PREDICCIÓN TAZA DE MATERNIDAD ADOLESCENTE GESTIÓN 2020-2024

Presentación

El conjunto de datos elegido ha sido extraido de la pagina web oficial del Ayuntamiento de Barcelona https://opendata-ajuntament.barcelona.cat/data/ca/dataset/est-sp-taxa-fec-mares-15-19-anys, corresponden al conjunto de datos de población y demografía.

Caracteristicas Generales

Este es un archivo csv, que tiene 56 Kb de tamaño, su fecha ultima de actualizacion es 14/07/2017, lo que mas resalta en este conjunto de datos es que los registros de la tasa - Nombre en el conjunto estan o estan promediados por cada cuatro años. Los nombres de las columnas estan en Catalan porque corresponden al Ayuntamiento de Barcelona.

Definición de las variables

01.Anys_Periode.- Gestión en años a la que corresponden los datos, este dato es categorico.

02.Codi_Districte.- Codigo numerico del distrito

03.Nom_Districte.- Nombre del distrito

04.Codi_Barri.- Codigo del barrio, este dato es numerico.

05.Nom_Barri.- Nombre del barrio

06.Nombre.- Taza de fecundidad en mujeres de 15 a 19 años. El dato es numerico.

Objetivo

El objetivo de este proyecto es contruir un modelo predictivo, para predecir la taza de fecundidad en mujeres adolescentes de 15 a 19 años la ultima gestion (2020-2024) en cada distrito y barrio de la ciudad de Barcelona.

Procedimiento

In [27]:
```
import pandas as pd
df_main = pd.read_csv('2020_est_salut_publica_tax_espec_fec_mares_15_19a.csv')
df_main.head(10)
```

Out[27]:

	Anys_Periode	Codi_Districte	Nom_Districte	Codi_Barri	Nom_Barri	Nombre
0	2003-2007	1	Ciutat Vella	1	el Raval	19.4
1	2003-2007	1	Ciutat Vella	2	el Barri Gòtic	11.6
2	2003-2007	1	Ciutat Vella	3	la Barceloneta	11.6
3	2003-2007	1	Ciutat Vella	4	Sant Pere, Santa Caterina i la Ribera	14.6
4	2003-2007	2	Eixample	5	el Fort Pienc	4.7
5	2003-2007	2	Eixample	6	la Sagrada Família	5.8
6	2003-2007	2	Eixample	7	la Dreta de l'Eixample	2.6
7	2003-2007	2	Eixample	8	l'Antiga Esquerra de l'Eixample	3.5
8	2003-2007	2	Eixample	9	la Nova Esquerra de l'Eixample	3.1
9	2003-2007	2	Eixample	10	Sant Antoni	7.0

In [28]:
```
df_main.shape
```

Out[28]:
```
(1022, 6)
```

In [29]:
```
df_main.describe().transpose()
```

Out[29]:

	count	mean	std	min	25%	50%	75%	max
Codi_Districte	1022.0	6.246575	2.789701	1.0	4.0	7.0	8.000	10.0
Codi_Barri	1022.0	37.000000	21.081624	1.0	19.0	37.0	55.000	73.0
Nombre	1022.0	8.226908	8.301110	0.0	2.7	5.7	11.175	66.0

In [30]:
```
df_main.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1022 entries, 0 to 1021
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Anys_Periode    1022 non-null   object
 1   Codi_Districte  1022 non-null   int64
 2   Nom_Districte   1022 non-null   object
 3   Codi_Barri      1022 non-null   int64
 4   Nom_Barri       1022 non-null   object
 5   Nombre          1022 non-null   float64
dtypes: float64(1), int64(2), object(3)
memory usage: 48.0+ KB
```

In [31]:
```
df_main.columns
```

Out[31]:
```
Index(['Anys_Periode', 'Codi_Districte', 'Nom_Districte', 'Codi_Barri',
       'Nom_Barri', 'Nombre'],
      dtype='object')
```

In [32]:
```
df_main['Anys_Periode'].unique()
```

Out[32]:
```
array(['2003-2007', '2004-2008', '2005-2009', '2006-2010', '2007-2011',
       '2008-2012', '2009-2013', '2010-2014', '2011-2015', '2012-2016',
       '2013-2017', '2014-2018', '2015-2019', '2016-2020'], dtype=object)
```

