Integrantes: *Maria Lucía Castillo García, Juliana González Sánchez y Ana Daniela Paredes Tovar*

1.1 Descripción General

El Sistema Nacional de Información de la Educación Superior (SNIES) en Colombia es una plataforma fundamental para la recopilación y gestión de datos relacionados con las instituciones de educación superior y los programas académicos que ofrecen. Estos datos incluyen información clave sobre procesos como la inscripción, admisión, matrícula y graduación, constituyendo una base esencial para la toma de decisiones estratégicas en el sector educativo.

El análisis de estas tendencias permite identificar oportunidades de mejora en el diseño curricular de programas académicos, asegurando que sean pertinentes, atractivos y alineados con las necesidades de los estudiantes. Asimismo, el estudio de datos históricos y patrones emergentes resulta crucial para la planificación estratégica de las instituciones, facilitando decisiones informadas sobre la apertura de nuevos programas, la modificación de los existentes y la asignación eficiente de recursos.

Su objetivo principal es procesar grandes volúmenes de información proveniente de diferentes fuentes, como admitidos, graduados, inscritos, matriculados y matriculados en primer curso, consolidándolos en un único DataFrame que permite realizar análisis eficientes y personalizados.

El sistema permite a los usuarios filtrar los datos por palabras clave y rangos de años específicos. Además, incluye la capacidad de exportar los resultados en formatos versátiles como XLSX, CSV y JSON, facilitando su integración en otros sistemas o plataformas.

A través de una interfaz interactiva creada con Streamlit, los usuarios pueden cargar archivos desde su sistema local o navegador, parametrizar búsquedas, y visualizar los datos procesados. También ofrece herramientas para graficar y analizar tendencias basadas en las métricas seleccionadas, como admitidos o graduados, durante los años disponibles en los datos.

La estructura del proyecto se organiza en componentes especializados:

1. **Gestores**: Encargados de leer y exportar datos en diversos formatos.
2. **SNIESController**: Responsable de consolidar y procesar la información en función de las solicitudes del usuario.
3. **View**: Maneja la interacción con el usuario a través de la interfaz gráfica, validando entradas y guiando al usuario durante el proceso.

4. **Settings**: Define rutas y configuraciones clave para los archivos de entrada y salida.

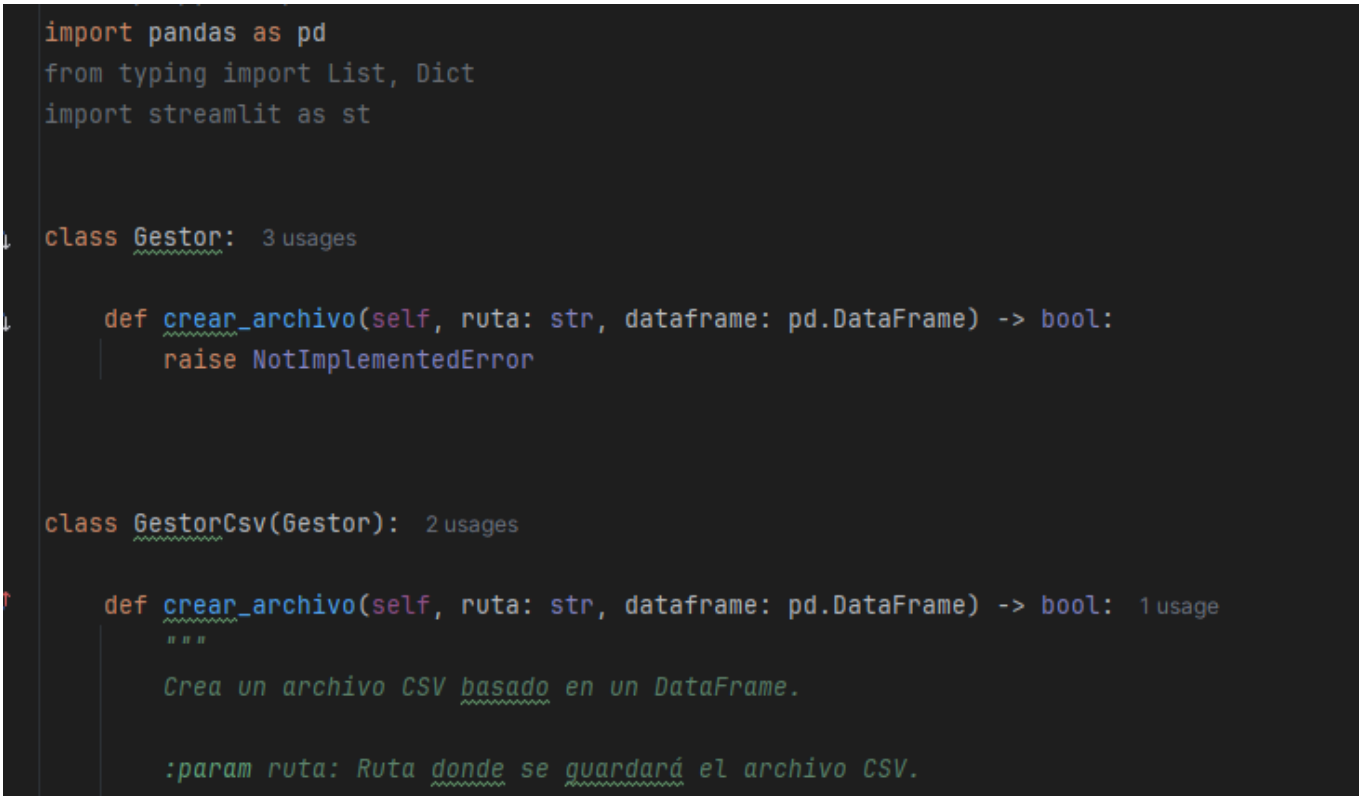
En conjunto, "SNIES Extractor" es una herramienta robusta diseñada para facilitar el análisis y la toma de decisiones basada en datos educativos en Colombia.

