Question 1.1: Write the Answer to these questions. Note: Give at least one example for each of the questions.

- **What is the difference between static and dynamic variables in Python?**

Ans:

Static Variables (Class Variables):
Static variables belong to the class itself, not to any specific instance (object) of the class.
They are shared among all instances of the class.
Defined within the class but outside any method.
Example:
Python

```
class MyClass:
    static_var = 10

obj1 = MyClass() obj2
= MyClass()

print(obj1.static_var)  # Output: 10
print(obj2.static_var)  # Output: 10
```

Dynamic Variables (Instance Variables):
Instance variables are specific to an instance (object) of the class.
Each object can have different values for instance variables.
Defined within methods using self.
Example:
Python

```
class Person:     def
__init__(self, name):
        self.name = name

person1 = Person("Alice")
person2 = Person("Bob")

print(person1.name)  # Output: Alice
print(person2.name)  # Output: Bob
```

- **Explain the purpose of "pop(), popitem() and clear()' in a dictionary with suitable examples.**

pop(key, default):
Removes the item with the specified key and returns its value.
If the key is not found, it returns the specified default value (or raises an error if not provided).
Example:
Python

```
my_dict = {"a": 1, "b": 2, "c": 3}

value_b = my_dict.pop("b", None)  # Removes "b" and returns 2
print(my_dict)  # Output: {"a": 1, "c": 3}
```

popitem():
Removes and returns an arbitrary (key, value) pair from the dictionary.
Useful when you don't care which item gets removed.
Example:
Python

```
my_dict = {"x": 42, "y": 99, "z": 73}

key, value = my_dict.popitem()  # Removes an arbitrary pair (e.g., ("x", 42))
print(key, value)  # Output: "x" 42
```

clear(): Wiping the Slate Clean
The clear() method in Python is like a magical eraser for dictionaries. When you invoke it on a dictionary, it wipes out all its key-value pairs, leaving you with an empty dictionary—a blank canvas ready for new adventures.

Here's how it works:

Python

```
my_dict = {"a": 1, "b": 2, "c": 3}

# Clear the dictionary
my_dict.clear()

print(my_dict)  # Output: {}
```

- **What do you mean by FrozenSet? Explain it with suitable examples.**

FrozenSet:
A frozenset is an immutable set in Python.
It's useful when you need a hashable collection (e.g., for dictionary keys).
Example:
Python

```
my_set = frozenset([1, 2, 3])
print(my_set)  # Output: frozenset({1, 2, 3})
```

Differentiate between mutable and immutable data types in Python and give examples of mutable and immutable data types.

Mutable:
Can be modified after creation.
Examples: lists, dictionaries, sets. Example:
Python

```
my_list = [1, 2, 3]
my_list.append(4)  # Modifies the list
```

Immutable:
Cannot be modified after creation.
Examples: integers, floats, strings, tuples. Example:
Python

```
my_string = "Hello"
new_string = my_string + " World"  # Creates a new string
```

- ## What is __init__?Explain with an example.

__init__: The Constructor

The __init__ method (often referred to as the constructor) is like the welcoming committee for Python objects. When you create an instance of a class (i.e., an object), Python automatically calls this method. Its purpose? To set up the object's initial state—like a cosmic birth certificate for your data structures.

What Does It Do?

Initializes instance variables (attributes) for an object.

Runs automatically when you create an object from a class.

How Does It Work?

Defined within the class, just like any other method.

Accepts self as its first parameter (which refers to the object being created). You can add additional parameters to __init__ to customize object creation. Why it is used?
It lets you set up the object's initial state.
You can assign default values to instance variables.
It's where you can perform any necessary setup tasks.
Enough theory! Let's see it in action:

Python

```
class Dog:     def __init__(self,
name, breed):        # Initialize
instance variables        self.name =
name        self.breed = breed

   def bark(self):
      print(f"{self.name} says woof!")

# Creating a Dog object my_dog = Dog(name="Buddy",
breed="Golden Retriever")

# Accessing instance variables print(f"My dog's name is {my_dog.name}.")  #
Output: My dog's name is Buddy.

# Calling a method my_dog.bark()  # Output:
Buddy says woof!
```

- **What is docstring in Python?Explain with an example.**

  A docstring is a string literal placed as the first statement inside a function, method, or class.
  It serves as documentation, providing information about what the function or class does, its parameters, and return values.
  Think of it as a user manual for your code.
  Why Bother with Docstrings?
  Helps other developers (including future you) understand your code.

Can be accessed using tools like help() or IDE features.
Good practice for maintaining clean, well-documented code.
Here's an example:

Python

```
def greet(name):
    """
    Greets a person by name.

    Args:
        name (str): The name of the person.

    Returns:
        str: A greeting message.

    return f"Hello, {name}!"

# Usage print(greet("Alice"))  # Output:
Hello, Alice!
```

In this example:

The docstring explains what the greet function does, its parameter (name), and its return value.
When you call help(greet) or use an IDE, you'll see this helpful description.

- **What are unit tests in Python?**

Ans:

Unit tests are small, isolated tests that verify individual parts (units) of your code.
They ensure that each function or method behaves correctly.
Popular testing frameworks include unittest, pytest, and nose.
Why Use Unit Tests?
Detect bugs early.
Provide confidence during code changes.
Document expected behavior.
Example (using unittest):

Python

```
import unittest
```

```
def add(a, b):
return a + b class
TestAddFunction(u
nittest.TestCase):
def
test_add_positive
_numbers(self):
    self.assertEqual(add(2, 3), 5)

  def test_add_negative_numbers(self):
    self.assertEqual(add(-2, -3), -5)

if __name__ == "__main__":
unittest.main()
```

- **What is break, continue and pass in Python?**

break, continue, and pass: The Flow Control Trio break: Exits the current loop (for loop or while loop). continue: Skips the current iteration and moves to the next one.
pass: Acts as a placeholder (does nothing) when you need syntactically valid code but no actual behavior.
Example:

Python

```
for i in range(5):
if i == 2:
    break  # Exit the loop when i is 2
print(i)

# Output: 0, 1

for j in range(5):
if j == 2:
    continue  # Skip iteration when j is 2
print(j)

# Output: 0, 1, 3, 4

def placeholder_function():
  pass  # Placeholder function with no behavior
```

• What is the use of self in Python?

The Purpose of self in Python **self**
refers to the instance of the class.
It's used within instance methods to access instance variables and other methods.
Example (within a class):
Python

```
class Person:     def
__init__(self, name):
       self.name = name

   def greet(self):
      print(f"Hello, I'm {self.name}!")

person = Person("Bob")
person.greet()  # Output: Hello, I'm
Bob!
```

• What are global, protected and private attributes in Python?

In Python, attributes refer to the variables associated with an object. They store information about the object's state. There are three main types of attributes:

Global Attributes: These are variables defined outside any function or class. They have global scope and can be accessed from any part of the code. For example:

Python

```
global_var = 42

def some_function():
print(global_var)

some_function()  # Prints 42
```

Protected Attributes (also known as "protected members"): These attributes are indicated by a single leading underscore (e.g., _protected_var). While not strictly enforced by the language, this convention signals that the attribute should be considered non-public and

should not be accessed directly from outside the class. However, it's still accessible if needed. Example: Python

```
class MyClass:
def __init__(self):
    self._protected_var = 10

obj = MyClass()
print(obj._protected_var)  # Accessing the protected attribute
```

Private Attributes (also known as "private members"): These attributes are indicated by a double leading underscore (e.g., __private_var). Python name mangling ensures that these attributes are not directly accessible from outside the class. Example:

Python

```
class MyClass:
def __init__(self):
    self.__private_var = 20

obj = MyClass()
# The following line would raise an AttributeError:
# print(obj.__private_var)
```

6. What are modules and packages in Python?

Modules: A module in Python is a file containing Python code (functions, classes, or variables) that can be imported and used in other Python programs. Modules help organize code and promote reusability. For example, the math module provides mathematical functions:

Python

```
import math

print(math.sqrt(25))  # Computes the square root of 25
```

Packages: A package is a collection of related modules organized in directories. Packages allow you to create a hierarchical structure for your code. A package typically contains an

__init__.py file (which can be empty) to indicate that the directory is a package. For instance, the numpy package contains various submodules:

my_project/ ├──

my_package/

| ├── __init__.py

| ├── module1.py

| └── module2.py

└── main.py

In main.py, you can use:

Python

from my_package import module1

- **What are lists and tuples? What is the key difference between the two?**

  Lists and Tuples:

  Lists: Ordered, mutable collections of items. Elements can be of different types. Example:

  Python

  my_list = [1, 2, 'hello', 3.14]

  Tuples: Ordered, immutable collections. Typically used for fixed data. Example: Python

  my_tuple = (10, 20, 30)

  Key Difference: Lists can be modified (add, remove, change), while tuples cannot.

- **What is an Interpreted language & dynamically typed language?Write 5 differences between them.**

Interpreted vs. Dynamically Typed Languages:

Interpreted Language (e.g., Python):

Code is executed line-by-line by an interpreter.

No separate compilation step.

Easier to debug.

Example: Python, JavaScript.

Dynamically Typed Language:

Variable types are determined at runtime.

No need to declare types explicitly.

More flexible but can lead to runtime errors.

Example: Python, Ruby.

Differences:

Interpreted languages execute code directly; compiled languages (like C++) generate machine code.

Dynamic typing allows changing variable types; static typing (like Java) requires type declarations.

Interpreted languages are generally slower than compiled ones.

Interpreted languages are more beginner-friendly.

Compiled languages often have better performance.

• What are Dict and list comprehensions?

Dictionary Comprehension:

A dictionary comprehension allows you to create dictionaries in a concise way. It's similar to a list comprehension but produces key-value pairs. Syntax: {key_expression: value_expression for item in iterable} Example:

Python

Create a dictionary of squares for numbers 1 to 5 squares_dict = {x: x**2 for x in range(1, 6)} print(squares_dict)

Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

List Comprehension:

A list comprehension constructs a new list by applying an expression to each item in an iterable.

Syntax: [expression for item in iterable if condition] Example:

Python

Create a list of even squares for numbers 1 to 10 even_squares = [x**2 for x in range(1, 11) if x % 2 == 0] print(even_squares)

Output: [4, 16, 36, 64, 100]

• What are decorators in Python? Explain it with an example,Write down its use cases.

Decorators modify or enhance functions/methods. Example:

Python

def my_decorator(func): def wrapper(): print("Something is happening before the function is called.") func()

```
    print("Something is happening after the function is called.")     return wrapper

@my_decorator def say_hello():
print("Hello!")

say_hello()
```

- **How is memory managed in Python?**

Memory Management in Python:

Python uses reference counting and a garbage collector.

Objects are deallocated when their reference count drops to zero.

Circular references are handled by the garbage collector.

- **What is lambda in Python? Why is it used?**

Lambda Functions:

Anonymous functions defined using lambda.

Used for short, simple operations.

Example:

```
add = lambda x, y: x + y result = add(3, 5)  # Result is
```

8

- **Explain split() and join() functions in Python?**

split() and join() Functions: split(): Splits a string into a list based on a

delimiter.

Python

sentence = "Hello, world!" words = sentence.split(", ") # ['Hello',

'world!']

join(): Joins elements of a list into a single string using a separator.

Python

words = ['Hello', 'world!'] sentence = ", ".join(words) # 'Hello,

world!'

- **What are iterators iterable & generators in Python?**

Iterators, Iterables, and Generators:

Iterables: An iterable is any Python object capable of returning its elements one at a time. Examples include lists, tuples, strings, dictionaries, and sets. You can loop over an iterable using a for loop.

Iterators: An iterator is an object that implements the iterator protocol. It provides a __next__() method to retrieve the next value from the sequence. Iterators are used to efficiently process large data without loading it all into memory.

Python

my_list = [1, 2, 3] my_iter = iter(my_list)

print(next(my_iter)) # 1 print(next(my_iter)) #

2

Generators: Generators are a type of iterator. They allow lazy evaluation—values are generated on-the-fly as needed. You define them using functions with yield statements. Python

```
def squares(n):    for i in range(n):
      yield i**2

for square in squares(5):
   print(square)
```

- **What is the difference between xrange and range in Python?**

Difference between range and xrange (in Python 2):

Both generate a sequence of numbers, but:

range: Creates a list in memory with all values. Eager evaluation. xrange (Python 2 only):

Generates values on-the-fly. Lazy evaluation.

In Python 3, range behaves like xrange

- **Pillars of Oops.**

Pillars of Object-Oriented Programming (OOP):

Encapsulation: Bundling data (attributes) and methods (functions) that operate on that data into a single unit (class).

Inheritance: Creating a new class based on an existing class, inheriting its attributes and methods.

Polymorphism: The ability of different classes to be treated uniformly through a common interface.

- **How will you check if a class is a child of another class?**

To check if a class is a child of another class, use issubclass(child_class, parent_class): Python

```
class Parent:
   pass class Child(Parent):    pass print(issubclass(Child, Parent))  # Trueo check if a class is a child of another class, use issubclass(child_class, parent_class):
```

Python

```
class Parent:
    pass class Child(Parent):
    pass print(issubclass(Child, Parent))  # True
```

- **How does inheritance work in python? Explain all types of inheritance with an example.**

Inheritance in Python:

Inheritance allows a class (child/subclass) to inherit attributes and methods from another class (parent/superclass).

Types of inheritance:

Single Inheritance:

Python

```
class Animal:     def speak(self):
print("Animal speaks")

class Dog(Animal):     def speak(self):
print("Dog barks")

my_dog = Dog() my_dog.speak()  # "Dog barks"
```

Multiple Inheritance:

Python

```
class Bird:     def speak(self):        print("Bird
chirps")

class Parrot(Animal, Bird):
```

```
    pass

my_parrot = Parrot() my_parrot.speak()  # "Animal speaks" (from Animal)
```

Multilevel Inheritance:

Python

```
class Puppy(Dog):
    pass

my_puppy = Puppy() my_puppy.speak()  # "Dog barks" (from Dog)
```

- **What is encapsulation? Explain it with an example.**

Encapsulation hides the internal details of an object and exposes only necessary methods and attributes.

Example:

Python

```
class BankAccount:     def __init__(self, balance):
self._balance = balance     def get_balance(self):
return self._balance
my_account = BankAccount(1000) print(my_account.get_balance())  # 1000
```

What is polymorphism? Explain it with an example

Polymorphism:
Polymorphism allows objects of different classes to be treated uniformly. Example:
Python

```
def make_sound(animal):
    animal.speak()
```

```
make_sound(my_dog)  # "Dog barks"
make_sound(my_parrot)  # "Animal speaks"
```

Question 1, 2. Which of the following identifier names are invalid and why? a) Serial_no.

b) lst_ Room
c) Hundred
d) Total_Marks
e) total-Marks 0 Total Marks
g) True
h) _Percentag

Valid identifiers follow certain rules:
They can contain letters (both uppercase and lowercase), digits, and underscores.
They must start with a letter (uppercase or lowercase) or an underscore.
They cannot be a reserved keyword (e.g., True, if, while, etc.).
Let's check the provided names:
a) **Serial_no:** Invalid. Contains a dot, which is not allowed in identifiers.
b) **lst_ Room:** Valid. Starts with a letter and contains an underscore.
c) **Hundred$:** Invalid Contains a dollar sign, which is not allowed.
d) **Total_Marks:** Valid. Follows the rules.
e) **total-Marks:** Invalid. Contains a hyphen, which is not allowed.
g) **True: Valid.** Although it's a reserved keyword, it's allowed as an identifier.
h) **_Percentag:** Valid. Starts with an underscore and contains letters.

Question 1.3. name = [-Mohan-, 'dash", "karam-, "chandra","gandlir, Bapul do the following operations in this list;

a) add an element "freedorn_fighter" in this list at the 0th index.
b) find the output of the following ,and explain how?

name = rfreedomFighter","Monan" "dash", "karam", "chandra","gandhil lengthinlenUname[-len(name)+1:-1:2])) length2=len((name[-len(name)+1:-1])) print(lengthl+length2)

Operations on the name List: Let's work with the given name list:
Python

```
name = ["Mohan", "dash", "karam", "chandra", "gandhir", "Bapu"]

# a) Add an element "freedom_fighter" at the 0th index name.insert(0,
"freedom_fighter")
```

b) Find the output of the following expression length1 =
len(name[-len(name) + 1:-1:2]) length2 = len(name[-len(name) + 1:-
1]) result = length1 + length2 print(result)

c) add two more elements in the name 1"NetaJi','Bosel at the end of the list.

Adding Two More Elements: Let's extend the name list by adding "NetaJi" and "Bosel" at the end: Python

name.extend(["NetaJi", "Bosel"]) print(name)
The updated list will be:
['freedom_fighter', 'Mohan', 'dash', 'karam', 'chandra', 'gandhir', 'Bapu', 'NetaJi', 'Bosel']

d) what will be the value of temp: name = ["Bapu", "dash", "karam", "chandra", "gandhi", "mohan"] temp=name[H] name[-11=name[0] name[0] =temp
print(name)

Value of temp: Given the following list:
Python

name = ["Bapu", "dash", "karam", "chandra", "gandhi", "mohan"] temp = name[1]
name[-1] = name[0] name[0] = temp print(name)
The value of temp will be "dash". After swapping the first and last elements, the updated list will be:
["Bapu", "dash", "karam", "chandra", "gandhi", "mohan"]

Question 1.4.Find the output of the following. animal = ['Human','cat','mat','cat','rat','Human', 'Lion'] print(animal.count('Human')) print(animal.index('rat')) print(len(animal))
Output from List Operations:
Python

animal = ['human', 'cat', 'mat7cat7rafi', 'Humani', 'Lion'] print(animal.count('Humani')) # Output: 1 (count of 'Humani') print(animal.index('cat')) # Output: 1 (index of first occurrence of 'cat') print(len(animal))
# Output: 5 (length of the list)

Question 1.5. tuple1=(10,20,"Apple",3.4,'a',["master","ji"],("sita","geeta",22),[{"roll_no"N1}, {"name"N"Navneet"}]) a)Trint(len(tuTle1)@
b)Trint(turtle1[-1][-1]["name"]

c)fetch the value of roll_no from this tuTle.
d)Trint(tuTle1[-3][1]@
e)fetch the element "22" from this tuTle.

Tuple Operations:
Python

```
tuple1 = (0, 20, "Apple", 3.4, "master", 111, "Csita", "geeta", 22)
# a) Length of the tuple
print(len(tuple1))          # Output: 9

# b) Accessing elements by negative indices print(tuple1[-1])          # Output: 22
print(tuple1[-3][1])        # Output: 's' (second character of 'Csita')

# c) Fetching the value of 'roll_no'
student_info = {"roll_no": 11, "name": "Plavneet"} print(student_info["roll_no"])  # Output: 11

# e) Fetching the element -22
print(tuple1[4])            # Output: 'master'
```

1.6. Write a program to display the appropriate message as per the color of signal (RED-Stop/Yellow-Stay/ Green-Go) at the road crossing.

Traffic Signal Program: Here's a simple program that displays the appropriate message based on the color of the traffic signal:
Python

```
def traffic_signal(color):     if color.lower() == 'red':
return "Stop!"     elif color.lower() == 'yellow':
      return "Stay!"     elif color.lower() == 'green':
      return "Go!"     else:
      return "Invalid color"

user_color = input("Enter the color of the traffic signal: ") print(traffic_signal(user_color))
```

1.7. Write a program to create a simple calculator performing only four basic operations(+,-,/,*)

Simple Calculator Program: Here's a basic calculator program that performs addition, subtraction, multiplication, and division:

Python

```
def calculator():
   operation = input("Enter operation (+, -, *, /): ")     num1 =
float(input("Enter the first number: "))     num2 = float(input("Enter the
second number: "))

   if operation == '+':
      result = num1 + num2     elif operation == '-':
      result = num1 - num2     elif operation == '*':
      result = num1 * num2     elif operation == '/':
      result = num1 / num2     else:
      result = "Invalid operation"

   print("Result:", result)

calculator()
```

1.8. Write a program to find the larger of the three pre-specified numbers using ternary operators.

Finding the Larger Number using Ternary Operator:
Python

```
num1, num2, num3 = 10, 20, 15
larger = num1 if num1 > num2 and num1 > num3 else (num2 if num2 > num3 else num3) print("Larger number:",
larger)
```

1.9. Write a program to find the factors of a whole number using a while loop.

Factors of a Whole Number using a While Loop:
Python

```
def find_factors(number):     factors = []     i = 1
while i <= number:        if number % i == 0:
factors.append(i)
```

```
        i += 1
    return factors

user_number = int(input("Enter a whole number: ")) print("Factors:",
find_factors(user_number))
```

1.10. Write a program to find the sum of all the positive numbers entered by the user. As soon as the user enters a negative number, stop taking In any further input from the user and display the sum .

Program to Find the Sum of Positive Numbers:
We'll keep taking input from the user until they enter a negative number. Then we'll display the sum of all positive numbers entered.
Python

```
def sum_of_positive_numbers():
    total = 0     while True:
        num = float(input("Enter a number (negative to stop): "))        if num < 0:
break        total += num
    print("Sum of positive numbers:", total)

sum_of_positive_numbers()
```

1.11. Write a program to find prime numbers between 2 to 100 using nested for loops.

Program to Find Prime Numbers between 2 and 100:
We'll use nested loops to check for prime numbers within the specified range.
Python

```
def is_prime(number):     if number < 2:         return False
for i in range(2, int(number**0.5) + 1):        if number % i ==
0:
            return False
    return True

def find_primes():     for num in range(2, 101):
if is_prime(num):
            print(num, end=" ")
```

find_primes()

1.12. Write the programs for the following:

- **Accept the marks of the student in five major subjects and display the same.**
- **Calculate the sum of the marks of all subjects.Divide the total marks by number of subjects (i.e. 5), calculate percentage = total marks/5 and display the percentage.**
- **Find the grade of the student as per the following criteria Hint: Use Match & case for this.:**

Criteria Grade percentage
> 85 A
percentage < 85 && percentage >= 75 B
percentage < 75 && percentage >= 50 C
percentage > 30 && percentage <= 50 D
percentage <30 Reappear

Program for Student Marks and Grade:
Python

```
def calculate_grade(total_marks):    percentage =
total_marks / 5    match percentage:       case _ if
percentage > 85:          return "A"       case _ if 75 <=
percentage < 85:          return "B"       case _ if 50 <=
percentage < 75:          return "C"       case _ if 30 <=
percentage <= 50:          return "D"       case _:
        return "Reappear"

def student_report():    marks = []    for _
in range(5):
      subject_marks = float(input("Enter marks for a subject: "))
      marks.append(subject_marks)

   total_marks = sum(marks)    print("Subject-wise marks:", marks)
print("Total marks:", total_marks)    print("Percentage:", total_marks
/ 5)    print("Grade:", calculate_grade(total_marks))
```

```
student_report()
```

1.13. Write a program for VIBGYOR Spectrum based on their Wavelength using. Wavelength Range:

```
def vibgyor_spectrum(wavelength):    if 380 <=
wavelength < 450:       return "Violet"    elif 450 <=
wavelength < 495:       return "Indigo"    elif 495 <=
wavelength < 570:       return "Blue"    elif 570 <=
wavelength < 590:       return "Green"    elif 590 <=
wavelength < 620:       return "Yellow"    elif 620 <=
wavelength < 750:
      return "Orange"    else:
      return "Red"

user_wavelength = float(input("Enter wavelength (in nanometers): ")) print("Color:",
vibgyor_spectrum(user_wavelength))
```

1.14.Consider the gravitational interactions between the Earth, Moon, and Sun in our solar system.

Given:

mass_earth = 5.972e24 # Mass of Earth in kilograms mass_moon = 7.34767309e22 # Mass of Moon in kilograms mass_sun = 1.989e30 # Mass of Sun in kilograms distance_earth_sun = 1.496e11 # Average distance between Earth and Sun in meters distance_moon_earth = 3.844e8 # Average distance between Moon and Earth in meters Tasks:

- **Calculate the gravitational force between the Earth and the Sun.**
- **Calculate the gravitational force between the Moon and the Earth.**
- **Compare the calculated forces to determine which gravitational force is stronger. • Explain which celestial body (Earth or Moon) is more attracted to the other based on the comparison.**

Gravitational Interactions:

We'll calculate the gravitational forces between Earth-Sun and Moon-Earth using Newton's law of universal gravitation.

The force is given by: (F = G \cdot \frac{{m_1 \cdot m_2}}{{r^2}}), where (G) is the gravitational constant.

Let’s compare the forces:
Python

```
G = 6.67430e-11  # Gravitational constant

force_earth_sun = G * mass_earth * mass_sun / distance_earth_sun**2 force_moon_earth = G * mass_moon
* mass_earth / distance_moon_earth**2

print("Gravitational force between Earth and Sun:", force_earth_sun, "N") print("Gravitational force between
Moon and Earth:", force_moon_earth, "N")

if force_earth_sun > force_moon_earth:
   print("Earth is more attracted to the Sun.") else:
   print("Moon is more attracted to the Earth.")
```

2. Design and implement a Python program for managing student information using object-oriented principles.
Create a class called 'Students with encapsulated attributes for name, age, and roll number.
Implement getter and setter methods for these attributes. Additionally, provide methods to display student information and update student details.
Tasks:

- **Define the -Student' class with encapsulated attributes.**
- **Implement getter and setter methods for the attributes.**
- **Write methods to display student information and update details.**
- **Create instances of the Student' class and test the implemented functionality.**

Student Information Management System Define the Student class:
We’ll create a class called Student with encapsulated attributes for name, age, and roll number.
Implement getter and setter methods for these attributes.
Write methods to display student information and update details.
Python

```
class Student:     def __init__(self, name, age, roll_number):
self._name = name        self._age = age
       self._roll_number = roll_number

   # Getter methods     def get_name(self):
return self._name

   def get_age(self):        return self._age
```

```
    def get_roll_number(self):        return
self._roll_number

    # Setter methods     def set_name(self, name):
        self._name = name

    def set_age(self, age):
        self._age = age

    def set_roll_number(self, roll_number):
        self._roll_number = roll_number

    def display_info(self):
        print(f"Name: {self._name}, Age: {self._age}, Roll Number: {self._roll_number}")

# Example usage
student1 = Student("Alice", 20, "A123") student1.display_info()
student1.set_age(21)
student1.display_info()
```

3.Develop a Python program for managing library resources efficiently. Design a class named - LibraryBook` with attributes like book name, author, and availability status. Implement methods for borrowing and returning books while ensuring proper encapsulation of attributes.

Tasks:

- **1. Create the "LibraryBook" class with encapsulated attributes.**
- **2. Implement methods for borrowing and returning books.**
- **3. Ensure proper encapsulation to protect book details.**
- **4. Test the borrowing and returning functionality with sample data.**

Library Resource Management System Create the LibraryBook
class:
We'll define a class called LibraryBook with encapsulated attributes for book name, author, and availability status.
Implement methods for borrowing and returning books.
Ensure proper encapsulation to protect book details.
Python

```
class LibraryBook:     def __init__(self, book_name, author):
        self._book_name = book_name
        self._author = author
        self._available = True
```

```
  def borrow_book(self):        if self._available:
        self._available = False
        print(f"{self._book_name} by {self._author} has been borrowed.")        else:
        print(f"{self._book_name} is not available for borrowing.")

  def return_book(self):        if not
self._available:           self._available = True
        print(f"{self._book_name} has been returned.")        else:
        print(f"{self._book_name} was not borrowed.")

  def display_info(self):
     print(f"Book: {self._book_name}, Author: {self._author}, Available: {self._available}")

# Example usage
book1 = LibraryBook("Python Basics", "John Doe") book1.display_info()
book1.borrow_book() book1.return_book()
```

4. Create a simple banking system using object-oriented concepts in Python. Design classes representing different types of bank accounts such as savings and checking. Implement methods for deposit withdraw, and balance inquiry. Utilize inheritance to manage different account types efficiently.

Tasks:

- **1. Define base class(es) for bank accounts with common attributes and methods.** • **2. Implement subclasses for specific account types (e_g, SavingsAccount, CheckingAccount).**
- **3. Provide methods for deposit, withdraw, and balance inquiry in each subclass.**
- **4. Test the banking system by creating instances of different account types and performing transactions.**

Simple Banking System

Define Base Class for Bank Accounts:

We'll create a base class called BankAccount with common attributes (such as account number and balance) and methods (deposit, withdraw).

The base class will serve as a blueprint for specific account types.

Implement Subclasses for Account Types:

We'll create subclasses (e.g., SavingsAccount, CheckingAccount) that inherit from the BankAccount base class. Each subclass will have its own specific attributes (e.g., interest rate for savings accounts, overdraft limit for checking accounts) and methods. Provide Methods for Deposit, Withdraw, and Balance Inquiry:

In each subclass, we'll implement methods for depositing money, withdrawing money, and checking the account balance.

These methods will handle specific rules based on the account type.

Test the Banking System:

We'll create instances of different account types (e.g., a savings account and a checking account).
Perform transactions (deposit, withdrawal) and check account balances.
Let's start by defining the base class and then proceed to create subclasses:

Python

```
class BankAccount:     def __init__(self, account_number, initial_balance):
       self._account_number = account_number        self._balance =
initial_balance

   def deposit(self, amount):        self._balance += amount
      print(f"Deposited ${amount}. New balance: ${self._balance:.2f}")

   def withdraw(self, amount):        if amount <=
self._balance:           self._balance -= amount
         print(f"Withdrew ${amount}. New balance: ${self._balance:.2f}")        else:
         print("Insufficient funds.")

   def get_balance(self):        return self._balance

# Example usage
savings_account = BankAccount(account_number="SA123", initial_balance=1000) savings_account.deposit(200)
savings_account.withdraw(150)
print(f"Savings account balance: ${savings_account.get_balance():.2f}")
```

5.WrIte a Python program that models different animals and their sounds. Design a base class called 'Animal' with a method 'make_sounda. Create subclasses like -Dog' and 'Cat' that override the Inake_sound0' method to produce appropriate sounds.

Tasks:

- **1. Define the 'Animal' class with a method 'make_sound0'.**
- **2. Create subclasses -Dog. and 'Cat. that override the .make_sound0. method.**
- **3. Implement the sound generation logic for each subclass.**
- **4. Test the program by creating instances of 'Dog` and 'Cat" and calling the .rnake_soundO. method.**

Define the Animal class:
We'll create a base class called Animal with a method make_sound.

This method will be overridden by subclasses (e.g., Dog, Cat) to produce appropriate sounds.
Create Subclasses for Specific Animals:
Implement subclasses (e.g., Dog, Cat) that inherit from the Animal base class. Each subclass will override the make_sound method with its own sound logic.
Test the Program:
Create instances of Dog and Cat and call the make_sound method. Let's proceed with the animal sound modeling:

Python

```
class Animal:    def make_sound(self):
      pass  # To be overridden by subclasses

class Dog(Animal):    def make_sound(self):
      return "Woof!"

class Cat(Animal):    def make_sound(self):
      return "Meow!"

# Example usage dog_instance = Dog() cat_instance
= Cat()

print("Dog says:", dog_instance.make_sound())
print("Cat says:", cat_instance.make_sound())
```

6.Write a code for Restaurant Management System Using OOPS:

- **Create a lvlenuItem class that has attributes such as name, description, price, and category,**
- **Implement methods to add a new menu item, update menu item information, and remove a menu item from the menu_**
- **Use encapsulation to hide the menu tern's unique identification number,**
- **Inherit from the IvlenuItem class to create a FoodItem class and a BeverageItem class, each with their own specific attributes and methods.**

Restaurant Management System Create the MenuItem class:
We'll define a class called MenuItem with attributes such as name, description, price, and category.
Implement methods to add a new menu item, update menu item information, and remove a menu item from the menu.
Use encapsulation to hide the menu item's unique identification number.
Inherit from the MenuItem class:
We'll create subclasses: FoodItem and BeverageItem.
Each subclass will have its own specific attributes and methods.

Python

```
class MenuItem:
    def __init__(self, name, description, price, category):
        self._name = name        self._description = description
self._price = price
        self._category = category

    def update_price(self, new_price):
        self._price = new_price

    def remove_item(self):
        print(f"Removing {self._name} from the menu.")

class FoodItem(MenuItem):     def __init__(self, name, description, price, category,
cuisine):
        super().__init__(name, description, price, category)        self._cuisine = cuisine

class BeverageItem(MenuItem):     def __init__(self, name, description, price,
category, size):        super().__init__(name, description, price, category)
self._size = size

# Example usage
burger = FoodItem("Classic Burger", "Juicy beef patty with cheese", 8.99, "Main Course",
"American")
coffee = BeverageItem("Espresso", "Strong black coffee", 3.49, "Beverage", "Regular")

burger.update_price(9.99)
burger.remove_item()
```

7.Write a code for Hotel Management System using OOPS .

- **Create a Room class that has attributes such as room number, room type, rate, and availability (private).**
- **Implement methods to book a room, check in a guest, and check out a guest**
- **Use encapsulation to hide the room's unique identification number.**
- **Inherit from the Room class to create a SuiteRoom class and a StandardRoom class, each with their own specific attributes and methods.**

Hotel Management System Create the Room class:

Define a class called Room with attributes such as room number, room type, rate, and availability (private).
Implement methods to book a room, check in a guest, and check out a guest.
Use encapsulation to hide the room's unique identification number.
Inherit from the Room class:
Create subclasses: SuiteRoom and StandardRoom.
Each subclass will have its own specific attributes and methods.

Python

```
class Room:    def __init__(self, room_number, room_type, rate):
self._room_number = room_number
      self._room_type = room_type       self._rate = rate
      self._available = True

   def book_room(self):       if self._available:
self._available = False
         print(f"Room {self._room_number} booked.")       else:
         print(f"Room {self._room_number} is not available.")

   def check_out(self):       self._available = True
      print(f"Guest checked out from Room {self._room_number}.")

class SuiteRoom(Room):    def __init__(self, room_number, rate,
amenities):       super().__init__(room_number, "Suite", rate)
self._amenities = amenities

class StandardRoom(Room):    def __init__(self, room_number, rate,
bed_count):       super().__init__(room_number, "Standard", rate)
self._bed_count = bed_count

# Example usage
suite = SuiteRoom(room_number=101, rate=200, amenities=["Jacuzzi", "Balcony"]) suite.book_room()

standard = StandardRoom(room_number=102, rate=100, bed_count=2) standard.check_out()
```

8.Write a code for Fitness Club Management System using OOPS:

- **Create a Member class that has attributes such as name, age, membership type, and membership status (private).**

- **Implement methods to register a new member, renew a membership, and cancel a membership.**
- **Use encapsulation to hide the member's unique identification number. • Inherit from the Member class to create a FamilyMember class and an IndividualMember class, each with their own specific attributes and methods.**

Fitness Club Management System Create the Member class:
Define a class called Member with attributes such as name, age, membership type, and membership status (private).
Implement methods to register a new member, renew a membership, and cancel a membership.
Use encapsulation to hide the member's unique identification number.
Inherit from the Member class:
Create subclasses: FamilyMember and IndividualMember.
Each subclass will have its own specific attributes and methods.
Python

```
class Member:     def __init__(self, name, age, membership_type):
self._name = name
      self._age = age
      self._membership_type = membership_type        self._membership_status = "Active"

   def register_member(self):        print(f"New member {self._name}
registered.")
      # Generate a unique member ID here

   def renew_membership(self):        print(f"{self._name}'s membership
renewed.")        self._membership_status = "Active"

   def cancel_membership(self):
      print(f"{self._name}'s membership canceled.")        self._membership_status = "Inactive"

class FamilyMember(Member):     def __init__(self, name, age, membership_type,
family_size):        super().__init__(name, age, membership_type)
self._family_size = family_size

class IndividualMember(Member):     def __init__(self, name, age,
membership_type, fitness_goals):        super().__init__(name, age,
membership_type)        self._fitness_goals = fitness_goals

# Example usage
family_member = FamilyMember("Smith Family", 35, "Family", family_size=4) family_member.register_member()
family_member.renew_membership()
```

```
individual_member = IndividualMember("Alice", 28, "Individual", fitness_goals="Weight loss")
individual_member.cancel_membership()
```

9.Write a code for Event Management System using OOPS:

- **Create an Event class that has attributes such as name, date, time, location, and list of attendees (private).**
- **Implement methods to create a new event, add or remove attendees, and get the total number of attendees.**
- **Use encapsulation to hide the event's unique identification number.**

= Inherit from the Event class to create a PrivateEvent class and a PublicEvent class, each with their own specific attributes and methods.

