SPRINT 10

Análisis de datos en Pandas

Descripció

En aquesta tasca t'enfrontaràs a un exercici de neteja i analítica de dades.

Tenim un dataset provinent d'una enquesta als nostres treballadors i treballadors, i hem de garantir que les dades es processes correctament, tant en format com en llegibilitat.

Tascas Nivel 1

Desarrollo:

Procesamiento y limpieza de datos

En esta fase se realiza el tratamiento inicial del dataset utilizando Pandas.

Primero se importa el archivo **sprint10.xlsx** verificando que los encabezados e índices se lean correctamente y validando la unicidad del DNI como identificador.

Se generan nuevas columnas derivadas, como el nombre completo y una variable booleana para identificar nacionalidad española. También se normalizan las categorías de género mediante una función personalizada y se limpia la columna de salarios eliminando símbolos y formateos para convertirla en valores numéricos.

La redundancia entre las columnas “Hijos” y “Sin hijos” se resuelve unificándolas en una sola categoría (“Sí”/“No”) mediante una función aplicada por filas.

Posteriormente se realiza un análisis descriptivo con **groupby y crosstab** para obtener estadísticas de salario por género y país, complementado con un mapa de calor de Seaborn para visualizar las diferencias salariales.

Finalmente, se combinan las columnas de día, mes y año en un objeto **datetime**, lo que permite calcular la edad exacta de cada trabajador comparando la fecha de nacimiento con la fecha actual.

Nivell 1

1.

- Importa com un DataFrame l'arxiu sprint10.xlsx. Assegura't que el fitxer s'importa correctament, amb els noms de columnes que li corresponen, sense manipular l'arxiu original.
- Ordena el DataFrame pel país d'origen. En cas d'empat, ordena pel nom de la ciutat.
- Mostra les primeres 10 files.

Addicionalment, fes un *print* on comprovi que el DNI només té valors únics.

Codigos:

```
# Librerías y módulos para la ejecución de las tareas

import pandas as pd
import numpy as np
import seaborn as sns
from datetime import datetime, date, timedelta


# a. Importación: Los encabezados están en la fila 2 y el índice corresponde a "Unnamed: 0"

df = pd.read_excel(r"Archivos\sprint10.xlsx",header= 3,index_col = "Unnamed: 0")


# b. Control genérico del DataFrame importado

df.info()
print("\n" + "-"*80 + "\n")

display(round(df.describe()),2)

print("\n" + "-"*80 + "\n")
df


# Creo una copia de seguridad, además de realizar algunas limpiezas básicas para evitar errores en las próximas tareas

df_c = df.copy()

df_c.columns = df_c.columns.str.strip()

df_c = df_c.dropna(how="all")


# c. Ordeno el DataFrame por país de origen y ciudad

df_c.sort_values(by =["País d'origen","Ciutat"], ascending= True)


# d. Primera 10 filas

df_c.head(10)


# e. DNI únicos

if len(df_c) == df_c["DNI"].nunique():
    print("Correcto, en la tabla hay DNI unicos!")
print("\n" + "-"*40 + "\n")
print("Combrobación calculos.\n\nTotales filas de la tablas: \n",len(df_c),"\nTotales DNI unicos: \n",df_c["DNI"].nunique())
```

2.

- Crea una columna que sigui el nom complet.
- Crea una columna si la persona és nascuda a Espanya o no.
- Posa el DNI com a índex del DataFrame (noms de files).
- Substitueix el nom de les columnes Dia de Naixement, Mes de Naixement i Any de Naixement per Dia, Mes i Any.
- Substitueix H per Home, D per Dona, A per Altres i NC per una dada faltant (nan/null/na).

Mostra tots els canvis que has realitzat en una sola taula.

Codigos:

```python
# a. Nueva columna: "Nom complet"

df_c["Nom_complet"] = df["Nom"] +" "+ df["Cognoms"]
```

```python
# b. Nueva columna: "Nacionalitat espanyola"

df_c["Nacionalitat espanyola"] = df_c["Pais d\'origen"].apply(lambda x : "Sí" if x == "Espanya" else "No")
```

```python
# c. Defino el índice como "DNI" tras eliminar los espacios del nombre de la column

df_c = df_c.reset_index(drop=True).set_index("DNI")

df_c
```

```python
# d. Cambio de nombres de columnas

df_c = df_c.rename(columns={"Dia de Naixement":"Dia","Mes de Naixement":"Mes","Any de Naixement":"Any"})
df_c
```

```python
# e. Sustituye "H por Home", "D por "Dona", "A por Altres" y "NC por un dato faltante "NaN"

def cambios_col_gènere(x):

    if x == "H":
        return "Home"
    elif x == "D":
        return "Dona"
    elif x == "A":
        return "Altres"
    else:
        return np.nan


df_c["Gènere"] = df_c["Gènere"].apply(lambda x : cambios_col_gènere(x))
```

```python
# f. Muestro los cambios realizados

df_c
```

```python
# Control genérico de los cambios

df_c.info()
print("\n" + "-"*80 +"\n")
```

3.