1.2 Descripción del Proyecto SNIES

En el desarrollo del presente proyecto, se optó por abandonar el uso de C++ en favor de Python, dado que este último ofrece herramientas más adecuadas para el procesamiento y análisis de datos, así como una mayor facilidad para la creación de aplicaciones interactivas a través de bibliotecas como Streamlit. Esto hizo que clases como *Consolidado* y *ProgramaAcademico* pasaran a ser obsoletas por lo que consolidar en Python por medio de dataframes resulta ser más sencillo y eficiente.

Ahora bien, entre las clases que sí prevalecieron fueron: *Gestor* que funciona como una clase base para manejar diferentes tipos de archivos. De ella derivan las clases hijas *GestorCsv*, *GestorXlx* y *GestorJson* , que se especializan en la lectura y escritura de datos en diferentes formatos. Esto lo hicimos por medio de dataframes. También mantuvimos el *SNIESController* quien se encarga de controlar la interacción con el SNIES, facilitando la conexión y la gestión de datos necesarios para el análisis.

Imágenes de Gestor



```python
import pandas as pd
from typing import List, Dict
import streamlit as st


class Gestor:  3 usages

    def crear_archivo(self, ruta: str, dataframe: pd.DataFrame) -> bool:
        raise NotImplementedError



class GestorCsv(Gestor):  2 usages

    def crear_archivo(self, ruta: str, dataframe: pd.DataFrame) -> bool:  1 usage
        """
        Crea un archivo CSV basado en un DataFrame.

        :param ruta: Ruta donde se guardará el archivo CSV.
```

```python
    def crear_archivo(self, ruta: str, dataframe: pd.DataFrame) -> bool:  1 usage
        try:
            dataframe.to_csv(ruta, index=False, encoding='utf-8')  # Guarda el DataFrame directamente como CSV
            return True
        except Exception as e:
            print(f"Error al crear el archivo CSV: {e}")
            return False




class GestorJson(Gestor):  2 usages

    def crear_archivo(self, ruta: str, dataframe: pd.DataFrame) -> bool:  1 usage
        """
        Crea un archivo JSON basado en un DataFrame.

        :param ruta: Ruta donde se guardará el archivo JSON.
        :param dataframe: DataFrame que contiene los datos a guardar.
        :return: True si se crea el archivo correctamente.
        """
```

```python
    def crear_archivo(self, ruta: str, dataframe: pd.DataFrame) -> bool:  1 usage
        :return: True si se crea el archivo correctamente.
        """
        try:
            dataframe.to_json(ruta, orient='records', indent=4, force_ascii=False)  # Guarda el DataFrame como JSON
            return True
        except Exception as e:
            print(f"Error al crear el archivo JSON: {e}")
            return False


class GestorXlsx(Gestor):  5 usages

    @staticmethod  1 usage
    def leer_archivo(ruta, palabra_clave, columna_unica):
        try:
            df = pd.read_excel(ruta, header=None)

            # Limpiar filas completamente vacías
            df.dropna(how='all', inplace=True)
```

```python
    # Buscar la fila del encabezado correcto
    fila_encabezado = None
    for idx, fila in df.iterrows():
        if "PROGRAMA ACADÉMICO" in fila.values:  # Buscar la columna clave
            fila_encabezado = idx
            break

    if fila_encabezado is None:
        raise KeyError("No se encontró la columna 'PROGRAMA ACADÉMICO' en el archivo.")

    # Releer el archivo usando la fila del encabezado detectado
    df = pd.read_excel(ruta, header=fila_encabezado)

    # Estandarizar nombres de columnas
    df.columns = df.columns.str.strip().str.upper()

    # Validar si la columna clave está presente
    if "PROGRAMA ACADÉMICO" not in df.columns:
        raise KeyError("La columna 'PROGRAMA ACADÉMICO' no existe en el archivo después de procesarlo.")
```

```python
        # Filtrar por palabra clave
        df_palabra_clave = df[df["PROGRAMA ACADÉMICO"].str.contains(palabra_clave, case=False, na=False)]

        # Definir columnas a consolidar según `unico_dato`
        columnas_a_buscar = [
            "CÓDIGO SNIES DEL PROGRAMA", "SEMESTRE"
        ] if columna_unica else [
            "CÓDIGO SNIES DEL PROGRAMA",
            "METODOLOGÍA",
            "PROGRAMA ACADÉMICO",
            "INSTITUCIÓN DE EDUCACIÓN SUPERIOR (IES)",
            "PRINCIPAL O SECCIONAL",
            "DEPARTAMENTO DE OFERTA DEL PROGRAMA",
            "MUNICIPIO DE OFERTA DEL PROGRAMA",
            "NIVEL DE FORMACIÓN",
            "SEMESTRE"
        ]
```

```python
        # Selección y consolidación de datos
        ultima_columna = df.columns[-1]
        ultima_columna_df = df_palabra_clave[ultima_columna]
        buscado = df_palabra_clave[columnas_a_buscar]


        buscado = pd.concat( objs: [buscado, ultima_columna_df], axis=1)
        resultado = buscado.groupby(columnas_a_buscar).sum(numeric_only=True).reset_index()

        return resultado

    except KeyError as e:
        print(f"Error: {e}")
        return None
    except Exception as e:
        print(f"Error al procesar el archivo: {e}")
        return None
```

```python
def crear_archivo(self, ruta: str, dataframe: pd.DataFrame) -> bool:  1 usage
    """
    Crea un archivo XLSX basado en un DataFrame.

    :param ruta: Ruta donde se guardará el archivo XLSX.
    :param dataframe: DataFrame que contiene los datos a guardar.
    :return: True si se crea el archivo correctamente.
    """
    try:
        dataframe.to_excel(ruta, index=False, engine='openpyxl')  # Guarda el DataFrame como XLSX
        return True
    except Exception as e:
        print(f"Error al crear el archivo XLSX: {e}")
        return False
```