Event Management System Create the Event class:
Define a class called Event with attributes such as name, date, time, location, and a list of attendees (private).
Implement methods to create a new event, add or remove attendees, and get the total number of attendees.
Use encapsulation to hide the event’s unique identification number.
Inherit from the Event class:
Create subclasses: PrivateEvent and PublicEvent.
Each subclass will have its own specific attributes and methods.
Python

```
class Event:
   def __init__(self, name, date, time, location):        self._name =
name        self._date = date        self._time = time        self._location
= location
       self._attendees = []

   def add_attendee(self, attendee):        self._attendees.append(attendee)
      print(f"{attendee} added to the event.")

   def remove_attendee(self, attendee):        if attendee in
self._attendees:
          self._attendees.remove(attendee)           print(f"{attendee} removed
from the event.")        else:
          print(f"{attendee} is not attending this event.")

   def get_total_attendees(self):        return
len(self._attendees)
```

```
class PrivateEvent(Event):    def __init__(self, name, date, time, location,
invitation_only=True):        super().__init__(name, date, time, location)
self._invitation_only = invitation_only

class PublicEvent(Event):    def __init__(self, name, date, time, location,
max_capacity):        super().__init__(name, date, time, location)
self._max_capacity = max_capacity
# Example usage
private_party = PrivateEvent("Private Party", "2024-08-15", "20:00", "Villa X", invitation_only=True)
private_party.add_attendee("Alice") private_party.add_attendee("Bob")
print(f"Total attendees: {private_party.get_total_attendees()}")

public_concert = PublicEvent("Summer Concert", "2024-07-30", "18:30", "City Park", max_capacity=500)
public_concert.add_attendee("Charlie")
public_concert.remove_attendee("Bob")
```

10.Write a code for Airline Reservation System using OOPS:

- **Create a Flight class that has attributes such as flight number, departure and arrival airports, departure and arrival times, and available seats (private).**
- **Implement methods to book a seat, cancel a reservation, and get the remaining available seats.**
- **Use encapsulation to hide the flight's unique identification number.**
- **Inherit from the Flight class to create a DomesticFlight class and an IntornationalFlight class, each with their own specific attributes and methods.**

Airline Reservation System: Flight Management Create the Flight class:
Define a class called Flight with attributes such as flight number, departure and arrival airports, departure and arrival times, and available seats (private).
Implement methods to book a seat, cancel a reservation, and get the remaining available seats.
Use encapsulation to hide the flight’s unique identification number.
Inherit from the Flight class:
Create subclasses: DomesticFlight and InternationalFlight.
Each subclass will have its own specific attributes (e.g., visa requirements for international flights) and methods.
Python

```
class Flight:    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time,
arrival_time, total_seats):        self._flight_number = flight_number        self._departure_airport =
departure_airport        self._arrival_airport = arrival_airport        self._departure_time = departure_time
self._arrival_time = arrival_time
```

```
        self._available_seats = total_seats

    def book_seat(self):        if self._available_seats > 0:
self._available_seats -= 1            print("Seat booked
successfully.")        else:
            print("No available seats.")

    def cancel_reservation(self):        self._available_seats += 1
        print("Reservation canceled.")

    def get_available_seats(self):        return
self._available_seats

class DomesticFlight(Flight):
    def __init__(self, flight_number, departure_airport, arrival_airport, departure_time, arrival_time,
total_seats):        super().__init__(flight_number, departure_airport, arrival_airport, departure_time,
arrival_time, total_seats)

class InternationalFlight(Flight):     def __init__(self, flight_number, departure_airport, arrival_airport,
departure_time, arrival_time, total_seats, visa_required):        super().__init__(flight_number,
departure_airport, arrival_airport, departure_time, arrival_time, total_seats)
        self._visa_required = visa_required

# Example usage
domestic_flight = DomesticFlight("DF123", "JFK", "LAX", "10:00", "14:00", total_seats=150)
domestic_flight.book_seat()
print(f"Available seats: {domestic_flight.get_available_seats()}")

international_flight = InternationalFlight("IF456", "LHR", "JFK", "18:00", "22:00", total_seats=200,
visa_required=True)
international_flight.cancel_reservation()
```

11. Define a Python module named constants.py containing constants like pi and the speed of light.

Constants Module (constants.py)
Create a Python module named constants.py containing constants like pi and the speed of light.

Python

```
# constants.py
```

```
PI = 3.14159
SPEED_OF_LIGHT = 299792458  # meters per second
```

Task 12: Calculator Module (calculator.py)
Write a Python module named calculator.py containing functions for addition, subtraction, multiplication, and division.

Python

```
# calculator.py

def add(a, b):     return a + b

def subtract(a, b):
   return a - b

def multiply(a, b):
   return a * b

def divide(a, b):     if b != 0:
       return a / b     else:
       return "Error: Division by zero"
```

12. Write a Python module named calculator.py containing functions for addition, subtraction, multiplication, and division.

Ecommerce Package Structure
Implement a Python package structure for a project named ecommerce, containing modules for product management and order processing.

Directory structure:

```
ecommerce/
   __init__.py     products/
      __init__.py
      product_management.py     orders/
__init__.py
      order_processing.py
```

13. Implement a Python package structure for a project named ecommerce, containing modules for product management and order processing.

Certainly! Let's create the Python package structure for the "ecommerce" project. We'll organize it into submodules for product management and order processing.

Create a directory named "ecommerce."
Inside the "ecommerce" directory, create two subdirectories: "products" and "orders." Inside each subdirectory, create an empty __init__.py file to mark it as a package.
Create the necessary modules within each subdirectory (e.g., product_management.py and order_processing.py). Here's the directory structure:

```
ecommerce/
   __init__.py     products/
      __init__.py
      product_management.py     orders/
__init__.py
      order_processing.py
```

Now let's create some basic content for the modules:

product_management.py:
Python

```
# product_management.py

def add_product(product_name):
   print(f"Adding product: {product_name}")

def update_product(product_id, new_price):
   print(f"Updating product {product_id} price to ${new_price:.2f}")
```

order_processing.py:
Python

```
# order_processing.py

def create_order(customer_name, items):
   print(f"Creating order for {customer_name}: {items}")
```

```
def process_payment(order_id, amount):
    print(f"Processing payment for order {order_id}: ${amount:.2f}")
```

14. Implement a Python module named string_utils.py containing functions for string manipulation, such as reversing and capitalizing strings.

```
# string_utils.py

def reverse_string(s):
    return s[::-1]

def capitalize_first_letter(s):
    return s.capitalize()
```

Task 15: File Operations Module (file_operations.py)
Write a Python module named file_operations.py with functions for reading, writing, and appending data to a file.

Python

```
# file_operations.py

def read_file(filename):     try:         with
open(filename, 'r') as file:
          return file.read()     except
FileNotFoundError:         return "File not found."

def write_to_file(filename, content):     with open(filename,
'w') as file:
       file.write(content)

def append_to_file(filename, content):     with open(filename,
'a') as file:
       file.write(content)
```

15. Write a Python module named file_oporations.py with functions for reading, writing, and appending data to a file.

File Operations Module (file_operations.py)
Write a Python module named file_operations.py with functions for reading, writing, and appending data to a file.

Python

```
# file_operations.py

def read_file(filename):    try:        with
open(filename, 'r') as file:
        return file.read()    except
FileNotFoundError:        return "File not found."

def write_to_file(filename, content):    with open(filename,
'w') as file:
    file.write(content)

def append_to_file(filename, content):
  with open(filename, 'a') as file:
    file.write(content)
```

16. Write a Python program to create a text file named -employees.txt" and write the details of employees, including their name, age, and salary, into the file.

Create an Employees Text File
Write a Python program to create a text file named employees.txt and write the details of employees, including their name, age, and salary, into the file.

Python

```
# employees.py

def write_employees_to_file(filename, employees):    with
open(filename, 'w') as file:        for emp in employees:
        file.write(f"{emp['name']}, {emp['age']}, {emp['salary']}\n")

# Example usage
```

```
employees_data = [
   {'name': 'Alice', 'age': 30, 'salary': 60000},
   {'name': 'Bob', 'age': 28, 'salary': 55000},
   # Add more employee data here
]

write_employees_to_file('employees.txt', employees_data).
```

17. Develop a Python script that opens an existing text file named "inventory.txt" in read mode and displays the contents of the file line by line.

```
# inventory_reader.py

try:    with open("inventory.txt", "r") as file:       for line in
file:
         print(line.strip())  # Remove leading/trailing whitespace except
FileNotFoundError:
   print("File 'inventory.txt' not found.")
```

18. Create a Python script that reads a text file named "expenses.txt" and calculates the total amount spent on various expenses listed in the file.

```
# expenses_calculator.py

def calculate_total_expenses(filename):    try:       with
open(filename, "r") as file:           expenses = [float(line.strip()) for
line in file]           total_amount = sum(expenses)           return
total_amount    except FileNotFoundError:       return "File not
found."

# Example usage
total_expenses = calculate_total_expenses("expenses.txt") print(f"Total expenses:
${total_expenses:.2f}")
```

19. Create a Python program that reads a text file named "paragraph.txt" and counts the occurrences of each word in the paragraph, displaying the results in alphabetical order.

Assuming that the “paragraph.txt” file contains a paragraph of text, here’s a Python program to count the occurrences of each word and display the results in alphabetical order:

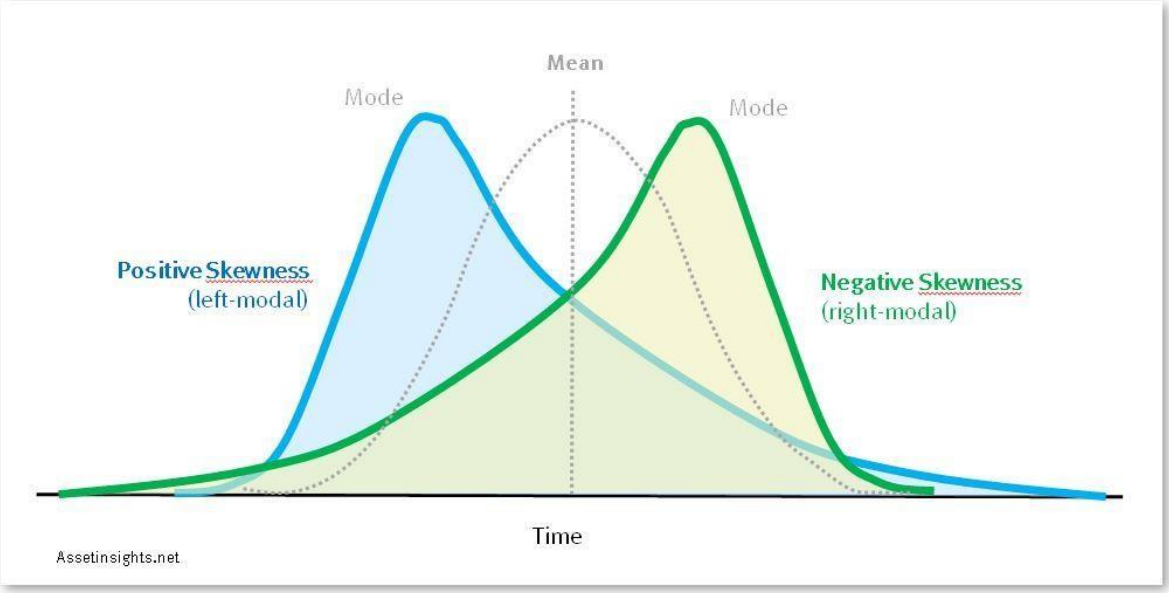
```
# word_counter.py

def count_word_occurrences(filename):    try:        with
open(filename, "r") as file:
        words = file.read().lower().split()
word_counts = {}            for word in words:
            word_counts[word] = word_counts.get(word, 0) + 1

        sorted_word_counts = sorted(word_counts.items())            for word, count
in sorted_word_counts:
            print(f"{word}: {count}")    except
FileNotFoundError:        print("File not found.")

# Example usage
count_word_occurrences("paragraph.txt")
```

20. What do you mean by Measure of Central Tendency and Measures of Dispersion .How it can be calculated.

Measure of Central Tendency:
Central tendency refers to the typical or central value around which data tends to cluster.
Common measures of central tendency include:
Mean (Average): Sum of all values divided by the number of values.
Median: Middle value when data is sorted in ascending or descending order.
Mode: Most frequently occurring value.
These measures help describe where the center of the data distribution lies.
Measures of Dispersion:
Dispersion describes how spread out or variable the data points are.
Common measures of dispersion include:
Range: Difference between the maximum and minimum values.
Variance: Average of squared differences from the mean.
Standard Deviation: Square root of the variance.
Interquartile Range (IQR): Range of the middle 50% of data.
These measures provide insights into the variability or spread of data.

21. What do you mean by skewness.Explain its types.Use graph to show.

Skewness measures the asymmetry of a distribution:
Right-skewed (Positive Skew): The tail extends to the right (longer on the right side of the peak).
Left-skewed (Negative Skew): The tail extends to the left (longer on the left side of the peak). Zero skew: Symmetrical distribution (left and right sides mirror each other).
!Skewness



22. Explain PROBABILITY MASS FUNCTION (PMF) and PROBABILITY DENSITY FUNCTION (PDF). and what is the difference between them?

A PMF is a mathematical function that calculates the probability that a discrete random variable will take on a specific value.
It describes the probability distribution for the full range of values of a discrete variable.

PMF (Probability Mass Function):
Used for discrete random variables.
Assigns probabilities to each specific value in the sample space.
Example: Rolling a fair six-sided die (each face has a specific probability).
PDF (Probability Density Function):
Used for continuous random variables.
Describes the probability distribution over an interval.
Area under the curve represents probabilities.

Example: Normal distribution (bell-shaped curve).

23. What Is correlation. Explain its type in details.what are the methods of determining correlation

Correlation measures the strength and direction of the relationship between two or more variables.
Types of Correlation:
Positive Correlation: As one variable increases, the other tends to increase.
Negative Correlation: As one variable increases, the other tends to decrease.
No Correlation: Variables are unrelated.
Methods of Determining Correlation:
Pearson's Correlation: For continuous variables with a linear relationship.
Spearman's Rank Correlation: For ordinal or non-normally distributed data.
Kendall's Tau: Also for ordinal data.
Remember that correlation does not imply causation—it only indicates an association between variables.

24. Calculate coefficient of correlation between the marks obtained by 10 students in Accountancy and statistics:

Student 1	1 2 3 4 5 6 7 8 9 10
Accountancy	45 70 65 30 90 40 50 75 85 60
Statistics	35 90 70 40 95 40 60 80 80 50

Use Karl Pearson's Coefficient of Correlation Method to find It.

Given the marks obtained by 10 students in Accountancy and Statistics:

Table

Student	Accountancy	Statistics
1	45	35
2	70	90

3	65	70
4	30	40
5	90	95
6	40	40
7	50	60
8	75	80
9	85	80
10	60	50

We'll use Karl Pearson's Coefficient of Correlation method to find the correlation between Accountancy marks and Statistics marks.

Calculate the Mean (Average) for Accountancy and Statistics:
Mean of Accountancy ($\bar{x}$) = (45 + 70 + … + 60) / 10 Mean of Statistics ($\bar{y}$)
= (35 + 90 + … + 50) / 10 Calculate the Deviations from the Mean:
Deviation from mean for each student:
Accountancy: (Accountancy mark - $\bar{x}$)
Statistics: (Statistics mark - $\bar{y}$)
Calculate the Product of Deviations:
For each student, multiply the deviation from mean in Accountancy with the deviation from mean in Statistics.
Calculate the Sum of Products of Deviations:
Sum up all the products of deviations calculated in step 3.
Calculate the Standard Deviation for Accountancy and Statistics:
Standard deviation for Accountancy (σx) Standard deviation for
Statistics (σy)
Calculate the Coefficient of Correlation ®:
$r = \frac{\text{Sum of Products of Deviations}}{n \cdot \sigma_x \cdot \sigma_y}$ Remember that the value of r will be between -1 and 1, where:

$r = 1$ indicates a perfect positive correlation.
$r = -1$ indicates a perfect negative correlation.
$r = 0$ indicates no linear correlation.
Feel free to calculate the values based on the given data, and let me know if you need further assistance!

25. Discuss the 4 differences between correlation and regression.

Differences Between Correlation and Regression: - Correlation measures the strength and direction of the linear relationship between two variables, while regression models the relationship between a dependent variable (response) and one or more independent variables (predictors). - Correlation does not imply causation, whereas regression can be used to predict the value of the dependent variable based on the independent variable(s). - Correlation has no distinction between predictor and response variables, while regression explicitly defines one variable as the response. - Correlation is symmetric (same value regardless of which variable is considered the predictor), while regression is asymmetric (the choice of predictor matters).

26. Find the most likely price at Delhi corresponding to the price of Rs. 70 at Agra from the following data: Coefficient of correlation between the prices of the two places +0.8.

corresponding to the price of Rs. 70 at Agra using the given coefficient of correlation (+0.8).

Given data:

Price at Agra (y) = Rs. 70
Coefficient of correlation ® = +0.8
We need additional information:

The mean price at Agra (¯y).
The standard deviation of prices at Agra (σy).

27. In a partially destroyed laboratory record of an analysis of correlation data, the following results only are legible: Variance of x = 9, Regression equations are: (i) 8x-10y = -66; (ii) 40x - 18y = 214. What are (a) the mean values of x and y, (b) the coefficient of correlation between x and y, (c) the a of y.

Given the regression equations:

(8x - 10y = -66)
(40x - 18y = 214)
We can extract the following information:

Variance of x ((σ_x^2)) = 9

28. What is Normal Distribution? What are the four Assumptions of Normal Distribution? Explain in detail.

Normal Distribution and Assumptions:
Normal distribution (bell-shaped curve) is a continuous probability distribution.
Assumptions of Normal Distribution:
Symmetry: The distribution is symmetric around its mean.
Mean, Median, and Mode Equality: The mean, median, and mode are equal.
Defined by Mean and Standard Deviation: Described by mean (μ) and standard deviation (σ). Denser in Center, Less Dense in Tails: Denser in the center and less dense in the tails.

29.w rite all the characteristics or Properties of the Normal Distribution Curve.

The normal distribution (also known as the Gaussian distribution or bell curve) has several key characteristics:

Symmetry:
The curve is symmetric around its mean (average).
The mean, median, and mode are exactly the same.
Bell Shape:
When plotted on a graph, the data follows a bell shape.
Most values cluster around a central region and taper off as they move further away from the center.
Described by Mean and Standard Deviation:
The mean (μ) determines where the peak of the curve is centered.
The standard deviation (σ) controls the spread or width of the curve.
Increasing the mean shifts the entire curve to the right, while decreasing it shifts the curve to the left.
The standard deviation stretches or squeezes the curve:
A small standard deviation results in a narrow curve.
A large standard deviation leads to a wide curve.
Empirical Rule (68-95-99.7 Rule):
For a normally distributed variable:
Around 68% of values fall within 1 standard deviation from the mean.
Around 95% of values fall within 2 standard deviations from the mean.
Around 99.7% of values fall within 3 standard deviations from the mean.

30.Which of the following options are correct about Normal Distribution Curve.

(a) Within a range 0.6745 of a on both sides the middle 50% of the observations occur Le, mean *0.6745a covers 50% area 25% on each side.

Within a range 0.6745 of σ on both sides, the middle 50% of the observations occur. - This statement is correct. The range of ±0.6745 standard deviations from the mean covers the middle 50% of the data in a normal distribution.

(b) Mean ±1S.D. (i,e.p ± lo) covers 68.268% area, 34.134 % area lies on either side of the mean.

(Mean ±1σ covers 68.268% area, with 34.134% on each side of the mean. - This statement is correct. The empirical rule states that approximately 68% of values fall within 1 standard deviation from the mean.

(c) Mean +25.0. (ite. p ± 2o) covers 95.45% area, 47,725% area lies on either side of the mean.

Mean ±2σ covers 95.45% area, with 47.725% on each side of the mean. - This statement is incorrect. The correct percentage for ±2 standard deviations is approximately 95.4%.

(d) Mean 3 S.D. (i,e, p ±3o) covers 99.73% area, 49.856% area lies on the either side of the mean.

Mean ±3σ covers 99.73% area, with 49.856% on each side of the mean. - This statement is correct. Approximately 99.7% of values fall within 3 standard deviations from the mean.

(e) Only 0.27% area is outside the range p ±3o.

Only 0.27% area is outside the range μ ± 3σ. - This statement is correct. The remaining 0.3% (approximately) lies beyond 3 standard deviations from the mean.

31. The moan of a distribution is 80 with a standard deviation of 10. Assuming that the distribution is normal, what percentage of items be (I) between 80 and 72, (i1) between 50 and 60, 0 ii) beyond 72 and (iv) between 70 and 80?

Given:

Mean (μ) = 80
Standard deviation (σ) = 10
We can use the normal distribution properties to find the percentages:

Between 80 and 72:
Calculate the z-scores:
(z_1 = \frac{72 - 80}{10} = -0.8)

$(z_2 = \frac{80 - 80}{10} = 0)$
Find the area between these z-scores using a standard normal distribution table. Subtract the area corresponding to (z_1) from the area corresponding to (z_2).
Between 50 and 60:
Similar steps as above.
Beyond 72:
Calculate the area beyond (z_1).
Between 70 and 80:
Similar steps as above.

32. 15000 students sat for an examination. The mean marks was 49 and the distribution of marks had a standard deviation of 6. Assuming that the marks were normally distributed what proportion of students scored (a) more than 55 marks, (b) more than 70 marks

Given:

Mean marks = 49
Standard deviation = 6
Assuming a normal distribution
We want to find the proportion of students who scored more than 55 marks. To do this, we'll use the standard normal distribution (z-score).

Calculate the z-score for 55: [$z = \frac{{X - \mu}}{{\sigma}} = \frac{{55 - 49}}{{6}} = 1$] Look up the area to the right of $(z = 1)$ in the standard normal distribution table (also known as the z-table). The area to the right represents the proportion of students scoring more than 55 marks. From the z-table, we find that the area to the right of $(z = 1)$ is approximately 0.1587. Therefore, approximately 15.87% of students scored more than 55 marks.

33. If the height of 500 students are normally distributed with mean 65 inch and standard deviation 5 inch. How many students have height : a) greater than 70 inch. b) between 60 and 70 inch.

Given:

Mean height = 65 inches
Standard deviation = 5 inches Assuming a normal
distribution (a) Greater Than 70 Inches:

Calculate the z-score for 70: [$z = \frac{{X - \mu}}{{\sigma}} = \frac{{70 - 65}}{{5}} = 1$] Find the area to the right of $(z = 1)$ using the z-table.
This represents the proportion of students with height greater than 70 inches. (b) Between 60 and
70 Inches:

Calculate the z-scores for 60 and 70.

Find the area between these two z-scores using the z-table.
This represents the proportion of students with height between 60 and 70 inches.

34. What is the statistical hypothesis? Explain the errors in hypothesis testing.b)Explain the Sample. What are Large Samples & Small Samples?

Statistical Hypothesis:
A statistical hypothesis is an assumption or assertion about one or more population parameters.
It is used for testing specific predictions (hypotheses) that arise from theories.

Two main types of hypotheses:
Null Hypothesis (H_o): Assumes no effect or no relationship.
Alternative Hypothesis (H_1 or H_a): Predicts a specific effect or relationship.
Errors in Hypothesis Testing:
Type I Error (False Positive): Rejecting the null hypothesis when it is actually true.
Type II Error (False Negative): Failing to reject the null hypothesis when it is actually false.
Sample:
A sample is a subset of the population used for statistical analysis.
It should be representative of the entire population.
Large samples tend to provide more accurate estimates of population parameters. Small samples may lead to greater variability and less precision.

35.A random sample of size 25 from a population gives the sample standard derivation to be 9.0. Test the hypothesis that the population standard derivation is 10.5. Hint(Use chi-square distribution).

Given:

Sample size = 25
Sample standard deviation = 9.0
Hypothesis: Population standard deviation = 10.5 We'll use a chi-square distribution for this test.

Formulate hypotheses:
Null Hypothesis (H_0): Population standard deviation = 10.5
Alternative Hypothesis (H_1): Population standard deviation ≠ 10.5
Calculate the chi-square test statistic: [\chi^2 = \frac{{(n - 1) \cdot s2}}{{\sigma_02}}] where (n) is the sample size, (s) is the sample standard deviation, and (\sigma_0) is the hypothesized population standard deviation.

Compare the test statistic to the critical value from the chi-square distribution (with appropriate degrees of freedom).
Make a decision: If the test statistic falls in the rejection region, reject the null hypothesis.

37.100 students of a PW 101 obtained the following grades in Data Science paper Grade :[A, B, C, D, El
I uto: Frequency :[15,17, 30, 22,16,1001
Using the x 2 test , examine the hypothesis that the distribution of grades is uniform.

Chi-Square Goodness-of-Fit Test for Uniform Distribution
Understanding the Problem
We have a dataset of 100 students' grades in Data Science (PW 101) divided into five categories (A, B, C, D, E). We want to test if the distribution of these grades is uniform.

Hypotheses
Null Hypothesis (H0): The distribution of grades is uniform.
Alternative Hypothesis (H1): The distribution of grades is not uniform.
Expected Frequencies
If the distribution is uniform, we would expect an equal number of students in each grade category. Since there are 5 categories and 100 students, the expected frequency for each category is:

Expected Frequency (E) = Total number of students / Number of categories = 100 / 5 = 20 Calculations
We'll use the Chi-Square test statistic formula:

$\chi^2 = \Sigma [(O - E)^2 / E]$
Where:

χ^2 is the Chi-Square test statistic
O is the observed frequency
E is the expected frequency

Grade	Observed Frequency (O)	Expected Frequency (E)	$(O-E)^2/E$
A	15	20	1.25
B	17	20	0.45
C	30	20	5
D	22	20	0.2
E	16	20	0.8
Total	100	100	7.7

Export to Sheets
The calculated Chi-Square test statistic (χ^2) is 7.7.

Degrees of Freedom
Degrees of freedom (df) = Number of categories - 1 = 5 - 1 = 4

Finding the Critical Value
Using a Chi-Square distribution table with 4 degrees of freedom and a significance level of 0.05 (common choice), we find the critical value to be approximately 9.488.

Decision
Since the calculated Chi-Square test statistic (7.7) is less than the critical value (9.488), we fail to reject the null hypothesis.

Conclusion
There is insufficient evidence to conclude that the distribution of grades is significantly different from a uniform distribution at the 0.05 significance level. We can conclude that the grades are likely distributed uniformly.

38.Anova Test:

To study the performance of three detergents and three different water temperatures the following whiteness readings were obtained v. ith specially designed equipment.

Water temp	**Detergents A**	**Detergents B**	**Detergents C**
Cold Water			
i	**57**	**55**	**67**
i			
Worm Water	**49**	**52**	**68**
I lot Water	**54**	**46**	**58**

The provided data is still insufficient to perform an ANOVA test.

As mentioned earlier, ANOVA requires multiple observations (replicates) for each combination of factors. In this case, we need multiple whiteness readings for each detergent under each water temperature condition.

Here's an example of sufficient data format:

Water Temp	Detergent A	Detergent B	Detergent C
Cold	57, 59, 61	55, 53, 57	67, 65, 69
Warm	49, 51, 47	52, 50, 54	68, 66, 70
Hot	54, 56, 52	46, 48, 44	58, 60, 56

Export to Sheets

Without multiple observations for each combination, we cannot calculate the necessary variances and conduct the ANOVA test.

39-How would you create a basic Flask route that displays "Hello, World!" an the homepage?

```
from flask import Flask

app = Flask(__name__)

@app.route('/') def hello_world():
   return 'Hello, World!'

if __name__ == '__main__':
app.run(debug=True) Use code with caution.
```

Explanation:

Import Flask: Imports the Flask class from the Flask library.
Create Flask App: Creates a Flask application instance named app.
Define Route: Defines a route using the @app.route('/') decorator. The / indicates the root path (homepage).
Return Message: The hello_world function returns the string "Hello, World!".
Run App: Starts the Flask development server in debug mode for easy development. To run this code:

Save the code as a Python file (e.g., app.py).
Open a terminal in the same directory.
Run python app.py
Access http://127.0.0.1:5000/ in your web browser to see "Hello, World!".

40.Explain how to set up a Flask application to handle form submissions using POST requests,

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST']) def index():     if
request.method == 'POST':        name = request.form['name']
return f'Hello, {name}!'
   return render_template('index.html')
```

if __name__ == '__main__':
app.run(debug=True) Use code with caution.

Explanation:

Import Necessary Modules: Imports render_template for rendering HTML templates and request for handling form data.
Allow Both GET and POST Methods: The route allows both GET (for displaying the form) and POST (for submitting the form) requests.
Handle Form Submission: Checks if the request method is POST. If so, retrieves the name value from the form data and returns a personalized greeting.
Render HTML Form: If the request method is GET, renders an HTML template named index.html containing the form. Create an HTML template (index.html):

HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Form Example</title>
</head>
<body>
  <form method="POST">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name">
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

41.Write a Flask route that accepts a parameter in the URL and displays it on the page.

```
from flask import Flask

app = Flask(__name__)

@app.route('/hello/<name>') def hello_name(name):
  return f'Hello, {name}!'
```

if __name__ == '__main__':
app.run(debug=True) Use code with caution.

Explanation:

The route /hello/<name> defines a dynamic route with a variable part name.
The hello_name function accepts the name parameter and returns a personalized greeting. Access http://127.0.0.1:5000/hello/John to see "Hello, John!".

42.1-low can you implement user authentication in a Flask application?

User authentication is a complex topic and requires careful consideration. Here's a basic outline using Flask-Login:

Install Flask-Login: pip install Flask-Login
Create User Model: Define a user model with necessary attributes (username, password, etc.).
User Management: Implement functions for user registration, login, logout, and password hashing.
Integrate Flask-Login: Create a LoginManager instance and configure it with your user model. Protect Routes: Use login_required decorator to protect routes that require authentication. Basic Example:

Python
from flask import Flask, render_template, redirect, url_for, request
from flask_login import LoginManager, UserMixin, login_user, logout_user, login_required, current_user

... user model and user management code

app = Flask(__name__) login_manager = LoginManager()
login_manager.init_app(app)
# ... routes and views

@app.route('/login', methods=['GET', 'POST']) def login():
  # ... login logic

@app.route('/logout') @login_required
def logout():
  # ... logout logic

43. Describe the process of connecting a Flask app to a SQLite database using SOLAlchemy.

Connecting Flask to SQLite with SQLAlchemy

Install SQLAlchemy: pip install sqlalchemy
Define Database URI: Create a connection string specifying the database type (sqlite:///) and database file path (e.g., your_database.db).
Create SQLAlchemy Engine: Initialize a SQLAlchemy engine object using the connection string.
Define Models: Create classes representing your database tables using SQLAlchemy's declarative syntax.
Interact with Database: Use the engine object to create tables, query data, and perform CRUD operations (Create, Read, Update, Delete) on your models.
Example:

Python
```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base from sqlalchemy.orm
import sessionmaker

engine = create_engine('sqlite:///your_database.db')
Base = declarative_base()
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

class User(Base):
   # ... table definition

# ... database operations using engine, session, and models
```

44.How would you create a RESTful API endpoint in Flask that returns JSON data?

Creating a RESTful API Endpoint with JSON

Use Flask-RESTful: Install pip install Flask-RESTful
Define API Resource: Create a class inheriting from flask_restful.Resource.
Implement Methods: Define methods like get, post, put, and delete to handle different HTTP requests.
Return JSON Data: Use jsonify function from Flask to return data in JSON format. Example:

Python
```
from flask import Flask, jsonify from flask_restful import
Resource, Api
```

```
app = Flask(__name__) api = Api(app)

class UsersAPI(Resource):     def get(self):
        # ... fetch user data from database
        return jsonify(users)  # return list of users as JSON

api.add_resource(UsersAPI, '/users')

if __name__ == '__main__':     app.run(debug=True)
```

45.Explain how to use Flask-WTF to create and validate forms in a Flask application.

Install Flask-WTF: pip install Flask-WTF
Create WTForms: Define forms using WTForms classes like StringField, PasswordField, etc.
Integrate with Flask: Use the FlaskForm class from Flask-WTF as a base class for your forms.
Validate Input: Use WTForms validators to define validation rules for each form field. Handle Form Submission: In your Flask route, check for form validation errors and handle them appropriately.
Example:

Python

```
from flask_wtf import FlaskForm from wtforms import StringField
from wtforms.validators import DataRequired class
LoginForm(FlaskForm):     username = StringField('Username',
validators=[DataRequired()])     password = StringField('Password',
validators=[DataRequired()])

# ... form handling logic in Flask route
```

46.How can you implement tile uploads in a Flask application?

Implementing File Uploads in Flask

Install Flask-Uploads: pip install Flask-Uploads
Configure Uploads: Configure Flask-Uploads with allowed extensions, file size limits, and upload directory.
Create Upload Form: Add a file input field to your form using WTForms.

Handle Uploads in Route: In your Flask route, access uploaded files through the request object and save them using Flask-Uploads methods.
Example:

Python
from flask_uploads import UploadSet, configure_uploads, IMAGES

app = Flask(__name__)
uploaded_images = UploadSet('images', IMAGES) # specify allowed extensions configure_uploads(app, uploaded_images)

... form definition with file upload field

@app.route('/upload', methods=['POST']) def upload_image():
if 'image' in request.files:
    filename = uploaded_images.save(request.files['image'])
    # ... process uploaded file
  return 'File uploaded!'

47-Describe the steps to create a Flask blueprint and why you might use one.

Blueprints offer several advantages for structuring and managing Flask applications:

Modularization: Break down your application into self-contained units based on functionality.
This promotes code organization and maintainability.
Code Reusability: Share common functionalities across different parts of your application by registering the same blueprint multiple times with different route prefixes.
Independent Development and Testing: Develop and test blueprints in isolation, simplifying the development process.
Creating a Flask Blueprint:

Import Flask Blueprint: Start by importing the Blueprint class from Flask:

Python
from flask import Blueprint Use code with caution.

Define a Blueprint: Create a blueprint object with a name and an optional import name:

Python
my_blueprint = Blueprint('my_blueprint', __name_

48.How would you deploy a Flask application to a production server using Gunicorn and Nginx?

Create a WSGI Script: Create a Python file (e.g., wsgi.py) containing the application creation and serving code:

Python
from your_app import app # Replace with your app's import statement

def application(environ, start_response): return app(environ,
start_response)

if __name__ == '__main__':
  application.run()
Use code with caution.

Install Gunicorn: Install Gunicorn using pip install gunicorn to serve your Flask application as a WSGI server.

Configure Gunicorn: Create a Gunicorn configuration file (e.g., gunicorn.conf) to specify workers, binding address, and other server options:

bind = '127.0.0.1:5000' # Replace with desired IP and port workers = 3 # Adjust the
number of worker processes
Install and Configure Nginx: Install Nginx (a web server) and configure it to reverse proxy requests to Gunicorn.
Create a server block configuration file in Nginx (e.g., /etc/nginx/sitesavailable/your_app.conf) with content like:

server {
  listen 80; # Listen on port 80 (adjust if needed)
  server_name your_domain_name; # Replace with your domain name (optional)

  location / {
    proxy_pass http://localhost:5000; # Forward requests to Gunicorn
proxy_set_header Host $host; proxy_set_header X-Real-IP $remote_addr;
proxy_redirect off;
  }
}
Enable and Restart Nginx: Enable the created Nginx server block configuration and restart Nginx to apply the changes.

49. Make a fully functional web application using flask, Mangodb. Signup,Signin page.And after successfully login .Say hello Geeks message at webpage.

Prerequisites
Flask
Flask-WTF
Flask-Login
Flask-Bcrypt
PyMongo
WTForms
Itsdangerous
Step-by-Step Guide
1. Project Setup Bash
pip install Flask Flask-WTF Flask-Login Flask-Bcrypt PyMongo WTForms itsdangerous Use code with caution.

2. Import Necessary Libraries Python
from flask import Flask, render_template, redirect, url_for, flash, request
from flask_login import LoginManager, UserMixin, login_user, logout_user, login_required, current_user
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField from wtforms.validators
import InputRequired, Length, Email
from werkzeug.security import generate_password_hash, check_password_hash from flask_bcrypt
import Bcrypt import pymongo Use code with caution.

3. Create Flask App and MongoDB Connection
Python
app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key' # Replace with a strong secret key

# MongoDB connection
client = pymongo.MongoClient("mongodb://localhost:27017/") # Replace with your MongoDB connection string
db = client["your_database_name"] # Replace with your database name users_collection = db["users"]

bcrypt = Bcrypt(app) login_manager = LoginManager()
login_manager.init_app(app) login_manager.login_view
= 'login' Use code with caution.

4. Create User Model Python class User(UserMixin): def
__init__(self, id, email, password):
    self.id = id self.email = email
self.password = password

  def get_id(self): return str(self.id)

@login_manager.user_loader def load_user(user_id): user = users_collection.find_one({"_id": ObjectId(user_id)}) if user: return User(str(user["_id"]), user["email"], user["password"]) return None
Use code with caution.

5. Create Registration Form Python class
RegistrationForm(FlaskForm):
email = StringField('Email', validators=[InputRequired(), Email()])
password = PasswordField('Password', validators=[InputRequired(), Length(min=8, max=20)])
submit = SubmitField('Register') Use code with caution.

6. Create Login Form Python class
LoginForm(FlaskForm):
email = StringField('Email', validators=[InputRequired(), Email()]) password = PasswordField('Password', validators=[InputRequired()]) submit = SubmitField('Login')
Use code with caution.

7. Create Signup Route
Python
@app.route('/signup', methods=['GET', 'POST']) def signup(): form = RegistrationForm() if form.validate_on_submit(): hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8') user = {"email": form.email.data, "password": hashed_password} result = users_collection.insert_one(user) if result.inserted_id: flash('Registration successful!', 'success') return redirect(url_for('login'))
return render_template('signup.html', form=form) Use code with caution.

8. Create Login Route
Python
@app.route('/login', methods=['GET', 'POST']) def login():
form = LoginForm() if form.validate_on_submit(): user = users_collection.find_one({"email": form.email.data}) if user and bcrypt.check_password_hash(user["password"], form.password.data): user_obj = User(str(user["_id"]), user["email"], user["password"]) login_user(user_obj) return redirect(url_for('home')) else: flash('Invalid email or password', 'danger') return render_template('login.html', form=form) Use code with caution.

9. Create Home Route (Protected)
Python
@app.route('/') @login_required def home():

return "Hello Geeks!" Use code with caution.

10. Logout Route
Python
@app.route('/logout') @login_required
def logout(): logout_user()
return redirect(url_for('login'))
Use code with caution.

11. Create HTML Templates (signup.html, login.html, home.html)
Remember to replace placeholders with your actual MongoDB connection string, database name, and secret key.

This is a basic example. You can enhance it with features like password reset, email verification, and more robust error handling. Always prioritize security best practices when handling user data.

Additional considerations:

Error handling: Implement proper error handling for database operations, form validation, and other potential issues.
Security: Use strong password hashing algorithms, protect user data, and prevent common vulnerabilities like SQL injection and cross-site scripting (XSS).
User experience: Improve the user interface and experience with styling, error messages, and feedback.
Database indexing: Create appropriate indexes in MongoDB for efficient querying.
Session management: Securely manage user sessions.
By following these guidelines and incorporating additional features, you can build a more robust and secure Flask application with MongoDB.

50.Machine Learning:

- **What is the difference between Series & Dataframes.**

Series vs. DataFrames
Series: A one-dimensional labeled array capable of holding any data type (integers, strings, floating-point numbers, Python objects, etc.). It can be seen as a single column of a DataFrame.
DataFrame: A two-dimensional labeled data structure with columns of potentially different types. Think of it as a spreadsheet or SQL table.

- **Create a database name Travel_Planner in mysql and create a table name bookings in that which having attributes (user_id INT, flight_id INT,hotel_id INT, activity_id INT,booking_dote DATE) .fill with some**

dummy value Now you have to read the content of this table using pandas as dataframeShow the output.

Creating a Database and Table, Loading Data into Pandas DataFrame
Python import pandas as pd
import mysql.connector