- Junta les columnes Fills i No Fills en una sola columna, utilitzant el mètode .apply() i definint una funció que resolgui el problema. La columna nova ha de dir-se "Fills" i prendre els valors "Sí" o "No".

Codigos:

```python
# Preparo los datos

df_c["Fills"] = df_c["Fills"].fillna(0)
df_c["No Fills"] = df_c["No Fills"].fillna(0)

# Función

def fun(fila):

    if fila["Fills"] > fila["No Fills"]:
        return "Si"
    elif fila["Fills"] < fila["No Fills"]:
        return "No"
    else:
        return np.nan

# Ejecución

df_c["N_Fills"] = df_c.apply(fun,axis = 1)

df_c
```

```python
# Elimino columnas innecesarias y renombro la nueva columna "Fills"

del df_c["Fills"]
del df_c["No Fills"]

df_c.rename(columns = {'N_Fills':'Fills'})
```

```python
# Control valores "NaN" , hay 60 en columna "Gènere"

df_c.isna().sum()
```

4.

- Crea una taula resum que permeti veure el sou mig, medià, mínim i màxim per Gènere.
- Ordena la taula en funció del sou mig.

Codigos:

```
# Preparo columna "Salari mensual": Replace(),Strip(),Astype()
df_c["Salari mensual"] = df_c["Salari mensual"].str.replace("€","")
df_c["Salari mensual"] = df_c["Salari mensual"].str.replace(".","")
df_c["Salari mensual"] = df_c["Salari mensual"].str.strip()
df_c["Salari mensual"] = df_c["Salari mensual"].astype("i")
df_c
```

```
# Groupby y Agg
df_c.groupby("Gènere")["Salari mensual"].agg(Mitjana="mean",Mediana = "median", Mínim= "min",Màxim= "max").round(2)
```

5.

- Crea una taula resum amb el salari mig per gènere (files) i país d'origen (columnes).
- Afegeix-hi les mitjanes als marges de la taula.

(EXTRA): Aplica format condicional a la taula per veure en un color més intens els valors més elevats

Codigos:

```
# a. Uso Crosstable()
pd.crosstab(df_c["Gènere"],df_c["País d'origen"],df_c["Salari mensual"], aggfunc = "mean").round(2)
```

```
# b. Añado los márgenes y renombro la fila y columna "All" como "Tothom"
tabla = pd.crosstab(df_c["Gènere"],df_c["País d'origen"],df_c["Salari mensual"],margins=True, aggfunc = "mean").round(2)
tabla = tabla.rename(index={"All": "Tothom"}, columns={"All": "Tothom"})
tabla
```

```
# Extra: genero un heatmap con Seaborn como alternativa al formato condicional tradicional
tabla = pd.crosstab(df_c["País d'origen"],df_c["Gènere"],df_c["Salari mensual"], aggfunc = "mean")
sns.heatmap(tabla,cmap="YlGnBu",annot=True,cbar=False,fmt=".0f")
```

6.

- Crea una columna nova que sigui la data de naixament en format Datetime a partir de les columnes dia, mes i any. Utilitzant aquesta columna crea una funció que donada una data, et calculi l'edat actual a dia d'avui.
- Utilitza la funció que acabes de crear per generar una columna nova al DataFrame amb l'edat actual.

Codigos:

```python
# a. Nueva columna "Data de naixament" en formato datetime

df_c["Data de naixament"] = df_c.apply(lambda fila : datetime(fila["Any"],fila["Mes"],fila["Dia"]), axis= 1)

df_c
```

```python
# b. Funciòn

def calculo_edat(fila):

    hoy = datetime.now()

    edad = hoy.year - fila.year

    if (hoy.month,fila.month) < (hoy.day,fila.month):
        edad -= 1

    return edad
```

```python
# c. Nueva columna "Edat"

df_c["Edat"] = df_c["Data de naixament"].apply(calculo_edat)

df_c
```

```
(function) def calculo_edat(fila: Any) -> Any
```

Tascas Nivel 2:

