELECT * FROM table_name WHERE column_name IS NULL;
```

Aliases and Calculated Columns

```
-- Alias for column names
SELECT column_name AS alias_name FROM table_name;

-- Calculated columns in SELECT
SELECT column1, column2, column1 + column2
AS calculated_column FROM table_name;
```

Conditional Statements

```
-- Simple CASE expression
SELECT column_name, CASE WHEN column_name = value1 THEN
'Result 1' WHEN column_name = value2 THEN 'Result 2'
ELSE 'Default Result' END AS result FROM table_name;
```

Working with Dates

```
-- Get the current date and time
SELECT GETDATE();

-- Format dates
SELECT CONVERT(varchar, date_column, 103)
AS formatted_date FROM table_name;

-- Extract parts of a date
SELECT DATEPART(year, date_column) AS year
FROM table_name;
```

Creating and Modifying Tables

```
-- Create a new table
CREATE TABLE table_name (
column1 datatype1 constraint1,
column2 datatype2 constraint2
);

-- Add a new column to an existing table
ALTER TABLE table_name ADD column_name datatype;

-- Modify an existing column
ALTER TABLE table_name ALTER COLUMN column_name
new_datatype;

-- Add a primary key constraint
ALTER TABLE table_name ADD CONSTRAINT pk_con-
straint
PRIMARY KEY (column_name);
```

Indexes

```
-- Create an index
CREATE INDEX index_name ON table_name(column_name);

-- Delete an index
DROP INDEX table_name.index_name;
```

Example

Sample Data

1.1 Employees Table

Employee ID	First Name	Last Name	DepartmentID	Salary
1	John	Doe	1	50000
2	Jane	Smith	2	60000
3	Michael	Johnson	1	55000
4	Emily	Williams	3	52000
5	William	Brown	2	48000

1.2 Departments Table

Department ID	Department Name
1	Sales
2	Marketing
3	HR
4	Finance

Selecting Data

```
-- Select all employees
SELECT * FROM Employees;

-- Select specific columns from Employees table
SELECT FirstName, LastName, Salary FROM Employees;

-- Filter employees in the Sales department
SELECT FirstName, LastName FROM Employees
WHERE DepartmentID = 1;

-- Select employees with a salary greater than 52000
SELECT FirstName, LastName, Salary
FROM Employees WHERE Salary > 52000;
```

Aggregation Functions

```
-- Calculate the total salary of all employees
SELECT SUM(Salary) AS TotalSalary FROM Employees;

-- Calculate the average salary
SELECT AVG(Salary) AS AverageSalary FROM Employees;

-- Get the highest salary
SELECT MAX(Salary) AS MaxSalary FROM Employees;

-- Count the number of employees in the Marketing
department
SELECT COUNT(*) AS NumberOfEmployees
FROM Employees WHERE DepartmentID = 2;
```

Sorting and Grouping

```
-- Order employees by salary in descending order
SELECT FirstName, LastName, Salary
FROM Employees ORDER BY Salary DESC;

-- Group employees by department and calculate the
total salary for each department
SELECT DepartmentID, SUM(Salary) AS TotalSalary
FROM Employees GROUP BY DepartmentID;
```

Joins

```
-- Inner Join to get employee details along with
their department names
SELECT e.FirstName, e.LastName, d.DepartmentName
FROM Employees e INNER JOIN Departments d
ON e.DepartmentID = d.DepartmentID;

-- Left Join to include all departments, even
those without employees
SELECT d.DepartmentName, e.FirstName, e.LastName
FROM Departments d LEFT JOIN Employees e
ON d.DepartmentID = e.DepartmentID;
```

Filtering

```
-- Using the LIKE operator to filter employees
with names starting with 'J'
SELECT FirstName, LastName FROM Employees
WHERE FirstName LIKE 'J%';

-- Using the IN operator to filter employees from
specific departments
SELECT FirstName, LastName FROM Employees
WHERE DepartmentID IN (1, 2);

-- Using the BETWEEN operator to filter employees
with salaries between 50000 and 55000
SELECT FirstName, LastName FROM Employees
WHERE Salary BETWEEN 50000 AND 55000;

-- Using the IS NULL operator to find employees
without a department
SELECT FirstName, LastName FROM Employees
WHERE DepartmentID IS NULL;
```

Aliases and Calculated Columns