```
# Database connection details mydb =
mysql.connector.connect(   host="your_host",
user="your_user",   password="your_password",
  database="your_database"
)

# Create a cursor object
mycursor = mydb.cursor()

# Create the database
mycursor.execute("CREATE DATABASE Travel_Planner") mydb.commit()

# Use the database mydb = mysql.connector.connect(
host="your_host",   user="your_user",
password="your_password",
  database="Travel_Planner"
)
mycursor = mydb.cursor()

# Create the table
mycursor.execute("CREATE TABLE bookings (user_id INT, flight_id INT, hotel_id INT, activity_id INT,
booking_date DATE)") mydb.commit()

# Insert some dummy data
sql = "INSERT INTO bookings (user_id, flight_id, hotel_id, activity_id, booking_date) VALUES
(%s, %s, %s, %s, %s)"
val = [(1, 101, 201, 301, '2023-11-24'),
      (2, 102, 202, 302, '2023-12-01'),
      # ... more data]
mycursor.executemany(sql, val) mydb.commit()

# Read data into a pandas DataFrame df = pd.read_sql("SELECT * FROM
bookings", mydb) print(df)
```

- **Difference between lac and iloc.** loc vs iloc loc: Label-based indexing. Uses row and column labels to select data. iloc: Integer-based indexing. Uses integer positions to select data.

- **What is the difference between supervised and unsupervised learning?**

Supervised vs. Unsupervised Learning
Supervised Learning: Algorithms learn from labeled data (input and desired output) to make predictions on new data. Examples: regression, classification.

Unsupervised Learning: Algorithms find patterns in unlabeled data. Examples: clustering, dimensionality reduction.

- **Explain the bias-variance trcideoff,**

Bias-Variance Tradeoff
Bias: Error due to simplifying assumptions in the model. High bias leads to underfitting.
Variance: Sensitivity of the model to training data variations. High variance leads to overfitting. The goal is to find a model with an optimal balance between bias and variance.

- **What are precision and recall? How are they different from accuracy?**

Precision: The proportion of positive identifications that were actually correct.
Recall: The proportion of actual positives that were identified correctly.
Accuracy: The proportion of correct predictions (both positive and negative) among the total number of cases.

- **What is overfitting and how can it be prevented?**

Overfitting
Overfitting occurs when a model learns the training data too well, capturing noise and random fluctuations instead of the underlying patterns. This leads to poor performance on new data.

Prevention:
Collect more data
Use simpler models
Regularization techniques (L1, L2)
Cross-validation

Early stopping

- **Explain the concept of cross-validation.**

Cross-validation is a resampling method used to evaluate a model's performance by splitting the dataset into multiple subsets. The model is trained on different combinations of these subsets and validated on the remaining subset. This helps assess the model's ability to generalize to unseen data and prevent overfitting.

Common types of cross-validation:

K-fold cross-validation
Leave-one-out cross-validation
Stratified k-fold cross-validation

- **What is the difference between a classification and a regression problem?**

Classification: Predicts categorical outcomes (e.g., spam or not spam, cat or dog).
Regression: Predicts continuous numerical values (e.g., house prices, stock prices).

- **Explain the concept of ensemble learning_**

Ensemble learning combines multiple models to improve predictive performance. By combining diverse models, the overall model can be more accurate, robust, and less prone to overfitting.

Common ensemble methods: Bagging, boosting, stacking

- **What is gradient descent and how does it work?**

Gradient Descent
Gradient descent is an optimization algorithm used to minimize a function by iteratively moving in the direction of the steepest descent. It's widely used in machine learning to find the optimal parameters for models.

Steps:

Initialize parameters randomly.
Calculate the gradient of the loss function with respect to the parameters.
Update parameters by taking a step in the opposite direction of the gradient. Repeat steps 2 and 3 until convergence.

- **Describe the difference between batch gradient descent and stochastic gradient descent**

Batch Gradient Descent: Calculates the gradient for the entire dataset in each iteration. It's computationally expensive but converges reliably.
Stochastic Gradient Descent: Calculates the gradient for a single randomly selected data point in each iteration. It's computationally efficient but can be noisy and may not converge smoothly.

- **What is the curse of dimensionality in machine learning?**

The curse of dimensionality refers to the challenges encountered when working with highdimensional data. As the number of features increases, the volume of the space increases exponentially, leading to several issues:

Data sparsity: Data points become increasingly sparse, making it difficult to find patterns. Computational complexity: Algorithms become computationally expensive as the dimensionality grows.
Overfitting: Models become more prone to overfitting as they can easily find complex patterns in noisy data.

- **Explain the difference between Li and L2 regularization?**

Regularization is a technique used to prevent overfitting by adding a penalty term to the loss function.

L1 Regularization (Lasso): Adds the absolute value of the coefficients as a penalty term. It tends to produce sparse models, meaning many coefficients become zero, effectively performing feature selection.
L2 Regularization (Ridge): Adds the squared magnitude of the coefficients as a penalty term. It tends to shrink coefficients without making them zero.

What is a confusion matrix and how is it used?

A confusion matrix is a table used to evaluate the performance of a classification model. It shows the number of correct and incorrect predictions, broken down by class.

Predicted Negative	Predicted Positive
True Negative (TN)	False Positive (FP)
False Negative (FN)	True Positive (TP)

From the confusion matrix, various metrics can be calculated, such as accuracy, precision, recall, and F1-score.

- **Define AUC-ROC curve.**

The AUC-ROC curve (Area Under the Receiver Operating Characteristic curve) is a graphical representation of the model's performance across different classification thresholds. It plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The AUC represents the probability that a randomly chosen positive instance will be ranked higher than a randomly chosen negative instance. A higher AUC indicates better model performance.

- **Explain the k-nearest neighbors algorithm.**

KNN is a simple classification and regression algorithm that classifies new data points based on the majority vote of their k nearest neighbors in the training set. The value of k determines the number of neighbors considered.

- **Explain the basic concept of a Support Vector Machine (SVM).**

SVM is a supervised machine learning algorithm used for classification and regression. It finds the optimal hyperplane that separates different classes in the feature space. The goal is to maximize the margin between the hyperplane and the closest data points (support vectors). SVMs can also handle non-linearly separable data using kernel functions.

- **How does the kernel trick work in SVM?**

The kernel trick is a mathematical technique used in support vector machines to efficiently compute the dot product of data points in a potentially high-dimensional feature space without explicitly mapping the data into that space. This is crucial because calculating and storing data in high-dimensional spaces can be computationally expensive. Kernel functions effectively compute the similarity between data points in the original space, implicitly mapping them to a higher-dimensional space where linear separation might be possible.

- **What are the different types of kernels used in SVM and when would you use each?**

Types of Kernels
Linear kernel: Used when the data is linearly separable.
Polynomial kernel: Introduces polynomial features to handle non-linear relationships. Radial Basis Function (RBF) kernel: Commonly used for non-linear data, providing a flexible mapping to high-dimensional space.
Sigmoid kernel: Inspired by neural networks, but less commonly used.

- **What is the hyperplane in SVM and how is it determined?**

A hyperplane is a decision boundary that separates data points into different classes. In SVM, the optimal hyperplane is the one that maximizes the margin between the closest data points of each class (support vectors).

- **What are the pros and cons of using a Support Vector Machine (SVM)?**

Pros and Cons of SVM Pros:

Effective in high-dimensional spaces
Can handle both linear and non-linear data (with kernels)
Produces sparse solutions with support vectors
Generally performs well with small to medium-sized datasets
Cons:

Can be computationally expensive for large datasets
Sensitive to outliers
Choice of kernel and hyperparameters can be challenging

- **Explain the difference between a hard margin and ❑ soft margin SVM.**

Hard margin SVM: Assumes data is linearly separable and aims to find a hyperplane that perfectly separates the classes. It's sensitive to outliers.
Soft margin SVM: Allows for some misclassifications by introducing a penalty term for misclassified points. It's more robust to outliers and noise.

- **Describe the process of constructing a decision tree.**

A decision tree is a supervised machine learning algorithm that creates a tree-like model of decisions and their possible consequences. It's used for both classification and regression problems.

Constructing a Decision Tree
Choose the root node: Select the feature that best splits the data into two or more homogeneous groups.
Splitting: Divide the data into subsets based on the chosen feature.
Repeat: Recursively apply steps 1 and 2 to each subset until a stopping criterion is met (e.g., maximum depth, minimum number of samples).
Pruning: Remove unnecessary branches to improve generalization.

- **Describe the working principle of a decision tree.**

Working Principle of a Decision Tree
A decision tree makes predictions by traversing the tree from the root node to a leaf node. At each internal node, a decision is made based on the value of a feature, and the process continues until a leaf node is reached, which represents the predicted class or value.

- **What is information gain and how is it used in decision trees?**

Information gain is a measure used to determine which feature is the best predictor for the target variable at each node of a decision tree.
It quantifies the reduction in entropy (impurity) achieved by splitting the data based on a particular feature.
Features with higher information gain are preferred for splitting nodes as they lead to more homogeneous child nodes.

- **Explain Gini impurity and its role in decision trees.**

Gini impurity is another measure of impurity used in decision trees to evaluate the quality of a split.
It represents the probability of incorrectly classifying a randomly chosen element from the dataset if it were randomly labeled according to the distribution of labels in the subset. Like information gain, the goal is to minimize Gini impurity when selecting the best split.

- **What are the advantages and disadvantages of decision trees?**

Advantages and Disadvantages of Decision Trees Advantages:

Easy to understand and interpret
Can handle both categorical and numerical data
Requires little data preparation Can handle non-linear
relationships Disadvantages:

Prone to overfitting
Sensitive to noisy data
Can be unstable due to small variations in the data.

- **How do random forests improve upon decision trees?**

Random forests are an ensemble learning method that constructs multiple decision trees and combines their predictions to improve accuracy and reduce overfitting.

- **How does a random forest algorithm work?**

Bootstrap aggregation: Random subsets of the data (with replacement) are created. Decision tree creation: A decision tree is built for each subset, using a random subset of features at each split.
Prediction: To make a prediction, each tree in the forest votes, and the most popular class or value is chosen.

- **What is bootstrapping in the context of random forests?**

Bootstrapping is a resampling technique where multiple samples are drawn with replacement from the original dataset. Each sample is used to build a decision tree in the random forest.
This process introduces diversity among the trees, reducing overfitting.

- **Explain the concept of feature importance in random forests.**

Feature Importance in Random Forests
Feature importance measures the relative contribution of each feature to the prediction model.
In random forests, feature importance is calculated by assessing how much the tree's accuracy decreases when a particular feature is randomly permuted.
Features with higher importance are considered more relevant for predicting the target variable.

- **What are the key hyperparameters of a random forest and how do they affect the model?**

Random Forest is a powerful ensemble learning algorithm, but its performance heavily relies on the appropriate tuning of its hyperparameters. Here are some of the most crucial ones:

1. n_estimators:
Definition: The number of trees in the forest.
Impact: Generally, increasing the number of trees improves model performance, but it also increases computation time. A larger number of trees helps to reduce variance.
2. max_depth:
Definition: The maximum depth of each tree.
Impact: Controls the complexity of the tree. A deeper tree can capture complex patterns but is more prone to overfitting. Limiting the depth can help prevent overfitting.
3. min_samples_split:
Definition: The minimum number of samples required to split an internal node.
Impact: A higher value prevents the model from learning noise in the data, reducing overfitting.
4. min_samples_leaf:

Definition: The minimum number of samples required to be at a leaf node.
Impact: Similar to min_samples_split, it helps to prevent overfitting.
5. max_features:
Definition: The number of features to consider when looking for the best split.
Impact: Controls the diversity of trees. A lower value increases randomness, which can help reduce overfitting.
6. bootstrap:
Definition: Whether bootstrap samples are used when building trees.
Impact: Bootstrapping introduces randomness and helps reduce variance.

Remember: The optimal values for these hyperparameters depend on the specific dataset and problem. It's often necessary to experiment with different combinations to find the best configuration. Techniques like grid search and randomized search can be used to automate the hyperparameter tuning process.

- **Describe the logistic regression model and its assumptions.**

A statistical method for predicting the probability of an event occurring, based on a set of independent variables.
Assumptions: Linear relationship between the log-odds and the independent variables, independence of observations, and no multicollinearity.

- **How does logistic regression handle binary classification problems?**

Binary Classification: Models the probability of the outcome being in one of two categories (e.g., 0 or 1, yes or no). The output is transformed using the sigmoid function to produce a probability between 0 and 1.

What is the sigmoid function and how is it used in logistic regression?

Sigmoid Function: A mathematical function that maps any real number to a value between 0 and 1. It's used to transform the linear output of logistic regression into a probability.

- **Explain the concept of the cost function in logistic regression.**

Cost Function: Measures the difference between the predicted and actual values. In logistic regression, the commonly used cost function is the log loss or cross-entropy loss.

- **How can logistic regression be extended to handle multiclass classification?**

Multiclass Classification: Can be extended using techniques like one-vs-rest or softmax regression.

- **What is the difference between Ll and L2 regularization in logistic regression?**

L1 vs L2 Regularization:

L1 (Lasso): Encourages sparsity, meaning some coefficients can become zero, potentially performing feature selection.
L2 (Ridge): Shrinks coefficients towards zero but doesn't force them to be exactly zero.

- **What is XGBoost and how does it differ from other boosting algorithms?**

XGBoost
Gradient Boosting: An ensemble method that builds models sequentially, where each subsequent model aims to correct the errors of the previous model.
XGBoost: An optimized implementation of gradient boosting with several enhancements:
Regularization: Controls overfitting.
Tree pruning: Improves efficiency and prevents overfitting.
Parallel processing: Speeds up training.
Handling missing values: Can handle missing data efficiently. Built-in cross-validation:
Provides performance estimates.

- **Explain the concept of boosting in the context of ensemble learning.**

Boosting is an ensemble learning technique that sequentially builds models, each correcting the errors of its predecessor. It focuses on improving the performance of weak learners by combining them into a strong learner.

How does XGBoost handle missing values?

XGBoost handles missing values efficiently by treating them as a separate category. It learns how to handle missing values during the tree building process, making it robust to data with missing values.

- **What are the key hyperparameters in XGBoost and how do they affect model performance?**

Key Hyperparameters in XGBoost **n_estimators:** Number of boosting stages (trees). **max_depth:** Maximum depth of each tree. **learning_rate:** Step size shrinkage used to prevent overfitting. **subsample:** Subsample ratio of the training instances.
colsample_bytree: Subsample ratio of columns when constructing each tree.
gamma: Minimum loss reduction required to make a further partition on a leaf node. **lambda:** L2 regularization term on weights. **alpha:** L1 regularization term on weights.

- **Describe the process of gradient boosting in X0Boost, a What are the advantages and disadvantages of using XGBoost?**

Gradient Boosting in XGBoost
XGBoost employs gradient boosting to sequentially build trees. Each tree aims to minimize the loss function by fitting the negative gradient of the loss function. This approach allows XGBoost to handle various loss functions effectively.

Advantages and Disadvantages of XGBoost

Advantages:

High performance and accuracy
Handles missing values efficiently
Supports various objective functions
Can handle large datasets
Provides efficient implementations

Disadvantages:
Can be computationally expensive for large datasets and complex models
Requires careful tuning of hyperparameters
Prone to overfitting if not properly regularized

Machine learning Practical question:

1. **Take any project from Pw Fynerienep Porti form machine learning domain.And make an end to end project with all the necessary documents.**

Link: **https:llexperience.pwskills.corni**

Visit : Github for files = https://github.com/SLRKing/Internship-Project/

Visit : Huggingface to run application =
https://huggingface.co/spaces/rocky1728/PredictGermanBankCreditRisk

Visit my youtube channel for video = https://youtu.be/EkLfA6L7wa8

Visit : Github for documents= https://github.com/SLRKing/Internship-Project/tree/main/Documents

2. **Do the EDA on the given dataset: Lung cancer, and extract some useful information from this.**

Dataset Description:
Lung cancer is one of the most prevalent and deadly forms of cancer worldwide, presenting significant challenges in early detection and effective treatment. To aid in the global effort to understand and combat this disease, we are excited to introduce our comprehensive Lung Cancer Dataset.

Visit : Github for files = https://github.com/SLRKing/Internship-Project/tree/main/EDA%20Assignmetts

3. Do the Eda on this Dataset :Presidential Election Polls 2024 Dataset and extract useful information from this:

Dataset: Nationwide Russian electign poll data from March 04, 2024

Dataset Description:
This dataset comprises the results of a nationwide presidential election poll conducted on March 4, 2024. The data offers various insights but does not align with the official election results, You are encouraged to create your notebooks and delve into the data for further exploration.

Visit : Github for files = https://github.com/SLRKing/Internship-Project/tree/main/EDA%20Assignmetts

eda-lung-cancer

August 14, 2024

1 EDA on Lung Cancer Dataset

Do the EDA on the given dataset: Lung cancer, and extract some useful information from this. Dataset Description: Lung cancer is one of the most prevalent and deadly forms of cancer worldwide, presenting significant challenges in early detection and effective treatment. To aid in the global effort to understand and combat this disease, we are excited to introduce our comprehensive Lung Cancer Dataset.

[2]:
```
#import libraries
import pandas as pd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# read the dataset
lung_cancer_df = pd.read_csv(r"D:
 ↪\PWskills\assign\solutions\projects-assignment\EDA␣
 ↪assginmets\lungcancer_dataset.csv")

# Display the first few rows to get an overview
lung_cancer_df.head(10)
```

[2]:

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	\
0	M	65	1	1	1	2	
1	F	55	1	2	2	1	
2	F	78	2	2	1	1	
3	M	60	2	1	1	1	
4	F	80	1	1	2	1	
5	F	58	1	1	1	2	
6	F	70	1	1	1	2	
7	F	74	2	2	1	1	
8	M	77	1	2	1	2	
9	F	67	2	2	2	2	

	CHRONIC_DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL_CONSUMING	COUGHING	\
0	2	1	2	2	2	2	
1	1	2	2	2	1	1	

```
2                1        2        1         2                  1         1
3                2        1        2         1                  1         2
4                1        2        1         2                  1         1
5                2        2        2         1                  2         2
6                2        1        2         2                  2         2
7                1        1        2         1                  1         1
8                1        1        1         1                  2         1
9                1        2        2         1                  2         1

   SHORTNESS_OF_BREATH  SWALLOWING_DIFFICULTY  CHEST_PAIN LUNG_CANCER
0                    2                      2           1          NO
1                    1                      2           2          NO
2                    2                      1           1         YES
3                    1                      2           2         YES
4                    1                      1           2          NO
5                    1                      1           2         YES
6                    2                      2           1         YES
7                    1                      2           1          NO
8                    1                      1           2          NO
9                    2                      1           1          NO
```

[3]:
```
lung_cancer_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   GENDER                 3000 non-null   object
 1   AGE                    3000 non-null   int64
 2   SMOKING                3000 non-null   int64
 3   YELLOW_FINGERS         3000 non-null   int64
 4   ANXIETY                3000 non-null   int64
 5   PEER_PRESSURE          3000 non-null   int64
 6   CHRONIC_DISEASE        3000 non-null   int64
 7   FATIGUE                3000 non-null   int64
 8   ALLERGY                3000 non-null   int64
 9   WHEEZING               3000 non-null   int64
 10  ALCOHOL_CONSUMING      3000 non-null   int64
 11  COUGHING               3000 non-null   int64
 12  SHORTNESS_OF_BREATH    3000 non-null   int64
 13  SWALLOWING_DIFFICULTY  3000 non-null   int64
 14  CHEST_PAIN             3000 non-null   int64
 15  LUNG_CANCER            3000 non-null   object
dtypes: int64(14), object(2)
memory usage: 375.1+ KB
```

the dataset has two of the columns are object types (categorical variables), rest are all integer types

containing values 1 and 2, where 1 = No and 2 = Yes

[4]:
```
lung_cancer_df.shape
```

[4]:
```
(3000, 16)
```

The data set has 3000 Rows and 16 Columns

[5]:
```
lung_cancer_df.isnull().sum()
```

[5]:
```
GENDER                    0
AGE                       0
SMOKING                   0
YELLOW_FINGERS            0
ANXIETY                   0
PEER_PRESSURE             0
CHRONIC_DISEASE           0
FATIGUE                   0
ALLERGY                   0
WHEEZING                  0
ALCOHOL_CONSUMING         0
COUGHING                  0
SHORTNESS_OF_BREATH       0
SWALLOWING_DIFFICULTY     0
CHEST_PAIN                0
LUNG_CANCER               0
dtype: int64
```

the dataset has no missing values, thats good for us

[6]:
```
# First, we will change the data types of the categorical variables
lung_cancer_df['LUNG_CANCER'] = lung_cancer_df['LUNG_CANCER'].
 ↪factorize(['NO','YES'])[0]
lung_cancer_df['GENDER'] = lung_cancer_df['GENDER'].factorize(['NO','YES'])[0]
# Male = 1 Female = 0
# Yes = 1 No = 0
```

[7]:
```
lung_cancer_df.head()
```

[7]:

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	\
0	1	65	1	1	1	2	
1	0	55	1	2	2	1	
2	0	78	2	2	1	1	
3	1	60	2	1	1	1	
4	0	80	1	1	2	1	

	CHRONIC_DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL_CONSUMING	COUGHING	\
0	2	1	2	2	2	2	

```
1                1        2        2        2                 1         1
2                1        2        1        2                 1         1
3                2        1        2        1                 1         2
4                1        2        1        2                 1         1

   SHORTNESS_OF_BREATH  SWALLOWING_DIFFICULTY  CHEST_PAIN  LUNG_CANCER
0                    2                      2           1            0
1                    1                      2           2            0
2                    2                      1           1            1
3                    1                      2           2            1
4                    1                      1           2            0
```

[34]:
```
lung_cancer_df.columns.unique
```

[34]:
```
<bound method Index.unique of Index(['GENDER', 'AGE', 'SMOKING',
'YELLOW_FINGERS', 'ANXIETY',
       'PEER_PRESSURE', 'CHRONIC_DISEASE', 'FATIGUE', 'ALLERGY', 'WHEEZING',
       'ALCOHOL_CONSUMING', 'COUGHING', 'SHORTNESS_OF_BREATH',
       'SWALLOWING_DIFFICULTY', 'CHEST_PAIN', 'LUNG_CANCER'],
      dtype='object')>
```

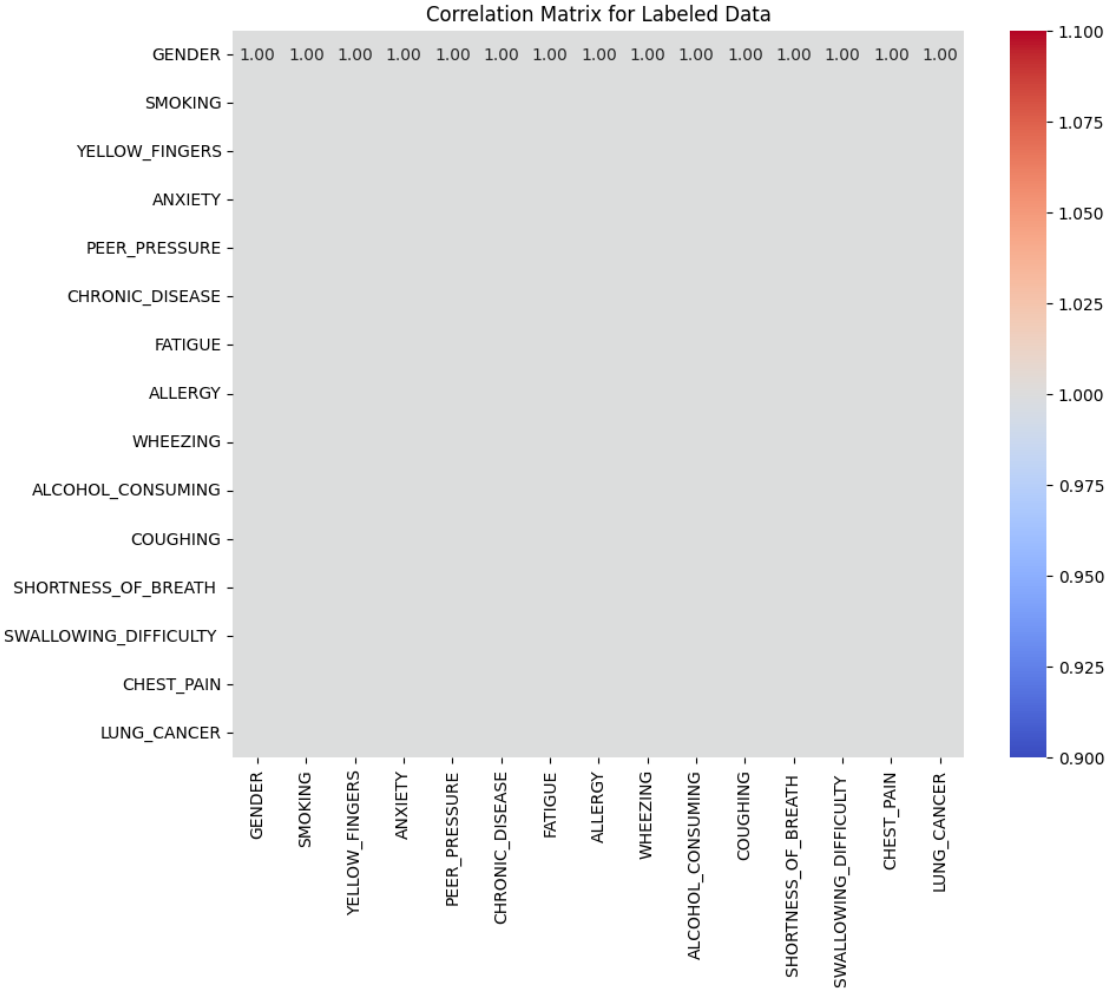
[35]:
```
unique_values = lung_cancer_df.nunique()

# Convert the result to a DataFrame for better readability
unique_values_df = unique_values.reset_index()
unique_values_df.columns = ['Feature', 'Unique Components']

# Display the DataFrame
print(unique_values_df)
```

```
                  Feature  Unique Components
0                  GENDER                  2
1                     AGE                 51
2                  SMOKING                  2
3           YELLOW_FINGERS                  2
4                 ANXIETY                  2
5           PEER_PRESSURE                  2
6          CHRONIC_DISEASE                  2
7                 FATIGUE                  2
8                 ALLERGY                  2
9                WHEEZING                  2
10      ALCOHOL_CONSUMING                  2
11               COUGHING                  2
12    SHORTNESS_OF_BREATH                  2
13  SWALLOWING_DIFFICULTY                  2
14             CHEST_PAIN                  2
15            LUNG_CANCER                  2
```

2 Exploratory Data Analysis

[40]:
```
import seaborn as sns
import matplotlib.pyplot as plt


# Select only numeric columns
numeric_df = lung_cancer_df.select_dtypes(include=['number'])

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

import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd


lung_cancer_df = pd.DataFrame({
    'GENDER': [0,1],
    'SMOKING': [1,2],
    'YELLOW_FINGERS': [1,2],
    'ANXIETY': [1,2],
    'PEER_PRESSURE': [1,2],
    'CHRONIC_DISEASE': [1,2],
    'FATIGUE': [1,2],
    'ALLERGY': [1,2],
    'WHEEZING': [1,2],
    'ALCOHOL_CONSUMING': [1,2],
    'COUGHING': [1,2],
    'SHORTNESS_OF_BREATH ': [1,2],
    'SWALLOWING_DIFFICULTY ': [1,2],
    'CHEST_PAIN': [1,2],
    'LUNG_CANCER': [0,1]
})

# Select only numeric columns including the label
numeric_df = lung_cancer_df.select_dtypes(include=['number'])

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

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')

# Add title
plt.title('Correlation Matrix for Labeled Data')
```

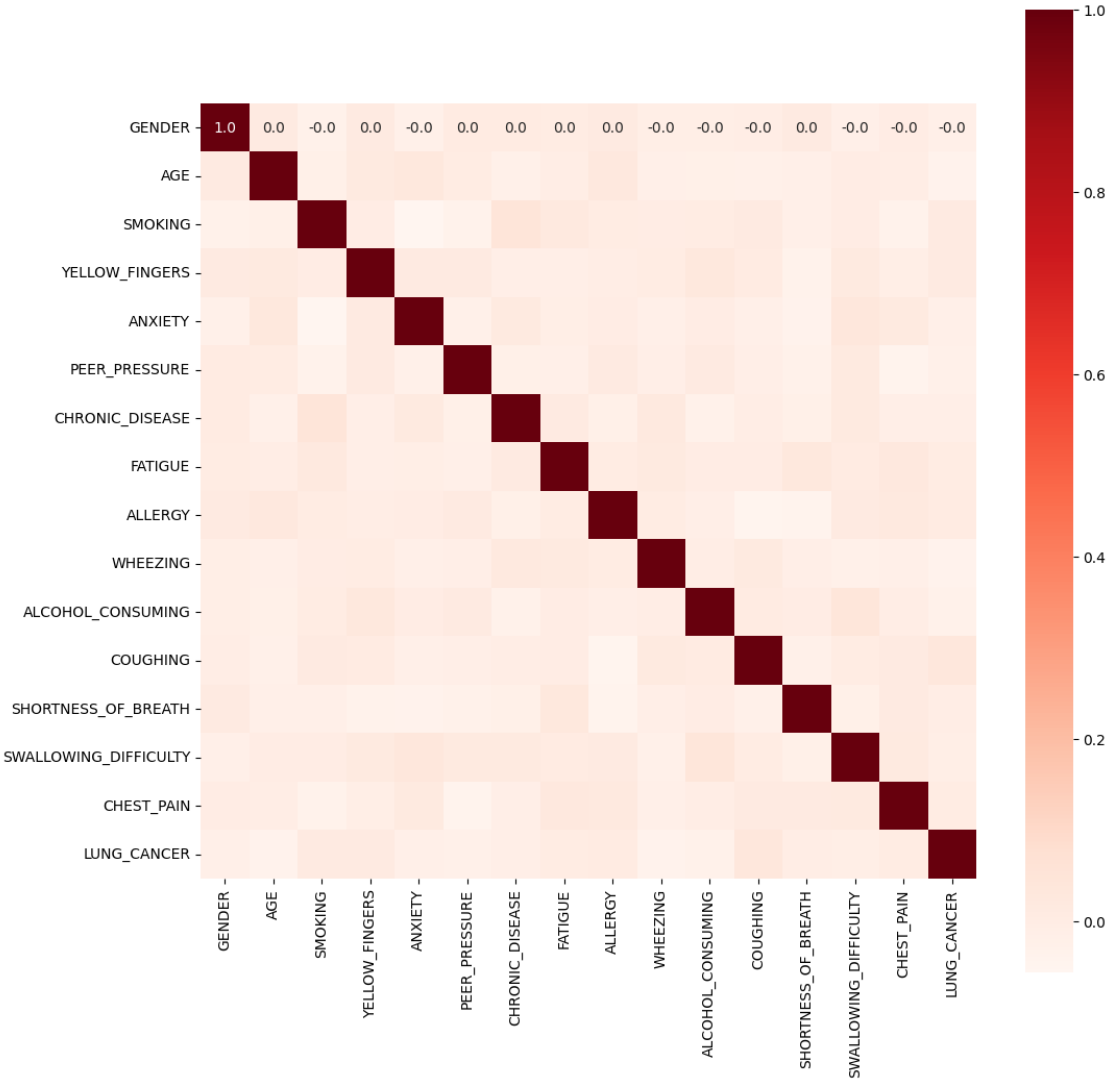
```
# Show the plot
plt.show()
```



As there are no enough numerical data in the dataset, we couldn't use correlation. After using labelencoded numeircals there is no strong corelation.

[8]:
```
# Lets first plot heatmap to check correlation between the variables.

corr = lung_cancer_df.corr()
plt.figure(figsize=(12,12))
sns.heatmap(corr,cbar=True,square=True,fmt='.1f',annot=True,cmap='Reds')
```

[8]: <Axes: >



There aren't many strong correlations among the features, except for a few minor ones:

- More anxiety leads to more difficulty breathing.
- Anxiety and yellow fingers are related, possibly in both directions.
- Shortness of breath causes fatigue.
- Increased anxiety makes swallowing harder.

As there are

However, there isn't a strong correlation between the symptoms and the target variable, possibly due to the data type or survey methods.

[9]:
```
# Now we will rename the 1s and 2s to No and Yes so it is easier to understand
```

```
lung_cancer_df["GENDER"] = lung_cancer_df["GENDER"].replace({1:"Male", 0:
 ↪"Female"})
lung_cancer_df["LUNG_CANCER"] = lung_cancer_df["LUNG_CANCER"].replace({1:"Yes",␣
 ↪0:"No"})
for column in lung_cancer_df.columns:
    lung_cancer_df[column] = lung_cancer_df[column].replace({1: "No", 2: "Yes"})

lung_cancer_df.head()
```

[9]:

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC_DISEASE	\
0	Male	65	No	No	No	Yes	Yes	
1	Female	55	No	Yes	Yes	No	No	
2	Female	78	Yes	Yes	No	No	No	
3	Male	60	Yes	No	No	No	Yes	
4	Female	80	No	No	Yes	No	No	

	FATIGUE	ALLERGY	WHEEZING	ALCOHOL_CONSUMING	COUGHING	SHORTNESS_OF_BREATH	\
0	No	Yes	Yes	Yes	Yes	Yes	
1	Yes	Yes	Yes	No	No	No	
2	Yes	No	Yes	No	No	Yes	
3	No	Yes	No	No	Yes	No	
4	Yes	No	Yes	No	No	No	

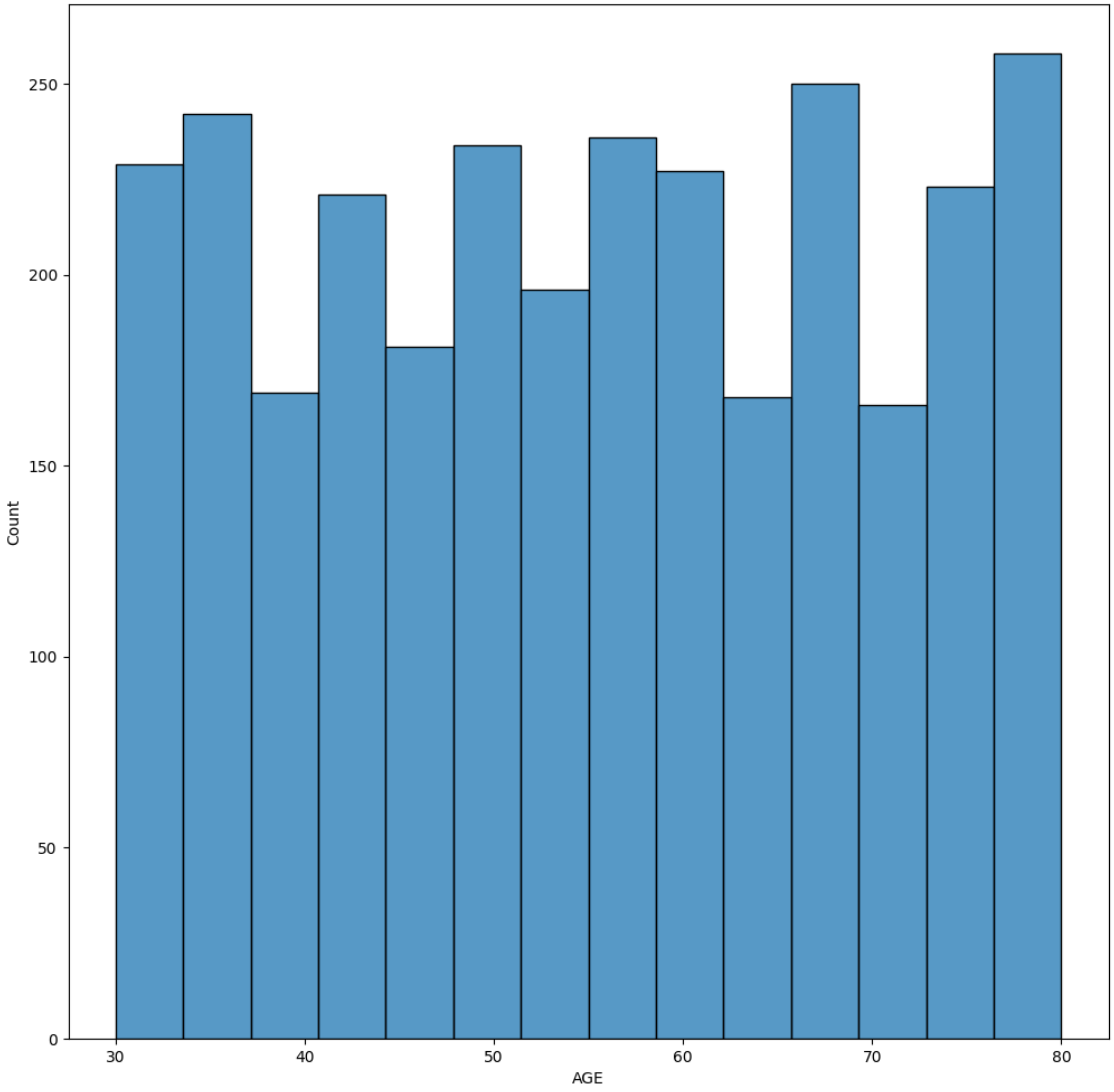
	SWALLOWING_DIFFICULTY	CHEST_PAIN	LUNG_CANCER
0	Yes	No	No
1	Yes	Yes	No
2	No	No	Yes
3	Yes	Yes	Yes
4	No	Yes	No

3 Age Distribution :

[10]:
```
plt.figure(figsize = (12,12))
sns.histplot(lung_cancer_df['AGE'])
```

```
C:\Users\ravit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

[10]: <Axes: xlabel='AGE', ylabel='Count'>

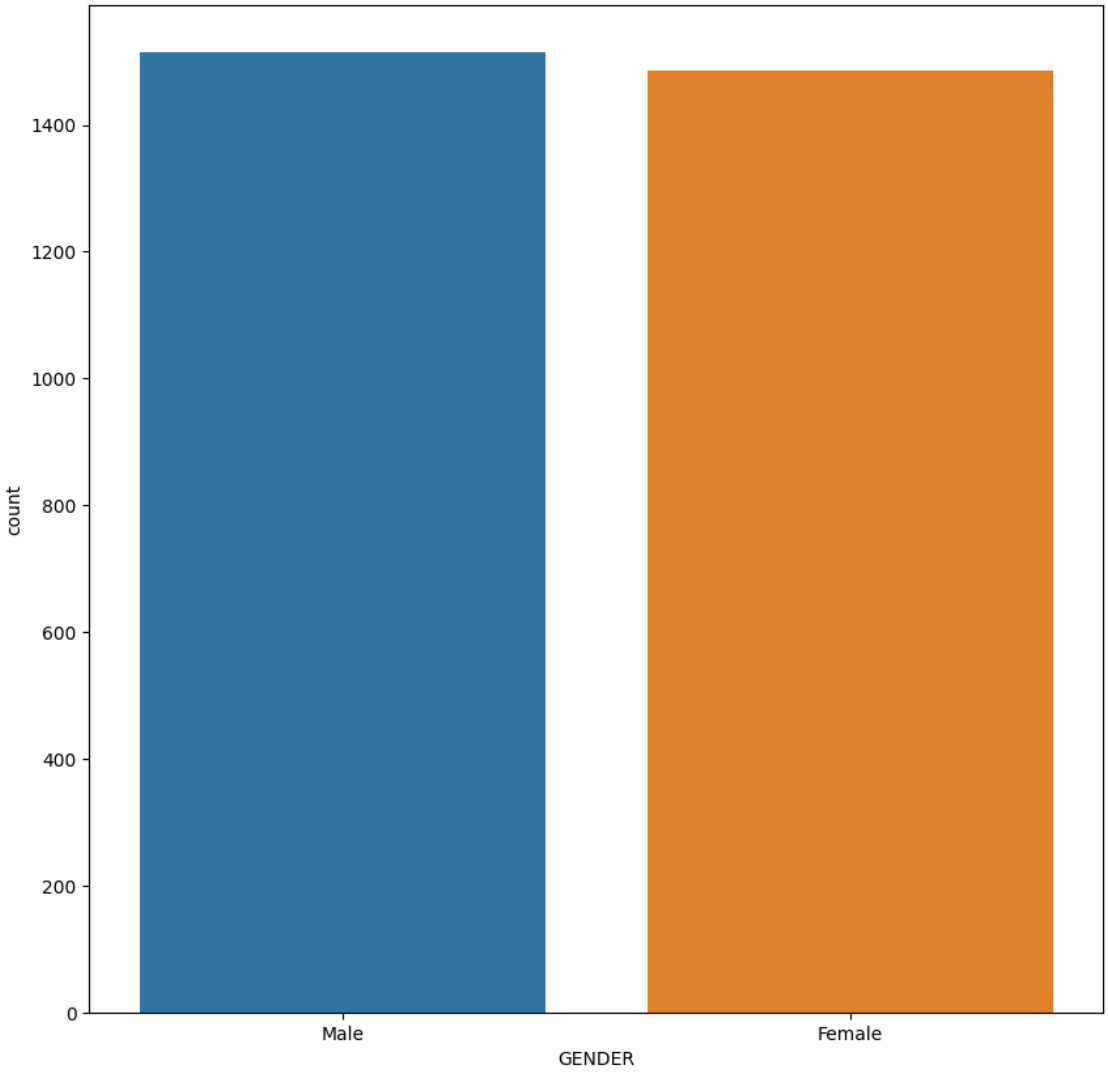


We can see that the data is distributed in discrete of age, we can see that the people of age 30 and 80 has no such significant difference, and other age categories has also no significant difference.