Nivell 2

1.
   - Utilitzant el següent DataFrame, adjunta la columna "Increment" al dataframe del nivell anterior.
   - Actualitza la columna salari en funció dels percentatges que s'adjunten. No modifiquis manualment els increments, escriu codi Python per fer les conversions necessàries.

```
df_increment = pd.DataFrame({"Grup":["Grup A","Grup B","Grup C", "Grup D" ] ,
"Increment":

    ["5%","3,5%","2%","8%"]})
```

Desarrollo:

En este segundo nivel se incrementa la complejidad del proyecto incorporando fuentes externas y automatizando procesos clave. Primero se enriquece el dataset mediante un left join que integra un DataFrame adicional con incrementos salariales basados en el “Grupo Profesional”.
A continuación, se implementa una actualización dinámica de salarios utilizando una estructura de control tipo match, que aplica automáticamente distintos porcentajes de aumento según el grupo del empleado, evitando cálculos manuales.
Se automatiza también la exportación de resultados mediante bucles for que filtran el DataFrame por grupo profesional y generan archivos Excel individuales para cada uno.
Finalmente, se construye un reporte ejecutivo que resume métricas esenciales —número de trabajadores, salario medio actualizado y mediana de edad— por grupo profesional, y se exporta para su uso administrativo.

Codigos:

```
# a.Creo nuevo dataframe y lo uno mediante la funcion merge() y borro la columna que sobra

df_increment = pd.DataFrame({"Grup":["Grup A","Grup B","Grup C", "Grup D"],
                             "Increment":["5%","3,5%","2%","8%"]})

df_increment = df_increment.rename(columns={"Grup":"Grup Professional"})

nuevo_df = df_c.merge(df_increment, left_on="Grup Professional", right_on="Grup Professional", how="left")
nuevo_df.index = df_c.index

nuevo_df
```

```
# b. Creo una función para actualizar los salarios "Salari mensual" en base al incremento.

def incremento_salario(fila):

    grupos = fila["Grup Professional"]

    match grupos:
        case "Grup A":
                return round(fila["Salari mensual"]*1.05)
        case "Grup B":
                return round(fila["Salari mensual"]*1.035)
        case "Grup C":
                return round(fila["Salari mensual"]*1.02)
        case "Grup D":
                return round(fila["Salari mensual"]*1.08)


nuevo_df["Salari mensual actualitzat"] = nuevo_df.apply(incremento_salario, axis = 1)

nuevo_df
```

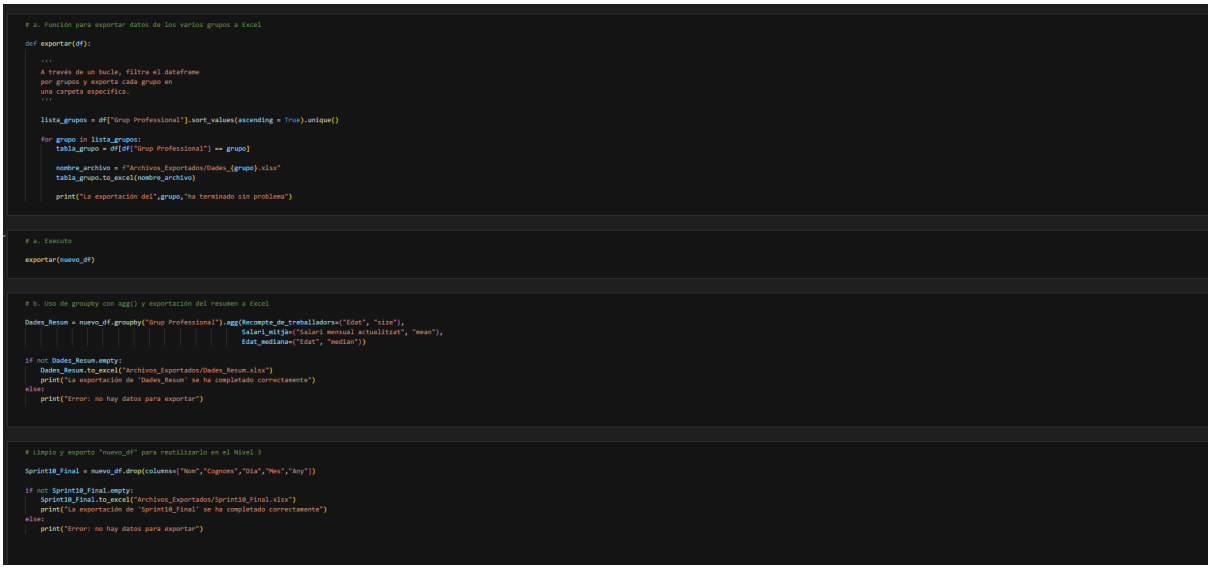
2.