In [33]:
```
barrios = df_main['Nom_Barri'].unique()
```

In [34]:
```
barrios.shape
```

Out[34]:
```
(73,)
```

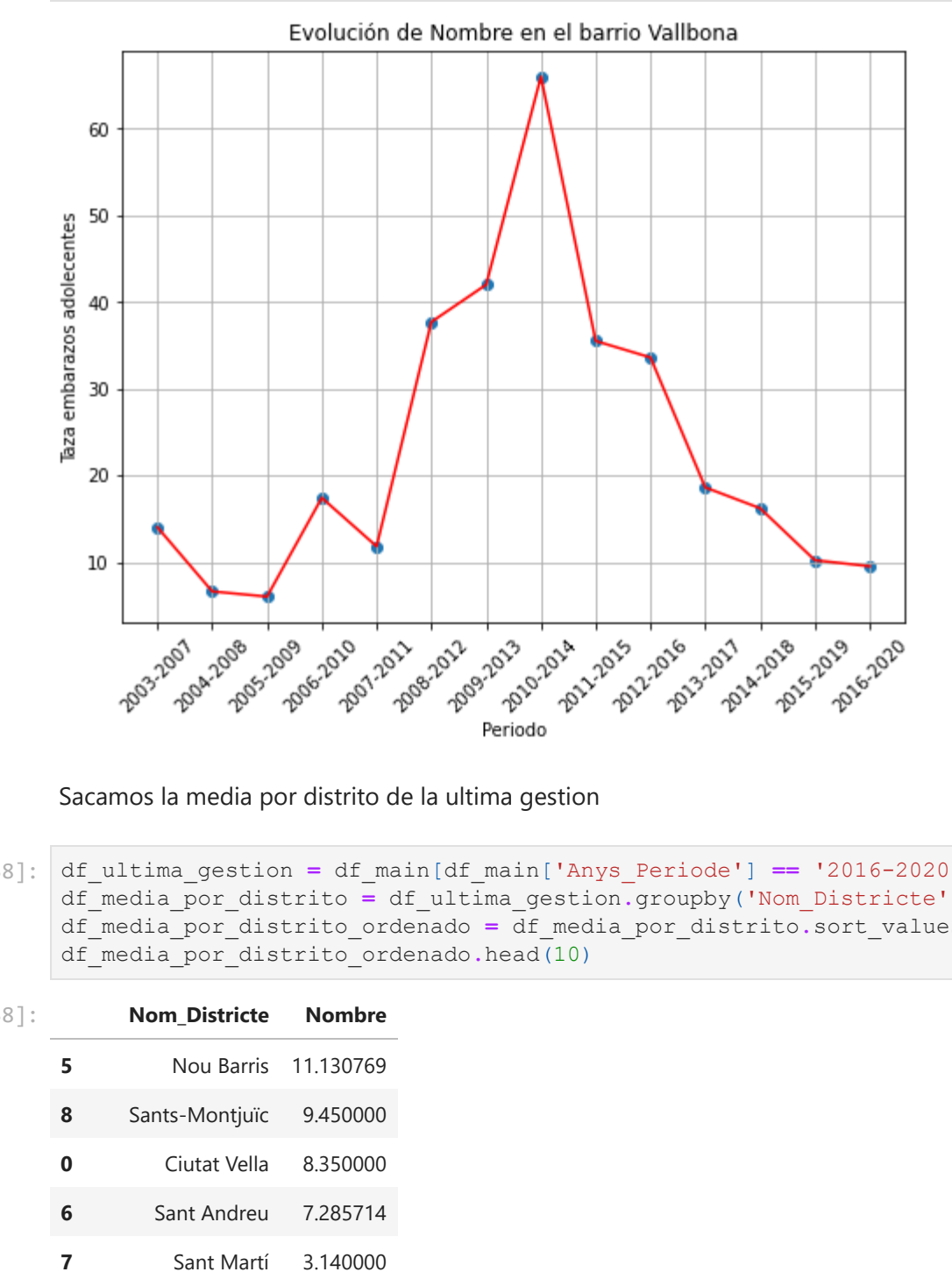
In [35]:
```
df_main[df_main['Nom_Barri'] == 'el Raval']
```

Out[35]:

	Anys_Periode	Codi_Districte	Nom_Districte	Codi_Barri	Nom_Barri	Nombre
0	2003-2007	1	Ciutat Vella	1	el Raval	19.4
73	2004-2008	1	Ciutat Vella	1	el Raval	19.3
146	2005-2009	1	Ciutat Vella	1	el Raval	19.8
219	2006-2010	1	Ciutat Vella	1	el Raval	18.7
292	2007-2011	1	Ciutat Vella	1	el Raval	18.7
365	2008-2012	1	Ciutat Vella	1	el Raval	17.1
438	2009-2013	1	Ciutat Vella	1	el Raval	15.6
511	2010-2014	1	Ciutat Vella	1	el Raval	17.5
584	2011-2015	1	Ciutat Vella	1	el Raval	13.1
657	2012-2016	1	Ciutat Vella	1	el Raval	13.2
730	2013-2017	1	Ciutat Vella	1	el Raval	13.5
803	2014-2018	1	Ciutat Vella	1	el Raval	13.2
876	2015-2019	1	Ciutat Vella	1	el Raval	12.7
949	2016-2020	1	Ciutat Vella	1	el Raval	11.5