```
-- Alias for column names
SELECT FirstName AS First, LastName AS Last
FROM Employees;

-- Calculated columns in SELECT
SELECT FirstName, LastName, Salary, Salary * 12
AS AnnualSalary FROM Employees;
```

Conditional Statements

```
-- Simple CASE expression to categorize employees
based on salary
SELECT FirstName, LastName,
CASE
WHEN Salary >= 55000 THEN 'High'
WHEN Salary >= 50000 THEN 'Medium'
ELSE 'Low'
END AS SalaryCategory FROM Employees;
```

Working with Dates

```
-- Get the current date
SELECT GETDATE() AS CurrentDate;

-- Format dates using CONVERT
SELECT FirstName, LastName,
CONVERT(varchar, HireDate, 103)
AS FormattedHireDate FROM Employees;

-- Extract the year from the HireDate
SELECT FirstName, LastName,
DATEPART(year, HireDate) AS HireYear
FROM Employees;
```

Creating and Modifying Tables

```
-- Create a new table
CREATE TABLE Customers (
CustomerID INT PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
Email VARCHAR (100)
);

-- Add a new column to an existing table
ALTER TABLE Employees ADD Age INT;

-- Modify an existing column
ALTER TABLE Employees ALTER COLUMN Salary
DECIMAL(10, 2);

-- Add a primary key constraint
ALTER TABLE Departments ADD CONSTRAINT PK_DepartmentID
PRIMARY KEY (DepartmentID);
```

Indexes

```
-- Create an index
CREATE INDEX IDX_Employees_DepartmentID
ON Employees ( DepartmentID);