Utilitzant un bucle, exporta en 4 fitxers (format .xlsx o .csv) les dades de cada Grup Professional.

Per exemple: "dades_GrupA.xlsx" , "dades_GrupB.xlsx" ...

Exporta un 5è DataFrame en format .xlsx o .csv que contingui quants treballadors hi ha per cada Grup Professional, quin és el seu sou mig i quina és la seva edat mediana.

Codigos:



```python
# a. Función para exportar datos de los varios grupos a Excel

def exportar(df):
    '''
    A través de un bucle, filtra el dataframe
    por grupos y exporta cada grupo en
    una carpeta específica.
    '''
    lista_grupos = df["Grup Professional"].sort_values(ascending = True).unique()

    for grupo in lista_grupos:
        tabla_grupo = df[df["Grup Professional"] == grupo]

        nombre_archivo = f"Archivos_Exportados/Dades_{grupo}.xlsx"
        tabla_grupo.to_excel(nombre_archivo)

        print("La exportación del",grupo,"ha terminado sin problema")
```

```python
# a. Executo

exportar(nuevo_df)
```

```python
# b. Uso de groupby con agg() y exportación del resumen a Excel

Dades_Resum = nuevo_df.groupby("Grup Professional").agg(Recompte_de_treballadors=("Edat", "size"),
                                                        Salari_mitjà=("Salari mensual actualitzat", "mean"),
                                                        Edat_mediana=("Edat", "median"))

if not Dades_Resum.empty:
    Dades_Resum.to_excel("Archivos_Exportados/Dades_Resum.xlsx")
    print("La exportación de 'Dades_Resum' se ha completado correctamente")
else:
    print("Error: no hay datos para exportar")
```

```python
# Limpio y exporto "nuevo_df" para reutilizarlo en el Nivel 3

Sprint10_Final = nuevo_df.drop(columns=["Nom","Cognoms","Dia","Mes","Any"])

if not Sprint10_Final.empty:
    Sprint10_Final.to_excel("Archivos_Exportados/Sprint10_Final.xlsx")
    print("La exportación de 'Sprint10_Final' se ha completado correctamente")
else:
    print("Error: no hay datos para exportar")
```

Tascas Nivel 3:

Nivell 3

El nivell 3 d'aquest sprint és totalment diferent a d'altres sprints que has fet fins ara, ja que són exercicis més abstractes que requereixen barallar-s'hi bastant. No continuen amb el mateix dataset dels nivells anteriors, sinó que et plantegen dues situacions noves totalment diferents entre elles.

1.

Crea una funció que prengui un dataframe com a paràmetre d'entrada.

La funció ha de crear (i exportar) un gràfic automàticament per a cada columna del dataframe. Per exemple:

- un histograma/boxplot si la variable és numèrica
- unes barres dels valors més freqüents si és categòrica
- unes barres dels anys més freqüents si la dada està en format data.

La idea és crear una funció que funcioni per **qualsevol** dataframe, no només amb el que hem treballat fins ara.

Mostra el resultat de la funció en algun dels datasets d'exemple que conté el paquet seaborn. Per exemple, *iris*, *penguins* o *titanic*.

Tingues en consideració que en el següent sprint treballaràs exclusivament amb gràfics. L'objectiu d'aquest exercici no és crear gràfics molt elaborats, sinó resoldre una necessitat de manera ràpida i automàtica.

Desarrollo:

El tercer nivel se centra en resolver problemas más abstractos mediante programación avanzada. En primer lugar, se diseña una función genérica, **generar_graficos()**, capaz de analizar cualquier DataFrame.

Esta función realiza un preprocesamiento automático —limpieza de filas vacías, normalización de nombres de columnas y tratamiento de valores nulos mediante mediana o moda— y aplica una lógica inteligente que detecta el tipo de dato para generar el gráfico más adecuado (histogramas, gráficos de tarta o barras), guardando cada visualización de forma automática.

Además, se aborda un problema algorítmico clásico: la optimización de rutas mediante un algoritmo. Tras limpiar la matriz de distancias entre ciudades españolas, se implementa una función que selecciona iterativamente la ciudad no visitada más cercana. Finalmente, se amplía el análisis evaluando todas las ciudades como posibles puntos de partida para identificar la ruta global más corta para recorrer España por carretera.