Imágenes de SNIES

```python
import pandas as pd
from gestores import GestorXlsx
from Settings import Settings


class SNIESController:  3 usages
    def __init__(self):
        self.dataframe_principal = pd.DataFrame()  # DataFrame para consolidar toda la información
        self.gestor_xlsx = GestorXlsx()
        self.etiquetas_columnas = []
        self.ruta_admitidos = Settings.ADMITIDOS_FILE_PATH
        self.ruta_graduados = Settings.GRADUADOS_FILE_PATH
        self.ruta_inscritos = Settings.INSCRITOS_FILE_PATH
        self.ruta_matriculados = Settings.MATRICULADOS_FILE_PATH
        self.ruta_matriculados_primer_curso = Settings.MATRICULADOS_PRIMER_CURSO_FILE_PATH

    def procesar_datos(self, anio1, anio2, palabra_clave):  1 usage
        try:
            anio11 = int(anio1)
            anio22 = int(anio2)

            anios = [anio for anio in range(min(anio11, anio22), max(anio11, anio22) + 1)]
```

```python
        anios = [anio for anio in range(min(anio11, anio22), max(anio11, anio22) + 1)]
        primero = True
        gestor = GestorXlsx()

        for anio in anios:
            DIRECCIONES = [
                Settings.ADMITIDOS_FILE_PATH,
                Settings.GRADUADOS_FILE_PATH,
                Settings.INSCRITOS_FILE_PATH,
                Settings.MATRICULADOS_FILE_PATH,
                Settings.MATRICULADOS_PRIMER_CURSO_FILE_PATH
            ]

            for direccion in DIRECCIONES:
                ruta_archivo = f"{direccion}{anio}.xlsx"
                dataframe_minimizado = gestor.leer_archivo(ruta_archivo, palabra_clave, not primero)

                dataframe_minimizado["CÓDIGO SNIES DEL PROGRAMA"] = dataframe_minimizado["CÓDIGO SNIES DEL PROGRAMA"].astype(str)
                dataframe_minimizado["SEMESTRE"] = dataframe_minimizado["SEMESTRE"].astype(str)
```

```python
            if primero:

                dataframe_minimizado.rename(columns={"ADMITIDOS": f'{"ADMITIDOS"}_{anio}'}, inplace=True)
                self.df = dataframe_minimizado
                primero = False

            else:
                for col in dataframe_minimizado.columns:
                    if not (col == "CÓDIGO SNIES DEL PROGRAMA" or col == "SEMESTRE"):
                        dataframe_minimizado.rename(columns={col: f'{col}_{anio}'}, inplace=True)

                self.df = pd.merge(
                    self.df, dataframe_minimizado,
                    on=["CÓDIGO SNIES DEL PROGRAMA", "SEMESTRE"],
                    how="left"
                )

    except FileNotFoundError:
        raise FileNotFoundError("Archivo necesario no está cargado al SNIES")
    except Exception as e:
```

```python
    except FileNotFoundError:
        raise FileNotFoundError("Archivo necesario no está cargado al SNIES")
    except Exception as e:
        raise Exception(f"Error al procesar los datos: {e}")

    return self.df
```

La aplicación que desarrollamos se llama *streamlitSNIES*. Esta es una aplicación interactiva desarrollada con Streamlit que permite analizar, filtrar y exportar datos relacionados con programas académicos del Sistema Nacional de Información de la Educación Superior (SNIES) en Colombia. La herramienta carga archivos en formatos como XLSX, CSV o JSON, y ofrece al usuario la posibilidad de filtrar información por años y palabras clave, consolidando los resultados en un único DataFrame para su análisis. Además, incluye funcionalidades para generar gráficos y visualizar datos seleccionados, ofreciendo una interfaz accesible y amigable

que guía al usuario en cada paso del proceso. StreamlitSNIES es especialmente útil para instituciones educativas, investigadores o administradores que necesitan procesar grandes volúmenes de datos del SNIES de manera rápida y eficiente.