-- Delete an index
DROP INDEX Employees.IDX_Employees_DepartmentID;
```

Probability

Cheat Sheet

Probability Basics

Experiment : A process that results in an outcome.

Sample Space (S) : The set of all possible outcomes of an experiment.

Event (E) : A subset of the sample space.

Probability (P) : A measure of the likelihood of an event occuring.

Probability Axioms

Non-Negativity : For all events A, $P(A) \geq 0$.

Additivity : For mutually exclusive events A and B,

- $P(A \cup B) = P(A) + P(B)$.

Normalization : The probability of the entire sample space S is 1:

- $P(S)=1$.

Discrete Uniform Law (Classical Probability) : For equally likely outcomes,

- $P(E) = \frac{Number\ of\ favorable\ outcomes}{Total\ number\ of\ outcomes}$

Relative Frequency Probability :

- $P(E) = \frac{Frequency\ of\ E\ occuring}{Total\ number\ of\ trials}$

Conditional Probability P (E|F) : Probability of E given that F has occurred. It is calculated as:

- $P(E \mid F) = \frac{P(E \cap F)}{P(F)}$ where $P(F) \neq 0$.

Bayes' Theorem

Bayes' Theorem helps us update our initial beliefs (prior probabilities) based on new information (likelihood) to arrive at a more accurate or informed estimate of the probability of an event occurring (posterior probability). Mathematically, Bayes' Theorem is stated as follows:

- $P(E \mid F) = \frac{P(F \mid E)\ .\ P(E)}{P(F)}$

Random Variables

Random Variable : A random variable assigns a numerical value to each outcome in a sample space. It can be discrete or continuous.

Expected Value : The expected value of a random variable represents the average or mean value it takes over all possible outcomes. It provides a measure of the central tendency of the distribution.

Variance : The variance of a random variable measures the extent to which the values of the variable deviate from its expected value. It quantifies the spread or dispersion of the distribution.

Discrete Random Variables

For a discrete random variable X, which takes on distinct values from a finite or countable set, we have the following key concepts:

Probability Mass Function (PMF): The PMF $P(X = x)$ of a discrete random variable X gives the probability that X takes on the value x. It describes the distribution of probabilities across all possible values of X.

Expected value of a discrete random variable X is calculated as the sum of each value x weighted by its probability $P(X = x)$:

- $E(X) = \sum_x x \cdot P(X = x)$

Variance of a discrete random variable X is calculated as the average of the squared differences between each value x and the expected value $E(X)$:

- $\mathrm{Var}(X) = E(X^2) - (E(X))^2$

Continuous Random Variables

For a continuous random variable X, which can take on any value within a certain range, we have the following key concepts:

Probability Density Function (PDF): The PDF $f(x)$ of a continuous random variable X provides the relative likelihood that X falls within a specific interval. Unlike discrete random variables, the probability that X takes on a specific value is generally zero. Instead, the probability is associated with intervals.

Expected value of a continuous random variable X is the integral of x multiplied by the $f(x)$ over the entire range of X:

- $E(X) = \int_{-\infty}^{\infty} x \cdot f(x)\, dx$

Variance of a continuous random variable X is calculated similarly:

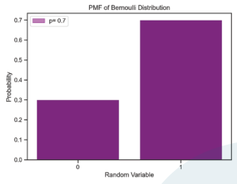
- $\mathrm{Var}(X) = E(X^2) - (E(X))^2$

Common Probability Distributions

Discrete Distributions

Bernoulli Distribution

The Bernoulli distribution models a random experiment with two possible outcomes, usually labeled as "success" and "failure".

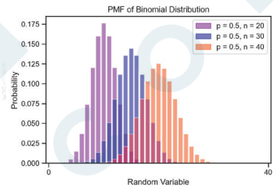


- **PMF :** $P(X=x) = p^x \cdot (1-p)^{1-x}$
- **Parameters :** p (probability of success)
- **Expected Value :** $E(X) = p$
- **Variance :** $Var(X) = p\ .\ (1-p)$

Binomial Distribution

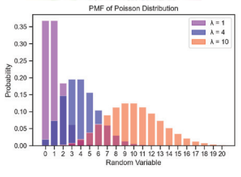
The binomial distribution describes the number of successes (S) in a fixed number of independent Bernoulli trials. Let us say you conduct "n" trials and you're interested in how many times you willl get a success. The binomial distribution provides the probabilities of getting 0, 1, 2, ..., n successes.

- **PMF :** $P(X = k) = C(n,k)\ .\ p^k\ .\ (1\text{-}p)^{n-k}$
- **Parameters :** n (number of trials), p (probability of success)
- **Expected Value :** $E(X) = n\ .\ p$
- **Variance :** $Var(X) = n\ .\ p\ .\ (1-p)$



Poisson Distribution

The Poisson distribution is a probability distribution that describes the number of events that occur in a fixed interval of time or space, given a certain average rate of occurrence.

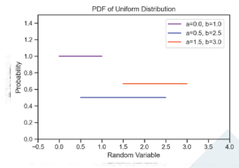


- **PMF :** $P(X = k) = \frac{\lambda^k\ .\ e^{-\lambda}}{k!}$
- **Parameters :** λ (average rate of events)
- **Expected Value :** $E(X) = \lambda$
- **Variance :** $Var(X) = \lambda$

Continuous Distributions

Uniform Distribution

The uniform distribution is a probability distribution that describes a situation where all outcomes in a given range are equally likely. In other words, each possible value within the range has the same probability of occurring.

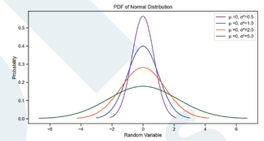


- **PMF :** $f(x) = \frac{1}{b-a}$ for $a \leq x \leq b$
- **Parameters :** a (lower bound), b (upper bound)
- **Expected Value :** $E(X) = a + b$
- **Variance :** $Var(X) = \frac{(b-a)^2}{12}$

Normal (Gaussian) Distribution

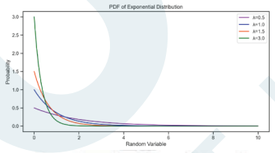
The normal distribution, also known as the Gaussian distribution or bell curve, is a fundamental concept in statistics and probability theory. It describes a symmetrical probability distribution that is characterized by its bell-shaped curve.

- **PDF :** $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
- **Parameters :** μ (mean), σ (standard deviation)
- **Expected Value :** $E(X) = \mu$
- **Variance :** $Var(X) = \sigma$



Exponential Distribution

The exponential distribution models the time between successive events in a process where events occur randomly and independently at a constant average rate.

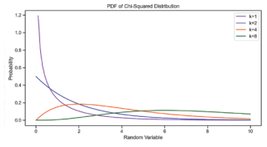


- **PDF :** $f(x) = \lambda e^{-\lambda x}$ for $x \geq 0$
- **Parameters :** λ (rate parameter)
- **Expected Value :** $E(X) = \frac{1}{\lambda}$
- **Variance :** $Var(X) = \frac{1}{\lambda^2}$

Chi-Squared Distribution

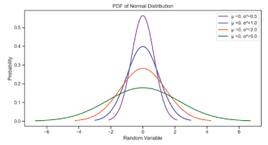
The chi-squared distribution emerges from the sum of the squares of independent standard normal random variables.

- **PDF :** $f(x) = \frac{1}{2^{k/2}\Gamma(k/2)} x^{k/2-1} e^{-x/2}$ for $x \geq 0$
- **Parameters :** k (degrees of freedom)
- **Expected Value :** $E(X) = k$
- **Variance :** $Var(X) = 2k$



Student's t-Distribution

The t-distribution models the variability of sample means when drawing small samples from a population. It is characterized by its bell-shaped curve, similar to the normal distribution.

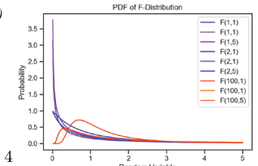


- **PDF :** Depends on degrees of freedom (v)
- **Parameters :** v (degrees of freedom)
- **Expected Value :** $E(X) = 0$ for $v > 1$
- **Variance :** $Var(X) = v$ for $v > \frac{2}{v-2}$

F-Distribution

The F-distribution models the ratio of two independent chi-squared distributions divided by their respective degrees of freedom.

- **PDF :** Depends on degrees of freedom ($d1$ and $d2$)
- **Parameters :** d_1 (numerator degrees of freedom), d_2 (denominator degrees of freedom)
- **Expected Value :** $E(X) = \frac{d_2}{d_2\text{-}2}$ for $d_2 > 2$
- **Variance :** $Var(X) = \frac{2d_2^2(d_1+d_2-2)}{d_1(d_2-2)^2(d_2-4)}$ for $d_2 > 4$



Fundamental Results

The Law of Large Numbers

Weak Law of Large Numbers (WLLN) : The sample mean of a large number of independent and identically distributed random variables approaches the expected value.

Strong Law of Large Numbers (SLLN) : The sample mean converges almost surely to the expected value.