Codigos:

```python
# Librerías y módulos esenciales para el desarrollo de las tareas

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

from pandas.api.types import is_numeric_dtype
from pandas.api.types import is_datetime64_any_dtype
```

```python
# Cargo dataset "Penguins" proporcionado por Seaborn

penguins = sns.load_dataset("penguins")
penguins
```

```python
# Verifico formatos columnas

penguins.info()
```

```python
def generar_graficos(df):

    '''
    Limpia el dataframe de forma genérica para evitar errores.

    - Elimina filas completamente vacías
    - Normaliza los nombres de las columnas
    - Elimina columnas con más del 75% de valores NaN
    - Rellena autosáticamente los valores NaN:
            · Mediana para columnas numéricas
            · Moda para columnas categóricas
            · Mediana para columnas Fechas
    '''

    df = df.dropna(how="all")

    df.columns = (df.columns.str.strip().str.replace(" ", "").str.lower())

    for columna in df.columns:
        if df[columna].isna().sum() == 0:
            continue

        if is_datetime64_any_dtype(df[columna]):
            df[columna] = df[columna].fillna(df[columna].median())

        elif is_numeric_dtype(df[columna]):
            df[columna] = df[columna].fillna(df[columna].median())
        else:
            moda = df[columna].mode(dropna=True)
            if not moda.empty:
                df[columna] = df[columna].fillna(moda.iloc[0])
            else:
                df[columna] = df[columna].fillna("")

    '''
```

```python
    '''
    Genera automáticamente gráficos según el tipo de columna del dataframe.

    Numéricas:
    - Pie si tienen ≤ 2 valores únicos
    - Boxplot si tienen entre 3 y 15 valores únicos
    - Histograma si tienen > 15 valores únicos

    Categóricas/objeto:
    - Pie si tienen ≤ 3 categorías
    - Barra si tienen más de 3

    Fechas:
    - Barra por años

    Muestra cada gráfico y lo exporta a una carpeta específica.
    '''

    for columna in df.columns:
        serie = df[columna]
        valores_unicos = serie.nunique()

        if is_datetime64_any_dtype(serie):
            plt.figure(figsize=(10,4))
            plt.title(columna)
            plt.ylabel("count")
            años = serie.dt.year.value_counts().sort_index()
            años.plot.bar()
            plt.xlabel(None)

        elif is_numeric_dtype(serie):

            if valores_unicos <= 15:
                if valores_unicos <= 2:
                    plt.figure(figsize=(6,6))
                    serie.value_counts().plot.pie(autopct="%1.1f%%")
                    plt.title(columna)
                    plt.ylabel(None)
                    plt.xlabel(None)
                else:
                    plt.figure(figsize=(6,4))
                    serie.plot.box()
                    plt.title(columna)
                    plt.ylabel(None)
                    plt.xticks([])
            else:
                plt.figure(figsize=(6,4))
                serie.plot.hist(bins=30)
                plt.title(columna)
                plt.ylabel("Frequency")

        elif isinstance(serie.dtype, pd.CategoricalDtype) or serie.dtype == object:

            if valores_unicos <= 3:
                plt.figure(figsize=(6,4))
                plt.title(columna)
                serie.value_counts().plot.pie(autopct="%1.1f%%")
                plt.ylabel(None)
                plt.ylabel(None)
            else:
                plt.figure(figsize=(6,4))
                plt.title(columna)
                plt.ylabel("count")
                serie.value_counts().plot.bar()
                plt.xlabel(None)


        else:
            print("Error")

        plt.savefig(f"Graficos_Exportados/{columna}.png", dpi=200, bbox_inches="tight")
        plt.show()
        plt.close()
```

2.

Carrega l'arxiu matriu_distancies.xlsx a pandas, de manera que els noms de files i els noms de columnes siguin els de les ciutats. Borra "Las Palmas de Gran Canaria" i "Palma" perquè poguem fer el trajecte en cotxe.

Font: Mejores Rutas

Ens interessa visitar totes les ciutats principals d'Espanya recorrent la mínima distància possible.