Imágenes de App

```python
import streamlit as st
import pandas as pd
import os
import matplotlib.pyplot as plt
import re
from view import View


st.title("¡Bienvenido al **_SNIES Extractor_**! 📌")
st.write("_Siga las instrucciones a continuación para realizar el análisis y exportación de datos._")

st.subheader("Parametrización Inicial")
st.write("Asegúrese de que la carpeta SNIES_EXTRACTOR esté correctamente configurada con los archivos necesarios.")

if st.button("Confirmar parametrización"):
    st.success("Parametrización confirmada. Puedes continuar con el análisis.")

st.subheader("Selección de parámetros")
anio1 = st.text_input("Escriba el primer año de búsqueda:", value="2020")
anio2 = st.text_input("Escriba el segundo año de búsqueda:", value="2021")

view = View()

valid_anio1, anio1_int = view.validar_entrada_anio(anio1)
valid_anio2, anio2_int = view.validar_entrada_anio(anio2)

if not (valid_anio1 and valid_anio2):
    st.error("Ambos años deben ser valores enteros válidos.")
else:
    palabra_clave = st.text_input("Ingrese una palabra clave para filtrar los datos:", value="")

    if not palabra_clave.strip():
        st.error("Debe ingresar una palabra clave válida.")
    else:
        formato = st.selectbox("Seleccione el formato de archivo para exportar los datos:", ["XLSX", "CSV", "JSON"])

        if st.button("Procesar y Exportar"):
            view.procesar_datos(anio1_int, anio2_int, palabra_clave, formato)

st.subheader("Cierre del programa")
if st.button("Salir"):
    st.success("Recuerde revisar la carpeta de outputs para los archivos exportados. ¡Programa cerrado con éxito!")


st.write("_Siga las instrucciones a continuación para generar los análisis deseados_")

# Opción de selección: Cargar archivo desde el sistema local o desde el navegador
opcion_archivo = st.radio("¿Cómo deseas cargar el archivo?", ('Desde el sistema local', 'Desde el navegador'))
```

```python
def extraer_anios_columnas(columnas):  2 usages
    """
    Extrae los años de las columnas que siguen el patrón <nombre>_<año>.
    """
    anios = []
    for col in columnas:
        match = re.search( pattern: r"_(\d{4})$", col)
        if match:
            anios.append(int(match.group(1)))
    return sorted(set(anios))


# Cargar archivos desde el sistema local
if opcion_archivo == 'Desde el sistema local':
    directorio_archivos = "C:/Users/casti/Documents/AYIYIWUA/OUTPUTS"
    archivos = [f for f in os.listdir(directorio_archivos) if f.endswith(('.csv', '.xlsx'))]
    archivo_seleccionado = st.selectbox("Selecciona un archivo", archivos)

    if archivo_seleccionado:
        ruta_archivo = os.path.join(directorio_archivos, archivo_seleccionado)

        try:
            if archivo_seleccionado.endswith(".csv"):
                df = pd.read_csv(ruta_archivo, delimiter=";").dropna()
```

```python
    elif archivo_seleccionado.endswith(".xlsx"):
        df = pd.read_excel(ruta_archivo)
    else:
        st.error("Formato de archivo no soportado.")
        df = None

    if df is not None:
        df.columns = df.columns.str.strip()
        st.success("Archivo cargado exitosamente.")
        st.write("Vista previa del DataFrame:")
        st.dataframe(df)

        anios_disponibles = extraer_anios_columnas(df.columns)

        if not anios_disponibles:
            st.warning("No se encontraron columnas con el formato esperado (<nombre>_<año>).")
        else:
            st.write("Años disponibles en los datos:", anios_disponibles)

            years_selected = st.multiselect("Selecciona los años para analizar", anios_disponibles)
            metric_options = ["ADMITIDOS", "GRADUADOS", "INSCRITOS", "MATRICULADOS", "PRIMER CURSO"]
            metric_selected = st.selectbox("Selecciona el tipo de dato a graficar", metric_options)
            chart_type = st.radio("Selecciona el tipo de gráfico", ["Barras", "Líneas de tendencia"])
```

```
            if not years_selected:
                st.error("Selecciona al menos un año para continuar.")
            else:
                filtered_columns = [
                    col for col in df.columns if
                    any(col.startswith(f"{metric_selected}_{year}") for year in years_selected)
                ]

                if not filtered_columns:
                    st.error("No se encontraron datos para la combinación seleccionada.")
                else:
                    filtered_df = df[filtered_columns]
                    st.write(f"Datos para {metric_selected} en los años seleccionados:")
                    st.dataframe(filtered_df)

                    try:
                        graph_data = filtered_df.sum().reset_index()
                        graph_data.columns = ['Column', 'Value']
                        graph_data['Year'] = graph_data['Column'].str.extract(r'_(\d{4})$')[0].astype(int)

                        fig, ax = plt.subplots(figsize=(12, 6))

                        if chart_type == "Barras":
                            grouped_data = graph_data.groupby("Year")["Value"].sum()
```

```