```python
# Import necessary libraries
import random
import matplotlib.pyplot as plt

# Function to simulate rolling a fair six-sided die
def roll_die():
    return random.randint(1, 6)

# Number of trials to perform for each sample size
num_trials = 10000

# List of sample sizes to investigate        ]
sample_sizes = [10, 50, 100, 500, 1000, 5000

# Iterate through each sample size
for sample_size in sample_sizes:
    trial_means = []

    # Perform num_trials trials for the current sample size
    for _ in range(num_trials):
        # Simulate rolling the die 'sample_size' times and
          calculate the mean                (sample_size)]
        sample = [roll_die() for _ in  range
        mean = sum(sample) / sample_size
        trial_means.append(mean)

    # Create a histogram of trial means with 20 bins and add
      it to the plot                        ,
    plt.hist(trial_means , bins=20, alpha=0.5    '  )
             label=f'Sample size: {sample_size}

    # Label the x and y axes and set the title of the plot
    plt.xlabel('Sample Mean')
    plt.ylabel('Frequency')
    plt.title('Law of Large Numbers')

    plt.legend()

plt.show()
```

The Central Limit Theorem

The distribution of the sum (or average) of a large number of independent, identically distributed random variables approaches a normal distribution.

```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt

# Define the size of the population and generate a random
population with an exponential distribution
population_size = 100000
population = np.random.exponential(scale=2, size=population_size)

# Define a list of sample sizes that we want to explore
sample_sizes = [10, 50, 100, 500]

# Iterate through each sample size
    sample_size in sample_sizes:
    # Generate 1000 random samples of the specified size from the
      population
    sample_means = [np.mean(np.random.choice(population,
                   size=sample size)) for _ in range(1000)]
    # Create a histogram of the sample means with 30 bins and add
      it to the plot
    plt.hist(sample_means, bins=30, alpha=0.5,
             label=f'Sample size: {sample_size}')

# Add labels and a title to the plot
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
plt.title('Central Limit Theorem')

plt.legend()
```