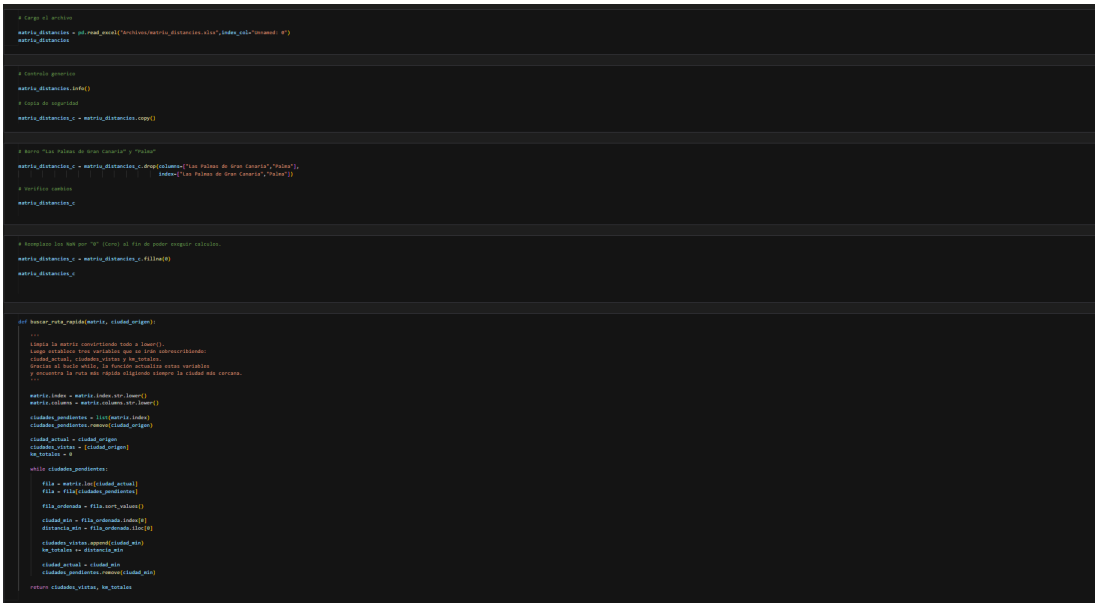
No cal que ho facis de forma òptima, ens interessa que desenvolupis una solució raonable utilitzant les eines que tens actualment.

Per exemple, una aproximació senzilla (que no òptima) seria anant sempre a la ciutat més propera que no haguem visitat encara

Fes una funció que donada la matriu de distàncies i la ciutat d'origen, faci una proposta de ruta que sigui el més curta possible que puguis, retornant una llista amb l'ordre de visita. Dóna també la distància total recorreguda.

(EXTRA) Des de quina ciutat la ruta seria més curta amb l'algoritme plantejat

Codigos:



```python
# Cargo el archivo

matriu_distancies = pd.read_excel("Archivos/matriu_distancies.xlsx",index_col="Unnamed: 0")
matriu_distancies
```

```python
# Controlo generico

matriu_distancies.info()

# Copia de seguridad

matriu_distancies_c = matriu_distancies.copy()
```

```python
# Borro "Las Palmas de Gran Canaria" y "Palma"

matriu_distancies_c = matriu_distancies_c.drop(columns=["Las Palmas de Gran Canaria","Palma"],
                                               index=["Las Palmas de Gran Canaria","Palma"])

# Verifico cambios

matriu_distancies_c
```

```python
# Reemplazo los NaN por "0" (Cero) al fin de poder eveguir calculos.

matriu_distancies_c = matriu_distancies_c.fillna(0)

matriu_distancies_c
```

```python
def buscar_ruta_rapida(matriz, ciudad_origen):
    '''
    Limpia la matriz convirtiendo todo a lower().
    Luego establece tres variables que se irán sobreescribiendo:
    ciudad_actual, ciudades_vistas y km_totales.
    Gracias al bucle while, la función actualiza estas variables
    y encuentra la ruta más rápida eligiendo siempre la ciudad más cercana.
    '''

    matriz.index = matriz.index.str.lower()
    matriz.columns = matriz.columns.str.lower()

    ciudades_pendientes = list(matriz.index)
    ciudades_pendientes.remove(ciudad_origen)

    ciudad_actual = ciudad_origen
    ciudades_vistas = [ciudad_origen]
    km_totales = 0

    while ciudades_pendientes:

        fila = matriz.loc[ciudad_actual]
        fila = fila[ciudades_pendientes]

        fila_ordenada = fila.sort_values()

        ciudad_min = fila_ordenada.index[0]
        distancia_min = fila_ordenada.iloc[0]

        ciudades_vistas.append(ciudad_min)
        km_totales += distancia_min

        ciudad_actual = ciudad_min
        ciudades_pendientes.remove(ciudad_min)

    return ciudades_vistas, km_totales
```

```python
# Ejecución, pregunto al usuario desde qué ciudad quiere empezar, garantizo que la entrada esté en lower() y uso un while hasta que coincida con las ciudades del dataset

while True:

    origen = input("Hola, me encantaría ayudarte a encontrar la mejor ruta para visitar toda España. ¿Desde qué ciudad quieres partir?").lower().strip()

    if origen in list(matriu_distancies_c.index.str.lower()) :
        break
    else:
        print("Lo siento, esta ciudad no está en nuestro sistema de datos.\nElliges entre: \n"
              "Barcelona, Hospitalet de Llobregat, Zaragoza, Valencia, Alicante, Murcia, Córdoba, Sevilla, Málaga, Valladolid, Gijón, Bilbao, Vigo")

resultado = buscar_ruta_rapida(matriu_distancies_c,origen)

ruta,km = resultado

print("La ruta mas rapida partiendo por (origen), es la siguiente :\n")
display(ruta)
print("\n" + "-"*30 + "\n")
print("El total de km percorsos son :",km)
```

```python
# Extra, encuentro ruta más rápida creando otra función y aprovechando la anterior

def mejor_ruta():
    '''
    Recorre todas las ciudades como posibles orígenes, calcula la ruta rápida
    para cada una y muestra los resultados. Guarda el origen con menor distancia
    total y devuelve la mejor ciudad, la ruta y los kilómetros.
    '''

    mejor_resultado = 100000000
    ruta_final = ""
    mejor_origen = ""

    print("Aquí encontramos todos los posibles resultados:\n\n")

    for ciudad in list(matriu_distancies_c.index):

        ruta,km = buscar_ruta_rapida(matriu_distancies_c,ciudad)
        destino = ruta[-1]

        print("Origen:",ciudad,"\nDestino:",destino,"\nKilómetros totales:",km)
        print("\n" + "-"*30 + "\n")

        if km < mejor_resultado:
            mejor_resultado = km
            ruta_final = ruta
            mejor_origen = ciudad

    return mejor_origen, ruta_final, mejor_resultado
```

```python
# Ejecución final

origen, ruta_final, mejor_resultado = mejor_ruta()

print("Resumen búsqueda del viaje por carretera más rápido:\n")

print("La ciudad desde donde empezar es:\n", origen.upper())
print("\n")
print("La ruta a seguir es la siguiente:\n", ruta_final)
print("\n")
print("Total de kilómetros:\n", mejor_resultado)
```