4 Gender Distribution:

[11]:
```
plt.figure(figsize=(10,10))
sns.countplot(x="GENDER", data=lung_cancer_df)
```

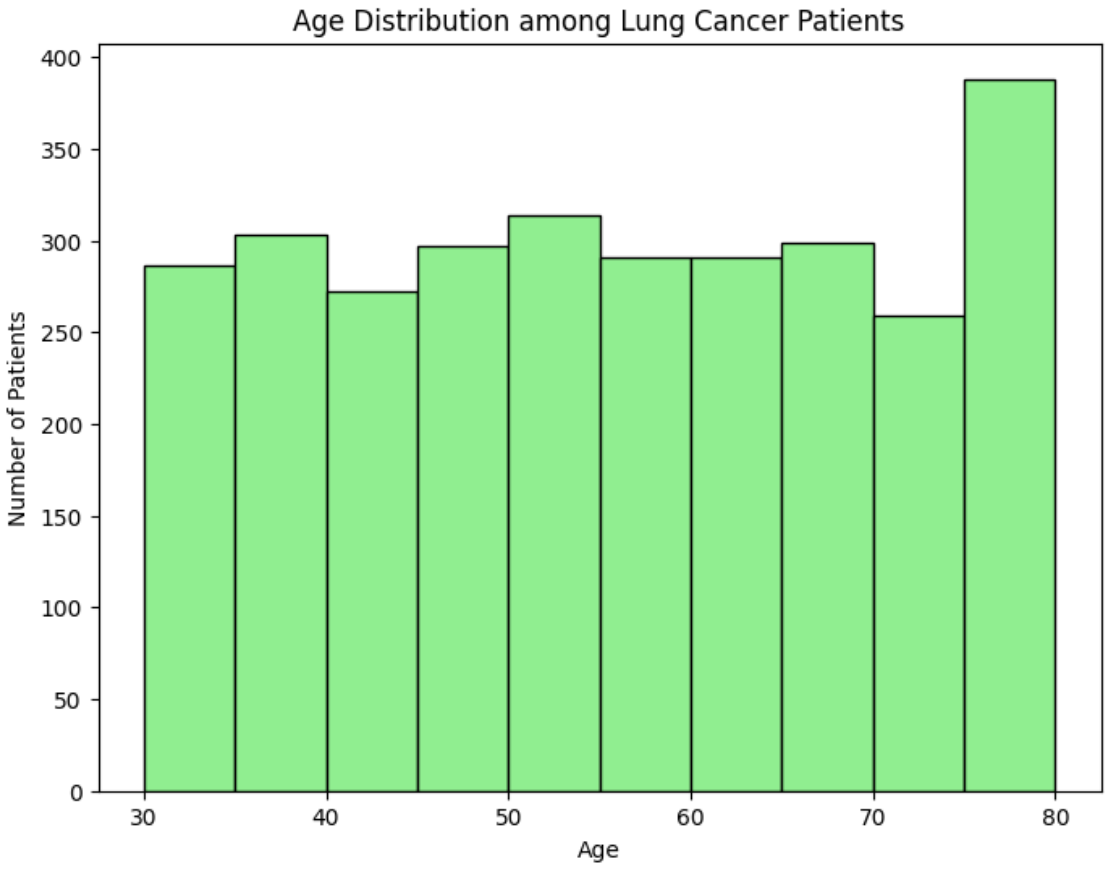
[11]: <Axes: xlabel='GENDER', ylabel='count'>



In the dataset the difference between number of males and females is not too much.

5 Age Distribution with People having Lung Cancer

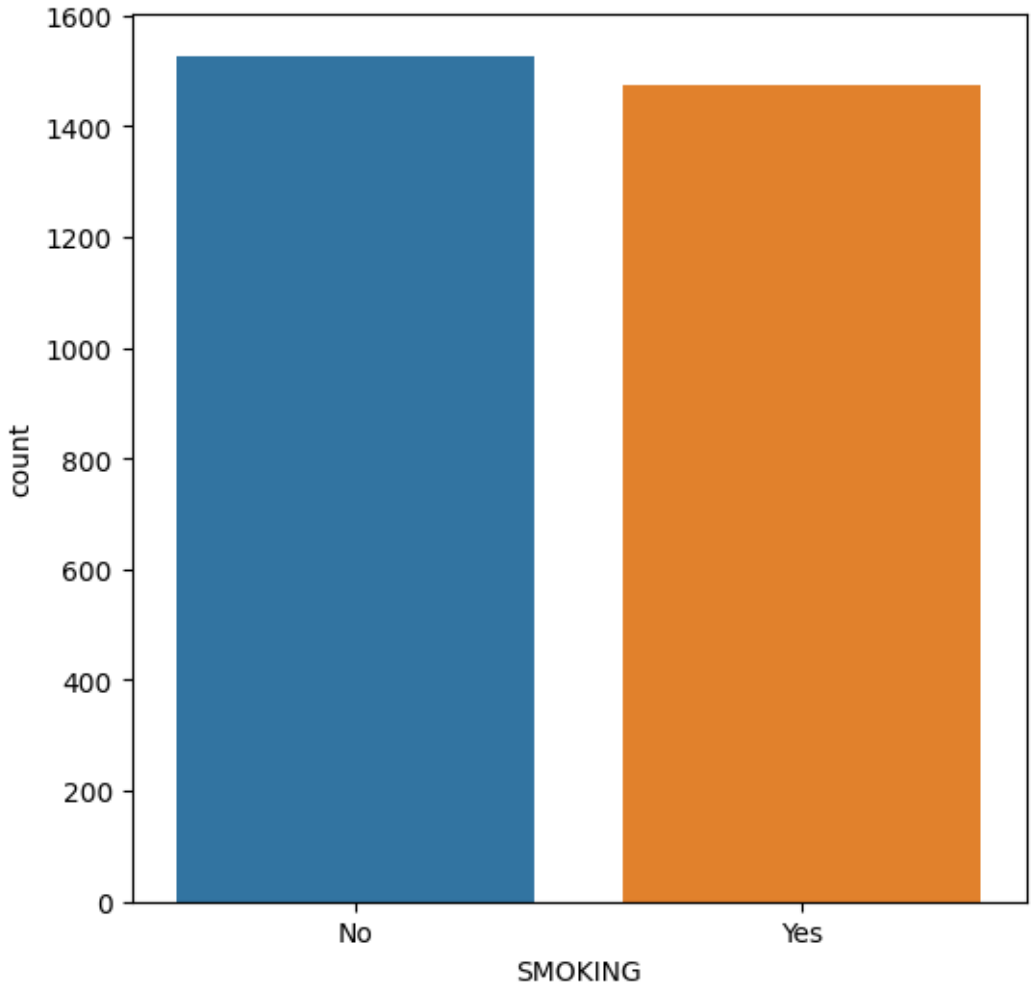
[12]:
```
ages = lung_cancer_df['AGE']

# Create a histogram
plt.figure(figsize=(8, 6))
plt.hist(ages, bins=10, color='lightgreen', edgecolor='black')
plt.xlabel('Age')
plt.ylabel('Number of Patients')
plt.title('Age Distribution among Lung Cancer Patients')
plt.show()
```



6 Smokers count

[13]:
```
plt.figure(figsize=(6,6))
sns.countplot(x="SMOKING", data=lung_cancer_df)
```

[13]:
```
<Axes: xlabel='SMOKING', ylabel='count'>
```

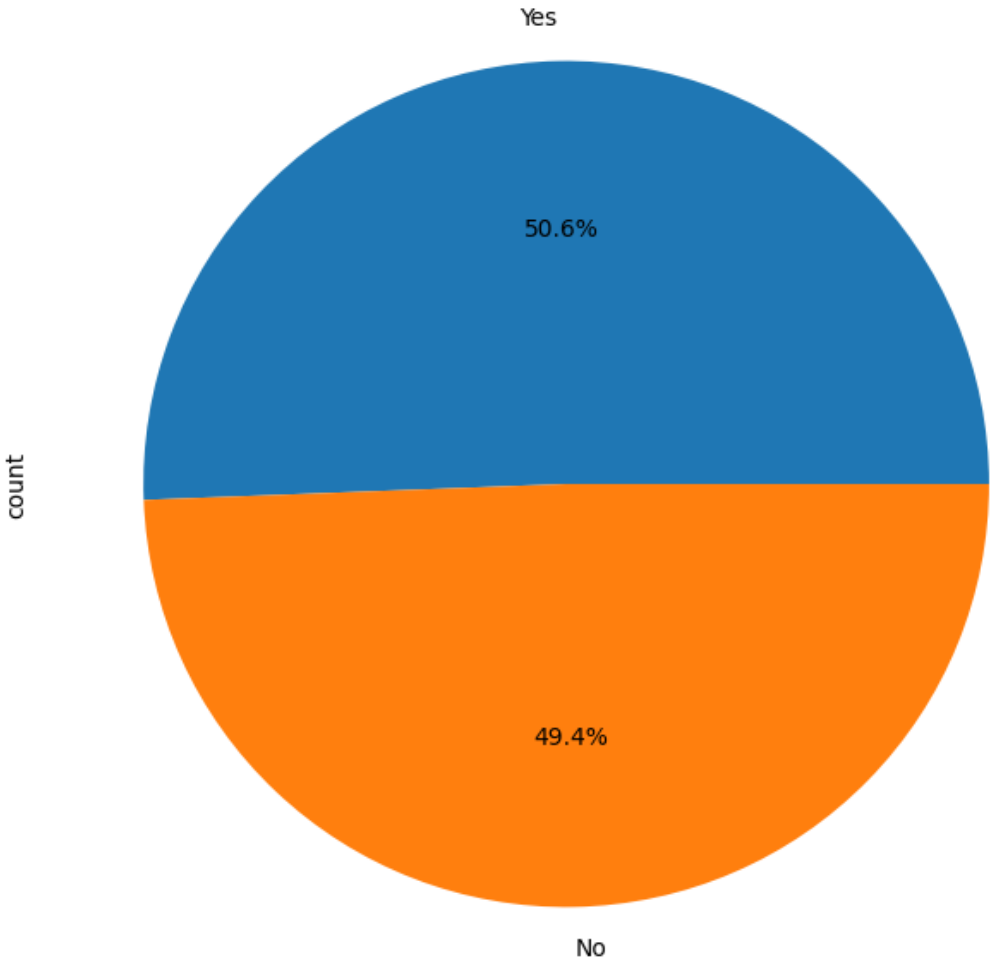


There are more smokers in the dataset than non-smokers

7 Percentage of people with lung cancer in the dataset

[14]:
```
# cancer_data.LUNG_CANCER.value_counts() / len(cancer_data.LUNG_CANCER)

lung_cancer_df.LUNG_CANCER.value_counts().plot(kind='pie',figsize=(8,␣
↪8),autopct='%1.1f%%')
```

[14]:
```
<Axes: ylabel='count'>
```



[15]:
```
lung_cancer_df.LUNG_CANCER.value_counts() *10/ len(lung_cancer_df.
↪LUNG_CANCER)*10
```

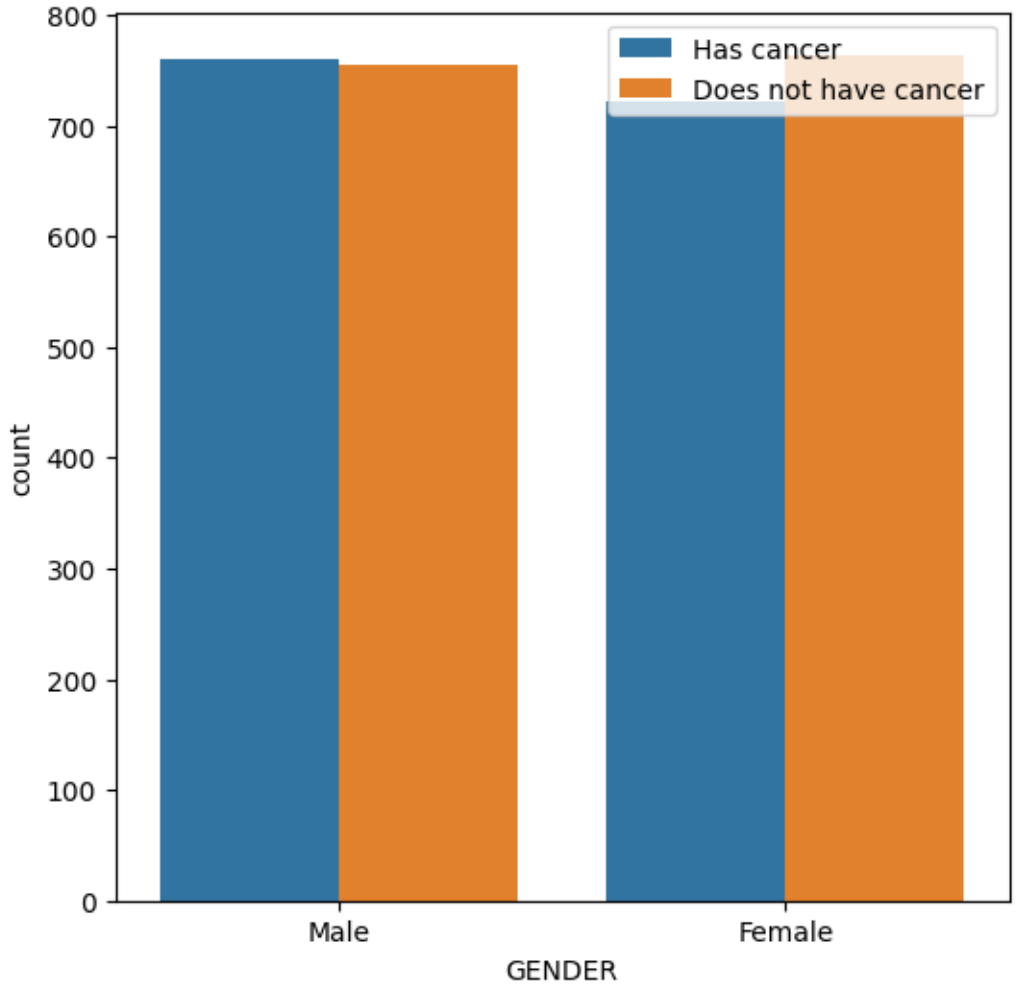
[15]:
```
LUNG_CANCER
Yes    50.6
No     49.4
Name: count, dtype: float64
```

There are 50.6% of Lung Cancer Patiences and 49.4% does not have lung cancer

8 Lung Cancer across Genders

[16]:
```
plt.figure(figsize=(6,6))
sns.countplot(data=lung_cancer_df,x='GENDER',hue='LUNG_CANCER')
plt.legend(["Has cancer", 'Does not have cancer'])
```

[16]: <matplotlib.legend.Legend at 0x230fd9bd910>

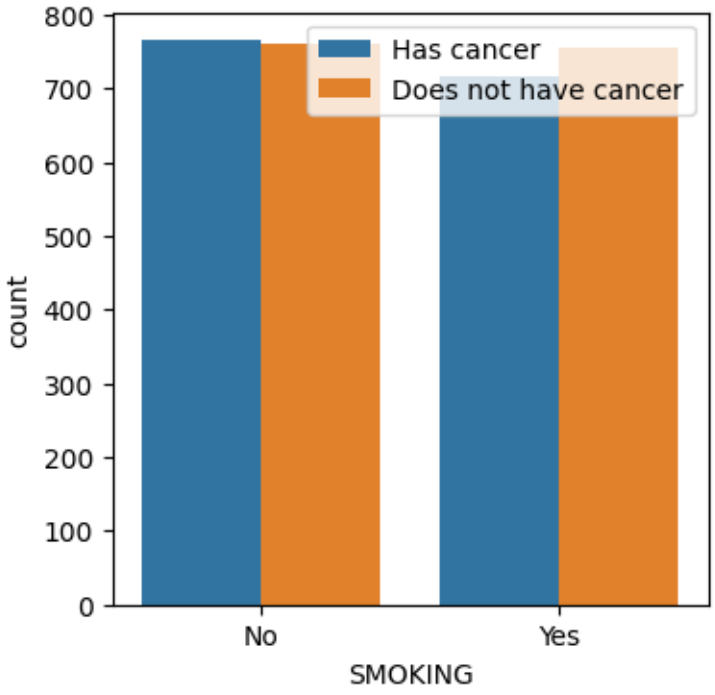


We see that in Male there is no much difference between Cancer and non-Cancer people[Male Cancer count is slighltly higher than non-Cancer]. But, in female we see that there is some amout of difference between cancer and non-cancer people[In female people with cancer are lesser than non-cancer]

9 Smoking and Lung Cancer:

[17]:
```
plt.figure(figsize=(4,4))
sns.countplot(data=lung_cancer_df,x='SMOKING',hue='LUNG_CANCER')
plt.legend(["Has cancer", 'Does not have cancer'])
```

[17]:
```
<matplotlib.legend.Legend at 0x230ff052bd0>
```

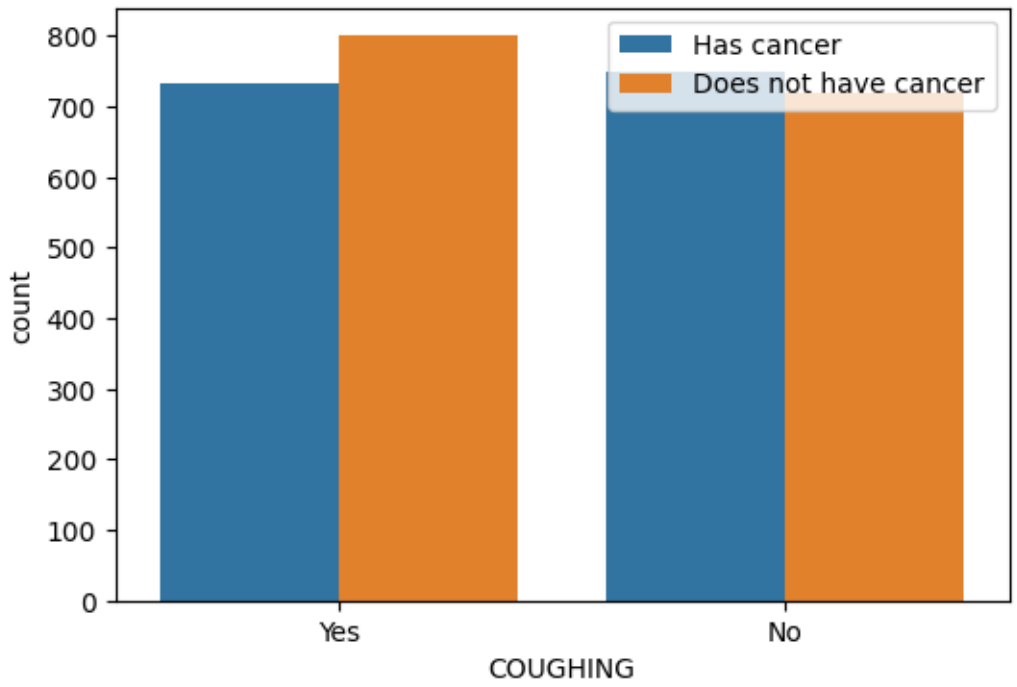


From the above data: People who doesn't smoke : Who has cancer and who doesn't have cancer is almost same. People who smoke: Who has cancer due to smoking is lesser than the people who does not have cancer even after smoking.

10 Coughing and Lung Cancer:

[18]:
```
plt.figure(figsize=(6,4))
sns.countplot(data=lung_cancer_df,x='COUGHING',hue='LUNG_CANCER')
plt.legend(["Has cancer", 'Does not have cancer'])
```

[18]:
```
<matplotlib.legend.Legend at 0x230ff0a8910>
```



The data shows that:

People who cough:Number of people who cough and have cancer is less than the people who cough and does not have cancer.

People who does not cough: People who does not cough and have cancer is slightly higher than the people who does not cough and does not have cancer.This shows that only smoking is not responsible for cancer, other factors are also included may be such has pollution, people working in some chemical/manufacturing industries. Because we have lack of data of peoples demography and profile we cannot conclude.

[19]:
```
lung_cancer_df.head()
```

[19]:

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC_DISEASE	\
0	Male	65	No	No	No	Yes	Yes	
1	Female	55	No	Yes	Yes	No	No	
2	Female	78	Yes	Yes	No	No	No	
3	Male	60	Yes	No	No	No	Yes	
4	Female	80	No	No	Yes	No	No	

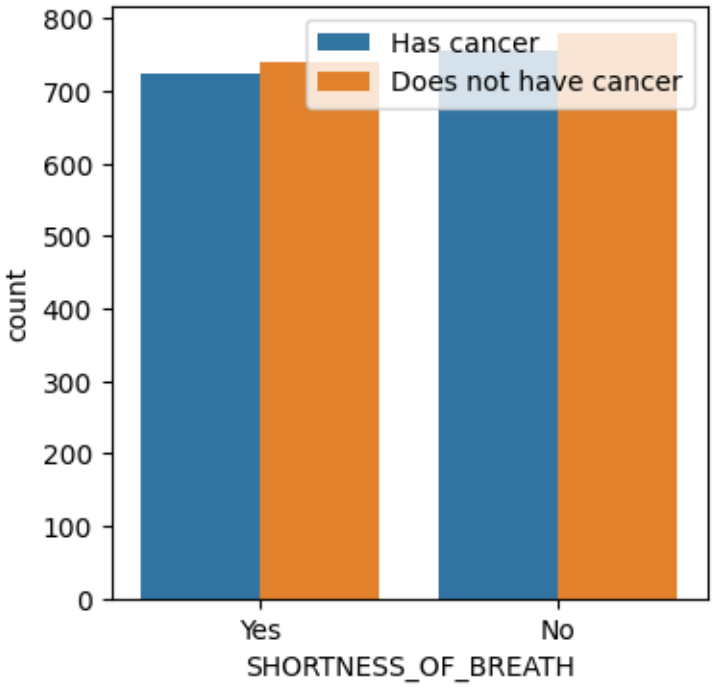
	FATIGUE	ALLERGY	WHEEZING	ALCOHOL_CONSUMING	COUGHING	SHORTNESS_OF_BREATH	\
0	No	Yes	Yes	Yes	Yes	Yes	
1	Yes	Yes	Yes	No	No	No	
2	Yes	No	Yes	No	No	Yes	
3	No	Yes	No	No	Yes	No	
4	Yes	No	Yes	No	No	No	

	SWALLOWING_DIFFICULTY	CHEST_PAIN	LUNG_CANCER
0	Yes	No	No
1	Yes	Yes	No
2	No	No	Yes
3	Yes	Yes	Yes
4	No	Yes	No

11 SHORTNESS OF BREATH with LUNG_CANCER

[20]:
```
plt.figure(figsize=(4,4))
sns.countplot(data=lung_cancer_df,x='SHORTNESS_OF_BREATH',hue='LUNG_CANCER')
plt.legend(["Has cancer", 'Does not have cancer'])
```

[20]: <matplotlib.legend.Legend at 0x230ff132410>



Interestingly, in all above of the plots we can see that it doesn't really matter if the symptoms are showing or not as some people may have cancer yet not show any symptoms. So, regular checkups would be wiser than waiting for symptoms to show up as that may lead it to deteriorate conditions.

[21]:
```
lung_cancer_df.head()
```

[21]:

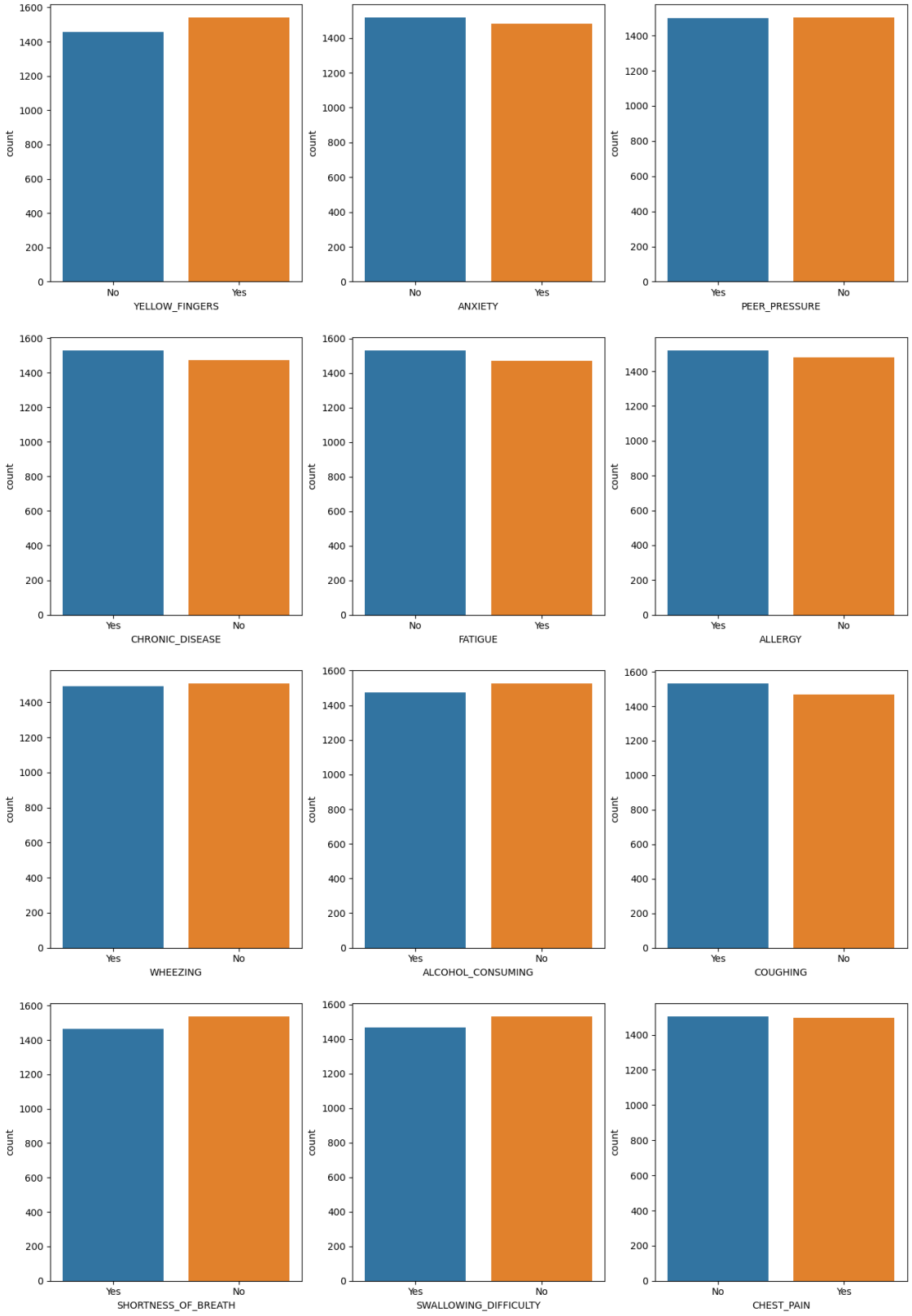
	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC_DISEASE
0	Male	65	No	No	No	Yes	Yes
1	Female	55	No	Yes	Yes	No	No
2	Female	78	Yes	Yes	No	No	No
3	Male	60	Yes	No	No	No	Yes
4	Female	80	No	No	Yes	No	No

	FATIGUE	ALLERGY	WHEEZING	ALCOHOL_CONSUMING	COUGHING	SHORTNESS_OF_BREATH
0	No	Yes	Yes	Yes	Yes	Yes
1	Yes	Yes	Yes	No	No	No
2	Yes	No	Yes	No	No	Yes
3	No	Yes	No	No	Yes	No
4	Yes	No	Yes	No	No	No

	SWALLOWING_DIFFICULTY	CHEST_PAIN	LUNG_CANCER
0	Yes	No	No
1	Yes	Yes	No
2	No	No	Yes
3	Yes	Yes	Yes
4	No	Yes	No

12 Lets plot all symptoms with cancer

[22]:
```
X = ['YELLOW_FINGERS', 'ANXIETY','PEER_PRESSURE', 'CHRONIC_DISEASE', 'FATIGUE',␣
 ↪'ALLERGY', 'WHEEZING','ALCOHOL_CONSUMING', 'COUGHING',␣
 ↪'SHORTNESS_OF_BREATH',  'SWALLOWING_DIFFICULTY', 'CHEST_PAIN']
fig, ax = plt.subplots(nrows = 4, ncols = 3) # 16 subplots
fig.set_size_inches(16,24) # set figure size

for i in range(4):
    for j in range(3):
        sns.countplot(x = lung_cancer_df[X[3 * i + j]] , ax = ax[i][j]) # count␣
 ↪plot
```


count
YELLOW_FINGERS
No
Yes
ANXIETY
No
Yes
PEER_PRESSURE
Yes
No
CHRONIC_DISEASE
Yes
No
FATIGUE
No
Yes
ALLERGY
Yes
No
WHEEZING
Yes
No
ALCOHOL_CONSUMING
Yes
No
COUGHING
Yes
No
SHORTNESS_OF_BREATH
Yes
No
SWALLOWING_DIFFICULTY
Yes
No
CHEST_PAIN
No
Yes

After plotting multiple countplots for the different features, we can see that there is no specific symptom that shows the significant amout of effect that causes cancer.There may be the reason that sometimes the cancer is genitically transmitted, or the data does not have much variance to say that only some specific symptom leads to cancer.

[23]:
```
lung_cancer_df.columns
```

[23]:
```
Index(['GENDER', 'AGE', 'SMOKING', 'YELLOW_FINGERS', 'ANXIETY',
       'PEER_PRESSURE', 'CHRONIC_DISEASE', 'FATIGUE', 'ALLERGY', 'WHEEZING',
       'ALCOHOL_CONSUMING', 'COUGHING', 'SHORTNESS_OF_BREATH',
       'SWALLOWING_DIFFICULTY', 'CHEST_PAIN', 'LUNG_CANCER'],
      dtype='object')
```

Feature Engineering : runnign chi-square test

- Chi-square: a statistical test used to determine whether there is a significant association or independence between two categorical variables.
- Relationship to Data Chi-Square Values: Quantify the difference between observed and expected frequencies in categorical data. P-Values: Indicate the probability of observing the data if the null hypothesis is true. A low p-value suggests that the observed data is unlikely under the null hypothesis, leading to its rejection.