In [36]:
```
#df_main[df_main['Codi_Barri'] == 56]
```

In [84]:
```
import matplotlib.pyplot as plt

Codi_Barri = 56

nombre_barri = df_main[df_main['Codi_Barri'] == Codi_Barri]['Nom_Barri'].iloc[0]

# Filtrar los datos para el Nom_Barri "La marina del prat vermell"
df_barri = df_main[df_main['Codi_Barri'] == Codi_Barri]

# Filtrar los datos para la evolución de Nombre por año (Anys_Periode)
df_evolucion = df_barri[['Anys_Periode', 'Nombre']]

# Ordenar los datos por Anys_Periode
df_evolucion = df_evolucion.sort_values('Anys_Periode')

# Crear la gráfica de puntos y línea
plt.figure(figsize=(8, 6))
plt.scatter(df_evolucion['Anys_Periode'], df_evolucion['Nombre'])
plt.plot(df_evolucion['Anys_Periode'], df_evolucion['Nombre'], 'r-')
plt.xlabel('Periodo')
plt.ylabel('Taza embarazos adolecentes')
plt.title(f"Evolución de Nombre en el barrio {nombre_barri}")
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



Sacamos la media por distrito de la ultima gestion

In [38]:
```
df_ultima_gestion = df_main[df_main['Anys_Periode'] == '2016-2020']
df_media_por_distrito = df_ultima_gestion.groupby('Nom_Districte')['Nombre'].mean().reset_index()
df_media_por_distrito_ordenado = df_media_por_distrito.sort_values('Nombre',ascending=False)
df_media_por_distrito_ordenado.head(10)
```

Out[38]:

	Nom_Districte	Nombre
5	Nou Barris	11.130769
8	Sants-Montjuïc	9.450000
0	Ciutat Vella	8.350000
6	Sant Andreu	7.285714
7	Sant Martí	3.140000
3	Horta-Guinardó	2.872727
2	Gràcia	2.480000
1	Eixample	1.950000
4	Les Corts	1.166667
9	Sarrià-Sant Gervasi	1.166667

Sacamos la media por barrio y la ultima por gestion

In [39]:
```
df_ultima_gestion = df_main[df_main['Anys_Periode'] == '2016-2020']
df_media_por_barri = df_ultima_gestion.groupby('Nom_Barri')['Nombre'].mean().reset_index()
df_media_por_barri_ordenado = df_media_por_barri.sort_values('Nombre',ascending=False)
#df_media_por_barri_ordenado.head(10)

df_top_10 = df_media_por_barri_ordenado.head(10)
df_top_10
```

Out[39]:

	Nom_Barri	Nombre
56	la Marina del Prat Vermell	36.0
71	les Roquetes	22.8
2	Can Peguera	21.2
64	la Trinitat Nova	16.4
16	el Turó de la Peira	14.4
4	Ciutat Meridiana	14.0
23	Torre Baró	14.0
65	la Trinitat Vella	13.8
32	el Bon Pastor	12.5
48	la Barceloneta	12.2

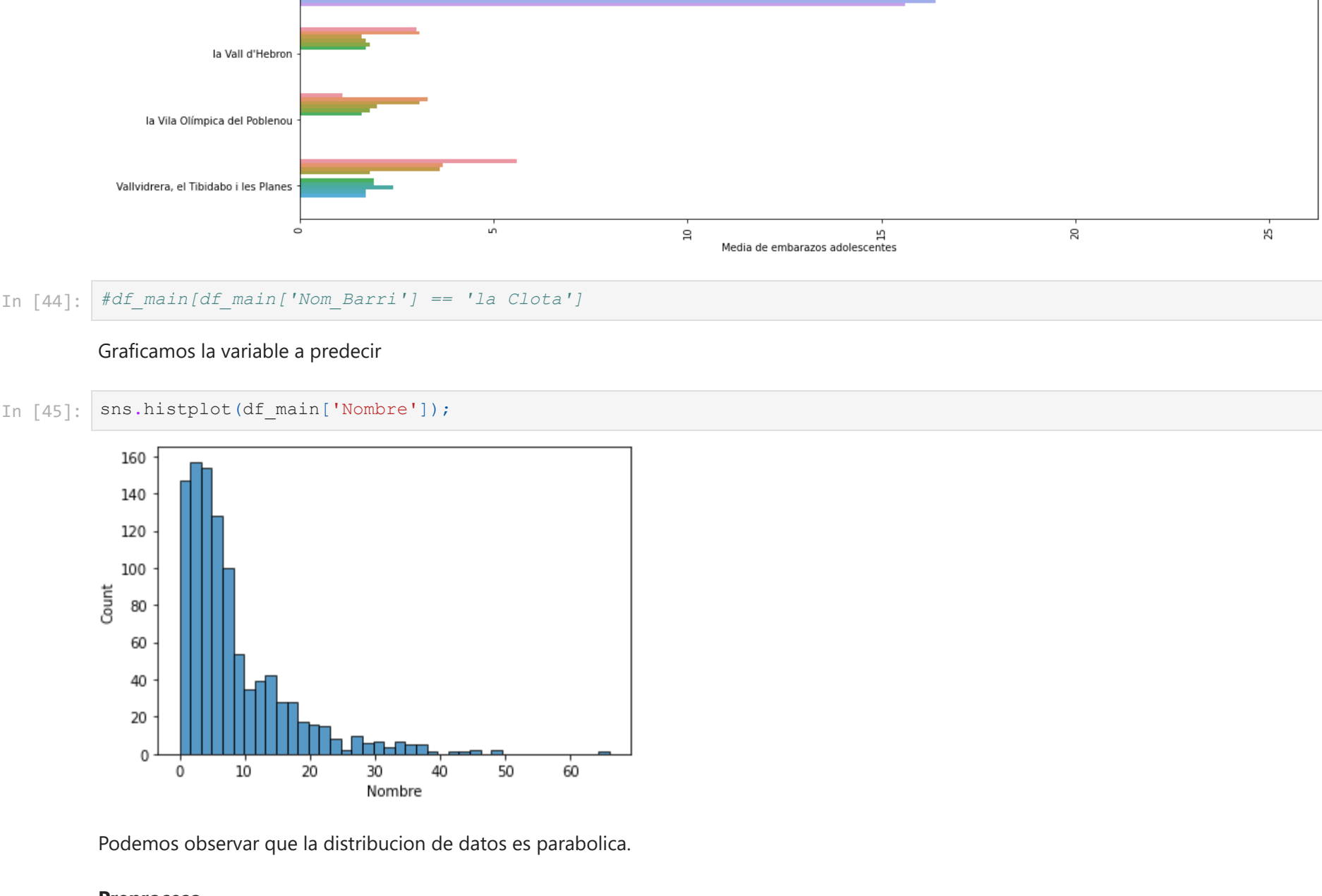
Graficamos los barrios con la media de la ultima gestion

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

# Ordenar los datos por la media en orden descendente
df_media_por_barri_ordenado = df_media_por_barri.sort_values('Nombre', ascending=False)

# Configurar el tamaño del gráfico
plt.figure(figsize=(18, 9))

# Crear la paleta de colores degradados
color_palette = sns.color_palette("viridis", len(df_media_por_barri_ordenado))

# Crear el gráfico de barras con colores degradados
plt.bar(df_media_por_barri_ordenado['Nom_Barri'][:35], df_media_por_barri_ordenado['Nombre'][:35],
        color=color_palette)

# Agregar etiquetas y título
plt.xlabel("Barrio")
plt.ylabel("Media")
plt.title('Media por Barrio en la Última Gestión (2016-2020)')

# Rotar las etiquetas del eje x para una mejor legibilidad
plt.xticks(rotation=90)

# Mostrar el gráfico
plt.show()
```