Visualización de la ruta con Folium

Desarrollo:

Para representar gráficamente los resultados obtenidos previamente, utilicé la librería **Folium**, la cual permite generar mapas interactivos directamente desde Python. Tras importar la librería, construí un mapa base utilizando las coordenadas calculadas en el análisis anterior. Sobre este mapa definí una función encargada de trazar la ruta: coloca marcadores numerados en cada punto del recorrido y dibuja una línea que conecta todas las ciudades en el orden correspondiente. Esta visualización facilita interpretar la secuencia del trayecto y verificar la coherencia de la solución obtenida. Finalmente, el mapa se muestra en pantalla y se exporta a una carpeta específica para su almacenamiento y consulta posterior.

Codigos:

```python
def mapa_ruta(ruta,coordenadas):
    '''
    Genera un mapa con Folium que muestre la ruta calculada:
    coloca un marcador numerado en cada ciudad y dibuja una
    línea roja que conecte todos los puntos de la ruta.
    En fin, lo muestra y lo exporta a una carpeta específica.
    '''

    lat0, lon0 = coordenadas[ruta[0]]
    mapa = folium.Map(location=[lat0, lon0], zoom_start=6)

    puntos = []

    for i, ciudad in enumerate(ruta):
        lat, lon = coordenadas[ciudad]
        puntos.append((lat, lon))

        folium.Marker(location=[lat, lon],
                      popup=f"{i+1}. {ciudad.title()}",
                      tooltip=f"Parada {i+1}",
                      icon=folium.Icon(color="blue")).add_to(mapa)

    folium.PolyLine(puntos,
                    color="red",
                    weight=4,
                    opacity=0.8).add_to(mapa)

    mapa.save(f"Mapas_Exportados/{ciudad}.html")

    return mapa
```