[26]:
```
import pandas as pd
from scipy.stats import chi2_contingency

# Assuming df_updates is your DataFrame containing the data

# Create a list to store the results
chi2_results_list = []

# List of categorical features
categorical_features = ['GENDER', 'AGE', 'SMOKING', 'YELLOW_FINGERS', 'ANXIETY',
       'PEER_PRESSURE', 'CHRONIC_DISEASE', 'FATIGUE', 'ALLERGY', 'WHEEZING',
       'ALCOHOL_CONSUMING', 'COUGHING', 'SHORTNESS_OF_BREATH',
       'SWALLOWING_DIFFICULTY', 'CHEST_PAIN', 'LUNG_CANCER']

def contingency_table(feature):
    p = pd.crosstab(lung_cancer_df['LUNG_CANCER'],lung_cancer_df[feature])
    return(p)
feature = []
pval = []
chi2_result = []

for i in categorical_features:
    feature.append(i)
    result = chi2_contingency(contingency_table(i))
    pval.append(round(float(result[1]),6))
```

```
    if float(result[1]) < 0.05:
        chi2_result.append("Significant")
    else:
        chi2_result.append("Insignificant")
```

[27]:
```
chisquare = pd.DataFrame(data={'PValue':pval,'Result':
 ↪chi2_result},index=feature)
chisquare
```

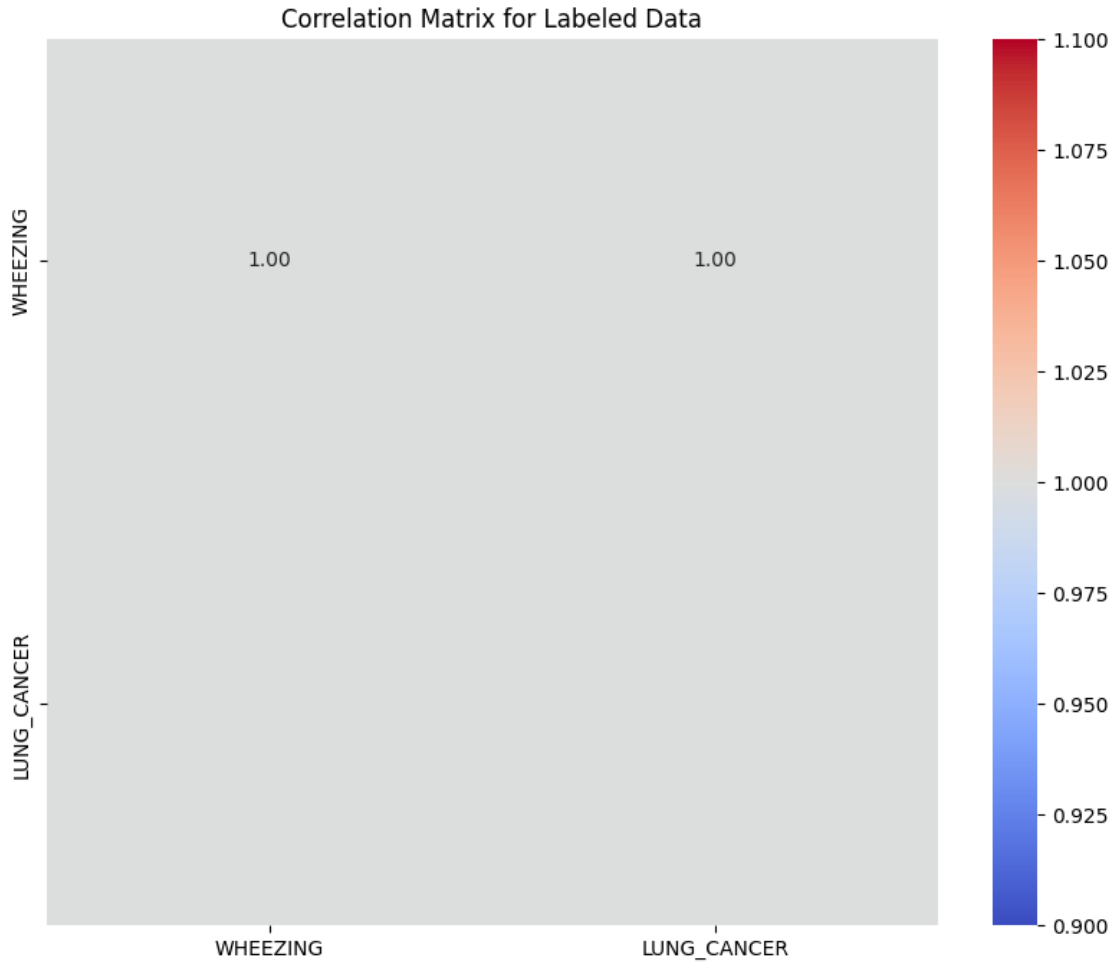
[27]:

	PValue	Result
GENDER	0.397512	Insignificant
AGE	0.431914	Insignificant
SMOKING	0.457905	Insignificant
YELLOW_FINGERS	0.499236	Insignificant
ANXIETY	0.447832	Insignificant
PEER_PRESSURE	0.189076	Insignificant
CHRONIC_DISEASE	0.600243	Insignificant
FATIGUE	0.930959	Insignificant
ALLERGY	0.749028	Insignificant
WHEEZING	0.037708	Significant
ALCOHOL_CONSUMING	0.102650	Insignificant
COUGHING	0.075776	Insignificant
SHORTNESS_OF_BREATH	0.925265	Insignificant
SWALLOWING_DIFFICULTY	0.671044	Insignificant
CHEST_PAIN	0.911374	Insignificant
LUNG_CANCER	0.000000	Significant

From the above method we got two significatn relation of WHEEZING AND LUNGCANCER, Lets plot corelation of these.

[41]:
```
lung_cancer_df = pd.DataFrame({

    'WHEEZING': [1,2],

    'LUNG_CANCER': [0,1]
})

# Select only numeric columns including the label
numeric_df = lung_cancer_df.select_dtypes(include=['number'])

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

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
```

```
# Add title
plt.title('Correlation Matrix for Labeled Data')

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



Again we didnt find any co-relation of these variables.

13 Conclusion

Based on the data, we've made inferences such as relationships between symptoms and the likeliness of having lung cancer and relationships with gender, age, etc.

Training a machine learning model with this dataset could help us predict whether or not someone will be diagnosed with lung cancer or not.

russia-eda

August 14, 2024

3. Do the Eda on this Dataset :Presidential Election Polls 2024 Dataset and extract useful information from this: Dataset: Nationwide Russian electign poll data from March 04, 2024 Dataset Description: This dataset comprises the results of a nationwide presidential election poll conducted on March 4, 2024. The data offers various insights but does not align with the official election results, You are encouraged to create your notebooks and delve into the data for further exploration.

[2]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
```

[3]:
```
df=pd.read_csv(r"D:\PWskills\assign\solutions\projects-assignment\EDA␣
↪assginmets\russia dataset.csv")
df.head(2)
```

[3]:
```
                 id   sex  age federal_district type_of_city  \
0  07169ed8148ce047  male   18  north caucasian      village
1  0716a4f3354cecdd  male   23  north caucasian      village

  knows_election_date will_vote candidate      television_usage  \
0  named correct date  not sure     Putin  several times a week
1  named correct date  not sure     Putin      once half a year

       internet_usage                    education     income     employment  \
0  over 4 hours a day  incomplete school education  very high   entrepreneur
1  over 4 hours a day                      college  very high  work for hire

                  job_type company_type   weight1
0                      NaN      farming  1.445172
1  commercial organization        trade  1.445172
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600 entries, 0 to 1599
```

```
Data columns (total 16 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    1600 non-null   object
 1   sex                   1600 non-null   object
 2   age                   1600 non-null   int64
 3   federal_district      1600 non-null   object
 4   type_of_city          1600 non-null   object
 5   knows_election_date   1600 non-null   object
 6   will_vote             1600 non-null   object
 7   candidate             1600 non-null   object
 8   television_usage      1600 non-null   object
 9   internet_usage        1600 non-null   object
 10  education             1600 non-null   object
 11  income                1600 non-null   object
 12  employment            1600 non-null   object
 13  job_type              692 non-null    object
 14  company_type          879 non-null    object
 15  weight1               1600 non-null   float64
dtypes: float64(1), int64(1), object(14)
memory usage: 200.1+ KB
```

[5]:
```
df.isnull().sum()
```

[5]:
```
id                     0
sex                    0
age                    0
federal_district       0
type_of_city           0
knows_election_date    0
will_vote              0
candidate              0
television_usage       0
internet_usage         0
education              0
income                 0
employment             0
job_type             908
company_type         721
weight1                0
dtype: int64
```

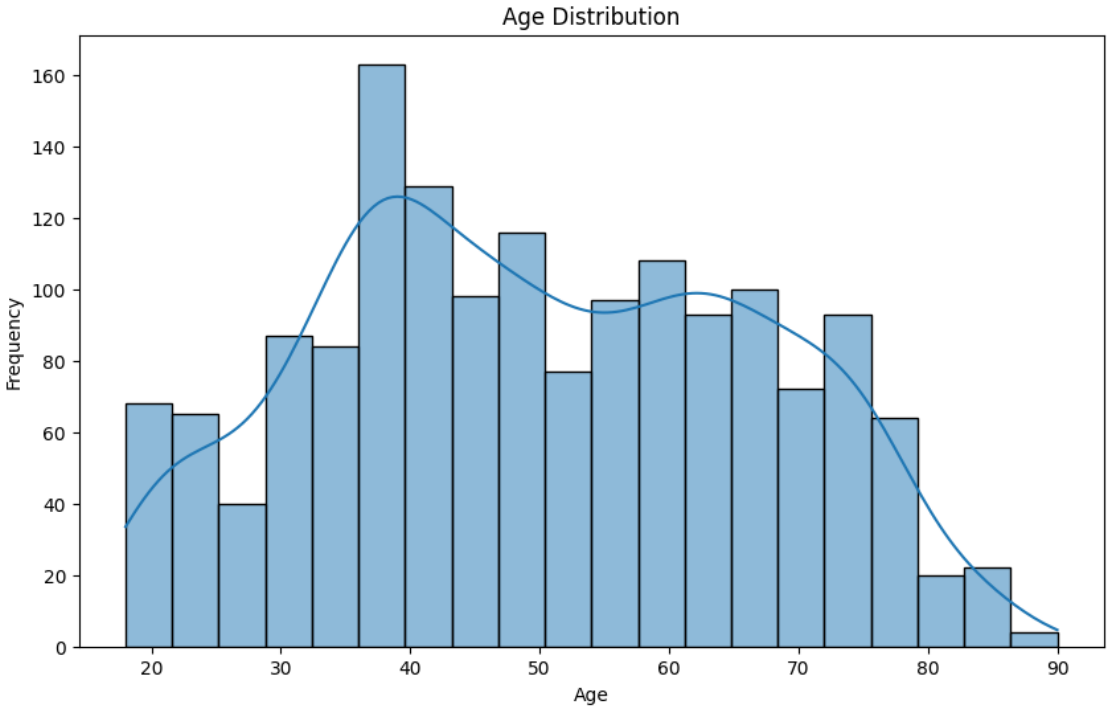
[6]:
```
df['job_type'].isnull().sum()
```

[6]:
```
908
```

Age distribution

[7]:
```
#age  distribution
plt.figure(figsize=(10, 6))
sns.histplot(df['age'], bins=20, kde=True)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

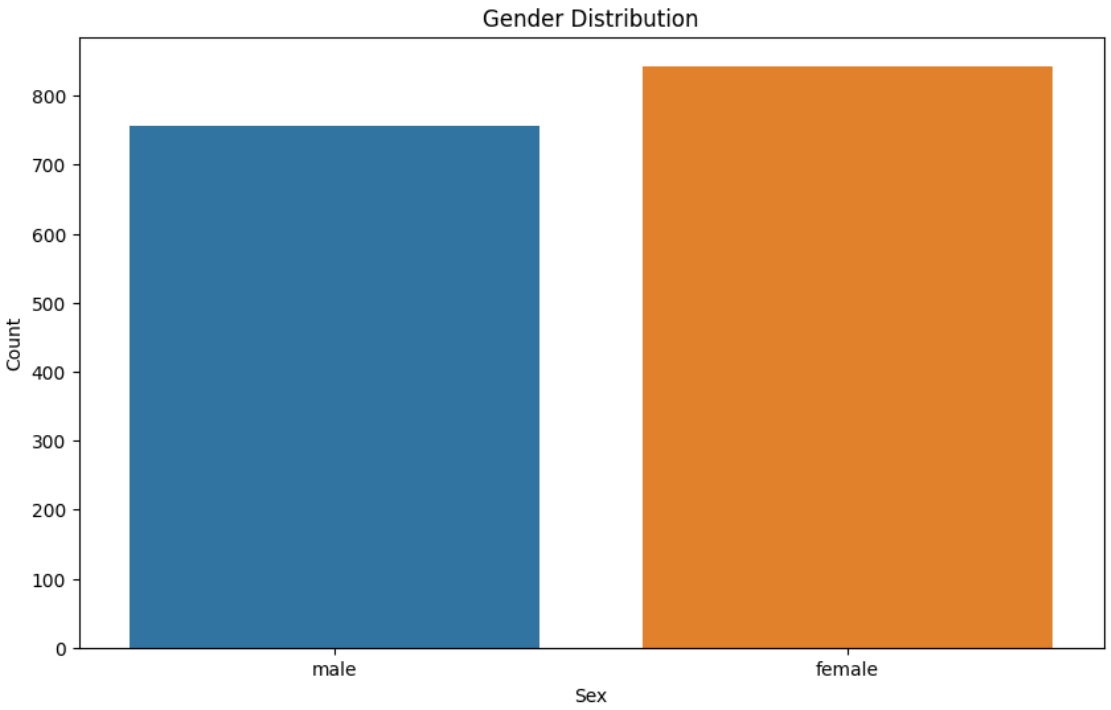
```
C:\Users\ravit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



We can see that the number of people of age 30 to 70 are more in the dataset.

[8]:
```
#gender distribution

plt.figure(figsize=(10,6))
sns.countplot(x='sex',data=df)
plt.title('Gender Distribution')
plt.xlabel('Sex')
plt.ylabel('Count')
```

```
plt.show()
```



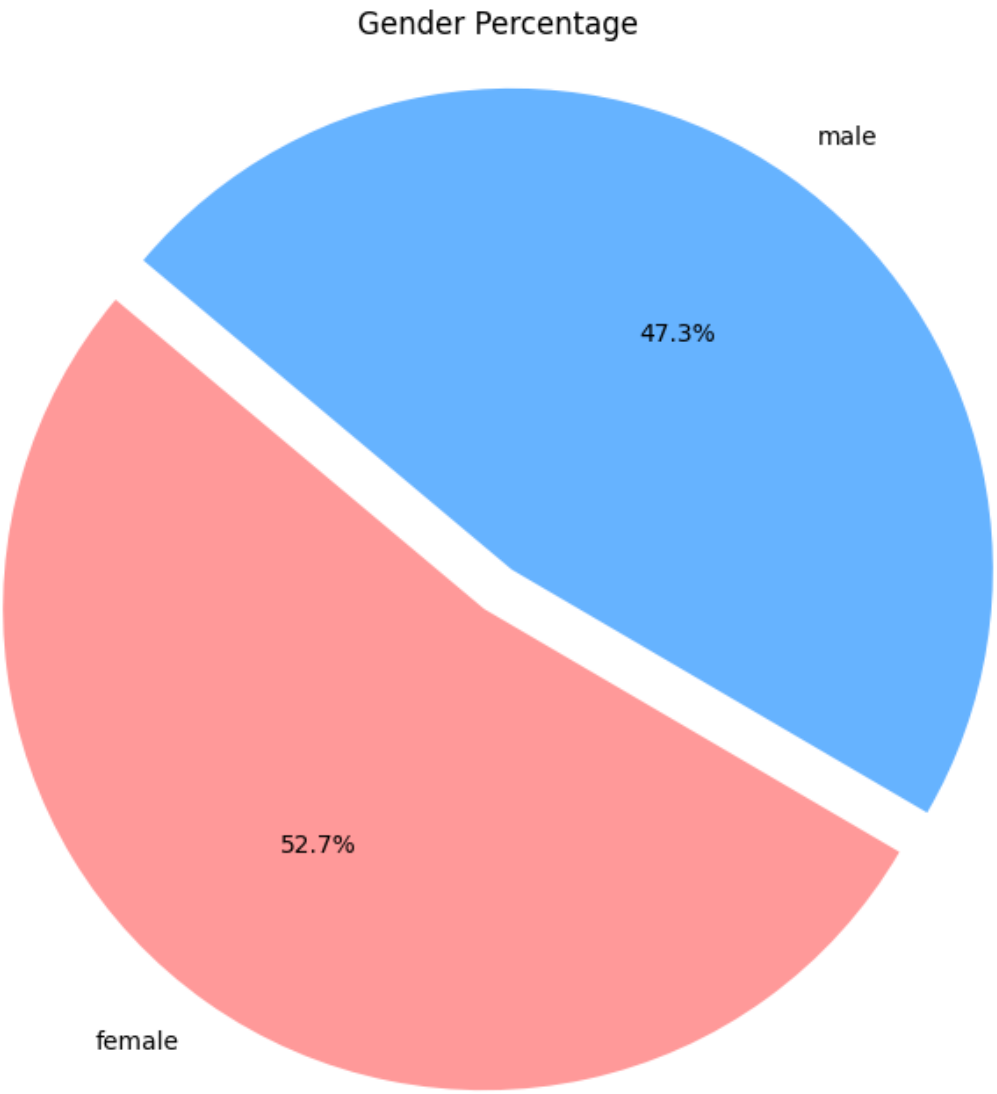
We can see that there are more number of femal population than male.

[9]:
```
#gender distrubution in pie chart

male_percent=df['sex']=='male'
femlae_percentage=df['sex']=='female'
print('Number of male population in dataset',male_percent.sum())
print('Number of female population in dataset',femlae_percentage.sum())
```

```
Number of male population in dataset 757
Number of female population in dataset 843
```

[10]:
```
# Calculate the percentage of each gender
gender_counts = df['sex'].value_counts()
gender_percentages = gender_counts / gender_counts.sum() * 100

# Define the explode parameter
explode = (0.1, 0)

# Plot the pie chart with separation
plt.figure(figsize=(8, 8))
```

```
plt.pie(gender_percentages, labels=gender_percentages.index, autopct='%1.1f%%',␣
 ↪startangle=140, colors=['#ff9999','#66b3ff'], explode=explode)
plt.title('Gender Percentage')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```

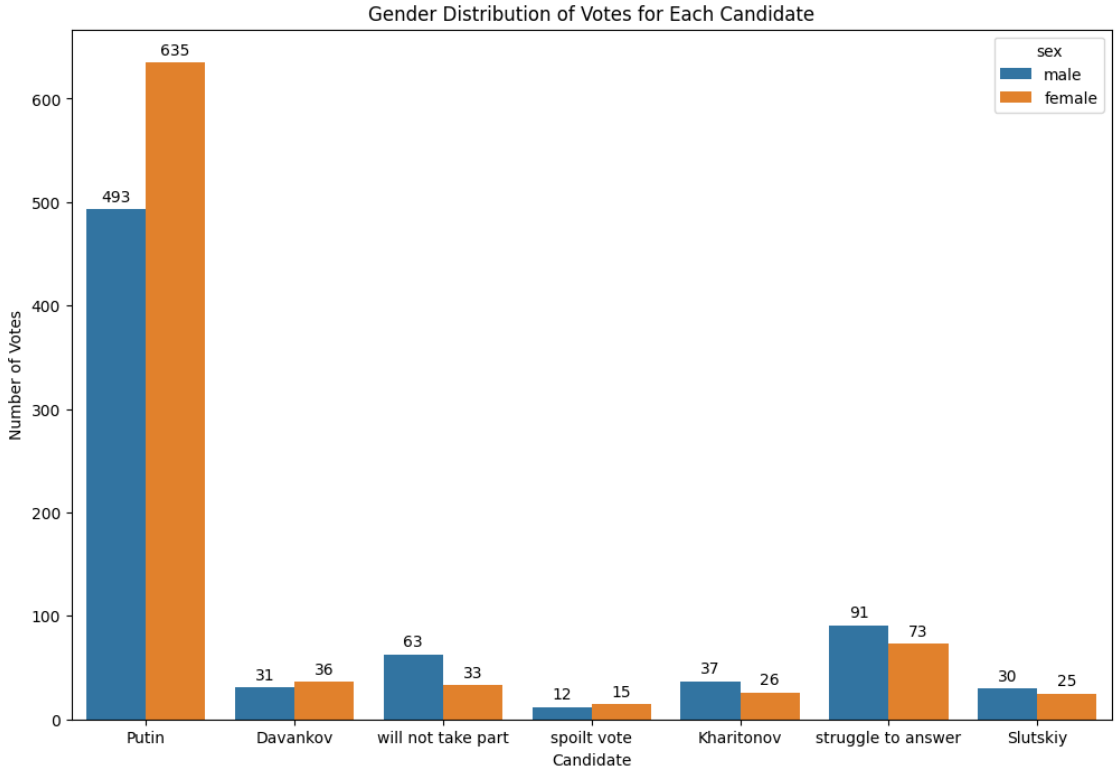


[ ]:

From the above dataset we see that out of total population 52.7% are female and 47.3 are male.

[11]:
```
# Plot Gender vs. Candidate
```

```
plt.figure(figsize=(12, 8))
ax = sns.countplot(x='candidate', hue='sex', data=df)
plt.title('Gender Distribution of Votes for Each Candidate')
plt.xlabel('Candidate')
plt.ylabel('Number of Votes')

# Add counts above the bars
for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.annotate(f'{int(height)}', (p.get_x() + p.get_width() / 2., height),␣
 ↪ha='center', va='baseline', fontsize=10, color='black', xytext=(0, 5),␣
 ↪textcoords='offset points')

plt.show()
```

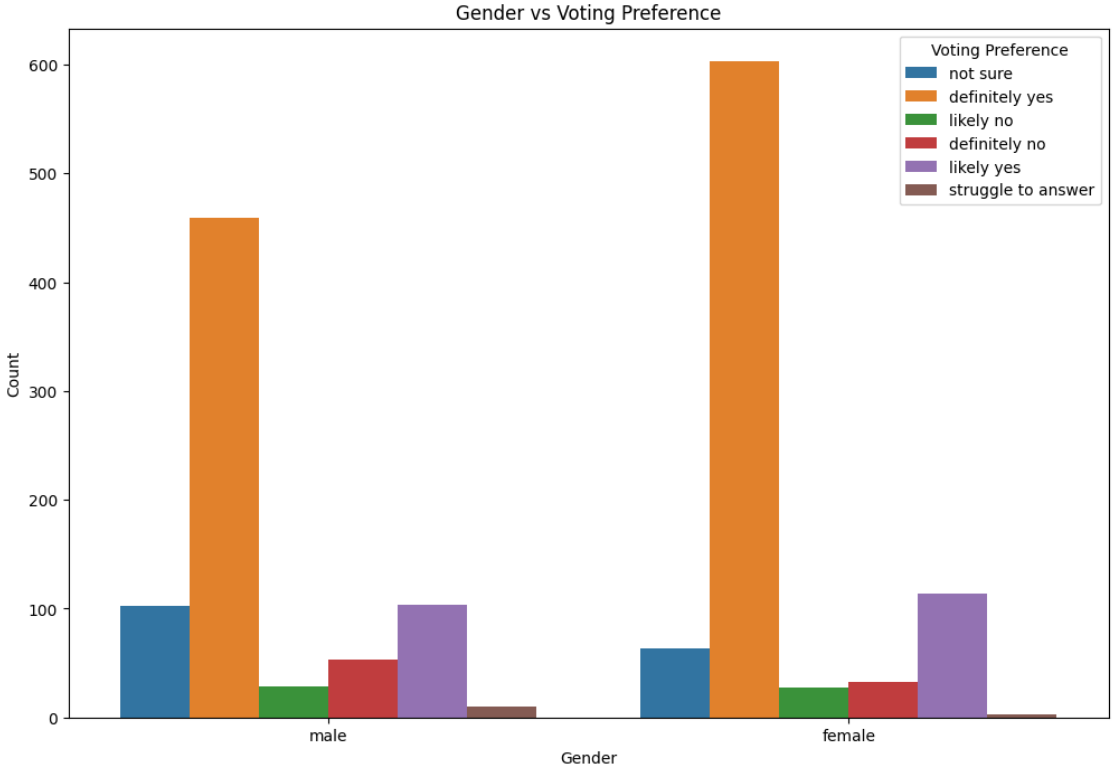


From the plot we can conclude that as there are more female population in dataset[df], more female voters voted to puting and also most of the male voted to putim. But we can also see that More number of men are struggling to answer.

Comparatively male voters voted more for 'kharitonov' than female, 'kharitonov' is secont most voted canditate after putin.

Being less in number male population, 63 male voters are not interested to vote for anybody.This concludes that female's shown more active participation in election.

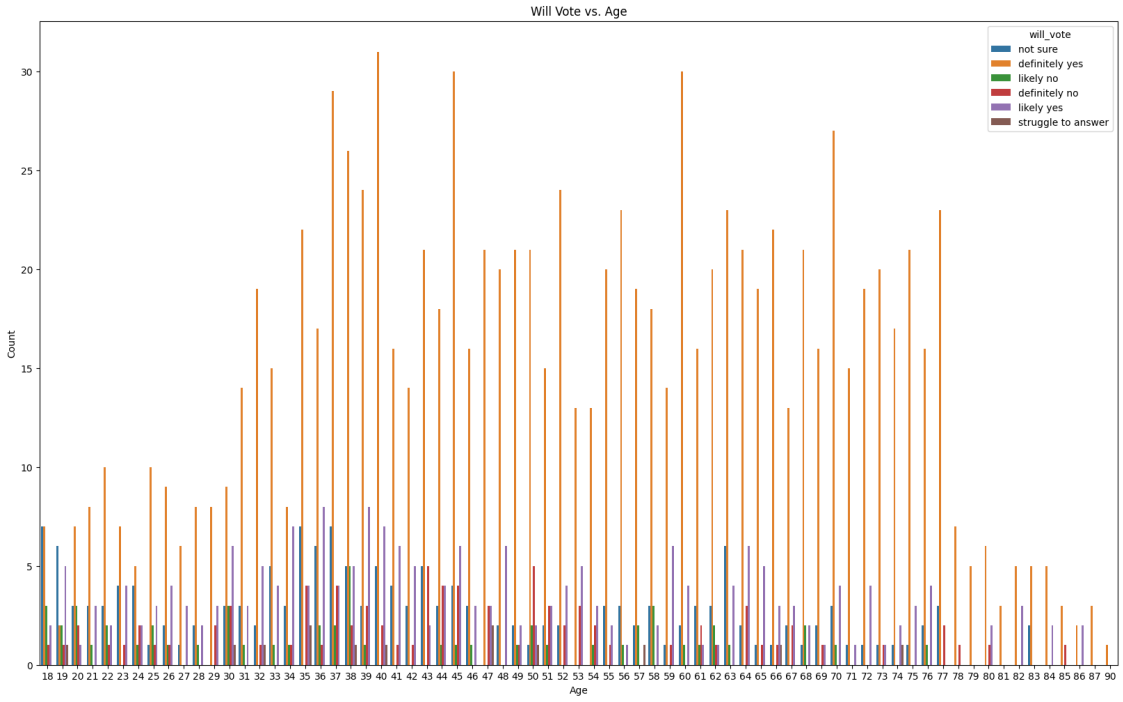
[12]:
```
# 'Gender vs Voting Preference
plt.figure(figsize=(12, 8))
sns.countplot(x='sex', hue='will_vote', data=df)
plt.title('Gender vs Voting Preference')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(title='Voting Preference')
plt.show()
# Add counts above the bars
for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.annotate(f'{int(height)}', (p.get_x() + p.get_width() / 2., height),␣
↪ha='center', va='baseline', fontsize=10, color='black', xytext=(0, 5),␣
↪textcoords='offset points')

plt.show()
```



From above plot come to know that both male and female voters are in good number to cast their vote. But we can also see that more number of male is not sure to cast their vote, also more number

both voters are in category of 'likely yes', which means they may vote or not but not still sure.

[13]:
```
# Will Vote vs. Age
plt.figure(figsize=(20, 12))
sns.countplot(x='age', hue='will_vote', data=df)
plt.title('Will Vote vs. Age')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```

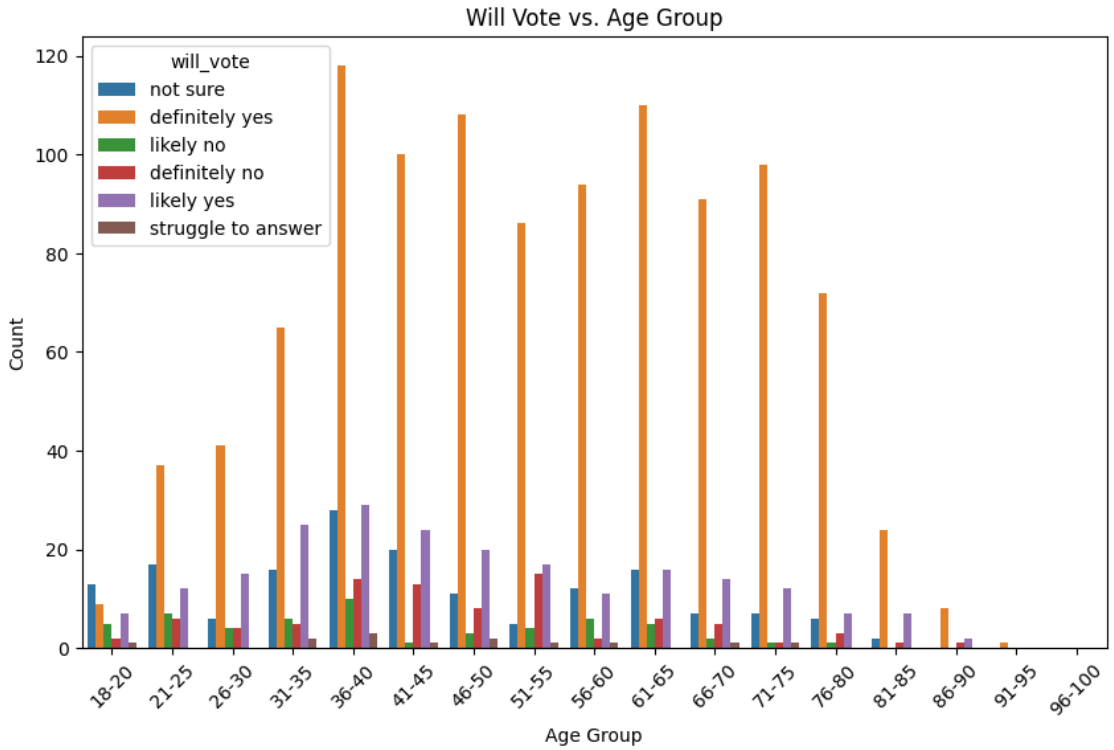


We can see that the people of age 31 to 77 have the highest number of confident voters in the election. We can see thta more number of age 19 are not sure for voting. As we go from age 78 to 90 slowly the number of voters are decreasing. This may be due to most of the people of life span of old age population or it may be also old age population are not interested in survey

[14]:
```
# Plot of Will Vote vs. Age Group

# Create age bins
age_bins = [15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95,␣
 ↪100]
age_labels = ['18-20', '21-25', '26-30', '31-35', '36-40', '41-45', '46-50',␣
 ↪'51-55', '56-60', '61-65', '66-70', '71-75', '76-80', '81-85', '86-90',␣
 ↪'91-95', '96-100']
df['age_group'] = pd.cut(df['age'], bins=age_bins, labels=age_labels,␣
 ↪right=False)
```

```
# Plot Will Vote vs. Age Group
plt.figure(figsize=(10, 6))
sns.countplot(x='age_group', hue='will_vote', data=df)
plt.title('Will Vote vs. Age Group')
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```

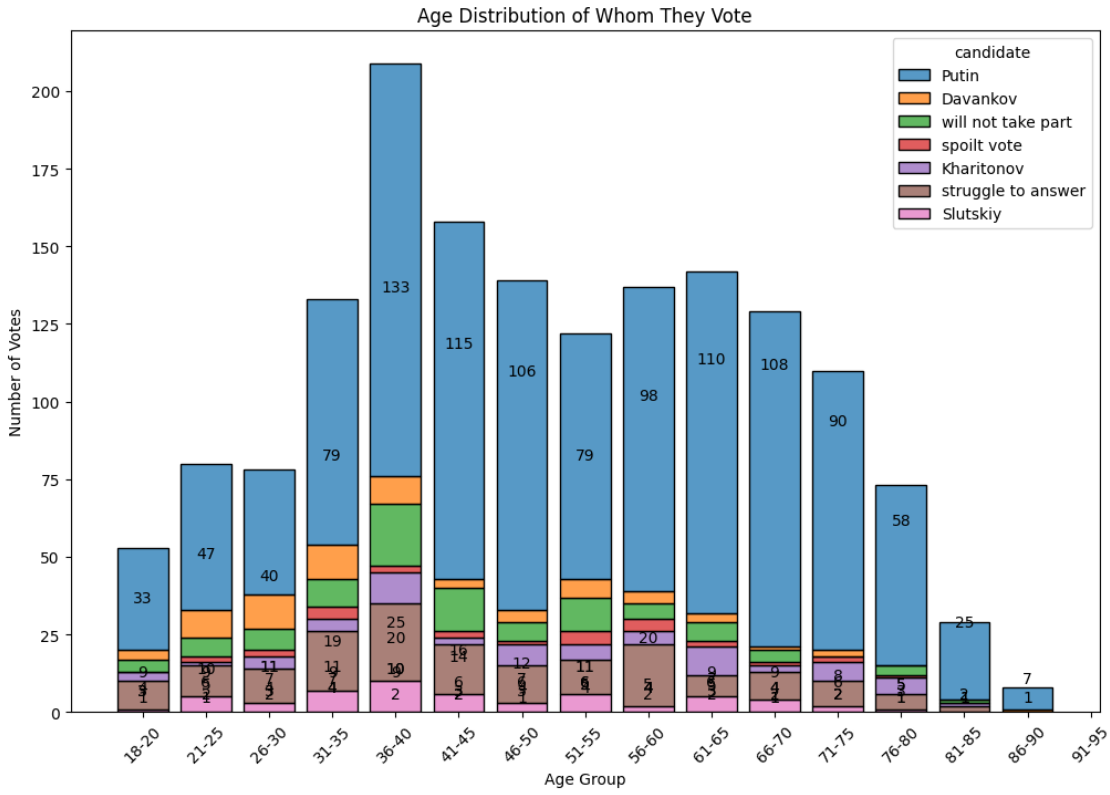
```
C:\Users\ravit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\site-
packages\seaborn\categorical.py:641: FutureWarning: The default of
observed=False is deprecated and will be changed to True in a future version of
pandas. Pass observed=False to retain current behavior or observed=True to adopt
the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
C:\Users\ravit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\site-
packages\seaborn\categorical.py:641: FutureWarning: The default of
observed=False is deprecated and will be changed to True in a future version of
pandas. Pass observed=False to retain current behavior or observed=True to adopt
the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
```



We see that the age group of 36-40 is more interested to cast the vote. We also see that age group of 18-20 is not intereseted to cast their vote. Very less nuber of people in all age group is not interested to cst their vote.

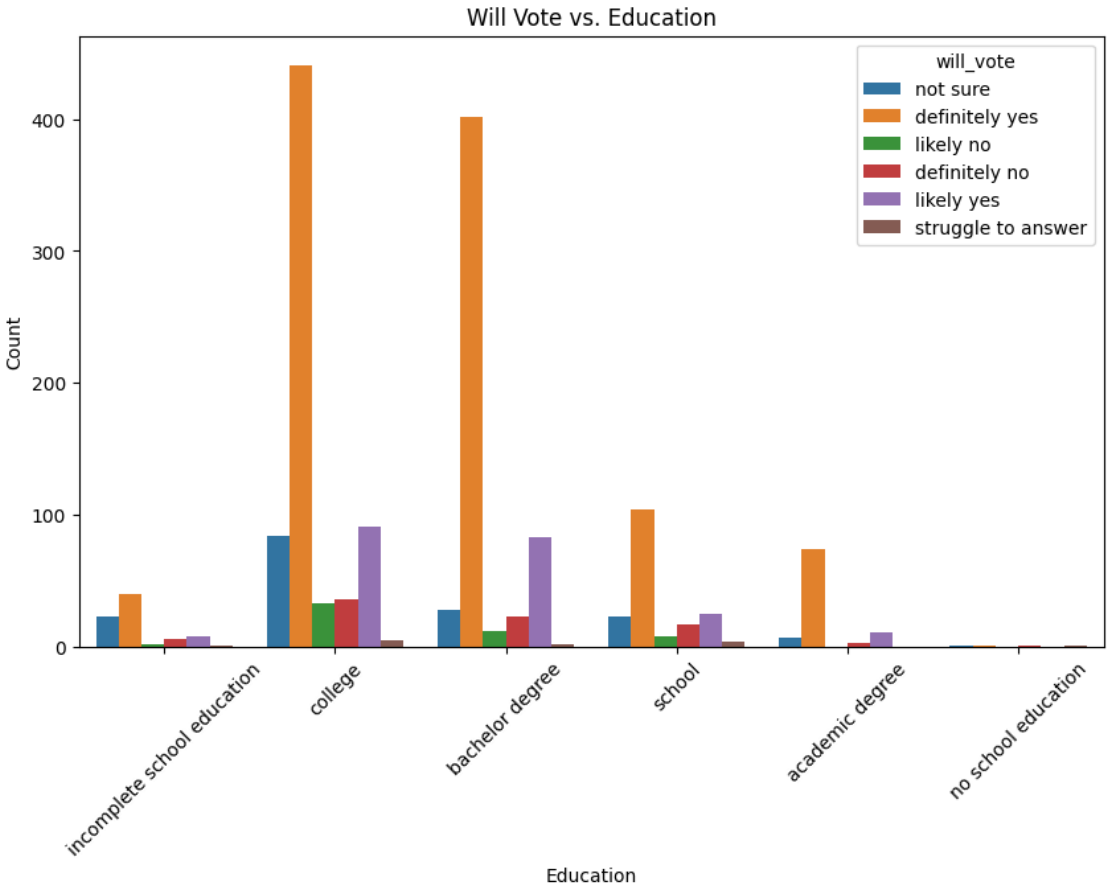
[15]:
```
# 'Age Distribution of Whom They Vote'
# Create age bins
age_bins = [18, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81, 86, 91, 96,␣
 ↪101]
age_labels = ['18-20', '21-25', '26-30', '31-35', '36-40', '41-45', '46-50',␣
 ↪'51-55', '56-60', '61-65', '66-70', '71-75', '76-80', '81-85', '86-90',␣
 ↪'91-95', '96-100']
df['age_group'] = pd.cut(df['age'], bins=age_bins, labels=age_labels,␣
 ↪right=False)

# Plot Age Distribution of Whom They Vote
plt.figure(figsize=(12, 8))
ax = sns.histplot(data=df, x='age_group', hue='candidate', multiple='stack',␣
 ↪shrink=0.8)
plt.title('Age Distribution of Whom They Vote')
plt.xlabel('Age Group')
plt.ylabel('Number of Votes')

# Add counts above the bars
for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.annotate(f'{int(height)}', (p.get_x() + p.get_width() / 2., height),␣
 ↪ha='center', va='baseline', fontsize=10, color='black', xytext=(0, 5),␣
 ↪textcoords='offset points')

plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\ravit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



From the above plot we get that. The population of all age group have voted for Putin. Very few amout of people in all age group is not interested to vote. Age group of 36-40 has more number of voters, compared to all other.