In [41]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Ordenar los datos por la media en orden descendente
df_media_por_barri_ordenado = df_media_por_barri.sort_values('Nombre', ascending=False)

# Configurar el tamaño del gráfico
plt.figure(figsize=(18, 9))

# Crear la paleta de colores degradados
color_palette = sns.color_palette("viridis", len(df_media_por_barri_ordenado))

# Crear el gráfico de barras con colores degradados
plt.bar(df_media_por_barri_ordenado['Nom_Barri'][35:], df_media_por_barri_ordenado['Nombre'][35:],
        color=color_palette)

# Agregar etiquetas y título
plt.xlabel("Barrio")
plt.ylabel("Media")
plt.title('Media por Barrio en la Última Gestión (2016-2020)')

# Rotar las etiquetas del eje x para una mejor legibilidad
plt.xticks(rotation=90)

# Mostrar el gráfico
plt.show()
```

In [42]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Filtrar los datos para los 10 barrios con el índice más alto de embarazos adolescentes
top_10_barrios = df_media_por_barri_ordenado.head(10)['Nom_Barri'].tolist()
df_top_10 = df_main[df_main['Nom_Barri'].isin(top_10_barrios)].sort_values(by="Nom_Barri", ascending=False)

# Configurar el tamaño del gráfico
plt.figure(figsize=(18, 12))

# Ordenar los valores únicos de Anys_Periode en orden ascendente
sorted_anys_periode = sorted(df_top_10['Anys_Periode'].unique())

# Crear el gráfico de barras con los datos de Anys_Periode ordenados en ascendente
sns.barplot(x='Nombre', y='Nom_Barri', hue='Anys_Periode', data=df_top_10, order=top_10_barrios, hue_order=sort

# Configurar título y etiquetas de los ejes
plt.title("Histórico de los 10 Barrios con alto índice de embarazos adolescentes")
plt.xlabel("Media de embarazos adolescentes")
plt.ylabel("Barrio")

# Rotar las etiquetas del eje x para una mejor legibilidad
plt.xticks(rotation=90)

# Mostrar el gráfico
plt.show()
```

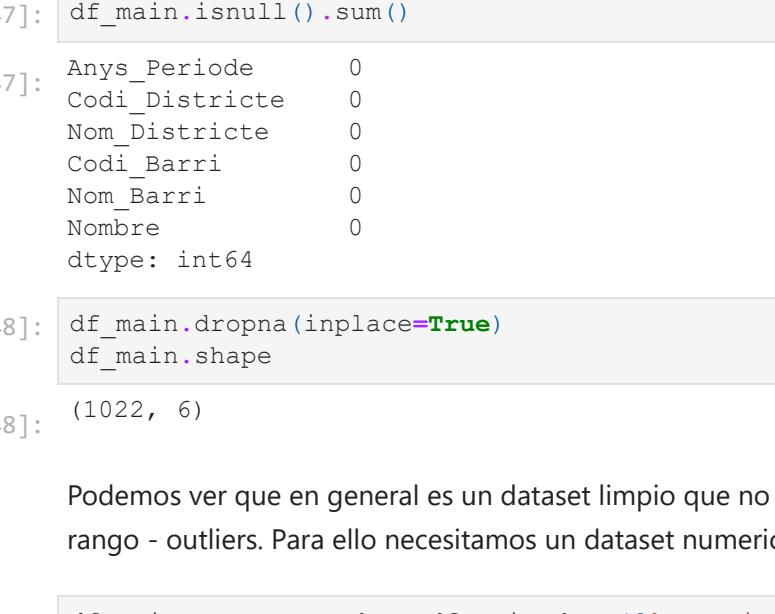
In [43]:
```
# Filtrar los datos para los 10 barrios con el índice más bajo de embarazos adolescentes
bottom_10_barrios = df_media_por_barri_ordenado.tail(10)['Nom_Barri'].tolist()
df_bottom_10 = df_main[df_main['Nom_Barri'].isin(bottom_10_barrios)].sort_values(by="Nom_Barri")

# Configurar el tamaño del gráfico
plt.figure(figsize=(18, 12))

# Ordenar los valores únicos de Anys_Periode en orden ascendente
sorted_anys_periode = sorted(df_bottom_10['Anys_Periode'].unique())

# Crear el gráfico de barras con los datos de Anys_Periode ordenados en ascendente
sns.barplot(x='Nombre', y='Nom_Barri', hue='Anys_Periode', data=df_bottom_10, order=bottom_10_barrios, hue_orde

# Configurar título y etiquetas de los ejes
plt.title("Histórico de los 10 Barrios con bajo índice de embarazos adolescentes")
plt.xlabel("Media de embarazos adolescentes")
plt.ylabel("Barrio")

# Rotar las etiquetas del eje x para una mejor legibilidad
plt.xticks(rotation=90)

# Mostrar el gráfico
plt.show()
```