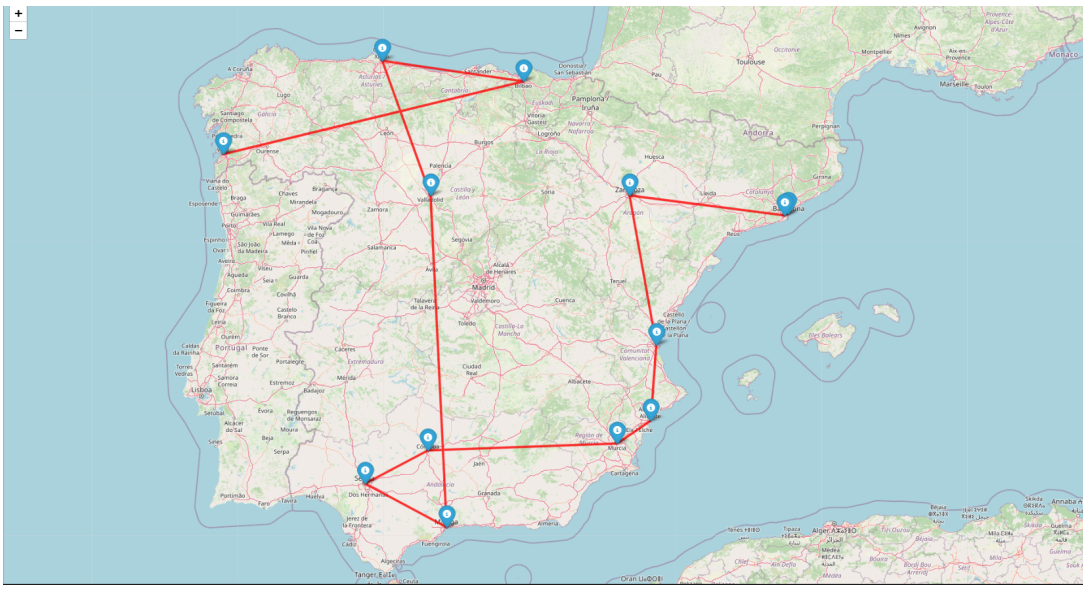
Python

```python
# Necesito reutilizar una parte del código anterior y combinarlo con la nueva función:

while True:

    origen = input("Hola, me encantaría ayudarte a encontrar la mejor ruta para visitar toda España. ¿Desde qué ciudad quieres partir?").lower().strip()

    if origen in list(matriu_distancies_c.index.str.lower()) :
        break
    else:
        print("Lo siento, esta ciudad no está en nuestro sistema de datos.\nElliges entre: \n"
              "Barcelona, Hospitalet de Llobregat, Zaragoza, Valencia, Alicante, Murcia, Córdoba, Sevilla, Málaga, Valladolid, Gijón, Bilbao, Vigo")

resultado = buscar_ruta_rapida(matriu_distancies_c,origen)

ruta,km = resultado

mapa_ruta(ruta,mapa_coords)
```

Python

```python
# Ruta mas rapida visualización con mapa:

origen, ruta_final, mejor_resultado = mejor_ruta()

print("Resumen búsqueda del viaje por carretera más rápido:\n")

print("La ciudad desde donde empezar es:\n", origen.upper())
print("\n")
print("La ruta a seguir es la siguiente:\n", ruta_final)
print("\n")
print("Total de kilómetros:\n", mejor_resultado)

mapa_ruta(ruta_final,mapa_coords)
```

Python



EXTRA:

Tarea 2 con NumPy Random

Aquí presento una segunda versión de la Tarea 2, algo más lenta en tiempo de búsqueda pero muy interesante. Esta variante utiliza el módulo numpy.random para generar rutas aleatorias y seleccionar la mejor entre ellas. Para usar la función **mejor_ruta**, recomiendo fijar un número de iteraciones en **ruta_rapida_random** menor o igual a 100 000, a menos que dispongas de bastante tiempo. Aun así, cuanto mayor sea el número de iteraciones, más precisa será la solución obtenida.

Codigo:

```python
import numpy as np

def ruta_rapida_random(matriz, ciudad_origen,numero_iteraciones):
    '''
    Genera un número configurable de rutas aleatorias.
    En cada iteración crea una ruta casual de las ciudades
    pendientes y calcula su distancia total. Gracias al break,
    descarta rápido las rutas peores que la mejor encontrada.
    En fin devuelve la ruta más corta y los km.
    '''


    matriz.index = matriz.index.str.lower()
    matriz.columns = matriz.columns.str.lower()
    ciudad_origen = ciudad_origen.lower()

    ciudades_pendientes = list(matriz.index)
    ciudades_pendientes.remove(ciudad_origen)

    rng = np.random.default_rng()

    mejor_ruta = None
    mejor_distancia = 100000000


    for _ in range(numero_iteraciones):
        ruta = [ciudad_origen]

        ruta_random = list(ciudades_pendientes)
        rng.shuffle(ruta_random)

        ruta.extend(ruta_random)

        distancia_total = 0
        for indice_ciudad in range(len(ruta) - 1):
            distancia_total += matriz.loc[ruta[indice_ciudad], ruta[indice_ciudad+1]]

            if distancia_total >= mejor_distancia:
                break

        if distancia_total < mejor_distancia:
            mejor_distancia = distancia_total
            mejor_ruta = ruta

    return mejor_ruta, mejor_distancia
```