                            ax.bar(grouped_data.index, grouped_data.values, color='skyblue')
                            ax.set_title(f"Comparativa de {metric_selected} entre los años seleccionados")
                            ax.set_xlabel("Año")
                            ax.set_ylabel(metric_selected)
                        elif chart_type == "Líneas de tendencia":
                            graph_data = graph_data.sort_values("Year")
                            ax.plot(
                                *args: graph_data["Year"],
                                graph_data["Value"],
                                marker='o',
                                linestyle='-',
                                color='b',
                                label=f"Tendencia de {metric_selected}"
                            )
                            ax.set_title(f"Tendencia de {metric_selected} entre los años seleccionados")
                            ax.set_xlabel("Año")
                            ax.set_ylabel(metric_selected)
                            ax.legend(title="Datos")

                        plt.xticks(rotation=45, ha='right')
                        st.pyplot(fig)
                    except Exception as e:
                        st.error(f"Error al generar el gráfico: {e}")
except Exception as e:
```

```
            st.error(f"Error al procesar el archivo: {e}")
else:
    uploaded_file = st.file_uploader("Sube un archivo CSV o Excel", type=["csv", "xlsx"])

    if uploaded_file is not None:
        file_extension = uploaded_file.name.split(".")[-1].lower()

        try:
            if file_extension == "csv":
                df = pd.read_csv(uploaded_file, delimiter=";").dropna()
            elif file_extension == "xlsx":
                df = pd.read_excel(uploaded_file)
            else:
                st.error("Formato de archivo no soportado.")
                df = None

            if df is not None:
                df.columns = df.columns.str.strip()
                st.success("Archivo subido exitosamente.")
                st.write("Vista previa del DataFrame:")
                st.dataframe(df)

                anios_disponibles = extraer_anios_columnas(df.columns)
```

```
    if not anios_disponibles:
        st.warning("No se encontraron columnas con el formato esperado (<nombre>_<año>).")
    else:
        st.write("Años disponibles en los datos:", anios_disponibles)

        years_selected = st.multiselect("Selecciona los años para analizar", anios_disponibles)
        metric_options = ["ADMITIDOS", "GRADUADOS", "INSCRITOS", "MATRICULADOS", "PRIMER CURSO"]
        metric_selected = st.selectbox("Selecciona el tipo de dato a graficar", metric_options)
        chart_type = st.radio("Selecciona el tipo de gráfico", ["Barras", "Líneas de tendencia"])

        if not years_selected:
            st.error("Selecciona al menos un año para continuar.")
        else:
            filtered_columns = [
                col for col in df.columns if
                any(col.startswith(f"{metric_selected}_{year}") for year in years_selected)
            ]

            if not filtered_columns:
                st.error("No se encontraron datos para la combinación seleccionada.")
            else:
                filtered_df = df[filtered_columns]
                st.write(f"Datos para {metric_selected} en los años seleccionados:")
                st.dataframe(filtered_df)
```

```
        try:
            graph_data = filtered_df.sum().reset_index()
            graph_data.columns = ['Column', 'Value']
            graph_data['Year'] = graph_data['Column'].str.extract(r'_(\d{4})$')[0].astype(int)

            fig, ax = plt.subplots(figsize=(12, 6))

            if chart_type == "Barras":
                grouped_data = graph_data.groupby("Year")["Value"].sum()
                ax.bar(grouped_data.index, grouped_data.values, color='skyblue')
                ax.set_title(f"Comparativa de {metric_selected} entre los años seleccionados")
                ax.set_xlabel("Año")
                ax.set_ylabel(metric_selected)
            elif chart_type == "Líneas de tendencia":
                graph_data = graph_data.sort_values("Year")
                ax.plot(
                    *args: graph_data["Year"],
                    graph_data["Value"],
                    marker='o',
                    linestyle='-',
                    color='b',
                    label=f"Tendencia de {metric_selected}"
                )
                ax.set_title(f"Tendencia de {metric_selected} entre los años seleccionados")
```

```
                    )
                    ax.set_title(f"Tendencia de {metric_selected} entre los años seleccionados")
                    ax.set_xlabel("Año")
                    ax.set_ylabel(metric_selected)
                    ax.legend(title="Datos")

                plt.xticks(rotation=45, ha='right')
                st.pyplot(fig)
            except Exception as e:
                st.error(f"Error al generar el gráfico: {e}")
    except Exception as e:
        st.error(f"Error al procesar el archivo: {e}")
```