In [44]:
```
#df_main[df_main['Nom_Barri'] == 'la Clota']
```

Graficamos la variable a predecir

In [45]:
```
sns.histplot(df_main["Nombre"]);
```



Podemos observar que la distribucion de datos es parabolica.

Preproceso

Verificamos si el conjunto de datos necesita limpieza y luego procedemos a limpiar si se requiere.

In [46]:
```
df_main.isna().sum()
```

Out[46]:
```
Anys_Periode      0
Codi_Districte    0
Nom_Districte     0
Codi_Barri        0
Nom_Barri         0
Nombre            0
dtype: int64
```

In [47]:
```
df_main.isnull().sum()
```

Out[47]:
```
Anys_Periode      0
Codi_Districte    0
Nom_Districte     0
Codi_Barri        0
Nom_Barri         0
Nombre            0
dtype: int64
```

In [48]:
```
df_main.dropna(inplace=True)
df_main.shape
```

Out[48]:
```
(1022, 6)
```

Podemos ver que en general es un dataset limpio que no tiene nulos ni registros vacios. Procedemos a verificar si existen datos fuera de rango - outliers. Para ello necesitamos un dataset numerico.

In [58]:
```
df_main_preprocesado = df_main.drop(['Nom_Districte','Nom_Barri'],axis=1)
df_main_preprocesado.head(10)
```

Out[58]:

	Anys_Periode	Codi_Districte	Codi_Barri	Nombre
0	2003-2007	1	1	19.4
1	2003-2007	1	2	11.6
2	2003-2007	1	3	11.6
3	2003-2007	1	4	14.6
4	2003-2007	2	5	4.7
5	2003-2007	2	6	5.8
6	2003-2007	2	7	2.6
7	2003-2007	2	8	3.5
8	2003-2007	2	9	3.1
9	2003-2007	2	10	7.0

In [59]:
```
from sklearn.preprocessing import LabelEncoder

# Crear una instancia de LabelEncoder
encoder = LabelEncoder()

# Ajustar y transformar la columna "Anys_Periode" en la tabla
df_main_preprocesado['Anys_Periode'] = encoder.fit_transform(df_main['Anys_Periode'])

# Mostrar la tabla actualizada
print(df_main_preprocesado.tail(10))
```

```
      Anys_Periode  Codi_Districte  Codi_Barri  Nombre
1012            13              10          64     3.6
1013            13              10          65     0.7
1014            13              10          66     1.5
1015            13              10          67     0.0
1016            13              10          68     0.9
1017            13              10          69     2.1
1018            13              10          70     9.0
1019            13              10          71     4.4
1020            13              10          72     3.4
1021            13              10          73     5.8
```

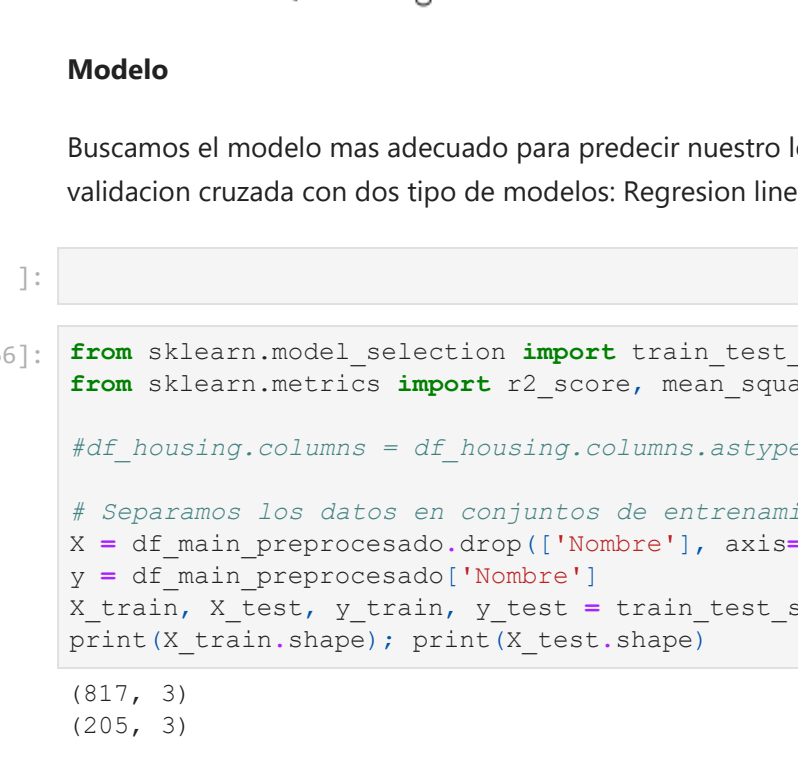
In [61]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import RobustScaler

df_main_outliers = df_main_preprocesado.copy()
# Seleccionar las columnas numericas en las que deseas identificar outliers
columnas_numericas = ['Anys_Periode','Codi_Districte', 'Codi_Barri', 'Nombre']