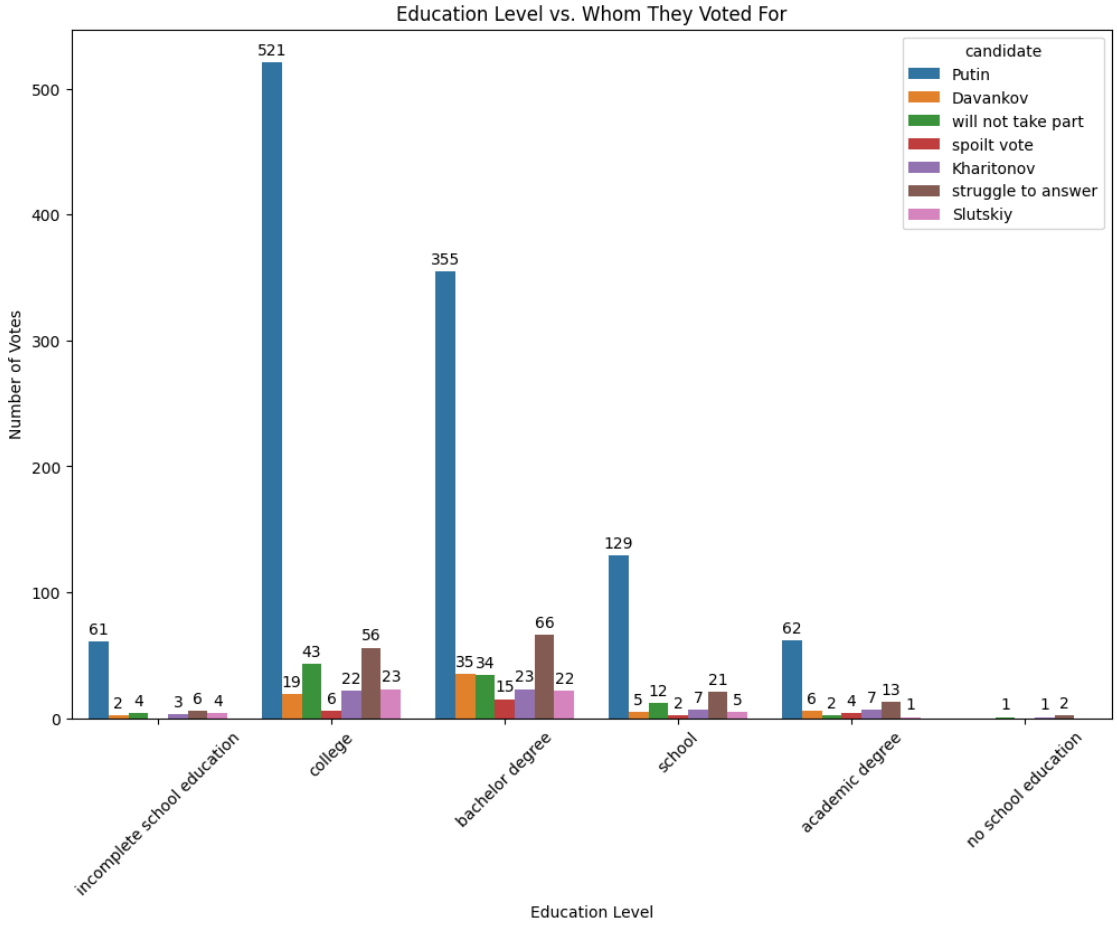
[16]:
```
# plot of ('Will Vote vs. Education'

plt.figure(figsize=(10, 6))
sns.countplot(x='education', hue='will_vote', data=df)
plt.title('Will Vote vs. Education')
plt.xlabel('Education')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



From the above plot we can see that People with college have more participation in voting, People with no school education have very less participation [it may be because that there are very less number of people who has no school education or they dont want to participate in election]. Some part of population with collage show high resistance for voting.

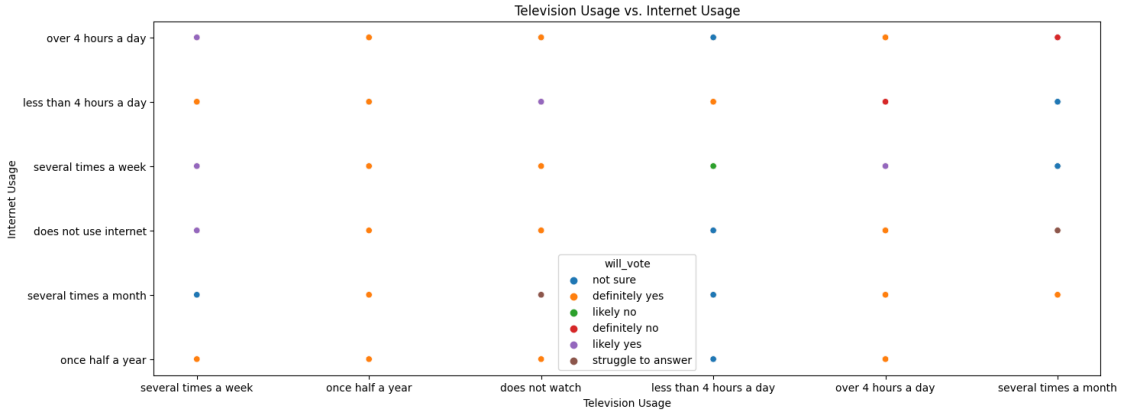
[17]:
```
# Plot Education vs. Candidate


plt.figure(figsize=(12, 8))
ax = sns.countplot(x='education', hue='candidate', data=df)
plt.title('Education Level vs. Whom They Voted For')
plt.xlabel('Education Level')
plt.ylabel('Number of Votes')
plt.xticks(rotation=45)

# Add counts above the bars
for p in ax.patches:
    height = p.get_height()
    if height > 0:
```

```
        ax.annotate(f'{int(height)}', (p.get_x() + p.get_width() / 2., height),␣
↪ha='center', va='baseline', fontsize=10, color='black', xytext=(0, 5),␣
↪textcoords='offset points')

plt.show()
```



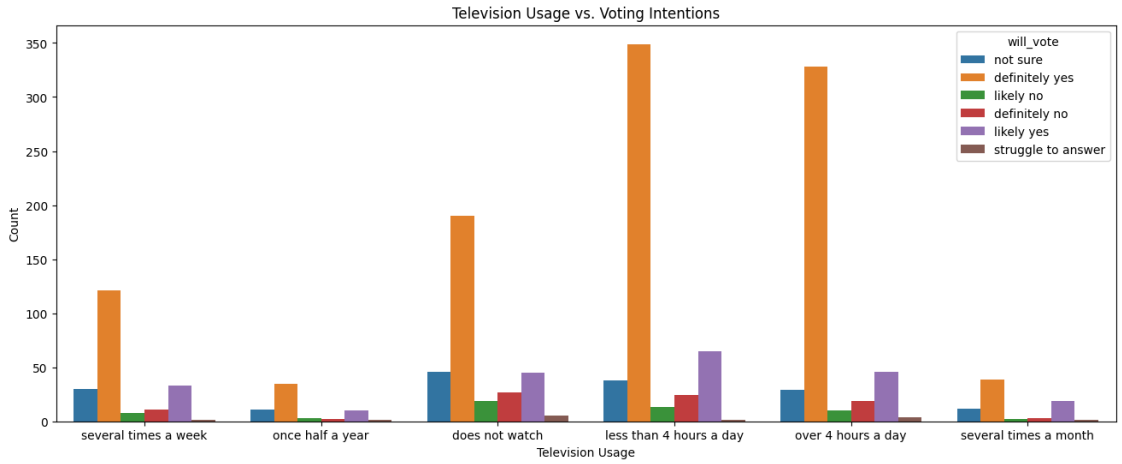
From above plot we witness that: People with college voted for putin in highest number, People with bachelor degree stands secont to vote for putin. people with no school education are very less but comparatively they voted more for 'kharatinov'

[18]:
```
plt.figure(figsize=(16, 6))
sns.scatterplot(x='television_usage', y='internet_usage', hue='will_vote',␣
↪data=df)
plt.title('Television Usage vs. Internet Usage')
plt.xlabel('Television Usage')
plt.ylabel('Internet Usage')
plt.show()
```



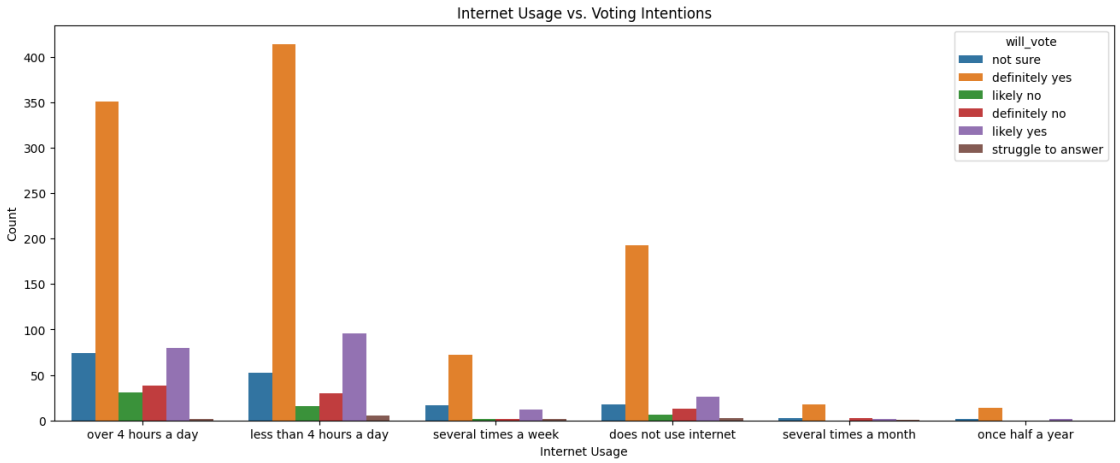
We see that almost all category with tv an internet consumption show strong support for voting. But we also notice that people who watch tv over 4+ hrs and internet usage less than 4 hrs has shown strong resistance for voting. similarly we also notice that people who watch tv several times a month and over 4+ hrs of internet usage has shown strong resistance for voting.

[19]:
```
# Media Usage vs. Voting Intentions
plt.figure(figsize=(16, 6))
sns.countplot(x='television_usage', hue='will_vote', data=df)
plt.title('Television Usage vs. Voting Intentions')
plt.xlabel('Television Usage')
plt.ylabel('Count')
plt.show()
```



From the above plot we see that, People who watch tv less and more than 4 hrs a day has highest voting participation. People who watch tv several times a month and once half a year show very less voting participation.

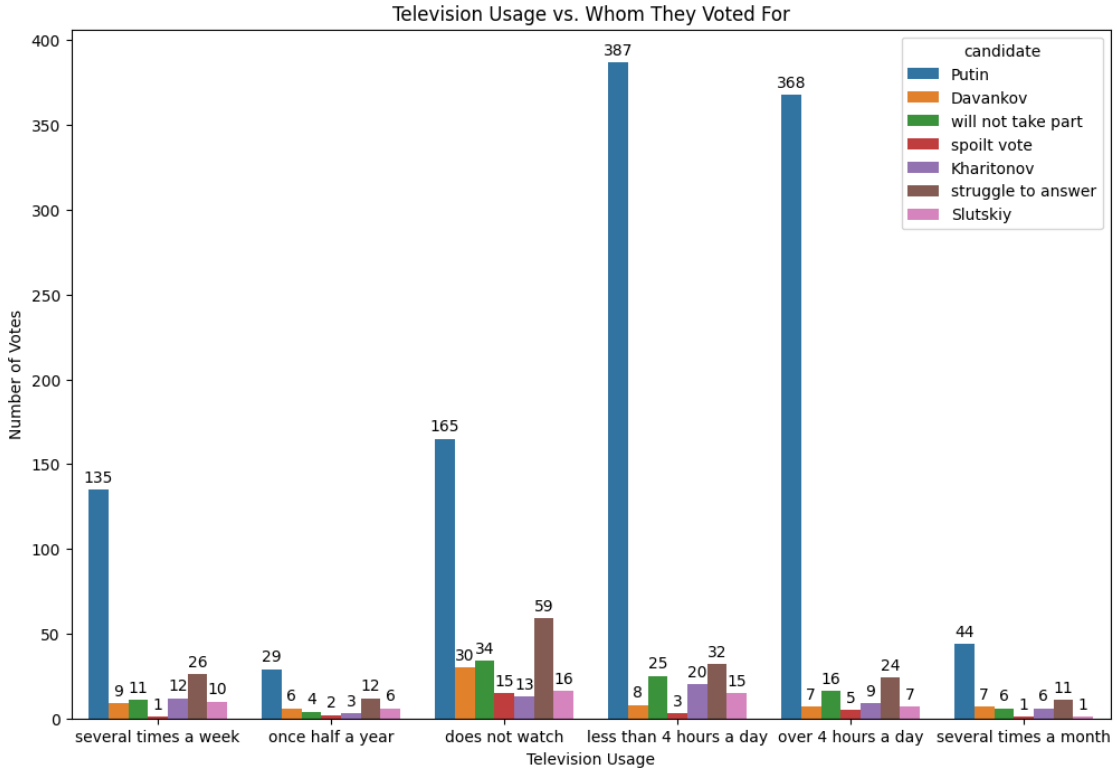
[20]:
```
# Internet Usage vs. Voting Intentions'
plt.figure(figsize=(16, 6))
sns.countplot(x='internet_usage', hue='will_vote', data=df)
plt.title('Internet Usage vs. Voting Intentions')
plt.xlabel('Internet Usage')
plt.ylabel('Count')
plt.show()
```

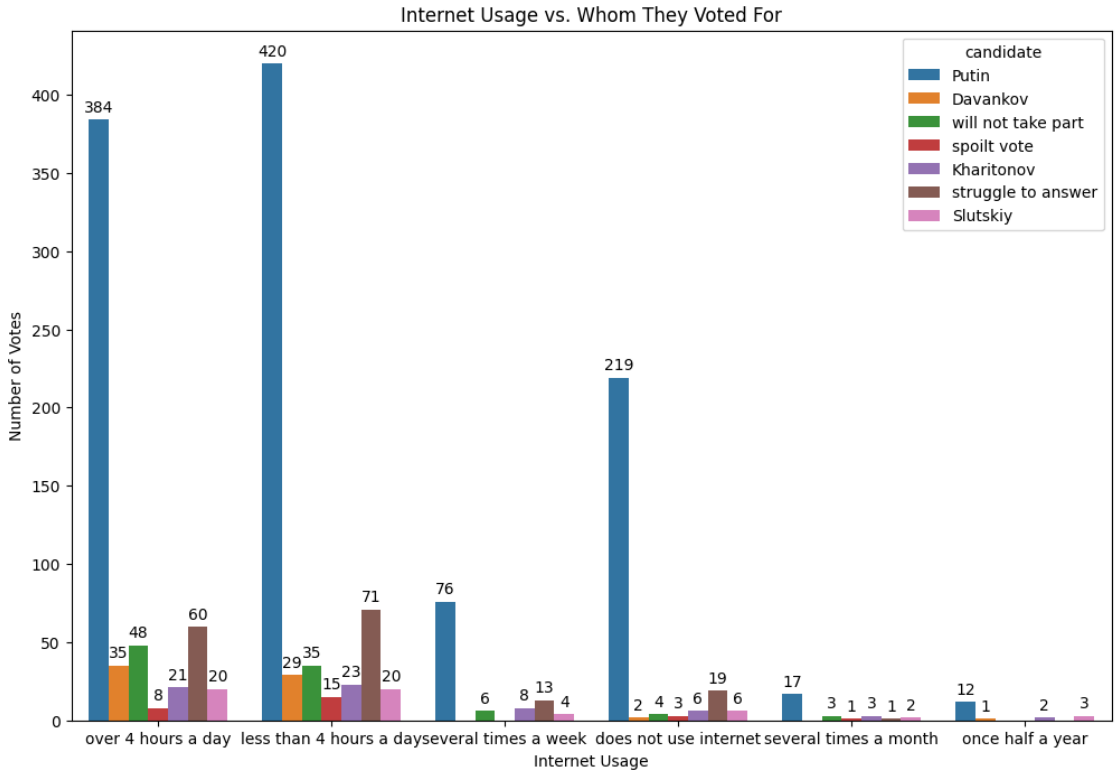


From the above plot we see that, People who use internet less and more than 4 hrs a day has highest voting participation. People who use internet several times a month and once half a year show very less voting participation.

[21]:
```
# Plot Television Usage vs. Whom They Voted For

plt.figure(figsize=(12, 8))
ax = sns.countplot(x='television_usage', hue='candidate', data=df)
plt.title('Television Usage vs. Whom They Voted For')
plt.xlabel('Television Usage')
plt.ylabel('Number of Votes')

# Add counts above the bars
for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.annotate(f'{int(height)}', (p.get_x() + p.get_width() / 2., height),␣
 ↪ha='center', va='baseline', fontsize=10, color='black', xytext=(0, 5),␣
 ↪textcoords='offset points')

plt.show()

# Plot Internet Usage vs. Whom They Voted For
```

```
plt.figure(figsize=(12, 8))
ax = sns.countplot(x='internet_usage', hue='candidate', data=df)
plt.title('Internet Usage vs. Whom They Voted For')
plt.xlabel('Internet Usage')
plt.ylabel('Number of Votes')

# Add counts above the bars
for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.annotate(f'{int(height)}', (p.get_x() + p.get_width() / 2., height),␣
 ↪ha='center', va='baseline', fontsize=10, color='black', xytext=(0, 5),␣
 ↪textcoords='offset points')

plt.show()
```





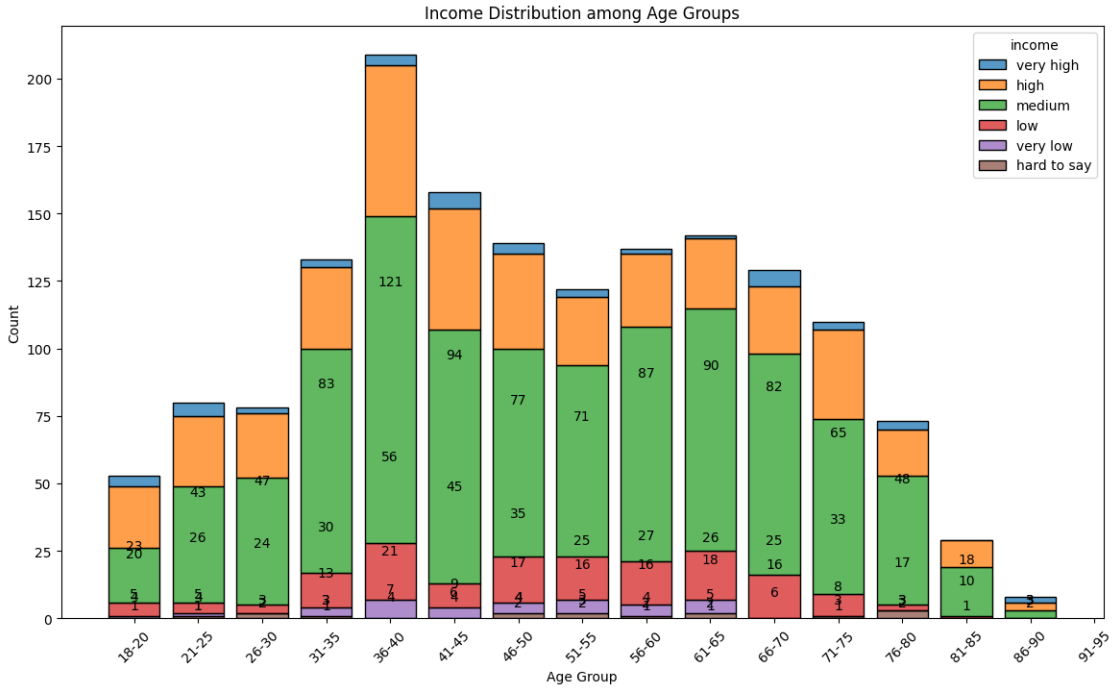
TV: From the above plot we see that, People who use T.v less and more than 4 hrs a day has voted putin. people who dont watch tv also casted good amout of vote for putin.

Internet usage: People who use Internet less and more than 4 hrs a day has voted putin. people who dont use internet also casted good amout of vote for putin.

[22]:
```
# Create age bins
age_bins = [18, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81, 86, 91, 96,␣
 ↪101]
age_labels = ['18-20', '21-25', '26-30', '31-35', '36-40', '41-45', '46-50',␣
 ↪'51-55', '56-60', '61-65', '66-70', '71-75', '76-80', '81-85', '86-90',␣
 ↪'91-95', '96-100']
df['age_group'] = pd.cut(df['age'], bins=age_bins, labels=age_labels,␣
 ↪right=False)

# Plot histogram of age group earning what income
plt.figure(figsize=(14, 8))
ax = sns.histplot(data=df, x='age_group', hue='income', multiple='stack',␣
 ↪shrink=0.8)
plt.title('Income Distribution among Age Groups')
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.xticks(rotation=45)
```

```
# Add counts above the bars
for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.annotate(f'{int(height)}', (p.get_x() + p.get_width() / 2., height),␣
 ↪ha='center', va='baseline', fontsize=10, color='black', xytext=(0, 5),␣
 ↪textcoords='offset points')

plt.show()
```

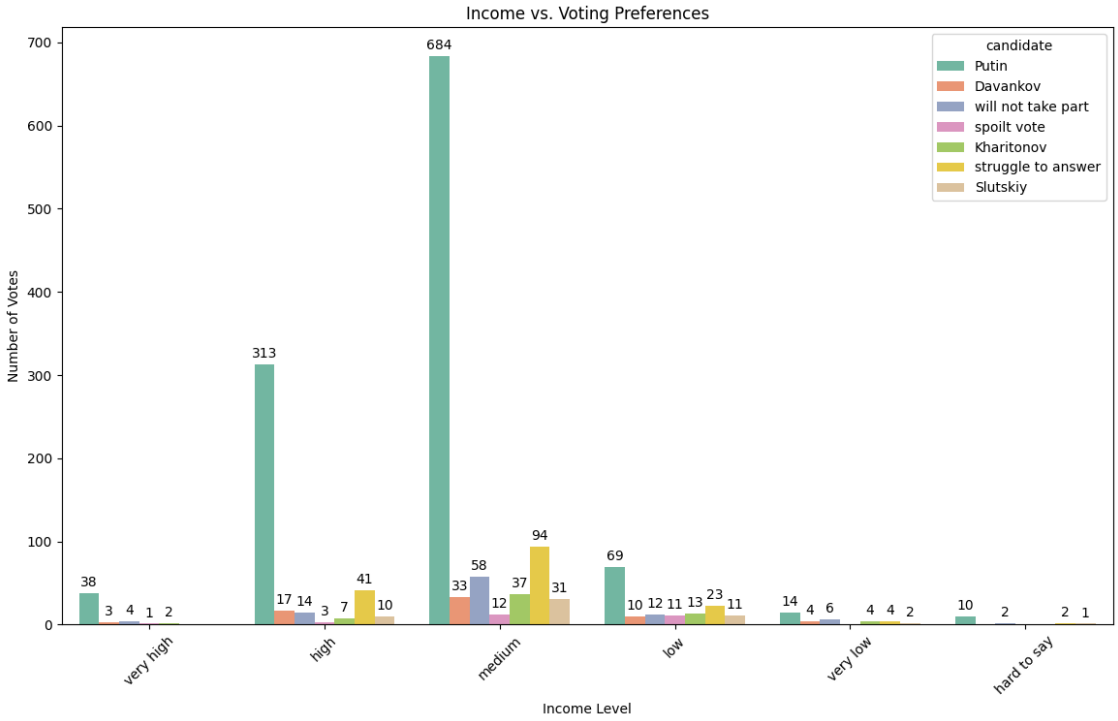
```
C:\Users\ravit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



We see that majority of all the age group has the medium income range. Age group of 36-40 has good number of high income people.

[23]:
```
# Plot Income vs. Voting Preferences
plt.figure(figsize=(14, 8))
ax = sns.countplot(x='income', hue='candidate', data=df, palette='Set2')
plt.title('Income vs. Voting Preferences')
```

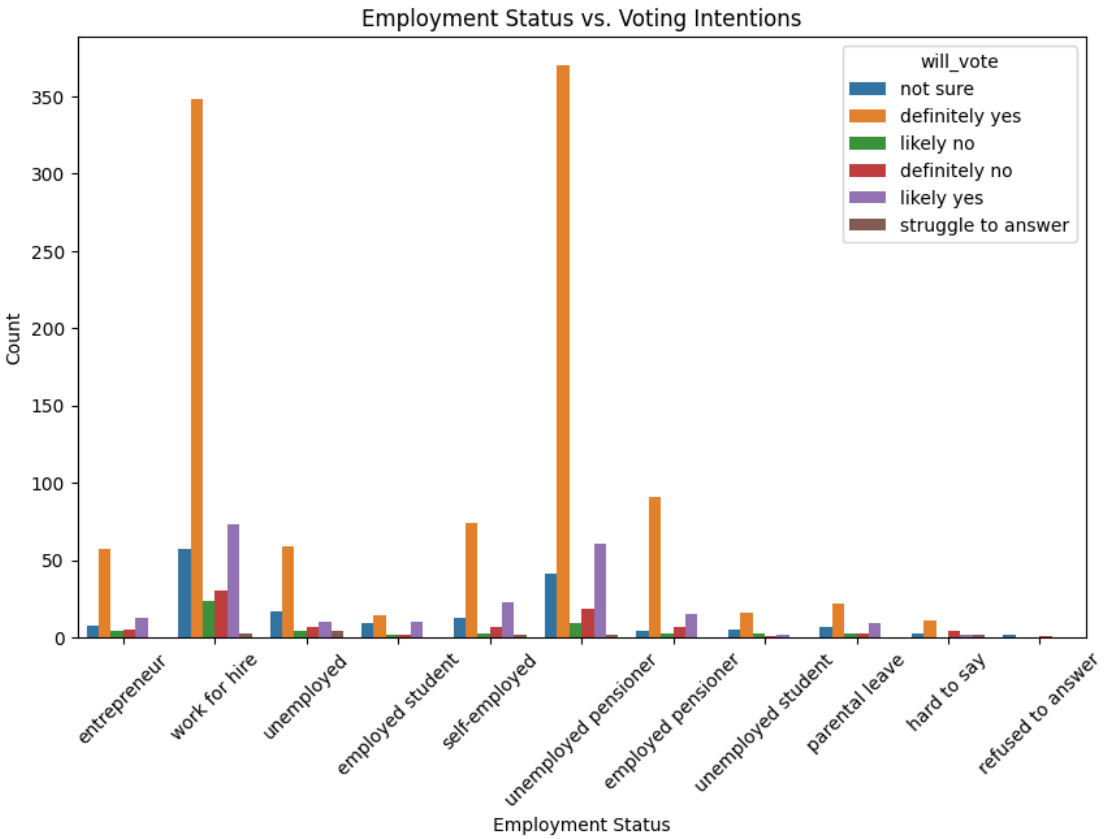
```
plt.xlabel('Income Level')
plt.ylabel('Number of Votes')
plt.xticks(rotation=45)

# Add counts above the bars
for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.annotate(f'{int(height)}', (p.get_x() + p.get_width() / 2., height),␣
 ↪ha='center', va='baseline', fontsize=10, color='black', xytext=(0, 5),␣
 ↪textcoords='offset points')

plt.show()
```



people with medium group has voted putin the most, and second most group voted for putin is high income group. People wiht very low incom group voted putin less.

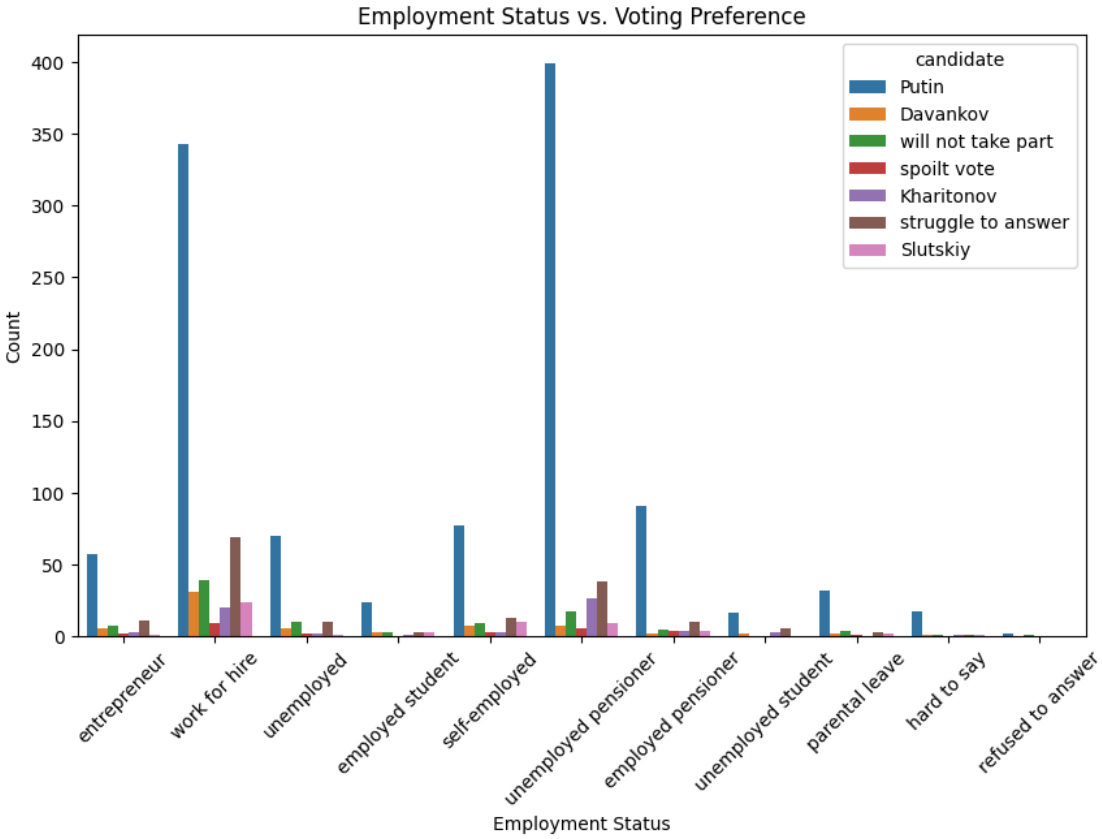
[24]:
```
#  Employment Status vs. Voting Intentions
plt.figure(figsize=(10, 6))
sns.countplot(x='employment', hue='will_vote', data=df)
plt.title('Employment Status vs. Voting Intentions')
plt.xlabel('Employment Status')
plt.ylabel('Count')
plt.xticks(rotation=45)
```

```
plt.show()
```



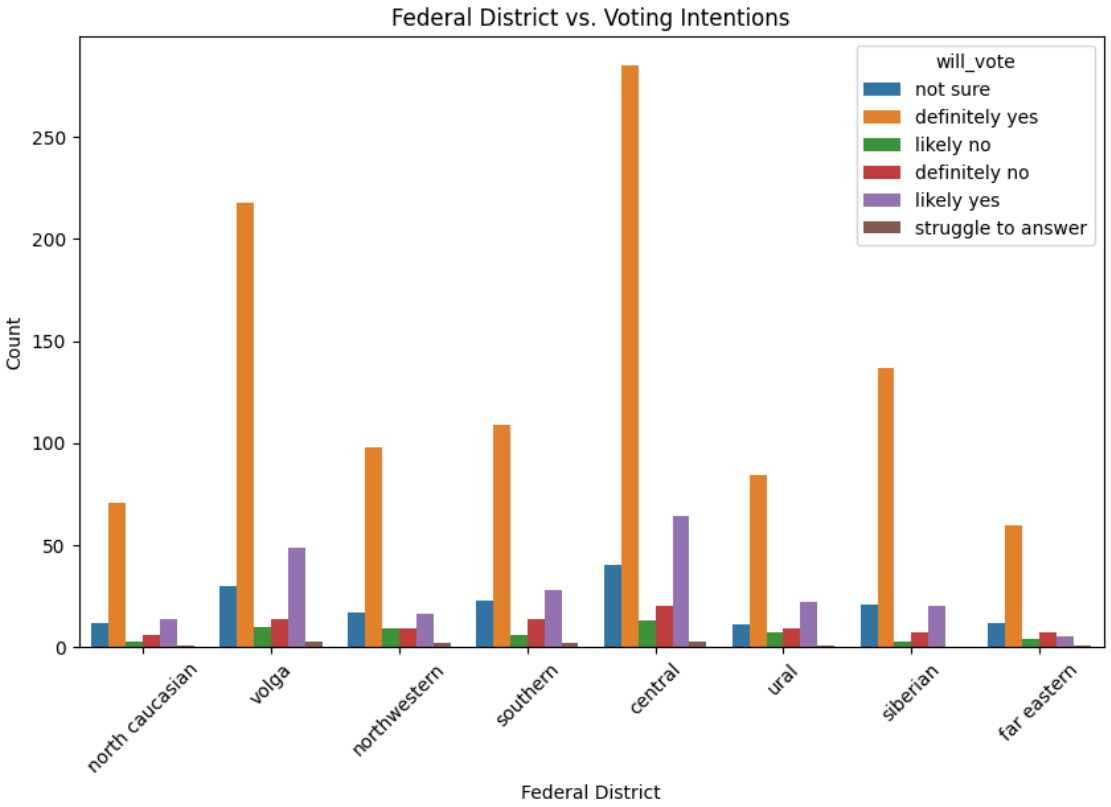
we see that most of the people who work for hire and self-employed has highest participation in election.

[25]:
```
# Employment Status vs. Voting Preference
plt.figure(figsize=(10, 6))
sns.countplot(x='employment', hue='candidate', data=df)
plt.title('Employment Status vs. Voting Preference')
plt.xlabel('Employment Status')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



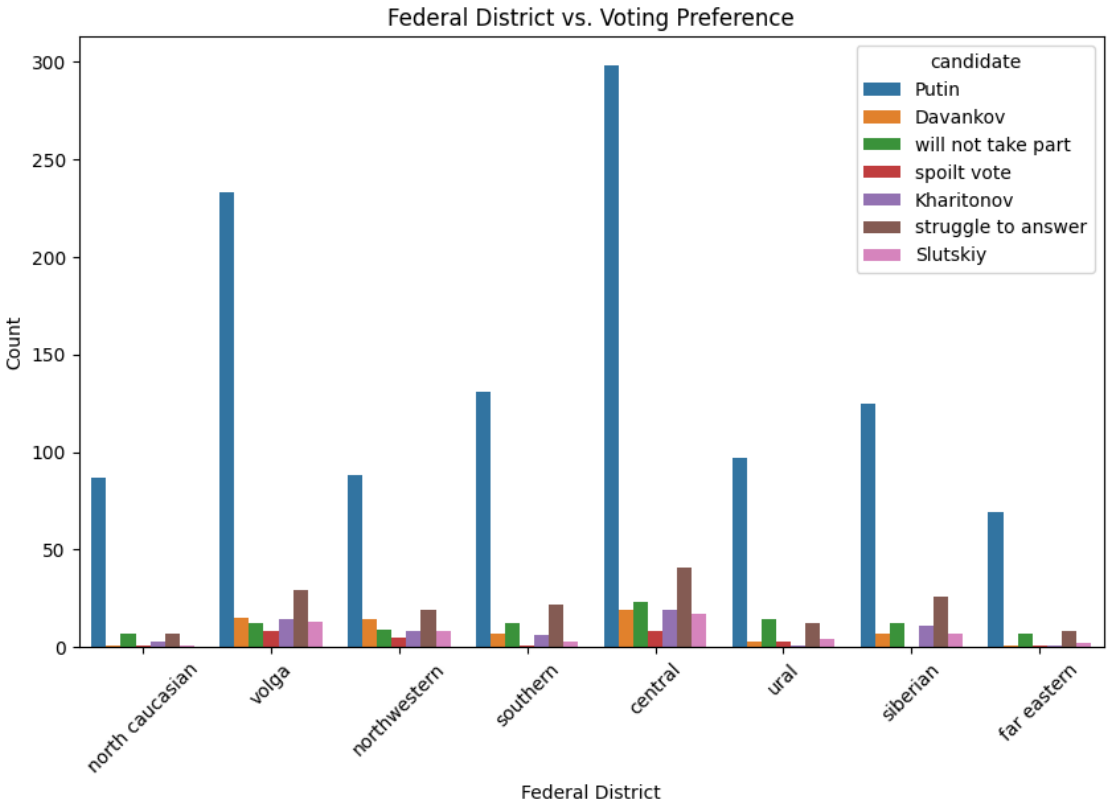
People who 'work for hire' and 'self employed' voted for putin in the highest number.

[26]:
```
#  Federal District vs. Voting Intentions
plt.figure(figsize=(10, 6))
sns.countplot(x='federal_district', hue='will_vote', data=df)
plt.title('Federal District vs. Voting Intentions')
plt.xlabel('Federal District')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



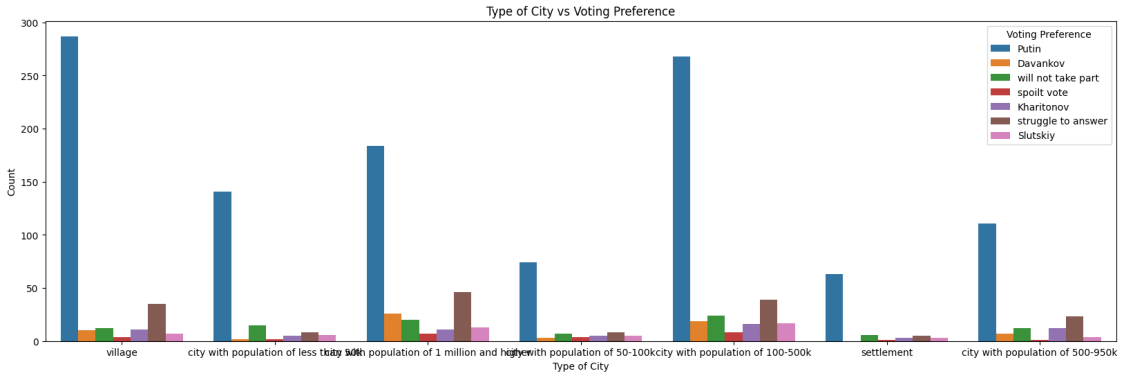
We see that majority of the people amoung all federal_district are actively participating in voting, central region has more voting participatin, and in sencon place volga region has secont most vote turnover. similary we also see that central and volga also has highest number of resistance to voting.