Por último, dejamos clases como el *View* para la visualización del menú y la clase *Settings* para el manejo de números mágicos.

Imágenes de View

```python
import streamlit as st
from SNIESController import SNIESController
from Settings import Settings
from gestores import GestorXlsx, GestorCsv, GestorJson


class View:  6 usages
    def __init__(self):
        self.controlador = SNIESController()

    @staticmethod  1 usage
    def is_convertible_to_int(value):
        try:
            int(value)
            return True
        except ValueError:
            return False

    def organizar_anios(self, anio1, anio2):  1 usage
        if int(anio2) < int(anio1):
            anio1, anio2 = anio2, anio1
        return anio1, anio2

    def validar_entrada_anio(self, anio):  2 usages
```

```python
def validar_entrada_anio(self, anio):  2 usages
    if View.is_convertible_to_int(anio):
        return True, int(anio)
    else:
        return False, None


def procesar_datos(self, anio1, anio2, palabra_clave, formato):  1 usage
    controlador = SNIESController()
    anio1, anio2 = View().organizar_anios(anio1, anio2)
    def_temp = controlador.procesar_datos(anio1, anio2, palabra_clave)

    if formato == "XLSX":
        gestor = GestorXlsx()
        ruta_salida = Settings.OUTPUTS_PATH_XLSX
    elif formato == "CSV":
        gestor = GestorCsv()
        ruta_salida = Settings.OUTPUTS_PATH_CSV
    elif formato == "JSON":
        gestor = GestorJson()
        ruta_salida = Settings.OUTPUTS_PATH_JSON
    else:
```

```python
    else:
        st.error("Formato no válido.")
        return False

    if gestor.crear_archivo(ruta_salida, def_temp):
        st.success(f"Datos exportados con éxito en el archivo: {ruta_salida}")
        return True
    else:
        st.error("Error al exportar los datos.")
        return False
```

Imágenes de Settings

```python
class Settings:  15 usages
    # Ruta base para los archivos
    BASE_PATH = "C:/Users/casti/Documents/AYIYIWUA/INPUTS/"

    # Rutas a los diferentes archivos
    ADMITIDOS_FILE_PATH = BASE_PATH + "Admitidos "
    GRADUADOS_FILE_PATH = BASE_PATH + "Graduados "
    INSCRITOS_FILE_PATH = BASE_PATH + "Inscritos "
    MATRICULADOS_FILE_PATH = BASE_PATH + "Matriculados "
    MATRICULADOS_PRIMER_CURSO_FILE_PATH = BASE_PATH + "Matriculados Primer Curso "
    OUTPUTS_PATH_XLSX = "C:/Users/casti/Documents/AYIYIWUA/OUTPUTS/datos_por_archivo.xlsx"
    OUTPUTS_PATH_CSV = "C:/Users/casti/Documents/AYIYIWUA/OUTPUTS/datos_por_archivo.csv"
    OUTPUTS_PATH_JSON = "C:/Users/casti/Documents/AYIYIWUA/OUTPUTS/datos_por_archivo.json"



    # Delimitador para archivos CSV
    DELIMITADOR = ';'

    # Constantes numéricas
    COLUMNAS_INFO_CONSOLIDADOS = 8
    DATOS_ACADEM_DEMOGRAF = 4
    COLUMNA_13 = 13
    COLUMNA_12 = 12
    FILAS_RESTANTES = 3
```