# Aplicar RobustScaler a las columnas numericas
scaler = RobustScaler()
df_main_outliers[columnas_numericas] = scaler.fit_transform(df_main_preprocesado[columnas_numericas])

# Calcular los limites para identificar los outliers
percentil_25 = np.percentile(df_main_preprocesado[columnas_numericas], 25, axis=0)
percentil_75 = np.percentile(df_main_preprocesado[columnas_numericas], 75, axis=0)
rango_intercuartil = percentil_75 - percentil_25
limite_inferior = percentil_25 - 1.5 * rango_intercuartil
limite_superior = percentil_75 + 1.5 * rango_intercuartil

# Identificar los outliers
outliers = df_main_outliers[
    (df_main_preprocesado[columnas_numericas] < limite_inferior) |
    (df_main_preprocesado[columnas_numericas] > limite_superior)
]

# Imprimir los outliers identificados
print("Outliers identificados:")
print(outliers)
#print(df_main_preprocesado.head(10))
```

```
Outliers identificados:
      Anys_Periode  Codi_Districte  Codi_Barri  Nombre
0              NaN             NaN         NaN     NaN
1              NaN             NaN         NaN     NaN
2              NaN             NaN         NaN     NaN
3              NaN             NaN         NaN     NaN
4              NaN             NaN         NaN     NaN
...            ...             ...         ...     ...
1017           NaN             NaN         NaN     NaN
1018           NaN             NaN         NaN     NaN
1019           NaN             NaN         NaN     NaN
1020           NaN             NaN         NaN     NaN
1021           NaN             NaN         NaN     NaN

[1022 rows x 4 columns]
```

In [65]:
```
#df_main_preprocesado
```

Graficamos los puntos y vemos la correlacion de los datos.

In [63]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.scatter(df_main_preprocesado['Anys_Periode'], df_main_preprocesado['Nombre'])
plt.xlabel('Anys_Periode')
plt.ylabel('Nombre')
plt.title('Distribución de los puntos')
plt.show()
```



In [64]:
```
corr_matrix = df_main_preprocesado.corr()
sns.heatmap(corr_matrix, annot=True)
plt.title('Heatmap de correlación')
plt.show()
```

Modelo

Buscamos el modelo mas adecuado para predecir nuestro los datos, preparamos y ajustamos el modelo. Comenzamos con utilizando validacion cruzada con dos tipo de modelos: Regresion lineal y random forest.

In [ ]:

In [66]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

#df_housing.columns = df_housing.columns.astype(str)

# Separamos los datos en conjunto de entrenamiento y de prueba
X = df_main_preprocesado.drop(['Nombre'], axis=1)
y = df_main_preprocesado['Nombre']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape); print(X_test.shape)
```

```
(817, 3)
(205, 3)
```

In [67]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor

# Crear los modelos
model_lr = LinearRegression()
model_rf = RandomForestRegressor()

# Definir los parámetros a ajustar para cada modelo
params_lr = {}
params_rf = {'n_estimators': [100, 200, 300], 'max_depth': [None, 5, 10]}

# Definir los scorers a utilizar para evaluar los modelos
scorers = {'MAE': 'neg_mean_absolute_error', 'MSE': 'neg_mean_squared_error'}

# Crear el objeto PolynomialFeatures

# Realizar la validación cruzada para cada modelo y obtener los mejores parámetros y métricas
grid_lr = GridSearchCV(model_lr, params_lr, scoring=scorers, refit='MSE', cv=5)
grid_rf = GridSearchCV(model_rf, params_rf, scoring=scorers, refit='MSE', cv=5)

# Transformar las características con PolynomialFeatures antes de ajustar los modelos

# Ajustar los modelos
grid_lr.fit(X_train, y_train)
grid_rf.fit(X_train, y_train)

# Obtener los mejores parámetros y métricas para cada modelo
best_params_lr = grid_lr.best_params_
best_metrics_lr = grid_lr.best_score_
best_params_rf = grid_rf.best_params_
best_metrics_rf = grid_rf.best_score_

# Imprimir los mejores parámetros y métricas para cada modelo
print("Linear Regression:")
print("Best Parameters:", best_params_lr)
print("Best Metrics:", best_metrics_lr)
print("\nRandom Forest Regression:")
print("Best Parameters:", best_params_rf)
print("Best Metrics:", best_metrics_rf)
```

```
Linear Regression:
Best Parameters: {}
Best Metrics: -64.257016281553

Random Forest Regression:
Best Parameters: {'max_depth': None, 'n_estimators': 300}
Best Metrics: -12.743617891026
```