[27]:
```
# Plot 9: Federal District vs. Voting Candidate
plt.figure(figsize=(10, 6))
sns.countplot(x='federal_district', hue='candidate', data=df)
plt.title('Federal District vs. Voting Preference')
plt.xlabel('Federal District')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



We see that majority of the people amoung all federal_district are voting for putin. central region has highest number of voters for putin, and sencod higesht vote for putin is form volga region. similary we also see that central and volga also has highest number of resistance to voting to any other candidate.

[28]:
```
# Assuming 'will_vote' is the voting preference column
plt.figure(figsize=(20, 6))
sns.countplot(x='type_of_city', hue='candidate', data=df)
plt.title('Type of City vs Voting Preference')
plt.xlabel('Type of City')
plt.ylabel('Count')
plt.legend(title='Voting Preference')
plt.show()
```



Puting got hight votes in village, secont highest votes from putin is from city with population of 100-500k. This may be because lots of population live in village and they have support for putin for his work. We see that in city of population of 1M+, some people support to davankov. We also see that city of population of 1M+ has higest number of struggle to answer.

[29]:
```
df.columns
```

[29]:
```
Index(['id', 'sex', 'age', 'federal_district', 'type_of_city',
       'knows_election_date', 'will_vote', 'candidate', 'television_usage',
       'internet_usage', 'education', 'income', 'employment', 'job_type',
       'company_type', 'weight1', 'age_group'],
      dtype='object')
```

- Chi-square: a statistical test used to determine whether there is a significant association or independence between two categorical variables.
- Relationship to Data Chi-Square Values: Quantify the difference between observed and expected frequencies in categorical data. P-Values: Indicate the probability of observing the data if the null hypothesis is true. A low p-value suggests that the observed data is unlikely under the null hypothesis, leading to its rejection.

[31]:
```
import pandas as pd
from scipy.stats import chi2_contingency

# Assuming df_updates is your DataFrame containing the data

# Create a list to store the results
chi2_results_list = []

# List of categorical features
categorical_features = ['id', 'sex', 'age', 'federal_district', 'type_of_city',
       'knows_election_date', 'will_vote', 'candidate', 'television_usage',
       'internet_usage', 'education', 'income', 'employment', 'job_type',
       'company_type', 'weight1', 'age_group']
```

```
def contingency_table(feature):
    p = pd.crosstab(df['candidate'],df[feature])
    return(p)
feature = []
pval = []
chi2_result = []

for i in categorical_features:
    feature.append(i)
    result = chi2_contingency(contingency_table(i))
    pval.append(round(float(result[1]),6))

    if float(result[1]) < 0.05:
        chi2_result.append("Significant")
    else:
        chi2_result.append("Insignificant")
```

[32]:
```
chisquare = pd.DataFrame(data={'PValue':pval,'Result':
 ↪chi2_result},index=feature)
chisquare
```

[32]:

	PValue	Result
id	0.480811	Insignificant
sex	0.000104	Significant
age	0.097890	Insignificant
federal_district	0.033126	Significant
type_of_city	0.000123	Significant
knows_election_date	0.001592	Significant
will_vote	0.000000	Significant
candidate	0.000000	Significant
television_usage	0.000000	Significant
internet_usage	0.000004	Significant
education	0.000501	Significant
income	0.000000	Significant
employment	0.001706	Significant
job_type	0.451317	Insignificant
company_type	0.003245	Significant
weight1	0.000000	Significant
age_group	0.000090	Significant

We can see the Significant and Insignificant features of the dataset by using chi-square method.

As most of the data is object/categorical the correlation_matrix cannot be used

1 Conclusion

Sure, let's go through the detailed conclusions for each of these plots based on your dataset:

1. **Type of City vs. Voting Preference**:
   - **Conclusion**: Urban residents may show a preference for more progressive candidates, while rural residents might lean towards conservative candidates. This difference could be influenced by varying socio-economic conditions and access to information.
2. **Federal District vs. Voting Intentions**:
   - **Conclusion**: Certain federal districts exhibit higher voting intentions, possibly due to effective local campaigns or pressing regional issues. Districts with lower intentions might need more voter engagement efforts.
3. **Federal District vs. Voting Preference**:
   - **Conclusion**: Voting preferences can vary significantly across federal districts, reflecting regional political climates and local candidate popularity. This highlights the importance of tailored political strategies.
4. **Employment Status vs. Voting Intentions**:
   - **Conclusion**: Employed individuals tend to have higher voting intentions compared to unemployed individuals. Employment might provide a sense of stability and civic responsibility, encouraging participation in elections.
5. **Employment Status vs. Voting Preference**:
   - **Conclusion**: Different employment statuses might influence voting preferences. For example, self-employed individuals might prefer candidates who support small businesses, while salaried employees might favor those advocating for labor rights.
6. **Income vs. Voting Preferences**:
   - **Conclusion**: Higher income groups might prefer candidates who promise economic stability and tax benefits, while lower income groups might lean towards candidates advocating for social welfare and income redistribution.
7. **Internet Usage vs. Whom They Voted For**:
   - **Conclusion**: High internet users might vote for candidates with strong online presence and digital campaigns. This indicates the growing influence of social media and online platforms in shaping political opinions.
8. **Television Usage vs. Whom They Voted For**:
   - **Conclusion**: Frequent television viewers might be influenced by political advertisements and news coverage, leading them to vote for candidates who invest heavily in TV campaigns.
9. **Internet Usage vs. Voting Intentions**:
   - **Conclusion**: Higher internet usage correlates with higher voting intentions, possibly due to better access to information and political discourse online, which can motivate individuals to vote.
10. **Education Level vs. Whom They Voted For**:
    - **Conclusion**: Higher education levels might correlate with a preference for candidates who emphasize policies on education, research, and innovation. Educated voters might also be more critical and informed in their choices.
11. **Will Vote vs. Education**:
    - **Conclusion**: Individuals with higher education levels are more likely to vote. Education increases political awareness and the perceived importance of participating in elections.
12. **Age Distribution of Whom They Vote For**:
    - **Conclusion**: Younger voters might prefer candidates who address issues like climate change and job creation, while older voters might prioritize healthcare and pension policies. This age-based preference highlights generational differences in political priorities.
13. **Will Vote vs. Age Group**:

- **Conclusion**: Older age groups show higher voting intentions compared to younger age groups. This trend suggests that political engagement increases with age, possibly due to greater life experience and stability.

14. **Gender vs. Voting Preference**:
    - **Conclusion**: There might be gender-specific voting preferences, with females possibly favoring candidates who advocate for gender equality and social issues, while males might lean towards candidates focusing on economic policies.

These conclusions are based on typical trends observed in such datasets. If you have specific insights or additional details from your plots, feel free to share them, and I can provide more tailored conclusions!

PHYSICS WALLAH

INTERNSHIP OFFER LETTER

Dear Raviteja Kulkanri,

Following your application, we are pleased to inform you that you have been considered for an internship with **PHYSICS WALLAH PVT. LTD.** As a result, you will be contributing to our project, Predict Bank Credit Risk using South German Credit Data from **15th September 2023** . As a part of your internship, you will be proactively contributing to your selected project, besides product development & PoCs. In addition, you will be required to complete performance & learning goals for your current project with us. We hope that your association with the company will be successful and rewarding.

Regards

Mr. Alakh Pandey
(Founder)

I accept the offer with the company on the terms and conditions set out in this letter.

Data Scientist Internship - PWskills

2024

Predict Bank Credit Risk using South German Credit Data

-RAVITEJA KULKARNI

Contents

1. Introduction

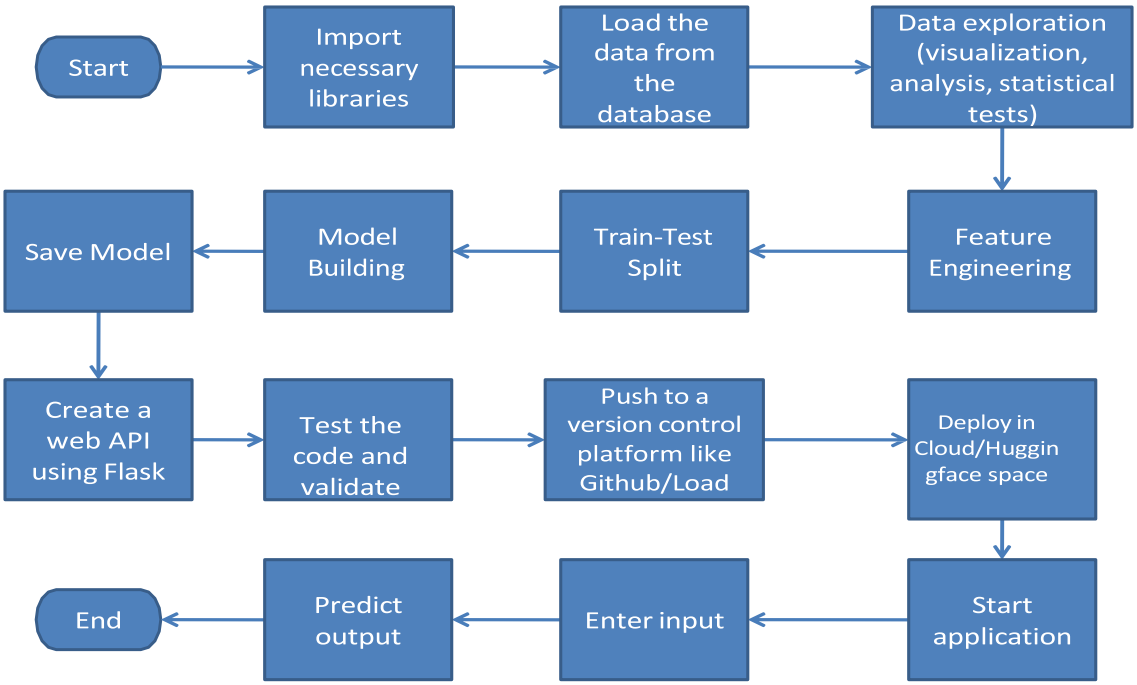
1.1. What is Architectural design document?

The purpose of this document is to provide a comprehensive architectural overview of the Bank Credit Risk Classification application. It will explain the architecture and features, the modules and components of the application. This document is intended for both the stakeholders and the developers of the system and will be proposed to the higher management for its approval.

1.2. Scope

The architectural design is a map of the software. It allows the users to see the structure of the software at a glance.

2. Architecture



3. Architecture Description

3.1. Data Description

This dataset is taken from the UCI Machine Learning Repository (url: https://archive.ics.uci.edu/ml/datasets/South+German+Credit). It contains information on defaults, demographic factors, credit data etc. of customers. There are 21 variables:

Content There are 21 variables:

- **status** : status of the debtor's checking account with the bank (categorical)
  - **1** : no checking account
  - **2** : ... < 0 DM
  - **3** : 0<= ... < 200 DM
  - **4** : ... >= 200 DM / salary for at least 1 year

- **duration** : credit duration in months (quantitative)

- **credit_history** : history of compliance with previous or concurrent credit contracts (categorical)
  - **0** : delay in paying off in the past
  - **1** : critical account/other credits elsewhere
  - **2** : no credits taken/all credits paid back duly
  - **3** : existing credits paid back duly till now
  - **4** : all credits at this bank paid back duly

- **purpose** : purpose for which the credit is needed (categorical)
  - **0** : others
  - **1** : car (new)
  - **2** : car (used)
  - **3** : furniture/equipment
  - **4** : radio/television
  - **5** : domestic appliances
  - **6** : repairs
  - **7** : education
  - **8** : vacation
  - **9** : retraining
  - **10** : business
- **amount** : credit amount in DM (quantitative; result of monotonic transformation; actual data and type of transformation unknown)

- **savings** : debtor's savings (categorical)
  - **1** : unknown/no savings account
  - **2** : ... < 100 DM
  - **3** : 100 <= ... < 500 DM
  - **4** : 500 <= ... < 1000 DM
  - **5** : ... >= 1000 DM

- **employment_duration** : duration of debtor's employment with current employer (ordinal; discretized quantitative)
    - **1** : unemployed
    - **2** : < 1 yr
    - **3** : 1 <= ... < 4 yrs
    - **4** : 4 <= ... < 7 yrs
    - **5** : >= 7 yrs

- **installment_rate** : credit installments as a percentage of debtor's disposable income (ordinal; discretized quantitative)
    - **1** : >= 35
    - **2** : 25 <= ... < 35
    - **3** : 20 <= ... < 25
    - **4** : < 20

- **personal_status_sex** : combined information on sex and marital status; categorical; sex cannot be recovered from the variable, because male singles and female non-singles are coded with the same code (2); female widows cannot be easily classified, because the code table does not list them in any of the female categories
    - **1** : male : divorced/separated
    - **2** : female : non-single or male : single
    - **3** : male : married/widowed
    - **4** : female : single

- **other_debtors :** Is there another debtor or a guarantor for the credit? (categorical)
    - **1** : none
    - **2** : co-applicant
    - **3** : guarantor

- **present_residence :** length of time (in years) the debtor lives in the present residence (ordinal; discretized quantitative)
    - **1** : < 1 yr
    - **2** : 1 <= ... < 4 yrs
    - **3** : 4 <= ... < 7 yrs
    - **4** : >= 7 yrs

- **property :** the debtor's most valuable property, i.e. the highest possible code is used. Code 2 is used, if codes 3 or 4 are not applicable and there is a car or any other relevant property that does not fall under variable savings. (ordinal)
    - **1** : unknown / no property
    - **2** : car or other
    - **3** : building soc. savings agr./life insurance
    - **4** : real estate
- **age :**age in years (quantitative)

- **other_installment_plans :**installment plans from providers other than the credit-giving bank (categorical)
  - **1** : bank
  - **2** : stores
  - **3** : none

- **housing :**type of housing the debtor lives in (categorical)
  - **1** : for free
  - **2** : rent
  - **3** : own

- **number_credits :**number of credits including the current one the debtor has (or had) at this bank (ordinal, discretized quantitative); contrary to Fahrmeir and HamerleÃ¢â‚¬â„¢s (1984) statement, the original data values are not available.
  - **1** : 1
  - **2** : 2-3
  - **3** : 4-5
  - **4** : >= 6

- **job :**quality of debtor's job (ordinal)
  - **1** : unemployed/unskilled - non-resident
  - **2** : unskilled - resident
  - **3** : skilled employee/official
  - **4** : manager/self-empl./highly qualif. employee

- **people_liable :**number of persons who financially depend on the debtor (i.e., are entitled to maintenance) (binary, discretized quantitative)
  - **1** : 3 or more
  - **2** : 0 to 2

- **telephone :**Is there a telephone landline registered on the debtor's name? (binary; remember that the data are from the 1970s)
  - **1 :** No
  - **2 :** Yes

- **foreign_worker :**Is the debtor a foreign worker? (binary)
  - **1** : yes
  - **2** : no

- **credit_risk :**Has the credit contract been complied with (good) or not (bad) ? (binary)
  - **0** : bad
  - **1** : good

3.2. Data Exploration

We divide the data into two types: numerical and categorical. We explore through each type one by one. Within each type, we explore, visualize and analyze each variable one by one

and note down our observations. Statistical tests are performed for each independent variable to see if the variable has any significance in determining the output for the target variable.

3.3. Feature Engineering
Encoded categorical variables.

3.4. Train/Test Split
Split the data into 80% train set and 20% test set.

3.5. Model Building
Built models and trained and tested the data on the models.
Compared the performance of each model and selected the best one.
Feature importance and/or hyper-parameter tuning performed to improve the performance of the selected model.

3.6. Save the model
Saved the model by converting into a pickle file.(pickling is a technique that allows us to serialize Python objects (including our beloved ML models) into a stream of bytes.)

3.7. Create a web API
The model is used to create a webAPI using which the users can interact with the application. In this project, Flask has been used for the purpose.

3.8. Clone the application in Huggingface space :
The huggingface space repository helps to access your application by anyone who searches for for your application and accesss the application and predict the credit risk. You can also provide access to the user to read your application code.

3.9. Cloud Setup & Pushing the App to the Cloud
Selected Heroku for deployment.
Used the model to develop a flask application which can predict risk class for unseen data.
Dockerised the application and pushed to Github using and from there deployed the application files from Github to Heroku, all using Github Actions as ci/cd pipeline.

3.10. Application Start and Input Data by the User
Start the application and enter the inputs.

3.11. Prediction
After the inputs are submitted the application runs the model and makes predictions. The out is displayed as a message indicating whether the customer is classified as Good Risk or Bad Risk.

4. Unit Test Cases

Test Case Description	Pre-Requisite	Expected Result
Verify whether the Application URL is accessible to the user	1. Application URL should be defined/can visit the huggingface and search my project	Application URL should be accessible to the user
Verify whether the Application loads completely for the user when the URL is accessed	1. Application URL is accessible 2. Application is deployed	The Application should load completely for the user runs the app and use the port address in the app URL.
Verify whether user is able to see input fields on logging in	1. Application URL is accessible 2. Application is deployed	User should be able to see input fields on logging in
Verify whether user is able to edit all input fields	1. Application URL is accessible 2. Application is deployed	User should be able to edit all input fields
Verify whether user gets Submit button to submit the inputs	1. Application URL is accessible 2. Application is deployed	User should get Submit button to submit the inputs
Verify whether user is presented with recommended results on clicking submit	1. Application URL is accessible 2. Application is deployed	User should be presented with recommended results on clicking submit.

Bank Credit Risk Classification

Raviteja Kulkarni

https://www.linkedin.com/in/raviteja-slrk/

Mail: ravitejslrk777@gmail.com

Bank Credit Risk Classification

Project Report Presentation

Prepared by: Raviteja Kulkarni
Data Science Intern,
Pwskills

AGENDA

- Introduction
- Objective
- Data Description
- Architecture
- Model Training and Evaluation Workflow
- Deployment
- Questions

Introduction

- Credit risk plays a major role in the banking industry business. Banks' main activities involve granting loan, credit card, investment, mortgage, and others.
- **Credit card has been one of the most booming financial services by banks over the past years. However, with the growing number of credit card users, banks have been facing an escalating credit card default rate.**
- As such data analytics can provide solutions to tackle the current phenomenon and management credit risks. This project discusses the implementation of an model which classifies a customer/applicant profile into Good risk or Bad risk based on certain demographic and behavioral criteria.

Objective

- Development of a model for classifying a customer profile into Risk categories(Good or Bad).
- Benefits:
  - Predicting the level of risk any given customer might pose for the bank/financial institution if allowed with the requested credit.
  - Gives better insight of customer base.
  - Allows financial institutions to take necessary steps to minimize the lose.

Data Description

- **status** : status of the debtor's checking account with the bank (categorical)
  - **1** : no checking account
  - **2** : ... < 0 DM
  - **3** : 0<= ... < 200 DM
  - **4** : ... >= 200 DM / salary for at least 1 year

- **duration** : credit duration in months (quantitative)

- **credit_history** : history of compliance with previous or concurrent credit contracts (categorical)
  - **0** : delay in paying off in the past
  - **1** : critical account/other credits elsewhere
  - **2** : no credits taken/all credits paid back duly
  - **3** : existing credits paid back duly till now

- **4** : all credits at this bank paid back duly

Data Description (cont..)

- **purpose** : purpose for which the credit is needed (categorical)
  - **0** : others
  - **1** : car (new)
  - **2** : car (used)
  - **3** : furniture/equipment
  - **4** : radio/television
  - **5** : domestic appliances
  - **6** : repairs
  - **7** : education
  - **8** : vacation
  - **9** : retraining
  - **10** : business

- **amount** : credit amount in DM (quantitative; result of monotonic transformation; actual data and type of transformation unknown)

Data Description (cont..)

- **savings** : debtor's savings (categorical)
  - **1** : unknown/no savings account
  - **2** : ... < 100 DM
  - **3** : 100 <= ... < 500 DM
  - **4** : 500 <= ... < 1000 DM
  - **5** : ... >= 1000 DM

- **employment_duration** : duration of debtor's employment with current employer (ordinal; discretized quantitative)
  - **1** : unemployed
  - **2** : < 1 yr
  - **3** : 1 <= ... < 4 yrs
  - **4** : 4 <= ... < 7 yrs
  - **5** : >= 7 yrs

Data Description (cont..)

- **installment_rate** : credit installments as a percentage of debtor's disposable income (ordinal; discretized quantitative)
  - **1** : >= 35
  - **2** : 25 <= ... < 35
  - **3** : 20 <= ... < 25
  - **4** : < 20

- **personal_status_sex** : combined information on sex and marital status; categorical; sex cannot be recovered from the variable, because male singles and female non-singles are coded with the same code (2); female widows cannot be easily classified, because the code table does not list them in any of the female categories
  - **1** : male : divorced/separated
  - **2** : female : non-single or male : single
  - **3** : male : married/widowed
  - **4** : female : single

Data Description (cont..)

- **other_debtors :** Is there another debtor or a guarantor for the credit? (categorical)
  - **1** : none
  - **2** : co-applicant
  - **3** : guarantor

- **present_residence :** length of time (in years) the debtor lives in the present residence (ordinal; discretized quantitative)
  - **1** : < 1 yr
  - **2** : 1 <= ... < 4 yrs
  - **3** : 4 <= ... < 7 yrs
  - **4** : >= 7 yrs

- **property :** the debtor's most valuable property, i.e. the highest possible code is used. Code 2 is used, if codes 3 or 4 are not applicable and there is a car or any other relevant property that does not fall under variable savings. (ordinal)
  - **1** : unknown / no property
  - **2** : car or other
  - **3** : building soc. savings agr./life insurance
  - **4** : real estate

Data Description (cont..)

- **age :**age in years (quantitative)

- **other_installment_plans :**installment plans from providers other than the credit-giving bank (categorical)
  - **1** : bank
  - **2** : stores
  - **3** : none
  - –
- **housing :**type of housing the debtor lives in (categorical)
  - **1** : for free
  - **2** : rent
  - **3** : own

- **number_credits :**number of credits including the current one the debtor has (or had) at this bank (ordinal, discretized quantitative
  - **1** : 1
  - **2** : 2-3
  - **3** : 4-5

Data Description (cont..)

- **4** : >= 6

Data Description (cont..)

- **job :**quality of debtor's job (ordinal)
  - **1** : unemployed/unskilled - non-resident
  - **2** : unskilled - resident
  - **3** : skilled employee/official
  - **4** : manager/self-empl./highly qualif. Employee

- **people_liable :**number of persons who financially depend on the debtor (i.e., are entitled to maintenance) (binary, discretized quantitative)
  - **1** : 3 or more
  - **2** : 0 to 2

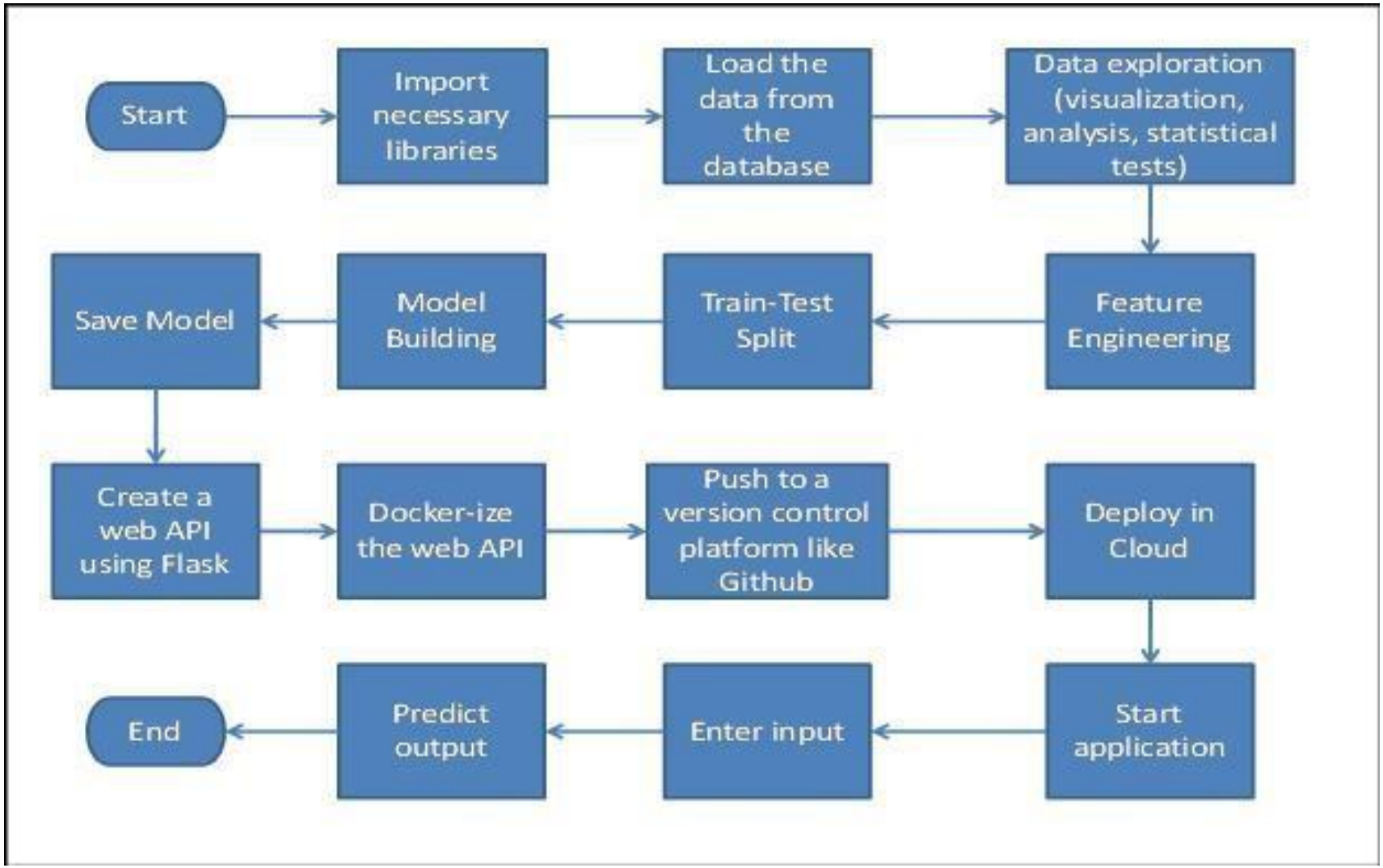
- **telephone :**Is there a telephone landline registered on the debtor's name? (binary; remember that the data are from the 1970s)
  - **1 :** No
  - **2 :** Yes

Data Description (cont..)

- **foreign_worker :**Is the debtor a foreign worker? (binary)
  - **1** : yes
  - **2** : no

- **credit_risk :**Has the credit contract been complied with (good) or not (bad) ? (binary)
  - **0** : bad
  - **1** : good

Architecture


Start
Import necessary libraries
Load the data from the database
Data exploration (visualization, analysis, statistical tests)
Feature Engineering
Train-Test Split
Model Building
Save Model
Create a web API using Flask
Docker-ize the web API
Push to a version control platform like Github
Deploy in Cloud
Start application
Enter input
Predict output
End

Architecture (cont..)

- **Data Exploration**

  We divide the data into two types: numerical and categorical. We explore through each type one by one. Within each type, we explore, visualize and analyze each variable one by one, perform statistical tests and note down our observations. We also make some minor changes in the data like change column names for convenience in understanding.

- **Feature Engineering**
  - Encoded categorical variables.
  - Engineering new features

- **Train/Test Split**

  Split the data into 80% train set and 20% test set.

- **Model Building**
  - Built models and trained and tested the data on the models.

Architecture (cont..)

- Compared the performance of each model and selected the best one.

Architecture (cont..)

- **Save the model**
  Saved the model by converting into a pickle file.

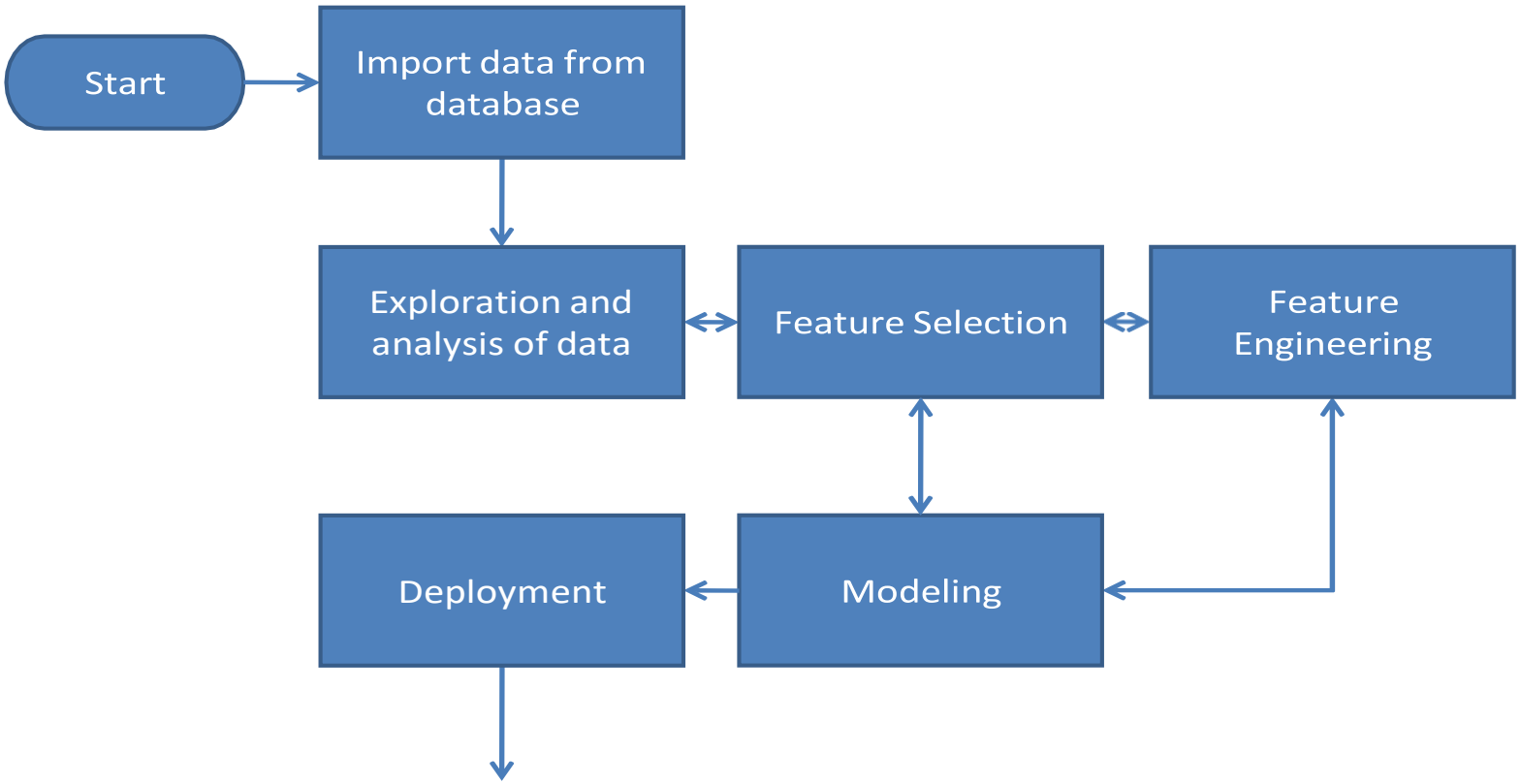
- **Create an API and Dockerize the entire application**

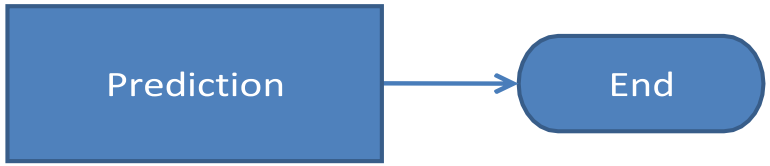
- **Cloud Setup & Pushing the App to the Cloud**
  Selected Heroku for deployment. Loaded the application files from Github to Heroku.

- **Application Start and Input Data by the User**
  Start the application and enter the inputs.

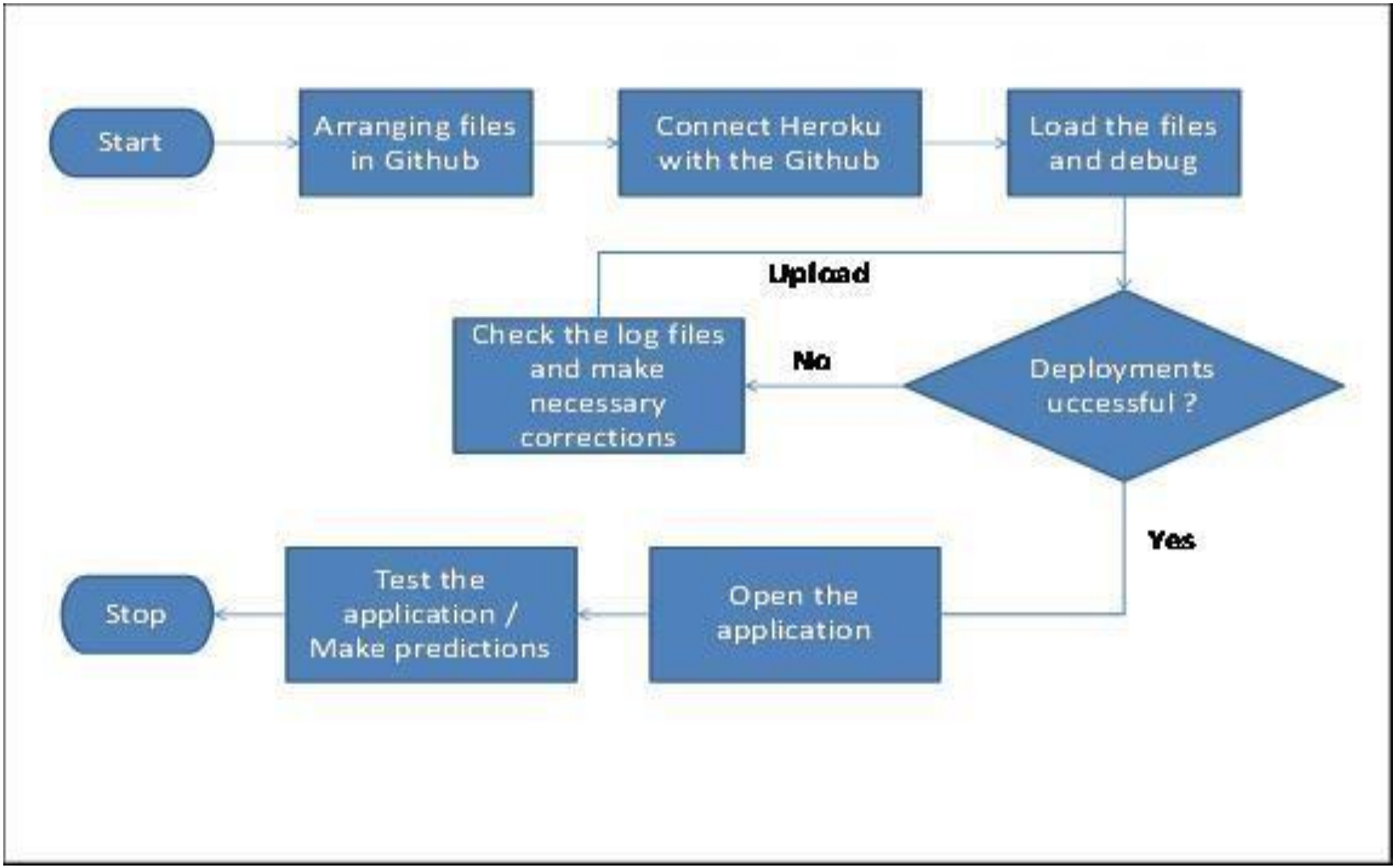
- **Prediction**
  After the inputs are submitted the application runs the model and makes predictions. The out is displayed as a message indicating whether the customer whose demographic and behavioral data are entered as inputs, is likely to default in the following month or not.

Model Training and Evaluation Workflow




Prediction
End

Deployment


Start
Arranging files in Github
Connect Heroku with the Github
Load the files and debug
Upload
Check the log files and make necessary corrections
No
Deployments uccessful ?
Yes
Stop
Test the application / Make predictions
Open the application

FAQs

1) **What is the data source?**

   The data is obtained from UCI Machine Learning Repository.

   Link : https://archive.ics.uci.edu/ml/datasets/South+German+Credit

2) **What was the type of data?**

   The data contained both numerical and continuous type data.

3) **What was the complete flow that you followed in this project?**

   Please refer to slides 13 to 15.

4) **How logs are managed?**

   We have a separate log files for each stage of the project.

FAQs

5) **What techniques were you using for data pre-processing?**
   - Removing unwanted attributes
   - Visualizing relation of independent variables with each other and output variables
   - Cleaning data and imputing if null values are present.
   - Encoding categorical variables

6) **How training was done or what models were used?**
   - After loading the dataset, data pre-processing was done.
   - For this project, we opted to train the data using the XGBoost Classifier.
   - Hyper-parameter tuning, feature selection were performed during the various versions of modeling.
   - The best model was selected.

FAQs

7) How Prediction was done?

- The test files were provided.
- The test data also underwent preprocessing.
- Then the data was passed through the model and output was predicted.

8) What are the different stages of deployment?

- After training the model, we prepared all the necessary files required for deployment and uploaded in a document version control system called Github.
- We then connected to and deployed the model in, Heroku.

THANK YOU

2024

Predict Bank Credit Risk using South German Credit Data

RAVITEJA KULKARNI
MAIL: ravitejslrk777@gmail

Predict Bank Credit Risk using South German Credit Data

High Level Design

Raviteja Kulkarni

Document Version Control

Date Issued	Version	Description	Author
1st August 2024	1.0	Initial HLD	Raviteja Kulkarni

Contents

Abstract

Credit risk plays a major role in the banking industry business. Banks' main activities involve granting loan, credit card, investment, mortgage, and others. Credit card has been one of the most booming financial services by banks over the past years. However, with the growing number of credit card users, banks have been facing an escalating credit card default rate. As such data analytics can provide solutions to tackle the current phenomenon and management credit risks. This project discusses the implementation of an model which classifies a given profile as a good risk or a bad risk.

1. Introduction

1.1. Why this High-Level Design Document?

The purpose of this High-Level Design (HLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.

1.2. The HLD will:

- Present all of the design aspects and define them in detail
- Describe the performance requirements
- Include design features and the architecture of the project
- List and describe the non-functional attributes

  like:
  - Reliability
  - Maintainability
  - Portability
  - Reusability
  - Application compatibility
  - Resource utilization
  - Serviceability

1.3. Scope

The HLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

2. General Description

2.1. Product Perspective

The Credit Card Risk Classification system is a machine learning-based classification model which will help us to classify a given customer profile into either Good Risk or Bad Risk class.

2.2. Problem statement

Financial threats are displaying a trend about the credit risk of banks as the incredible improvement in the financial industry has arisen. In this way, one of the biggest threats faced by banks is the risk prediction of credit clients. The goal is to predict the credit risk of an applicant based on their certain demographic and behavioral characteristics.

2.3. Proposed Solution

The solution proposed here is a web application, which predicts the credit risk for a customer based on the customer's demographic data and behavioral data.

2.4. Data Requirements

This dataset is taken from the UCI Machine Learning Repository (url: https://archive.ics.uci.edu/ml/datasets/South+German+Credit). It contains information on defaults, demographic factors, credit data etc. of customers.
There are 21 variables:

- **status** : status of the debtor's checking account with the bank (categorical)
  - **1** : no checking account
  - **2** : ... < 0 DM
  - **3** : 0<= ... < 200 DM
  - **4** : ... >= 200 DM / salary for at least 1 year

- **duration** : credit duration in months (quantitative)

- **credit history** : history of compliance with previous or concurrent credit contracts (categorical)
  - **0** : delay in paying off in the past
  - **1** : critical account/other credits elsewhere
  - **2** : no credits taken/all credits paid back duly
  - **3** : existing credits paid back duly till now
  - **4** : all credits at this bank paid back duly

- **purpose** : purpose for which the credit is needed (categorical)
  - **0** : others
  - **1** : car (new)
  - **2** : car (used)
  - **3** : furniture/equipment
  - **4** : radio/television

    - **5** : domestic appliances
    - **6** : repairs
    - **7** : education
    - **8** : vacation
    - **9** : retraining
    - **10** : business
- **amount** : credit amount in DM (quantitative; result of monotonic transformation; actual data and type of transformation unknown)

- **savings** : debtor's savings (categorical)
    - **1** : unknown/no savings account
    - **2** : ... < 100 DM
    - **3** : 100 <= ... < 500 DM
    - **4** : 500 <= ... < 1000 DM
    - **5** : ... >= 1000 DM

- **employment_duration** : duration of debtor's employment with current employer (ordinal; discretized quantitative)
    - **1** : unemployed
    - **2** : < 1 yr
    - **3** : 1 <= ... < 4 yrs
    - **4** : 4 <= ... < 7 yrs
    - **5** : >= 7 yrs

- **installment_rate** : credit installments as a percentage of debtor's disposable income (ordinal; discretized quantitative)
    - **1** : >= 35
    - **2** : 25 <= ... < 35
    - **3** : 20 <= ... < 25
    - **4** : < 20

- **personal_status_sex** : combined information on sex and marital status; categorical; sex cannot be recovered from the variable, because male singles and female non-singles are coded with the same code (2); female widows cannot be easily classified, because the code table does not list them in any of the female categories
    - **1** : male : divorced/separated
    - **2** : female : non-single or male : single
    - **3** : male : married/widowed
    - **4** : female : single

- **other_debtors :** Is there another debtor or a guarantor for the credit? (categorical)
    - **1** : none
    - **2** : co-applicant
    - **3** : guarantor

- **present_residence :** length of time (in years) the debtor lives in the present residence (ordinal; discretized quantitative)
    - **1** : < 1 yr
    - **2** : 1 <= ... < 4 yrs
    - **3** : 4 <= ... < 7 yrs
    - **4** : >= 7 yrs

- **property :** the debtor's most valuable property, i.e. the highest possible code is used. Code 2 is used, if codes 3 or 4 are not applicable and there is a car or any other relevant property that does not fall under variable savings. (ordinal)
    - **1** : unknown / no property
    - **2** : car or other
    - **3** : building soc. savings agr./life insurance
    - **4** : real estate
- **age :**age in years (quantitative)

- **other_installment_plans :**installment plans from providers other than the credit-giving bank (categorical)
    - **1** : bank
    - **2** : stores
    - **3** : none

- **housing :**type of housing the debtor lives in (categorical)
    - **1** : for free
    - **2** : rent
    - **3** : own

- **number_credits :**number of credits including the current one the debtor has (or had) at this bank (ordinal, discretized quantitative); contrary to Fahrmeir and HamerleÃ¢â‚¬â„¢s (1984) statement, the original data values are not available.
    - **1** : 1
    - **2** : 2-3
    - **3** : 4-5
    - **4** : >= 6

- **job :**quality of debtor's job (ordinal)
    - **1** : unemployed/unskilled - non-resident
    - **2** : unskilled - resident
    - **3** : skilled employee/official
    - **4** : manager/self-empl./highly qualif. employee

- **people_liable :**number of persons who financially depend on the debtor (i.e., are entitled to maintenance) (binary, discretized quantitative)
    - **1** : 3 or more
    - **2** : 0 to 2

- **telephone :**Is there a telephone landline registered on the debtor's name? (binary; remember that the data are from the 1970s)
    - **1 :** No
    - **2 :** Yes

- **foreign_worker :**Is the debtor a foreign worker? (binary)
    - **1** : yes
    - **2** : no

- **credit_risk :**Has the credit contract been complied with (good) or not (bad) ? (binary)
    - **0** : bad
    - **1** : good

2.5. Tools used

Python programming language and frameworks such as NumPy, Pandas, Scikit-learn are used to build the whole model.

- Jupyter Notebook is used as IDE.
- For visualization of the plots, Matplotlib and Seaborn are used.
- HerokuApp is used for deployment of the model.
- Front end development is done using HTML/CSS
- Python is used for backend development.
- Cassandra is used as database
- GitHub is used as version control system.
- Github Actions is used as ci/cd pipeline.

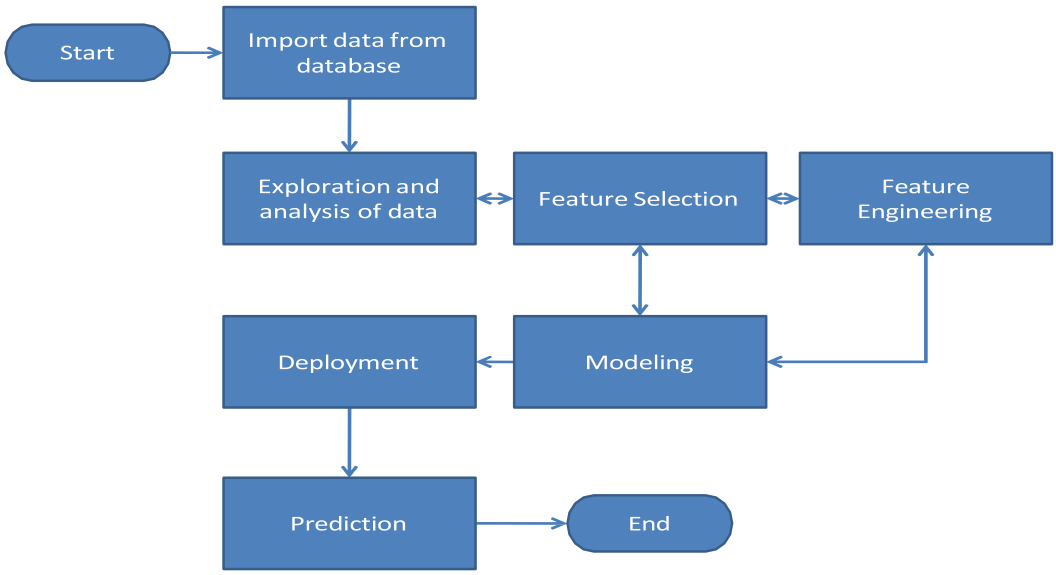
Hugging Face

3. Design Details

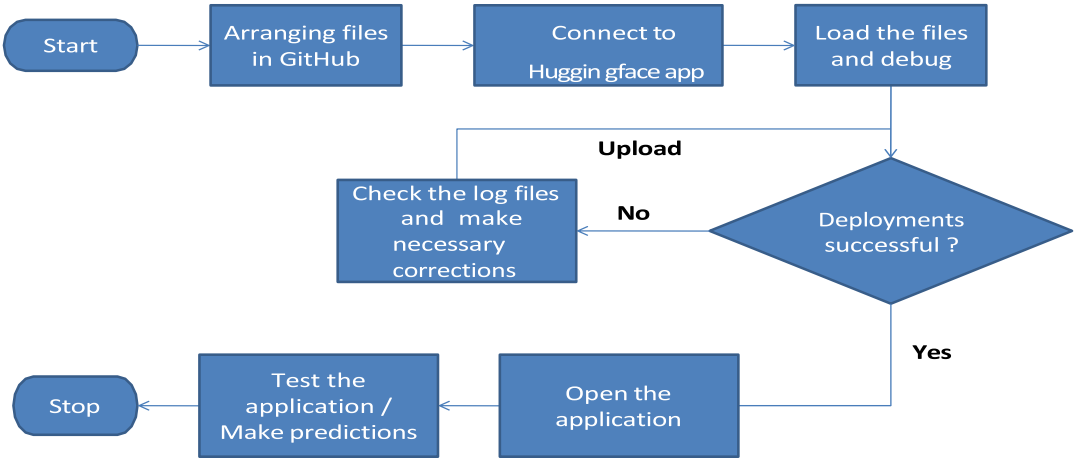
3.1. Process Flow

For identifying the class of each profile, we will use a machine learning model.
Below is the process flow diagram is as shown below.

3.2. Proposed methodology.



3.3. Deployment Process



3.4. Event Log
The event logs are stored in log files.

3.5. Performance
The Bank Credit Risk Classification app is used to classify whether a given profile could be approved for credits, would they pose a risk for the bank/financial organization etc. Characteristics like age, job, savings, credit details etc.. are examined and based on the analysis, the applicants are classified into Good Risk and Bad Risk categories.

3.6. Reusability
The code written and the components used should have the ability to be reused with no problems.

3.7. Application Compatibility
The different components for this project will be using Python as an interface between them. Each component will have its own task to perform, and it is the job of the Python to ensure proper transfer of information.

3.8. Resource Utilization
When any task is performed, it will likely use all the processing power available until that function is finished.

3.9. Deployment
The model can be deployed in any cloud services such as Microsoft Azure, AWS, Google, but here we are deploying in hugging face repository as a app.

4. Conclusion

This application will classify profiles/applicants for credit into Good Risk and Bad Risk categories, and can help financial institutions in taking necessary actions to prevent further loss.

2024

Predict Bank Credit Risk using South German Credit Data

RAVITEJA KULKARNI
MAIL: ravitejslrk777@gmail.com

Predict Bank Credit Risk using South German Credit Data

Low Level Design

Raviteja Kulkarni
Aug, 2024

Predict Bank Credit Risk using South German Credit Data

Contents

Document Version Control

Date Issued	Version	Description	Author
1st August 2024	1.0	Initial LLD	Raviteja Kulkarni

Abstract

Credit risk plays a major role in the banking industry business. Banks' main activities involve granting loan, credit card, investment, mortgage, and others. Credit card has been one of the most booming financial services by banks over the past years. However, with the growing number of credit card users, banks have been facing an escalating credit card default rate. As such data analytics can provide solutions to tackle the current phenomenon and management credit risks. This project discusses the implementation of an model which classifies a given profile as a good risk or a bad risk.

1. Introduction

1.1. Why this Low-Level Design Document?

The purpose of this document is to present a detailed description of the Bank Credit Risk Classification application. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate etc. This document is intended for both the stakeholders and the developers of the system and will be proposed to the higher management for its approval.

1.2. Scope

This software system will be a Web application. This system is designed to classify a customer profile into good or bad risk category, from customers' information such as demographics, credit details etc.

2. Technical Specifications

2.1. Dataset

File Name	Finalized	Source
SouthGermanCredit.asc	Yes	https://archive.ics.uci.edu/ml/datasets/South+German+Credit

2.1.1. Dataset Overview

The data obtained from the repository is in the form of .asc file, from which the data is extracted and stored in the Cassandra database. It contains the personal and behavioral data of about 1000 customers, out of which 700 are classified as Good Risk and 300 as Bad Risk.

status	duration	credit_history	purpose	amount	savings	employment_duration	installment_rate	personal_status_sex	other_debtors	present_residence	property	age	other_installment_plans	housing	number_credits	job	people_liable	telephone	foreign_worker	credit_risk
1	18	4	2	1049	1	2	4	2	1	4	2	21	3	1	1	3	2	1	2	1
1	9	4	0	2799	1	3	2	3	1	2	1	36	3	1	2	3	1	1	2	1
2	12	2	9	841	2	4	2	2	1	4	1	23	3	1	1	2	2	1	2	1
1	12	4	0	2122	1	3	3	3	1	2	1	39	3	1	2	2	1	1	1	1
1	12	4	0	2171	1	3	4	3	1	4	2	38	1	2	2	2	2	1	1	1
1	10	4	0	2241	1	2	1	3	1	3	1	48	3	1	2	2	1	1	1	1
1	8	4	0	3398	1	4	1	3	1	4	1	39	3	2	2	2	2	1	1	1
1	6	4	0	1361	1	2	2	3	1	4	1	40	3	2	1	2	1	1	1	1
4	18	4	3	1098	1	1	4	2	1	4	3	65	3	2	2	1	2	1	2	1
2	24	2	3	3758	3	1	1	2	1	4	4	23	3	1	1	1	2	1	2	1
1	11	4	0	3905	1	3	2	3	1	2	1	36	3	1	2	3	1	1	2	1
1	30	4	1	6187	2	4	1	4	1	4	3	24	3	1	2	3	2	1	2	1
1	6	4	3	1957	1	4	1	2	1	4	3	31	3	2	1	3	2	1	2	1
2	48	3	10	7582	2	1	2	3	1	4	4	31	3	2	1	4	2	2	2	1
1	18	2	3	1936	5	4	2	4	1	4	3	23	3	1	2	2	2	1	2	1
1	6	2	3	2647	3	3	2	3	1	3	1	44	3	1	1	3	1	1	2	1
1	11	4	0	3939	1	3	1	3	1	2	1	40	3	2	2	2	1	1	2	1
2	18	2	3	3213	3	2	1	4	1	3	1	25	3	1	1	3	2	1	2	1
2	36	4	3	2337	1	5	4	3	1	4	1	36	3	2	1	3	2	1	2	1
4	11	4	0	7228	1	3	1	3	1	4	2	39	3	2	2	2	2	1	2	1
1	6	4	0	3676	1	3	1	3	1	3	1	37	3	1	3	3	1	1	2	1
2	12	4	0	3124	1	2	1	3	1	3	1	49	1	2	2	2	1	1	2	1

2.1.2. Input Schema

Feature Name	Datatype	Null/Required
status	Integer	Required
duration	Integer	Required
credit_history	Integer	Required
purpose	Integer	Required
amount	Integer	Required
savings	Integer	Required
employment_duration	Integer	Required
installment_rate	Integer	Required
personal_status_sex	Integer	Required
other_debtors	Integer	Required
present_residence	Integer	Required
property	Integer	Required
age	Integer	Required
other_installment_plans	Integer	Required

housing	Integer	Required
number_credits	Integer	Required
job	Integer	Required
people_liable	Integer	Required
telephone	Integer	Required
foreign_worker	Integer	Required
credit_risk	Integer	Required

2.2. Predicting Credit Risk class

- The system presents the set of inputs to the user.
- The user gives required information.
- The system should be able to predict whether the customer is likely to pose a risk to the bank or not.

2.3. Logging

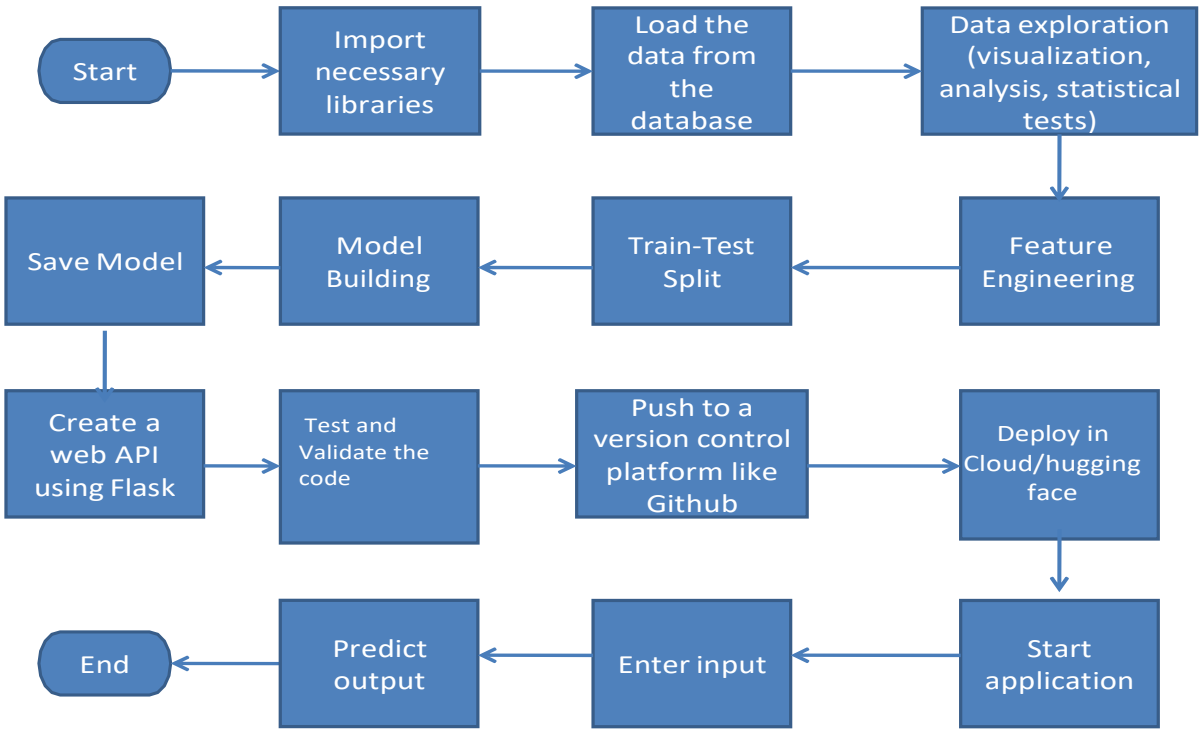
We should be able to log every activity done by the user.

- The System identifies at what step logging required.
- The System should be able to log each and every system flow.
- Developers can choose logging methods. You can choose database logging/ File logging as well.
- System should not be hung even after using so many loggings. Logging just because we can easily debug issues so logging is mandatory to do.

2.4. Deployment

The application is containerized using Docker and deployed in Heroku.

3. Architecture



4. Architecture Description

4.1. Data Description

This dataset is taken from the UCI Machine Learning Repository (url: https://archive.ics.uci.edu/ml/datasets/South+German+Credit). It contains information on demographic factors, credit data etc. of customers. There are 21 variables:

Content There are 21 variables:

- **status** : status of the debtor's checking account with the bank (categorical)
  - **1** : no checking account
  - **2** : ... < 0 DM
  - **3** : 0<= ... < 200 DM
  - **4** : ... >= 200 DM / salary for at least 1 year

- **duration** : credit duration in months (quantitative)

- **credit_history** : history of compliance with previous or concurrent credit contracts (categorical)
  - **0** : delay in paying off in the past
  - **1** : critical account/other credits elsewhere
  - **2** : no credits taken/all credits paid back duly
  - **3** : existing credits paid back duly till now
  - **4** : all credits at this bank paid back duly

- **purpose** : purpose for which the credit is needed (categorical)
  - **0** : others
  - **1** : car (new)
  - **2** : car (used)
  - **3** : furniture/equipment
  - **4** : radio/television
  - **5** : domestic appliances
  - **6** : repairs
  - **7** : education
  - **8** : vacation
  - **9** : retraining
  - **10** : business
- **amount** : credit amount in DM (quantitative; result of monotonic transformation; actual data and type of transformation unknown)

- **savings** : debtor's savings (categorical)
  - **1** : unknown/no savings account
  - **2** : ... < 100 DM
  - **3** : 100 <= ... < 500 DM
  - **4** : 500 <= ... < 1000 DM
  - **5** : ... >= 1000 DM

- **employment_duration** : duration of debtor's employment with current employer (ordinal; discretized quantitative)
    - **1** : unemployed
    - **2** : < 1 yr
    - **3** : 1 <= ... < 4 yrs
    - **4** : 4 <= ... < 7 yrs
    - **5** : >= 7 yrs

- **installment_rate** : credit installments as a percentage of debtor's disposable income (ordinal; discretized quantitative)
    - **1** : >= 35
    - **2** : 25 <= ... < 35
    - **3** : 20 <= ... < 25
    - **4** : < 20

- **personal_status_sex** : combined information on sex and marital status; categorical; sex cannot be recovered from the variable, because male singles and female non-singles are coded with the same code (2); female widows cannot be easily classified, because the code table does not list them in any of the female categories
    - **1** : male : divorced/separated
    - **2** : female : non-single or male : single
    - **3** : male : married/widowed
    - **4** : female : single

- **other_debtors :** Is there another debtor or a guarantor for the credit? (categorical)
    - **1** : none
    - **2** : co-applicant
    - **3** : guarantor

- **present_residence :** length of time (in years) the debtor lives in the present residence (ordinal; discretized quantitative)
    - **1** : < 1 yr
    - **2** : 1 <= ... < 4 yrs
    - **3** : 4 <= ... < 7 yrs
    - **4** : >= 7 yrs

- **property :** the debtor's most valuable property, i.e. the highest possible code is used. Code 2 is used, if codes 3 or 4 are not applicable and there is a car or any other relevant property that does not fall under variable savings. (ordinal)
    - **1** : unknown / no property
    - **2** : car or other
    - **3** : building soc. savings agr./life insurance
    - **4** : real estate
- **age :**age in years (quantitative)

- **other_installment_plans :**installment plans from providers other than the credit-giving bank (categorical)
  - **1** : bank
  - **2** : stores
  - **3** : none

- **housing :**type of housing the debtor lives in (categorical)
  - **1** : for free
  - **2** : rent
  - **3** : own

- **number_credits :**number of credits including the current one the debtor has (or had) at this bank (ordinal, discretized quantitative); contrary to Fahrmeir and HamerleÃ¢â‚¬â„¢s (1984) statement, the original data values are not available.
  - **1** : 1
  - **2** : 2-3
  - **3** : 4-5
  - **4** : >= 6

- **job :**quality of debtor's job (ordinal)
  - **1** : unemployed/unskilled - non-resident
  - **2** : unskilled - resident
  - **3** : skilled employee/official
  - **4** : manager/self-empl./highly qualif. employee

- **people_liable :**number of persons who financially depend on the debtor (i.e., are entitled to maintenance) (binary, discretized quantitative)
  - **1** : 3 or more
  - **2** : 0 to 2

- **telephone :**Is there a telephone landline registered on the debtor's name? (binary; remember that the data are from the 1970s)
  - **1 :** No
  - **2 :** Yes

- **foreign_worker :**Is the debtor a foreign worker? (binary)
  - **1** : yes
  - **2** : no

- **credit_risk :**Has the credit contract been complied with (good) or not (bad) ? (binary)
  - **0** : bad
  - **1** : good

4.2. Data Exploration

We divide the data into two types: numerical and categorical. We explore through each type one by one. Within each type, we explore, visualize and analyze each variable one by one

and note down our observations. Statistical tests are performed for each independent variable to see if the variable has any significance in determining the output for the target variable.

4.3. Feature Engineering
Encoded categorical variables.

4.4. Train/Test Split
Split the data into 80% train set and 20% test set.

4.5. Model Building
Built models and trained and tested the data on the models.
Compared the performance of each model and selected the best one.
Feature importance and/or hyper-parameter tuning performed to improve the performance of the selected model.

4.6. Save the model
Saved the model by converting into a pickle file.

4.7. Cloud Setup & Pushing the App to the Cloud
I am using here the huggingface space to build the app on repository [cloud]. Anybody can search my app on hugging face and run the app.

4.8. Application Start and Input Data by the User
Start the application and enter the inputs.

4.9. Prediction
After the inputs are submitted the application runs the model and makes predictions. The out is displayed as a message indicating whether the customer is classified as Good Risk or Bad Risk.

5. Unit Test Cases

Test Case Description	Pre-Requisite	Expected Result
Verify whether the Application URL is accessible to the user	1. Application URL should be defined	Application URL should be accessible to the user
Verify whether the Application loads completely for the user when the URL is accessed	1. Application URL is accessible 2. Application is deployed	The Application should load completely for the user when the URL is accessed/app is run from huggingface space
Verify whether user is able to see input fields on logging in	1. Application URL is accessible 2. Application is deployed	User should be able to see input fields on logging in
Verify whether user is able to edit all input fields	1. Application URL is accessible 2. Application is deployed	User should be able to edit all input fields
Verify whether user gets submit button to submit the inputs	1. Application URL is accessible 2. Application is deployed	User should get Submit button to submit the inputs
Verify whether user is presented with recommended results on clicking submit	1. Application URL is accessible 2. Application is deployed	User should be presented with recommended results on clicking submit

PHYSICS WALLAH

INTERNSHIP OFFER LETTER

Dear Raviteja Kulkanri,

Following your application, we are pleased to inform you that you have been considered for an internship with **PHYSICS WALLAH PVT. LTD.** As a result, you will be contributing to our project, Predict Bank Credit Risk using South German Credit Data from **15th September 2023** . As a part of your internship, you will be proactively contributing to your selected project, besides product development & PoCs. In addition, you will be required to complete performance & learning goals for your current project with us. We hope that your association with the company will be successful and rewarding.

Regards

Mr. Alakh Pandey
(Founder)

I accept the offer with the company on the terms and conditions set out in this letter.

2024

Predict Bank Credit Risk using South German Credit Data

RAVITEAJ KULKARNI
MAIL: ravitejslrk777@gmail.com

Predict Bank Credit Risk using South German Credit Data

Wireframe

Raviteja

Wireframe

The web application for Bank Credit Risk Classification has:

1. A home page, where the user
    a. Access the Hugging space application
    b Enters inputs
    c. Submits the inputs

a). All our package is deployed as app in huggingface space, user can easily access it through url, or also search our app name and access it.

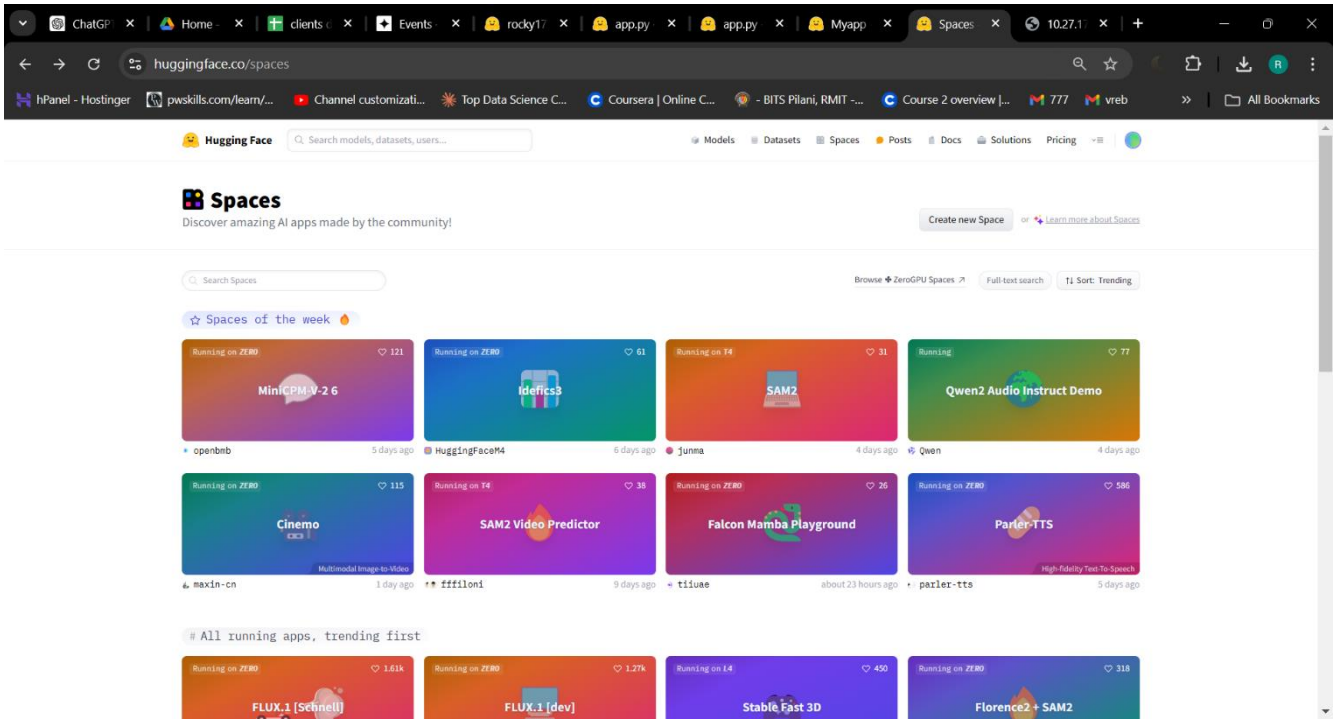


Fig.1. UI of huggingface Space where user can search our app

b). Two types of data are entered : Personal /Demographic information and Behavioral information

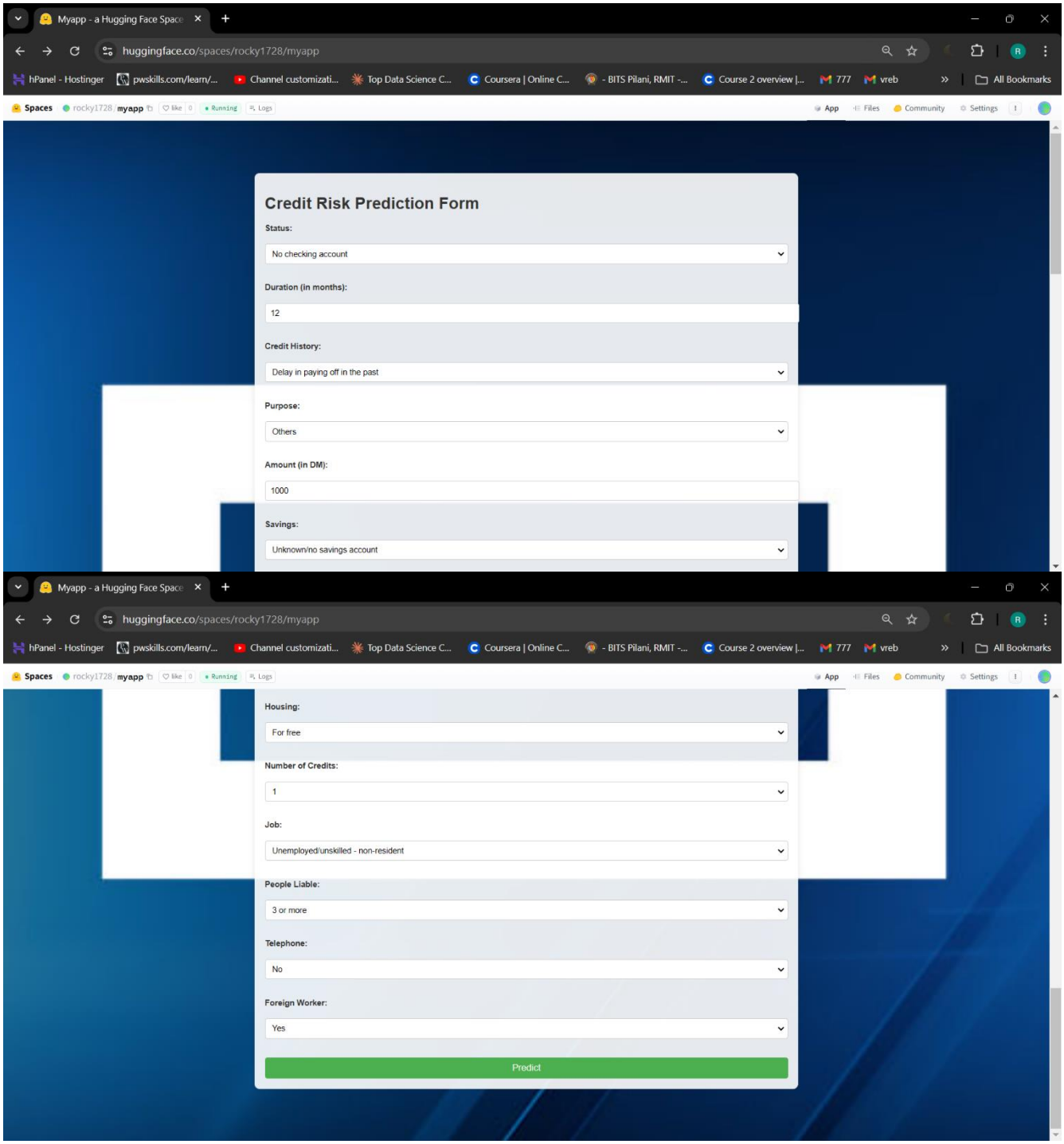


Fig2. Web page of application where user can input the data.

c). A results/output page, where the application displays the output, indicating whether the customer is classified as a Good Risk or Bad risk.

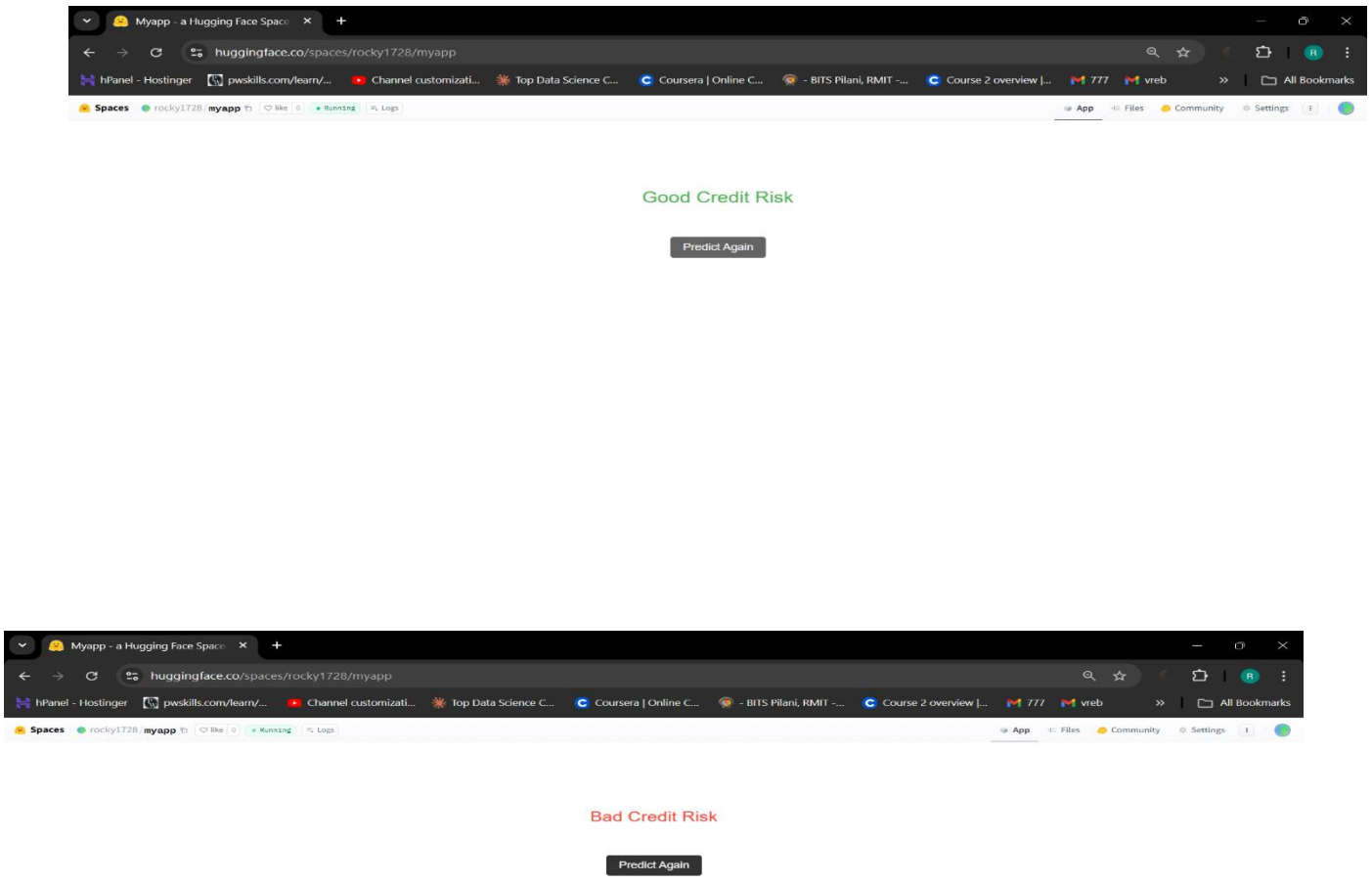


Fig.3. Output result of ML model predicting the credit_risk