Imágenes de Main

```python
from view import View

def main():  1 usage

    try:
        menu = View()
        menu.mostrar_pantalla_bienvenido()
        menu.salir()
    except Exception as e:
        print(f"Se produjo un error inesperado: {e}")


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

**2.2 Funcionamiento del proyecto

¡Bienvenido al *SNIES Extractor*!

Siga las instrucciones a continuación para realizar el análisis y exportación de datos.

Parametrización Inicial

Asegúrese de que la carpeta SNIES_EXTRACTOR esté correctamente configurada con los archivos necesarios.

Confirmar parametrización

Confirmar parametrización

Selección de parámetros

Escriba el primer año de búsqueda:

2020

Escriba el segundo año de búsqueda:

2021

Ingrese una palabra clave para filtrar los datos:

ZOOTECNIA

Seleccione el formato de archivo para exportar los datos:

XLSX

Procesar y Exportar

Seleccione el formato de archivo para exportar los datos:

XLSX

Procesar y Exportar

Datos exportados con éxito en el archivo:
C:/Users/casti/Documents/AYIYIWUA/OUTPUTS/datos_por_archivo.xlsx

Cierre del programa

Salir

Siga las instrucciones a continuación para generar los análisis deseados

¿Cómo deseas cargar el archivo?

- (•) Desde el sistema local
- ( ) Desde el navegador

Selecciona un archivo

datos_por_archivo.xlsx

Siga las instrucciones a continuación para generar los análisis deseados

¿Cómo deseas cargar el archivo?

- (•) Desde el sistema local
- ( ) Desde el navegador

Selecciona un archivo

datos_por_archivo.xlsx

datos_por_archivo.xlsx

Archivo cargado exitosamente.

Vista previa del DataFrame:

	CÓDIGO SNIES DEL PROGRAMA	METODOLOGÍA	PROGRAMA ACADÉMICO	INSTITUCIÓN DE
0	3	Presencial	ZOOTECNIA	UNIVERSIDAD NA
1	3	Presencial	ZOOTECNIA	UNIVERSIDAD NA

Vista previa del DataFrame:

	CÓDIGO SNIES DEL PROGRAMA	METODOLOGÍA	PROGRAMA ACADÉMICO	INSTITUCIÓN DE E
0	3	Presencial	ZOOTECNIA	UNIVERSIDAD NAC
1	3	Presencial	ZOOTECNIA	UNIVERSIDAD NAC
2	143	Presencial	ZOOTECNIA	UNIVERSIDAD NAC
3	143	Presencial	ZOOTECNIA	UNIVERSIDAD NAC