Evaluacion

In [68]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.ensemble import RandomForestRegressor

# Separamos los datos en conjunto de entrenamiento y de prueba
X = df_main_preprocesado.drop(['Nombre'], axis=1)
y = df_main_preprocesado['Nombre']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape); print(X_test.shape)

# Crear y entrenar el modelo de Random Forest
model = RandomForestRegressor(max_depth=None, n_estimators=300)
model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de entrenamiento y de prueba
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Evaluar el rendimiento del modelo en el conjunto de entrenamiento
r2_train = r2_score(y_train, y_pred_train)
mse_train = mean_squared_error(y_train, y_pred_train)
mae_train = mean_absolute_error(y_train, y_pred_train)
print("R2 score (train):", r2_train)
print("Mean Squared Error (train):", mse_train)
print("Mean Absolute Error (train):", mae_train)

# Evaluar el rendimiento del modelo en el conjunto de prueba
r2_test = r2_score(y_test, y_pred_test)
mse_test = mean_squared_error(y_test, y_pred_test)
mae_test = mean_absolute_error(y_test, y_pred_test)
print("R2 score (test):", r2_test)
print("Mean Squared Error (test):", mse_test)
print("Mean Absolute Error (test):", mae_test)
```

```
(817, 3)
(205, 3)
R2 score (train): 0.979385647122134
Mean Squared Error (train): 1.4343434848361232
Mean Absolute Error (train): 0.6097005303957552
R2 score (test): 0.8757216810552348
Mean Squared Error (test): 8.188087452032528
Mean Absolute Error (test): 1.577081300813009
```

Podemos observar que el modelo que mas se ajusta a nuestros datos es Random Forest.

Predicción

Ya sabemos que Random Forest es el modelo que mejor puede predecir nuestros datos ahora vamos utilizar este modelo para obtener los valores

In [70]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor

df_pred = df_main_preprocesado

# Crear el DataFrame df_result vacío
df_result = pd.DataFrame(columns=['Anys_Periode', 'Codi_Districte', 'Codi_Barri', 'Nombre'])

# Recorrer los distritos y barrios únicos en el dataset
distritos_barrios = df_pred[['Codi_Districte', 'Codi_Barri']].drop_duplicates()
for _, row in distritos_barrios.iterrows():
    codi_districte = row['Codi_Districte']
    codi_barri = row['Codi_Barri']

    # Filtrar los datos para el distrito y barrio actual
    df_pred_districte_barri = df_pred[(df_pred['Codi_Districte'] == codi_districte) & (df_pred['Codi_Barri'] ==

    # Filtrar los datos para períodos anteriores a '2020-2024'
    df_pred_anteriores = df_pred_districte_barri[df_pred_districte_barri['Anys_Periode'] < 14]

    if len(df_pred_anteriores) > 0:
        # Dividir los datos en variables independientes (X_pred) y variable dependiente (y_pred)
        X_pred = df_pred_anteriores[['Anys_Periode', 'Codi_Districte', 'Codi_Barri']]
        y_pred = df_pred_anteriores['Nombre']

        # Crear y entrenar el modelo de Random Forest con los mejores parámetros
        model = RandomForestRegressor(max_depth=None, n_estimators=100)
        #model = RandomForestRegressor(max_depth=10, n_estimators=300)
        model.fit(X_pred, y_pred)

        # Importancias de las características
        #importances = model.feature_importances_
        #for i, feature in enumerate(X_pred.columns):
        #    print(f"Feature: {feature}, Importance: {importances[i]}")


        # Realizar la predicción para el período '2020-2024'
        X_pred_prediccion = pd.DataFrame({'Anys_Periode': [14], 'Codi_Districte': [codi_districte], 'Codi_Barri'
        y_pred_prediccion = model.predict(X_pred_prediccion)

        # Agregar los resultados al DataFrame df_result
        df_result = pd.concat([df_result, pd.DataFrame({
            'Anys_Periode': [14],
            'Codi_Districte': codi_districte,
            'Codi_Barri': codi_barri,
            'Nombre': y_pred_prediccion[0]
        })], ignore_index=True)

# Imprimir el DataFrame df_result
print(df_result)
```

```
   Anys_Periode Codi_Districte Codi_Barri  Nombre
0            14              1          1  11.985
1            14              1          2   4.143
2            14              1          3  11.726
3            14              1          4   7.626
4            14              2          5   1.828
..          ...            ...        ...     ...
68           14             10         69   2.450
69           14             10         70  10.150
70           14             10         71   4.688
71           14             10         72   3.854
72           14             10         73   5.954

[73 rows x 4 columns]
```

In [71]:
```
df_result['Anys_Periode'] = '2020-2024'
df_result['Nom_Districte']=None
df_result['Nom_Barri']=None
```

In [72]:
```
df_result
```

Out[72]:

	Anys_Periode	Codi_Districte	Codi_Barri	Nombre	Nom_Districte	Nom_Barri
0	2020-2024	1	1	11.985	None	None
1	2020-2024	1	2	4.143	None	None
2	2020-2024	1	3	11.726	None	None
3	2020-2024	1	4	7.626	None	None
4	2020-2024	2	5	1.828	None	None
...	...	...	...	...	...	...
68	2020-2024	10	69	2.450	None	None
69	2020-2024	10	70	10.150	None	None
70	2020-2024	10	71	4.688	None	None
71	2020-2024	10	72	3.854	None	None
72	2020-2024	10	73	5.954	None	None