4	281	Presencial	MEDICINA VETERINARIA Y ZOOTECNIA	UNIVERSIDAD DE C
5	281	Presencial	MEDICINA VETERINARIA Y ZOOTECNIA	UNIVERSIDAD DE C
6	281	Presencial	MEDICINA VETERINARIA Y ZOOTECNIA	UNIVERSIDAD DE C
7	281	Presencial	MEDICINA VETERINARIA Y ZOOTECNIA	UNIVERSIDAD DE C
8	281	Presencial	MEDICINA VETERINARIA Y ZOOTECNIA	UNIVERSIDAD DE C
9	281	Presencial	MEDICINA VETERINARIA Y ZOOTECNIA	UNIVERSIDAD DE C

	INSTITUCIÓN DE EDUCACIÓN SUPERIOR (IES)	PRINCIPAL O SECCIONAL	DEPARTAMENTO DE OFERTA DEL PR(
0	UNIVERSIDAD NACIONAL DE COLOMBIA	Principal	Bogotá D.C.
1	UNIVERSIDAD NACIONAL DE COLOMBIA	Principal	Bogotá D.C.
2	UNIVERSIDAD NACIONAL DE COLOMBIA	Seccional	Valle del Cauca
3	UNIVERSIDAD NACIONAL DE COLOMBIA	Seccional	Valle del Cauca
4	UNIVERSIDAD DE CALDAS	Principal	Caldas
5	UNIVERSIDAD DE CALDAS	Principal	Huila
6	UNIVERSIDAD DE CALDAS	Principal	Nariño
7	UNIVERSIDAD DE CALDAS	Principal	Putumayo
8	UNIVERSIDAD DE CALDAS	Principal	Tolima
9	UNIVERSIDAD DE CALDAS	Principal	Valle del Cauca

Selecciona los años para analizar

2020 × 2021 ×

Selecciona el tipo de dato a graficar

ADMITIDOS

ADMITIDOS

GRADUADOS

INSCRITOS

MATRICULADOS

PRIMER CURSO

0	71	74
9	1	62

Selecciona el tipo de dato a graficar

INSCRITOS

Selecciona el tipo de gráfico

- (•) Barras
- ( ) Líneas de tendencia

Datos para INSCRITOS en los años seleccionados:

	INSCRITOS_2020	INSCRITOS_2021
0	71	74
1	67	64
2	81	71
3	72	65
4	389	62
5	389	62
6	389	62

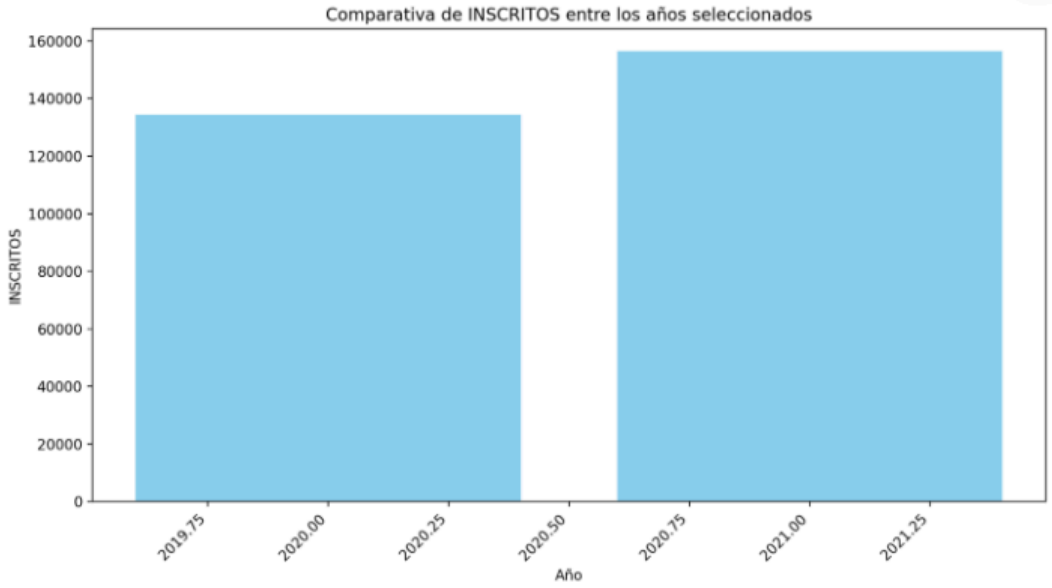
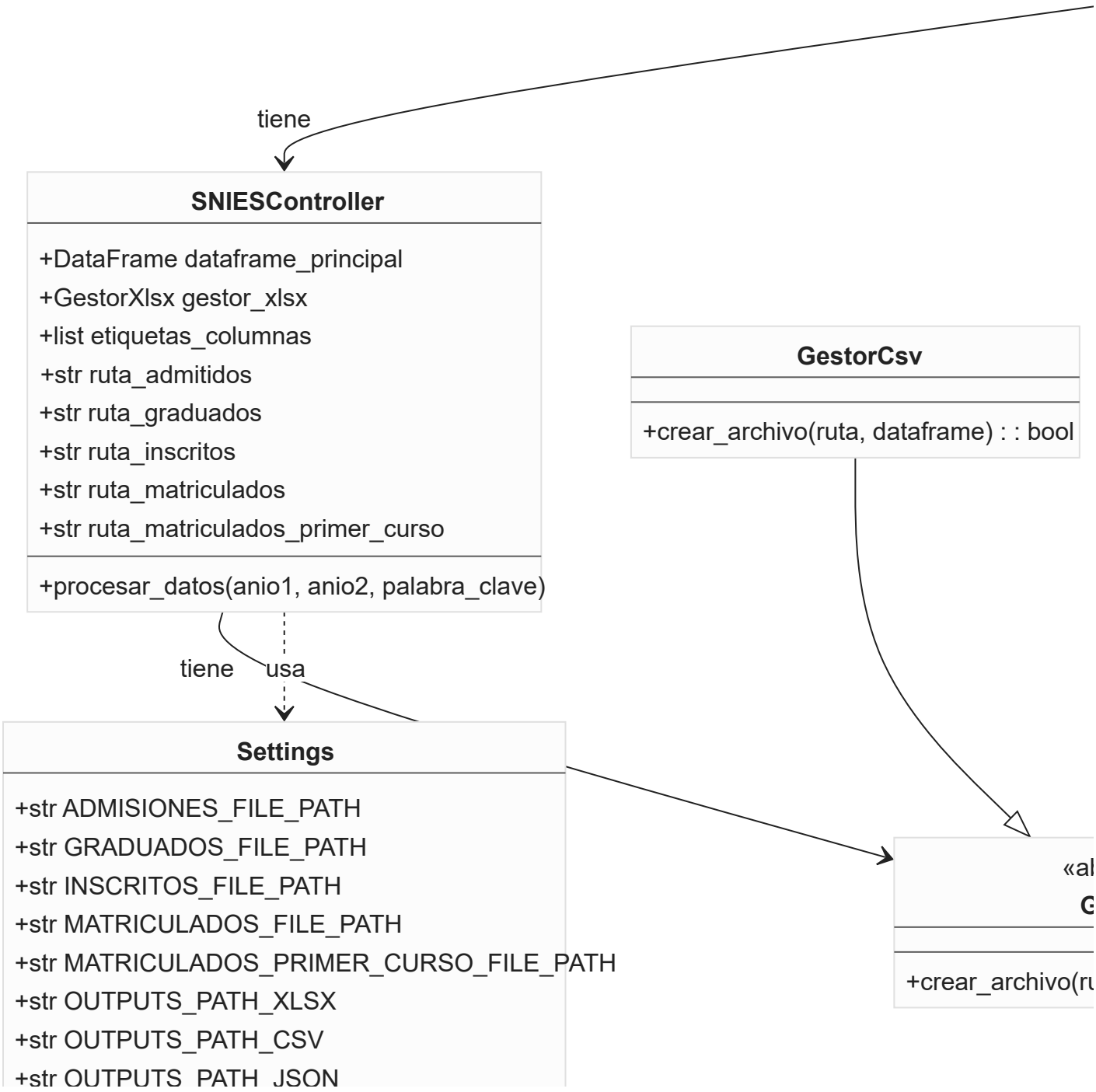


**Diagrama UML:


tiene
SNIESController
+DataFrame dataframe_principal
+GestorXlsx gestor_xlsx
+list etiquetas_columnas
+str ruta_admitidos
+str ruta_graduados
+str ruta_inscritos
+str ruta_matriculados
+str ruta_matriculados_primer_curso
+procesar_datos(anio1, anio2, palabra_clave)
GestorCsv
+crear_archivo(ruta, dataframe) : : bool
tiene
usa
Settings
+str ADMISIONES_FILE_PATH
+str GRADUADOS_FILE_PATH
+str INSCRITOS_FILE_PATH
+str MATRICULADOS_FILE_PATH
+str MATRICULADOS_PRIMER_CURSO_FILE_PATH
+str OUTPUTS_PATH_XLSX
+str OUTPUTS_PATH_CSV
+str OUTPUTS_PATH_JSON
+crear_archivo(ru