73 rows × 6 columns

In [78]:
```
# Obtener los valores únicos de Codi_Districte y Codi_Barri del DataFrame final
codi_districte_unique = df_main['Codi_Districte'].unique()
codi_barri_unique = df_main['Codi_Barri'].unique()

# Crear un diccionario para mapear los valores de Codi_Districte y Codi_Barri a Nom_Districte y Nom_Barri
mapping_dist = {}
for codi_districte in codi_districte_unique:
    nom_districte = df_main.loc[df_main['Codi_Districte'] == codi_districte, 'Nom_Districte'].iloc[0]
    mapping_dist[codi_districte] = nom_districte
mapping_barri = {}
for codi_barri in codi_barri_unique:
    nom_barri = df_main.loc[df_main['Codi_Barri'] == codi_barri, 'Nom_Barri'].iloc[0]
    mapping_barri[codi_barri] = nom_barri
#print (mapping_dist)
```

In [79]:
```
# Reemplazar los valores de Codi_Districte y Codi_Barri con sus respectivos nombres en el DataFrame final
df_result['Nom_Districte'] = df_result['Codi_Districte'].map(mapping_dist)
df_result['Nom_Barri'] = df_result['Codi_Barri'].map(mapping_barri)

# Mostrar el DataFrame final con los campos Nom_Districte y Nom_Barri llenos
print(df_result)
df_result.to_csv('df_result.csv', sep=',', index=False)
```

```
   Anys_Periode Codi_Districte Codi_Barri  Nombre Nom_Districte  \
0     2020-2024              1          1  11.985  Ciutat Vella
1     2020-2024              1          2   4.143  Ciutat Vella
2     2020-2024              1          3  11.726  Ciutat Vella
3     2020-2024              1          4   7.626  Ciutat Vella
4     2020-2024              2          5   1.828      Eixample
..          ...            ...        ...     ...           ...
68    2020-2024             10         69   2.450    Sant Martí
69    2020-2024             10         70  10.150    Sant Martí
70    2020-2024             10         71   4.688    Sant Martí
71    2020-2024             10         72   3.854    Sant Martí
72    2020-2024             10         73   5.954    Sant Martí

                                     Nom_Barri
0                                    el Raval
1                              el Barri Gòtic
2                             la Barceloneta
3         Sant Pere, Santa Caterina i la Ribera
4                                  el Fort Pienc
..                                         ...
68   Diagonal Mar i el Front Marítim del Poblenou
69                        el Besòs i el Maresme
70                       Provençals del Poblenou
71                        Sant Martí de Provençals
72                             la Verneda i la Pau

[73 rows x 6 columns]
```

In [81]:
```
df_final = pd.concat([df_main, df_result], ignore_index=True)

print(df_final)
df_final.to_csv('df_final.csv', sep=',', index=False)
```

```
     Anys_Periode Codi_Districte Nom_Districte Codi_Barri  \
0       2003-2007              1  Ciutat Vella          1
1       2003-2007              1  Ciutat Vella          2
2       2003-2007              1  Ciutat Vella          3
3       2003-2007              1  Ciutat Vella          4
4       2003-2007              2      Eixample          5
...           ...            ...           ...        ...
1090    2020-2024             10    Sant Martí         69
1091    2020-2024             10    Sant Martí         70
1092    2020-2024             10    Sant Martí         71
1093    2020-2024             10    Sant Martí         72
1094    2020-2024             10    Sant Martí         73

                                      Nom_Barri  Nombre
0                                      el Raval  19.400
1                                el Barri Gòtic  11.600
2                               la Barceloneta  11.600
3           Sant Pere, Santa Caterina i la Ribera  14.600
4                                  el Fort Pienc   4.700
...                                         ...     ...
1090  Diagonal Mar i el Front Marítim del Poblenou   2.450
1091                        el Besòs i el Maresme  10.150
1092                       Provençals del Poblenou   4.688
1093                      Sant Martí de Provençals   3.854
1094                           la Verneda i la Pau   5.954

[1095 rows x 6 columns]
```

Graficamos un solo barrio

In [83]:
```
import matplotlib.pyplot as plt

Codi_Barri = 56

nombre_barri = df_final[df_final['Codi_Barri'] == Codi_Barri]['Nom_Barri'].iloc[0]

# Filtrar los datos para el Nom_Barri "La marina del prat vermell"
df_barri = df_final[df_final['Codi_Barri'] == Codi_Barri]

# Filtrar los datos para la evolución de Nombre por año (Anys_Periode)
df_evolucion = df_barri[['Anys_Periode', 'Nombre']]

# Ordenar los datos por Anys_Periode
df_evolucion = df_evolucion.sort_values('Anys_Periode')

# Crear la gráfica de puntos y línea
plt.figure(figsize=(8, 6))
plt.scatter(df_evolucion['Anys_Periode'], df_evolucion['Nombre'])
plt.plot(df_evolucion['Anys_Periode'], df_evolucion['Nombre'], 'r-')
plt.xlabel('Periodo')
plt.ylabel('Taza embarazos adolecentes')
plt.title(f"Evolución de Nombre en el barrio {nombre_barri}")
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```


In [ ]: