Tarea1_Araneda_Wolf_D_M

May 5, 2025

[50]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
import sklearn
import scipy
from scipy.stats import nbinom
import seaborn as sns
from statsmodels.iolib.summary2 import summary_col

import warnings
warnings.filterwarnings("ignore")
```

1 observamos la naturaleza de los datos

[51]:
```
#leemos el archivo
df=pd.read_csv("machine_failure_data.csv")
df.describe(include="all")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142193 entries, 0 to 142192
Data columns (total 22 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Date              142193 non-null  object 
 1   Location          142193 non-null  int64  
 2   Min_Temp          141556 non-null  float64
 3   Max_Temp          141871 non-null  float64
 4   Leakage           140787 non-null  float64
 5   Evaporation       81350 non-null   float64
 6   Electricity       74377 non-null   float64
 7   Parameter1_Dir    132863 non-null  object 
 8   Parameter1_Speed  132923 non-null  float64
 9   Parameter2_9am    132180 non-null  object 
 10  Parameter2_3pm    138415 non-null  object 
 11  Parameter3_9am    140845 non-null  float64
```

```
 12  Parameter3_3pm    139563 non-null  float64
 13  Parameter4_9am    140419 non-null  float64
 14  Parameter4_3pm    138583 non-null  float64
 15  Parameter5_9am    128179 non-null  float64
 16  Parameter5_3pm    128212 non-null  float64
 17  Parameter6_9am    88536 non-null   float64
 18  Parameter6_3pm    85099 non-null   float64
 19  Parameter7_9am    141289 non-null  float64
 20  Parameter7_3pm    139467 non-null  float64
 21  Failure_today     140787 non-null  object
dtypes: float64(16), int64(1), object(5)
memory usage: 23.9+ MB
```

[52]:
```
df.dtypes
```

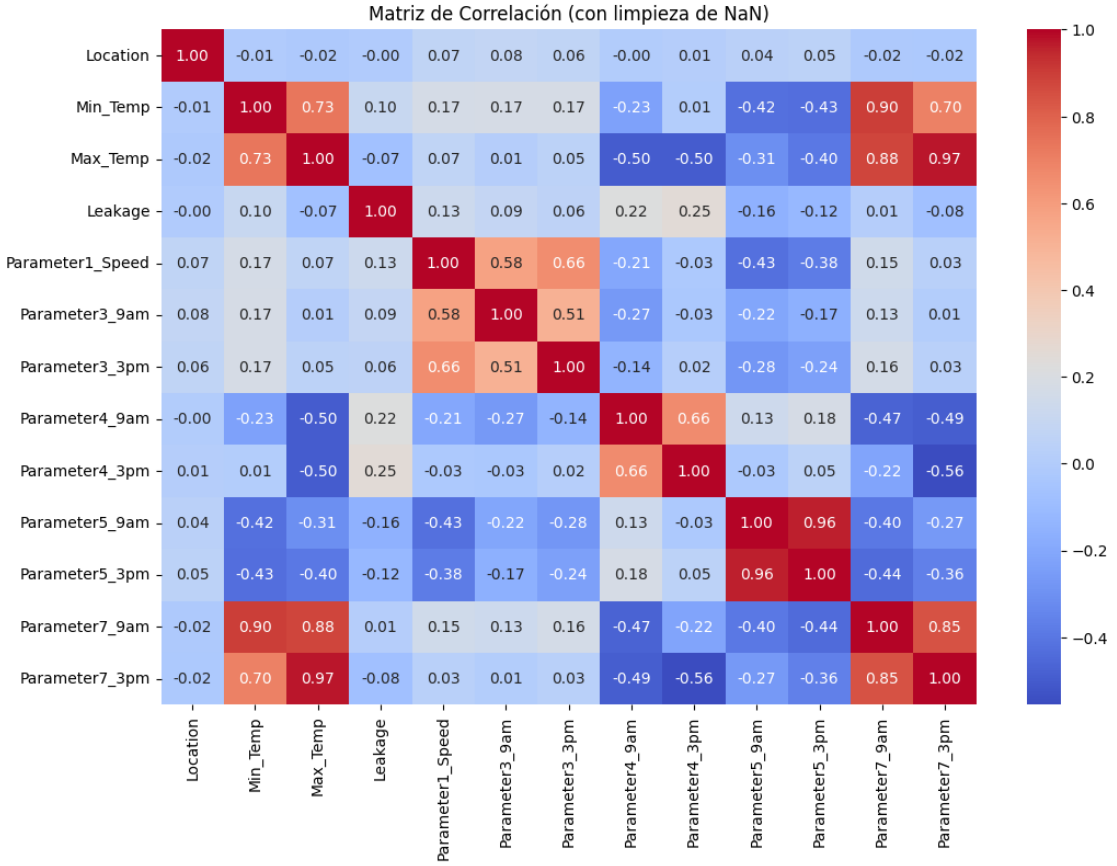
[52]:
```
Date                 object
Location              int64
Min_Temp            float64
Max_Temp            float64
Leakage             float64
Evaporation         float64
Electricity         float64
Parameter1_Dir       object
Parameter1_Speed    float64
Parameter2_9am       object
Parameter2_3pm       object
Parameter3_9am      float64
Parameter3_3pm      float64
Parameter4_9am      float64
Parameter4_3pm      float64
Parameter5_9am      float64
Parameter5_3pm      float64
Parameter6_9am      float64
Parameter6_3pm      float64
Parameter7_9am      float64
Parameter7_3pm      float64
Failure_today        object
dtype: object
```

Limpiamos los datos de nan y rellenamos con la media, y usamos las variables numericas para visulizar una matriz de correlacion para empezar a identificar variables utiles para nuestros modelos

[53]:
```
def matriz_correlacion_limpia(df, umbral_nan=0.1):
    df_numerico = df.select_dtypes(include='number').copy()

    # Eliminamos columnas con más del umbral permitido de nan
    columnas_validas = df_numerico.columns[df_numerico.isnull().mean() <=␣
 ↪umbral_nan]
```

```
    df_filtrado = df_numerico[columnas_validas]

    #rellenamos nan con la media
    df_filtrado = df_filtrado.fillna(df_filtrado.mean())

    #calculamos la matriz de correlacion
    corr = df_filtrado.corr()

    #grafico
    plt.figure(figsize=(12, 8))
    sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm')
    plt.title("Matriz de Correlación (con limpieza de NaN)")
    plt.show()

    return df_filtrado
```

[54]:
```
df_filtrado=matriz_correlacion_limpia(df)
```



[55]:
```
df_filtrado=pd.concat([df_filtrado, df["Failure_today"]])
```

[56]:
```
print(df.shape, "                ", df_filtrado.shape)
```

```
(142193, 22)                  (284386, 14)
```

[57]:
```
df_filtrado["Failure_today"]=df_filtrado["Failure_today"]
df_filtrado.info()
print("           ")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 284386 entries, 0 to 142192
Data columns (total 14 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Location          142193 non-null  float64
 1   Min_Temp          142193 non-null  float64
 2   Max_Temp          142193 non-null  float64
 3   Leakage           142193 non-null  float64
 4   Parameter1_Speed  142193 non-null  float64
 5   Parameter3_9am    142193 non-null  float64
 6   Parameter3_3pm    142193 non-null  float64
 7   Parameter4_9am    142193 non-null  float64
 8   Parameter4_3pm    142193 non-null  float64
 9   Parameter5_9am    142193 non-null  float64
 10  Parameter5_3pm    142193 non-null  float64
 11  Parameter7_9am    142193 non-null  float64
 12  Parameter7_3pm    142193 non-null  float64
 13  Failure_today     140787 non-null  object
dtypes: float64(13), object(1)
memory usage: 32.5+ MB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142193 entries, 0 to 142192
Data columns (total 22 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Date              142193 non-null  object
 1   Location          142193 non-null  int64
 2   Min_Temp          141556 non-null  float64
 3   Max_Temp          141871 non-null  float64
 4   Leakage           140787 non-null  float64
 5   Evaporation       81350 non-null   float64
 6   Electricity       74377 non-null   float64
 7   Parameter1_Dir    132863 non-null  object
 8   Parameter1_Speed  132923 non-null  float64
 9   Parameter2_9am    132180 non-null  object
 10  Parameter2_3pm    138415 non-null  object
 11  Parameter3_9am    140845 non-null  float64
```

```
 12  Parameter3_3pm    139563 non-null  float64
 13  Parameter4_9am    140419 non-null  float64
 14  Parameter4_3pm    138583 non-null  float64
 15  Parameter5_9am    128179 non-null  float64
 16  Parameter5_3pm    128212 non-null  float64
 17  Parameter6_9am    88536 non-null   float64
 18  Parameter6_3pm    85099 non-null   float64
 19  Parameter7_9am    141289 non-null  float64
 20  Parameter7_3pm    139467 non-null  float64
 21  Failure_today     140787 non-null  object
dtypes: float64(16), int64(1), object(5)
memory usage: 23.9+ MB
```

2 checkpoint limpiamos datos para el modelo (Pregunta1)

Librerias

[58]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
import sklearn
import scipy
from scipy.stats import nbinom
import seaborn as sns
from statsmodels.iolib.summary2 import summary_col

import warnings
warnings.filterwarnings("ignore")
```

[59]:
```
#leemos el archivo
df=pd.read_csv("machine_failure_data.csv")
df.describe(include="all")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142193 entries, 0 to 142192
Data columns (total 22 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Date              142193 non-null  object
 1   Location          142193 non-null  int64
 2   Min_Temp          141556 non-null  float64
 3   Max_Temp          141871 non-null  float64
 4   Leakage           140787 non-null  float64
 5   Evaporation       81350 non-null   float64
 6   Electricity       74377 non-null   float64
```

```
 7   Parameter1_Dir    132863 non-null  object
 8   Parameter1_Speed  132923 non-null  float64
 9   Parameter2_9am    132180 non-null  object
 10  Parameter2_3pm    138415 non-null  object
 11  Parameter3_9am    140845 non-null  float64
 12  Parameter3_3pm    139563 non-null  float64
 13  Parameter4_9am    140419 non-null  float64
 14  Parameter4_3pm    138583 non-null  float64
 15  Parameter5_9am    128179 non-null  float64
 16  Parameter5_3pm    128212 non-null  float64
 17  Parameter6_9am    88536 non-null   float64
 18  Parameter6_3pm    85099 non-null   float64
 19  Parameter7_9am    141289 non-null  float64
 20  Parameter7_3pm    139467 non-null  float64
 21  Failure_today     140787 non-null  object
dtypes: float64(16), int64(1), object(5)
memory usage: 23.9+ MB
```

[60]:
```
nan_por_columna = df.isna().sum()
nan_por_columna

# el parametro 6 9am posee
proporcion=53657/df.shape[0]
print("proporcion de nan del parametro 6 9am: ",proporcion*100,"%")
proporcion= nan_por_columna[-4]/df.shape[0]
print("proporcion de nan del parametro 6 3pm: ",proporcion*100,"%")
```

```
proporcion de nan del parametro 6 9am:  37.73533155640573 %
proporcion de nan del parametro 6 3pm:  40.15246882757942 %
```

[61]:
```
#se eliminan por un numero demasiado alto de nan

df = df.drop(columns=["Parameter6_3pm"])
df= df.drop(columns=["Parameter6_9am"])

# vemos el df
df

#esto ahora no tiene el parametro 6
```

[61]:

	Date	Location	Min_Temp	Max_Temp	Leakage	Evaporation	\
0	12/1/2008	3	13.4	22.9	0.6	NaN	
1	12/2/2008	3	7.4	25.1	0.0	NaN	
2	12/3/2008	3	12.9	25.7	0.0	NaN	
3	12/4/2008	3	9.2	28.0	0.0	NaN	
4	12/5/2008	3	17.5	32.3	1.0	NaN	
…	…	…	…	…	…	…	
142188	6/20/2017	42	3.5	21.8	0.0	NaN	

```
142189  6/21/2017        42       2.8      23.4      0.0          NaN
142190  6/22/2017        42       3.6      25.3      0.0          NaN
142191  6/23/2017        42       5.4      26.9      0.0          NaN
142192  6/24/2017        42       7.8      27.0      0.0          NaN

        Electricity Parameter1_Dir  Parameter1_Speed Parameter2_9am  \
0               NaN              W              44.0              W
1               NaN            WNW              44.0            NNW
2               NaN            WSW              46.0              W
3               NaN             NE              24.0             SE
4               NaN              W              41.0            ENE
…               …              …                …               …
142188          NaN              E              31.0            ESE
142189          NaN              E              31.0             SE
142190          NaN            NNW              22.0             SE
142191          NaN              N              37.0             SE
142192          NaN             SE              28.0            SSE

       Parameter2_3pm  Parameter3_9am  Parameter3_3pm  Parameter4_9am  \
0                 WNW            20.0            24.0            71.0
1                 WSW             4.0            22.0            44.0
2                 WSW            19.0            26.0            38.0
3                   E            11.0             9.0            45.0
4                  NW             7.0            20.0            82.0
…                  …              …               …               …
142188              E            15.0            13.0            59.0
142189            ENE            13.0            11.0            51.0
142190              N            13.0             9.0            56.0
142191            WNW             9.0             9.0            53.0
142192              N            13.0             7.0            51.0

        Parameter4_3pm  Parameter5_9am  Parameter5_3pm  Parameter7_9am  \
0                 22.0          1007.7          1007.1            16.9
1                 25.0          1010.6          1007.8            17.2
2                 30.0          1007.6          1008.7            21.0
3                 16.0          1017.6          1012.8            18.1
4                 33.0          1010.8          1006.0            17.8
…                 …             …               …               …
142188            27.0          1024.7          1021.2             9.4
142189            24.0          1024.6          1020.3            10.1
142190            21.0          1023.5          1019.1            10.9
142191            24.0          1021.0          1016.8            12.5
142192            24.0          1019.4          1016.5            15.1

        Parameter7_3pm Failure_today
0                 21.8            No
1                 24.3            No
```

```
2                23.2            No
3                26.5            No
4                29.7            No
…                …               …
142188           20.9            No
142189           22.4            No
142190           24.5            No
142191           26.1            No
142192           26.0            No

[142193 rows x 20 columns]
```

CORECCION DE NAN VARIABLES 2/21 PARAMETRO 6 AM Y PM

[62]:
```
def resumen_nan(df):
    total = df.shape[0]
    resumen = pd.DataFrame({
        "NaN_count": df.isna().sum(),
        "Total": total,
        "NaN_percent": df.isna().sum() / total
    })
    return resumen.sort_values("NaN_percent", ascending=False)


resumen_nan(df)
```

[62]:

	NaN_count	Total	NaN_percent
Electricity	67816	142193	0.476929
Evaporation	60843	142193	0.427890
Parameter5_9am	14014	142193	0.098556
Parameter5_3pm	13981	142193	0.098324
Parameter2_9am	10013	142193	0.070418
Parameter1_Dir	9330	142193	0.065615
Parameter1_Speed	9270	142193	0.065193
Parameter2_3pm	3778	142193	0.026570
Parameter4_3pm	3610	142193	0.025388
Parameter7_3pm	2726	142193	0.019171
Parameter3_3pm	2630	142193	0.018496
Parameter4_9am	1774	142193	0.012476
Failure_today	1406	142193	0.009888
Leakage	1406	142193	0.009888
Parameter3_9am	1348	142193	0.009480
Parameter7_9am	904	142193	0.006358
Min_Temp	637	142193	0.004480
Max_Temp	322	142193	0.002265
Date	0	142193	0.000000
Location	0	142193	0.000000

[63]:
```
#sacamos electricity y evaporation

df= df.drop(columns=["Electricity"])
df= df.drop(columns=["Evaporation"])

#df con un total de 3 columnas menos
```

[64]:
```
resumen_nan(df)

#ahora tenemos muchos menos datos porque eran nan generados por los datos ya␣
 ↪sacados(columnas anteriores)
```

[64]:

	NaN_count	Total	NaN_percent
Parameter5_9am	14014	142193	0.098556
Parameter5_3pm	13981	142193	0.098324
Parameter2_9am	10013	142193	0.070418
Parameter1_Dir	9330	142193	0.065615
Parameter1_Speed	9270	142193	0.065193
Parameter2_3pm	3778	142193	0.026570
Parameter4_3pm	3610	142193	0.025388
Parameter7_3pm	2726	142193	0.019171
Parameter3_3pm	2630	142193	0.018496
Parameter4_9am	1774	142193	0.012476
Failure_today	1406	142193	0.009888
Leakage	1406	142193	0.009888
Parameter3_9am	1348	142193	0.009480
Parameter7_9am	904	142193	0.006358
Min_Temp	637	142193	0.004480
Max_Temp	322	142193	0.002265
Location	0	142193	0.000000
Date	0	142193	0.000000

[65]:
```
df.dtypes
```

[65]:
```
Date                object
Location             int64
Min_Temp           float64
Max_Temp           float64
Leakage            float64
Parameter1_Dir      object
Parameter1_Speed   float64
Parameter2_9am      object
Parameter2_3pm      object
Parameter3_9am     float64
Parameter3_3pm     float64
Parameter4_9am     float64
Parameter4_3pm     float64
```

```
Parameter5_9am      float64
Parameter5_3pm      float64
Parameter7_9am      float64
Parameter7_3pm      float64
Failure_today        object
dtype: object
```

[66]:
```
#generamos una copia
df_copia=df.copy()

#realizaremos el experimento de borrar las filas donde haya almenos un nan, y␣
 ↪ver si se reducen en menos de un 10% los datos es aceptable
df_copia = df_copia.dropna()
```

[67]:
```
resumen_nan(df_copia)
```

[67]:

	NaN_count	Total	NaN_percent
Date	0	112925	0.0
Location	0	112925	0.0
Min_Temp	0	112925	0.0
Max_Temp	0	112925	0.0
Leakage	0	112925	0.0
Parameter1_Dir	0	112925	0.0
Parameter1_Speed	0	112925	0.0
Parameter2_9am	0	112925	0.0
Parameter2_3pm	0	112925	0.0
Parameter3_9am	0	112925	0.0
Parameter3_3pm	0	112925	0.0
Parameter4_9am	0	112925	0.0
Parameter4_3pm	0	112925	0.0
Parameter5_9am	0	112925	0.0
Parameter5_3pm	0	112925	0.0
Parameter7_9am	0	112925	0.0
Parameter7_3pm	0	112925	0.0
Failure_today	0	112925	0.0

2.1 fin limpieza de nan

TRABAJAMOS CON DF_COPIA1

[68]:
```
#ya que no tenemos nan, procedemos a cambiar el formato de las variables object
df_nan=df_copia.copy()

df_nan.info()

#corregimos los indicces
df_nan = df_nan.reset_index(drop=True)
df_nan
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 112925 entries, 0 to 142192
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Date              112925 non-null  object
 1   Location          112925 non-null  int64
 2   Min_Temp          112925 non-null  float64
 3   Max_Temp          112925 non-null  float64
 4   Leakage           112925 non-null  float64
 5   Parameter1_Dir    112925 non-null  object
 6   Parameter1_Speed  112925 non-null  float64
 7   Parameter2_9am    112925 non-null  object
 8   Parameter2_3pm    112925 non-null  object
 9   Parameter3_9am    112925 non-null  float64
 10  Parameter3_3pm    112925 non-null  float64
 11  Parameter4_9am    112925 non-null  float64
 12  Parameter4_3pm    112925 non-null  float64
 13  Parameter5_9am    112925 non-null  float64
 14  Parameter5_3pm    112925 non-null  float64
 15  Parameter7_9am    112925 non-null  float64
 16  Parameter7_3pm    112925 non-null  float64
 17  Failure_today     112925 non-null  object
dtypes: float64(12), int64(1), object(5)
memory usage: 16.4+ MB
```

[68]:

	Date	Location	Min_Temp	Max_Temp	Leakage	Parameter1_Dir	\
0	12/1/2008	3	13.4	22.9	0.6	W	
1	12/2/2008	3	7.4	25.1	0.0	WNW	
2	12/3/2008	3	12.9	25.7	0.0	WSW	
3	12/4/2008	3	9.2	28.0	0.0	NE	
4	12/5/2008	3	17.5	32.3	1.0	W	
…	…	…	…	…	…	…	
112920	6/20/2017	42	3.5	21.8	0.0	E	
112921	6/21/2017	42	2.8	23.4	0.0	E	
112922	6/22/2017	42	3.6	25.3	0.0	NNW	
112923	6/23/2017	42	5.4	26.9	0.0	N	
112924	6/24/2017	42	7.8	27.0	0.0	SE	

	Parameter1_Speed	Parameter2_9am	Parameter2_3pm	Parameter3_9am	\
0	44.0	W	WNW	20.0	
1	44.0	NNW	WSW	4.0	
2	46.0	W	WSW	19.0	
3	24.0	SE	E	11.0	
4	41.0	ENE	NW	7.0	
…	…	…	…	…	
112920	31.0	ESE	E	15.0	

```
112921              31.0             SE            ENE             13.0
112922              22.0             SE              N             13.0
112923              37.0             SE            WNW              9.0
112924              28.0            SSE              N             13.0

        Parameter3_3pm  Parameter4_9am  Parameter4_3pm  Parameter5_9am  \
0                 24.0            71.0            22.0          1007.7
1                 22.0            44.0            25.0          1010.6
2                 26.0            38.0            30.0          1007.6
3                  9.0            45.0            16.0          1017.6
4                 20.0            82.0            33.0          1010.8
…                    …               …               …               …
112920            13.0            59.0            27.0          1024.7
112921            11.0            51.0            24.0          1024.6
112922             9.0            56.0            21.0          1023.5
112923             9.0            53.0            24.0          1021.0
112924             7.0            51.0            24.0          1019.4

        Parameter5_3pm  Parameter7_9am  Parameter7_3pm Failure_today
0               1007.1            16.9            21.8            No
1               1007.8            17.2            24.3            No
2               1008.7            21.0            23.2            No
3               1012.8            18.1            26.5            No
4               1006.0            17.8            29.7            No
…                  …               …               …             …
112920          1021.2             9.4            20.9            No
112921          1020.3            10.1            22.4            No
112922          1019.1            10.9            24.5            No
112923          1016.8            12.5            26.1            No
112924          1016.5            15.1            26.0            No

[112925 rows x 18 columns]
```

2.2 TRANSFORMACION DE DATOS

funcion para ajustar los parametros que son object

[69]:
```
def reemplazar_por_primera_letra(df, nombre_columna):
    df[nombre_columna] = df[nombre_columna].astype(str).str.upper().str[0]
    return df
```

[70]:
```
#ajustamos parametro1_Dir parametro 2() am y pm), puesto que la diferencia es␣
 ↪poca y es probable que hayan diferencias mas notorias en este formato



df_nan=reemplazar_por_primera_letra(df_nan, "Parameter1_Dir")
```

```
df_nan=reemplazar_por_primera_letra(df_nan, "Parameter2_9am")

df_nan=reemplazar_por_primera_letra(df_nan, "Parameter2_3pm")


#vemos el df_nan con los parametros ajustados
print(df_nan["Parameter1_Dir"].unique(),
df_nan["Parameter2_9am"].unique(),
df_nan["Parameter2_3pm"].unique())
```

```
['W' 'N' 'S' 'E'] ['W' 'N' 'S' 'E'] ['W' 'E' 'N' 'S']
```

[71]:
```
df_nan["Failure_today"].value_counts()
```

[71]:
```
Failure_today
No     87556
Yes    25369
Name: count, dtype: int64
```

Nos queda por ajustar failure today a binario, y aplicar el modelo ols, recordar que leakage tiene correlacion con la varaible dependiente, parametro 7 en am y pm tiene mucha correlacion con la temperatura min y max segun la matriz, por lo que las abstendremos del modelo(a usar juntas)

[72]:
```
#pasamos a binario el failure today
def convertir_binario(df, columna):
    df[columna] = df[columna].astype(str).str.upper().map({'YES': 1, 'NO': 0})
    return df
df_nan=convertir_binario(df_nan, "Failure_today")
```

[73]:
```
df_nan["Failure_today"].value_counts()
```

[73]:
```
Failure_today
0    87556
1    25369
Name: count, dtype: int64
```

[74]:
```
df_nan.dtypes
```

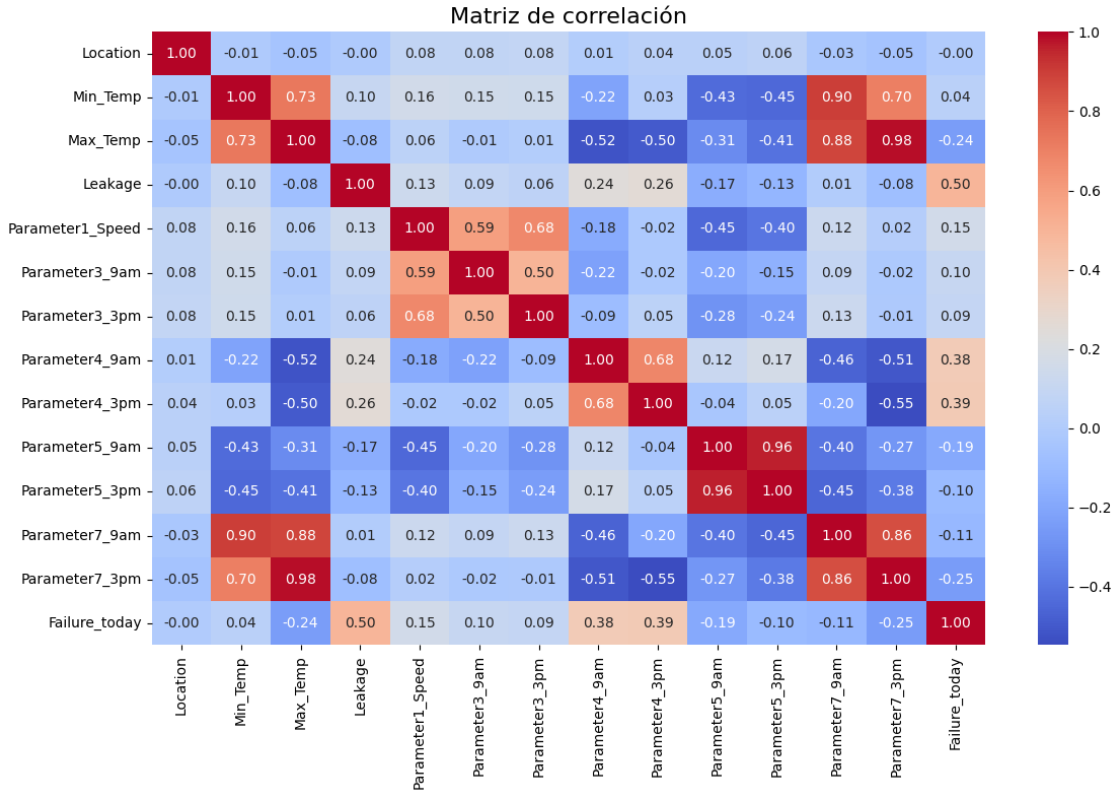
[74]:
```
Date                object
Location             int64
Min_Temp           float64
Max_Temp           float64
Leakage            float64
Parameter1_Dir      object
Parameter1_Speed   float64
Parameter2_9am      object
Parameter2_3pm      object
```

```
Parameter3_9am      float64
Parameter3_3pm      float64
Parameter4_9am      float64
Parameter4_3pm      float64
Parameter5_9am      float64
Parameter5_3pm      float64
Parameter7_9am      float64
Parameter7_3pm      float64
Failure_today         int64
dtype: object
```

HACEMOS LA MATRIZ DE CORRELACION NUEVAMENTE

[75]:
```
#FUNCION DE HACER LA MATRIZ
def matriz_correlacion(df, tamaño=(12, 8), cmap='coolwarm', annot=True, fmt='.
↪2f'):
    # Seleccionar solo columnas numéricas
    df_numerico = df.select_dtypes(include=['number'])

    # Calcular matriz de correlación
    correlacion = df_numerico.corr()

    # Crear el gráfico
    plt.figure(figsize=tamaño)
    sns.heatmap(correlacion, cmap=cmap, annot=annot, fmt=fmt)
    plt.title("Matriz de correlación", fontsize=16)
    plt.tight_layout()
    plt.show()
```

[76]:
```
matriz_correlacion(df_nan, tamaño=(12,8), cmap="coolwarm", annot=True, fmt=".
↪2f")
```



[77]:
```
df_nan.dtypes
```

[77]:
```
Date                object
Location             int64
Min_Temp           float64
Max_Temp           float64
Leakage            float64
Parameter1_Dir      object
Parameter1_Speed   float64
Parameter2_9am      object
Parameter2_3pm      object
Parameter3_9am     float64
Parameter3_3pm     float64
Parameter4_9am     float64
Parameter4_3pm     float64
Parameter5_9am     float64
Parameter5_3pm     float64
Parameter7_9am     float64
Parameter7_3pm     float64
Failure_today        int64
dtype: object
```

2.3 Fin pregunta 1

2.4 pregunta 2, Modelo OLS

[78]:
```
model = smf.ols("Failure_today ~ Parameter1_Speed + Parameter4_9am + Min_Temp +␣
↪Parameter7_3pm ", data=df_nan).fit()
print(model.summary()) #el parametro 4 de la mañana explica mas
#df_nan.columns
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          Failure_today   R-squared:                       0.236
Model:                            OLS   Adj. R-squared:                  0.236
Method:                 Least Squares   F-statistic:                     8715.
Date:                Fri, 25 Apr 2025   Prob (F-statistic):               0.00
Time:                        00:06:05   Log-Likelihood:                -46367.
No. Observations:              112925   AIC:                         9.274e+04
Df Residuals:                  112920   BIC:                         9.279e+04
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
Intercept     -0.3308      0.009    -34.882      0.000      -0.349
-0.312
Parameter1_Speed     0.0054   8.63e-05     62.752      0.000       0.005
0.006
Parameter4_9am       0.0072   7.11e-05    100.692      0.000       0.007
0.007
Min_Temp             0.0197      0.000     75.574      0.000       0.019
0.020
Parameter7_3pm      -0.0180      0.000    -66.661      0.000      -0.018
-0.017
==============================================================================
Omnibus:                    10967.998   Durbin-Watson:                   1.716
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            13218.282
Skew:                           0.812   Prob(JB):                         0.00
Kurtosis:                       2.587   Cond. No.                         735.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

Respuesta pregunta 2:

Se eligieron 4 variables para este modelo dada su baja correlacion entre otras variables, y su no alta correlacion con la variable dependiente. Luego tenemos que el aumento de una unidad de las variables: Parameter4_9am, Parameter7_3pm, Min_temp, y parameter1_speed, reflejan, respectivamente, un/a aumento(A)/Disminucion(D) de (A)0,72% ;(D) 1,8% ; (A)1,97% ; (A) 0.54% en la cifra de variable de interes: failure_today, que en este caso como es binario vendria siendo un aumento en terminos de "probabilidad", sin embargo, en realidad, por cada unidad que aumenten cada variable independiente, las cuales varian en su respectiva medida, la probabilidad de fallar cambia en aquellas unidades.

2.5 Pregunta 3 modelo probit

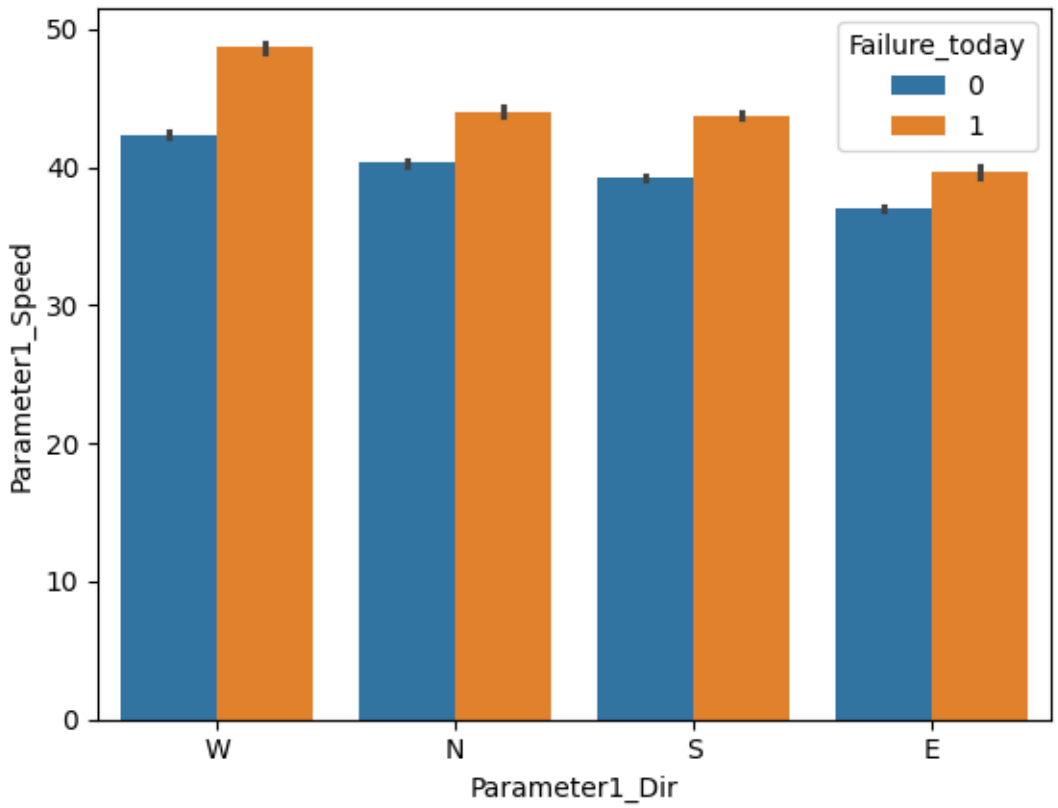
[79]:
```
probit_model = smf.probit("Failure_today ~ Parameter1_Speed + Parameter4_9am +␣
↪Min_Temp + Parameter7_3pm ", data=df_nan).fit()
print(probit_model.summary())

mfx = probit_model.get_margeff()
print(mfx.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.385756
         Iterations 7
                          Probit Regression Results
==============================================================================
Dep. Variable:          Failure_today   No. Observations:               112925
Model:                         Probit   Df Residuals:                   112920
Method:                           MLE   Df Model:                            4
Date:                Fri, 25 Apr 2025   Pseudo R-squ.:                  0.2759
Time:                        00:06:06   Log-Likelihood:                -43562.
converged:                       True   LL-Null:                       -60159.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
====
                        coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
----
Intercept           -3.3564      0.045    -75.269      0.000      -3.444
-3.269
Parameter1_Speed     0.0192      0.000     51.301      0.000       0.018
0.020
Parameter4_9am       0.0356      0.000     98.393      0.000       0.035
0.036
Min_Temp             0.1051      0.001     79.606      0.000       0.103
0.108
Parameter7_3pm      -0.1019      0.001    -73.766      0.000      -0.105
-0.099
==============================================================================
====
```

```
       Probit Marginal Effects
=====================================
Dep. Variable:          Failure_today
Method:                          dydx
At:                           overall
==============================================================================
====
                       dy/dx    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
----
Parameter1_Speed     0.0042   7.81e-05     53.306      0.000       0.004
0.004
Parameter4_9am       0.0077   6.91e-05    111.884      0.000       0.008
0.008
Min_Temp             0.0228      0.000     86.247      0.000       0.022
0.023
Parameter7_3pm      -0.0221      0.000    -78.940      0.000      -0.023
-0.022
==============================================================================
====
```

Respuesta pregunta 3: Los coeficientes son algo bajo(efectos marginales), donde tenemos que la velocidad y el parametro 4 afectan bastante poco la probabilidad de que falle la maquina, mas para el parametro 7 y la temperatura minima, este efecto es un poco mayor por unidad, donde es particular llama la atencion la temperatura minima, pueesto que tiene un amplio rango de valores, podria darse un efecto interesante en la probabilidad de que falle la maquina producto del aumento de la temperatura minima, donde de momento se tiene que por cada unidad que aumente la temperatura minima, la probabilidad de que falle la maquina aumenta en un 2,28%. por otro lado para el parametro 7 baja en un 2,21% la probabilidad que falle la maquina por cada unidad que aumente, la relacion contraria a la temperatura minima.

[80]:
```
sns.barplot(x="Parameter1_Dir", y="Parameter1_Speed",␣
 ↪data=df_nan,hue="Failure_today")
plt.show()
```



[81]:
```
##########################################################################################
##########################################################################################
##########################################################################################
##########################################################################################

# Crear un diccionario para almacenar los outliers
outliers_dict = {}

#buscamos los limites para cada columna y vamos comparando
for col in df_nan.select_dtypes(include='number').columns:
    Q1 = df_nan[col].quantile(0.25)
    Q3 = df_nan[col].quantile(0.75)
    IQR = Q3 - Q1

    # Límites para outliers
    limite_inf= Q1 - 1.5 * IQR
    limite_sup= Q3 + 1.5 * IQR

    # Filas que son outliers en esta columna
    outliers = df_nan[(df_nan[col] < limite_inf) | (df_nan[col] > limite_sup)]
```

```
    # Guardar índices de outliers
    outliers_dict[col] = outliers.index.tolist()

for col, idx in outliers_dict.items():
    print(f"Columna: {col} → {len(idx)} outliers")

#ignorar para failure
```

```
Columna: Location → 0 outliers
Columna: Min_Temp → 26 outliers
Columna: Max_Temp → 76 outliers
Columna: Leakage → 20331 outliers
Columna: Parameter1_Speed → 2630 outliers
Columna: Parameter3_9am → 1989 outliers
Columna: Parameter3_3pm → 2192 outliers
Columna: Parameter4_9am → 1493 outliers
Columna: Parameter4_3pm → 0 outliers
Columna: Parameter5_9am → 1210 outliers
Columna: Parameter5_3pm → 917 outliers
Columna: Parameter7_9am → 57 outliers
Columna: Parameter7_3pm → 167 outliers
Columna: Failure_today → 25369 outliers
```

2.6 Pregunta 4 Modelo logit

[82]:
```
logit_model = smf.logit("Failure_today ~ Parameter1_Speed + Parameter4_9am +␣
↪Min_Temp + Parameter7_3pm ", data=df_nan).fit()
print(logit_model.summary())

mfx = logit_model.get_margeff()
print(mfx.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.384726
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:          Failure_today   No. Observations:               112925
Model:                          Logit   Df Residuals:                   112920
Method:                           MLE   Df Model:                            4
Date:                Fri, 25 Apr 2025   Pseudo R-squ.:                  0.2778
Time:                        00:06:09   Log-Likelihood:                -43445.
converged:                       True   LL-Null:                       -60159.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
====
                       coef    std err          z      P>|z|      [0.025
0.975]
```

```
------------------------------------------------------------------------------
----
Intercept           -5.9588      0.081    -73.517      0.000      -6.118
-5.800
Parameter1_Speed     0.0330      0.001     50.185      0.000       0.032
0.034
Parameter4_9am       0.0643      0.001     96.665      0.000       0.063
0.066
Min_Temp             0.1879      0.002     78.841      0.000       0.183
0.193
Parameter7_3pm      -0.1822      0.003    -72.332      0.000      -0.187
-0.177
==============================================================================
====
        Logit Marginal Effects
=====================================
Dep. Variable:          Failure_today
Method:                          dydx
At:                           overall
==============================================================================
====
                      dy/dx    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
----
Parameter1_Speed     0.0041   7.71e-05     52.629      0.000       0.004
0.004
Parameter4_9am       0.0079   6.87e-05    115.005      0.000       0.008
0.008
Min_Temp             0.0231      0.000     87.053      0.000       0.023
0.024
Parameter7_3pm      -0.0224      0.000    -78.446      0.000      -0.023
-0.022
==============================================================================
====
```

2.7 Respuesta pregunta 4

R2 y significancia, sin cambios mayores. Este caso notemos que los cambios por unidad fluctuada de las variables en cuestion han aumentado otra vez, y se repite el patron en el cual la temperatura minima y el parametro 7 afectan mucho mas a la probabilidad de que falle la maquina, es decir, la probabilidad de que falle la maquina es mucho mas sensible a estas variables. los cambios esta vez fueron: - >(probit a logit) - parametro velocidad:0,42% -> 0,41% (bajó) - parametro 4 9am: 0,77% -> 0,79% (subio) - temperatura minima: 2,28% -> 2,31%% (subio) - parametro 7 3pm: -2,21%-> -2,24% (subio)

donde en esta ocasion un cambio en una unidad de temperaura minima de la maquina, puede afectar en un 2,31% la probabilidad que falle la maquina, y por otro lado, un cambio de una unidad en el

parametro 7 disminuye un 2,24% la probabilidad que ocurra una falla. ademas los cambios fueron bastantes bajos, mas es interesante como el efecto, si bien minimo, de la velocidad del viento, se ve reducido en esta ocasion.

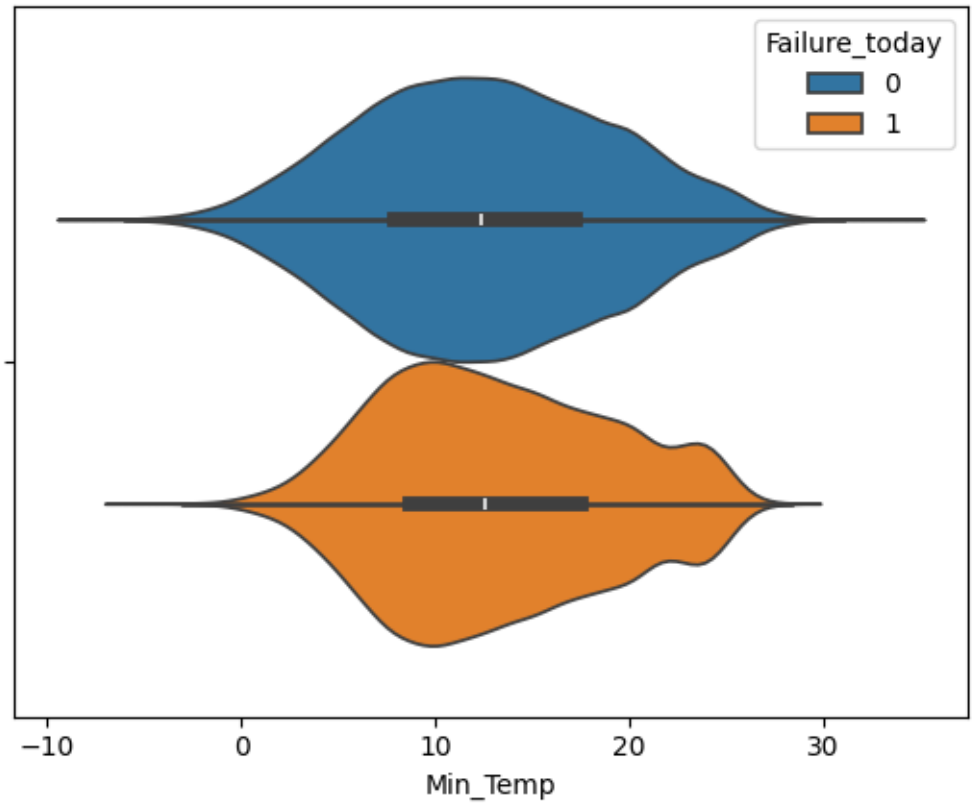
2.8 pregunta 5

Respuesta pregunta 5: el modelo ols es el peor, pues su estructura no interpreta bien variables que no sean continuas como es el caso de que falle o no la maquina(que es binario), por otro lado los modelos logit y probit estas mas adecuados a esta situación.

Tenemos que los aumentaron, una cierta cantidad, excepto en el parametro1_ la velocidad del viento. Luego, es mas conveniente usar un modelo logit a uno probit, dado que en este caso los datos tienen un rango de valores amplio y en algunos casos asimetricos, por lo que el probit se ajustará un poco peor que el logit, dado que usa la normal para calcular los coeficientes, distriucion que no es asimetrica ni trata de dispersar datos, al contario, de juntarlos en la llamada "campana", mas el modelo logit tiene la caracteristica de ajustarse mejor a los datos mas dispersos dada su naturaleza

[83]:
```
sns.violinplot( x=df_nan["Min_Temp"], hue=df_nan["Failure_today"])
```

[83]:
```
<Axes: xlabel='Min_Temp'>
```



2.9 pregunta 6

[84]:
```
# debemos agrupar la data por mes y por fecha
df_nan["Date"].info()

df_fecha=df_nan.copy()

df_nan["Date"].head(10)
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 112925 entries, 0 to 112924
Series name: Date
Non-Null Count   Dtype
--------------   -----
112925 non-null  object
dtypes: object(1)
memory usage: 882.4+ KB
```

[84]:
```
0     12/1/2008
1     12/2/2008
2     12/3/2008
3     12/4/2008
4     12/5/2008
5     12/6/2008
6     12/7/2008
7     12/8/2008
8     12/9/2008
9    12/10/2008
Name: Date, dtype: object
```

[85]:
```
# 1. Convertir la fecha correctamente
df_fecha["Date"] = pd.to_datetime(df_fecha["Date"], format="%m/%d/%Y",␣
 ↪errors="coerce")

# 2. Filtrar por año (por ejemplo, > 2008)
df_fecha = df_fecha[df_fecha["Date"].dt.year > 2008]

# 3. Crear columna año-mes para agrupar
df_fecha["Mes_Año"] = df_fecha["Date"].dt.to_period("M").astype(str)
```

[86]:
```
# pasamos a location a categorico
df_fecha["Location"] = df_fecha["Location"].astype("category")
df_fecha["Location"]
# ya tenemos listo location y fechas, procedemos a agrupar y calcular las␣
 ↪agregaciones
```

[86]:
```
28         3
29         3
```

```
30          3
31          3
32          3
           ..
112920     42
112921     42
112922     42
112923     42
112924     42
Name: Location, Length: 111179, dtype: category
Categories (44, int64): [1, 3, 4, 5, …, 46, 47, 48, 49]
```

[87]:
```
resumen_nan(df_fecha)
```

[87]:
```
                   NaN_count   Total  NaN_percent
Date                       0  111179          0.0
Location                   0  111179          0.0
Min_Temp                   0  111179          0.0
Max_Temp                   0  111179          0.0
Leakage                    0  111179          0.0
Parameter1_Dir             0  111179          0.0
Parameter1_Speed           0  111179          0.0
Parameter2_9am             0  111179          0.0
Parameter2_3pm             0  111179          0.0
Parameter3_9am             0  111179          0.0
Parameter3_3pm             0  111179          0.0
Parameter4_9am             0  111179          0.0
Parameter4_3pm             0  111179          0.0
Parameter5_9am             0  111179          0.0
Parameter5_3pm             0  111179          0.0
Parameter7_9am             0  111179          0.0
Parameter7_3pm             0  111179          0.0
Failure_today              0  111179          0.0
Mes_Año                    0  111179          0.0
```

[88]:
```
df_grouped = df_fecha.groupby(["Location", "Mes_Año"]).agg({
    "Min_Temp": "mean",
    "Max_Temp": "mean",
    "Leakage": "mean",
    "Parameter1_Speed": "mean",
    "Parameter3_9am": "mean",
    "Parameter3_3pm": "mean",
    "Parameter4_9am": "mean",
    "Parameter4_3pm": "mean",
    "Parameter5_9am": "mean",
    "Parameter5_3pm": "mean",
    "Parameter7_9am": "mean",
```

```
    "Parameter7_3pm": "mean",
    "Failure_today": "sum",  # <- suma
}).reset_index()
```

[89]:
```
df_grouped
```

[89]:

	Location	Mes_Año	Min_Temp	Max_Temp	Leakage	Parameter1_Speed	\
0	1	2009-01	17.975862	31.868966	0.041379	39.965517	
1	1	2009-02	18.855556	31.485185	0.029630	40.481481	
2	1	2009-03	15.220833	25.370833	0.891667	37.750000	
3	1	2009-04	13.207143	22.792857	2.628571	35.857143	
4	1	2009-05	10.966667	18.170833	1.966667	30.291667	
…	…	…	…	…	…	…	
4351	49	2017-02	19.546429	34.232143	0.000000	46.464286	
4352	49	2017-03	18.745161	33.732258	0.000000	43.612903	
4353	49	2017-04	13.572414	24.796552	1.403448	35.758621	
4354	49	2017-05	9.277419	20.938710	0.341935	33.580645	
4355	49	2017-06	5.952174	18.747826	0.008696	28.000000	

	Parameter3_9am	Parameter3_3pm	Parameter4_9am	Parameter4_3pm	\
0	10.448276	17.931034	38.689655	23.827586	
1	8.481481	18.185185	42.370370	29.407407	
2	8.583333	16.750000	59.166667	43.458333	
3	10.035714	16.500000	57.821429	45.857143	
4	8.458333	11.750000	73.916667	60.833333	
…	…	…	…	…	
4351	23.178571	20.928571	49.964286	24.285714	
4352	20.387097	18.419355	49.387097	21.806452	
4353	18.586207	17.172414	56.034483	38.379310	
4354	14.741935	17.290323	65.258065	37.677419	
4355	11.391304	13.391304	66.565217	36.608696	

	Parameter5_9am	Parameter5_3pm	Parameter7_9am	Parameter7_3pm	\
0	1014.327586	1012.324138	23.510345	30.579310	
1	1014.540741	1012.822222	23.292593	29.266667	
2	1017.641667	1016.483333	18.716667	23.825000	
3	1019.621429	1017.425000	17.610714	21.360714	
4	1024.575000	1022.308333	14.304167	17.200000	
…	…	…	…	…	
4351	1013.971429	1011.989286	23.560714	32.203571	
4352	1014.780645	1012.367742	22.170968	32.074194	
4353	1022.668966	1019.606897	18.596552	23.644828	
4354	1022.958065	1020.187097	13.806452	20.267742	
4355	1029.586957	1026.939130	10.556522	18.052174	

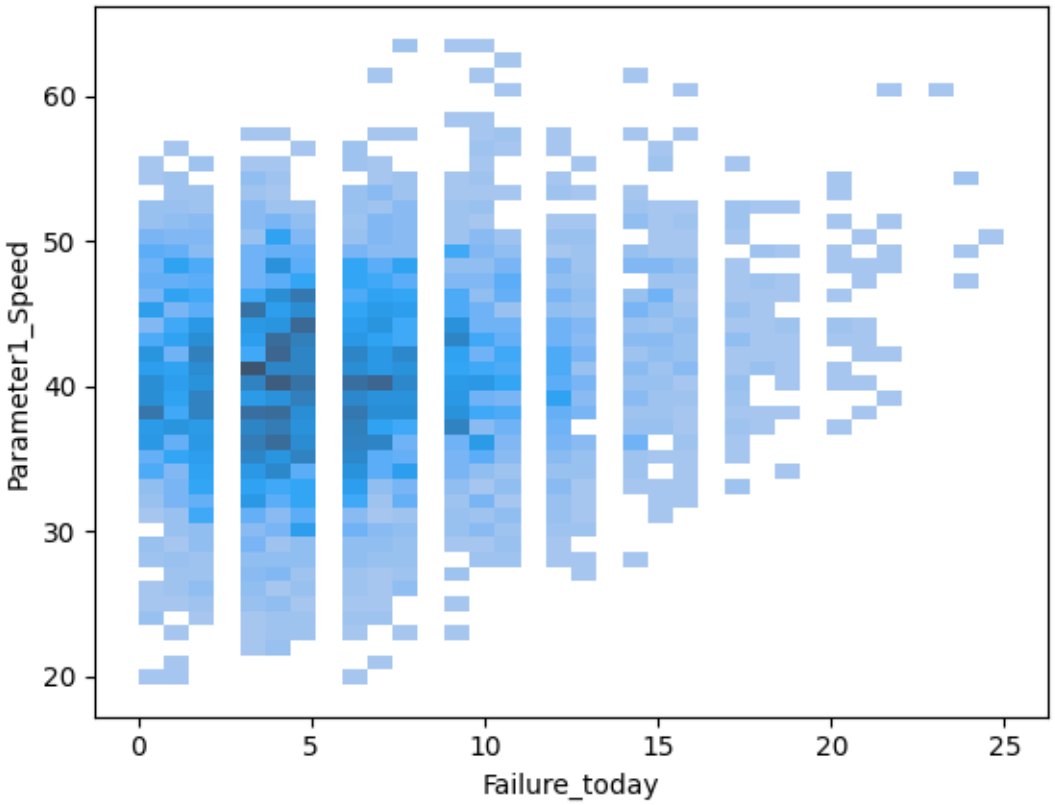
	Failure_today
0	0

```
1                    0
2                    4
3                    6
4                    6
…                  …
4351                 0
4352                 0
4353                 4
4354                 1
4355                 0

[4356 rows x 15 columns]
```

2.10 se obtiene el df_grouped, con la media de las variables, agrupadas por location y periodo mensual

[90]:
```
#graficos df_grouped

sns.histplot(y=df_grouped["Parameter1_Speed"], x=df_grouped["Failure_today"])
```

[90]:
```
<Axes: xlabel='Failure_today', ylabel='Parameter1_Speed'>
```



[91]:
```
# Ajustar modelo Poisson
poisson_model = smf.glm(
    formula="Failure_today ~ Parameter1_Speed + Parameter4_9am + Min_Temp +␣
 ↪Parameter7_3pm ",
    data=df_grouped,
    family=sm.families.Poisson()
).fit()

# resumen
print(poisson_model.summary())

# ver efectos marginales
mfx = poisson_model.get_margeff()
print(mfx.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:          Failure_today   No. Observations:                 4076
Model:                            GLM   Df Residuals:                     4071
Model Family:                 Poisson   Df Model:                            4
Link Function:                    Log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -9799.2
Date:                Fri, 25 Apr 2025   Deviance:                       6073.3
Time:                        00:06:13   Pearson chi2:                 5.53e+03
No. Iterations:                     5   Pseudo R-squ. (CS):             0.7980
Covariance Type:            nonrobust
==============================================================================
====
                       coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
----
Intercept            0.0678      0.112      0.607      0.544      -0.151
0.286
Parameter1_Speed     0.0186      0.001     16.767      0.000       0.016
0.021
Parameter4_9am       0.0254      0.001     32.304      0.000       0.024
0.027
Min_Temp             0.1050      0.003     40.294      0.000       0.100
0.110
Parameter7_3pm      -0.0998      0.003    -33.035      0.000      -0.106
-0.094
==============================================================================
====
        GLM Marginal Effects
=====================================
Dep. Variable:          Failure_today
Method:                          dydx
```

```
At:                              overall
================================================================================
====
                       dy/dx    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
----
Parameter1_Speed      0.1141      0.007     16.673      0.000       0.101
0.128
Parameter4_9am        0.1555      0.005     31.650      0.000       0.146
0.165
Min_Temp              0.6432      0.016     39.045      0.000       0.611
0.675
Parameter7_3pm       -0.6112      0.019    -32.336      0.000      -0.648
-0.574
================================================================================
====
```

- **RESPUESTA 6:** En este caso el modelo poisson, entrega los efectos respecto no a un aumento de la probabilidad que suceda o no, sino a la cantidad de veces que sucede(cantidad de fallos), por ende podemos ver que en este caso los efectos marginales de las variables son:
- Parameter1_Speed 0.1141
- Parameter4_9am 0.1555
- Min_Temp 0.6432

- Parameter7_3pm -0.6112 y estos indican, por ejemplo para el caso de MIn_temp, el aumento en una unidad tiene un efecto, en promedio, en aumentar la cantidad de fallas en el periodo mensual en 0,65 veces. por otro lado el aumento del parametro7_3pm implica una disminucion en promedio de 0,6 veces en el numero de fallas mensuales.

notemos que el efecto mas grande es otra vez con los parametros de min_temp y parametro 7, patron que se viene repitiendo de manera similar

2.11 Pregunta 7 SOBREDISPERSION

[ ]:
```
############################################################
################## SECCION CORREGIDA ######################
############################################################


poisson_model = smf.glm("Failure_today ~ Parameter1_Speed + Parameter4_9am +␣
 ↪Min_Temp + Parameter7_3pm",
                        data=df_grouped, family=sm.families.Poisson()).fit()

# Guardar el dataframe limpio
df_dispersion = poisson_model.model.data.frame
```

```python
# Obtener los valores ajustados (fitted values)
mu = poisson_model.fittedvalues

# Nos aseguramos de que los índices sean los mismos
df_dispersion = df_dispersion.loc[mu.index]

# funcion de sobredispersión
aux = ((df_dispersion["Failure_today"] - mu) ** 2 - mu) / mu

# Ajustar un modelo OLS a los residuos
auxr = sm.OLS(aux, mu).fit()

# Imprimir el resumen del modelo OLS
print(auxr.summary())
```

```
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:                      y   R-squared (uncentered):
0.034
Model:                            OLS   Adj. R-squared (uncentered):
0.033
Method:                 Least Squares   F-statistic:
141.3
Date:                Fri, 25 Apr 2025   Prob (F-statistic):
4.75e-32
Time:                        09:42:58   Log-Likelihood:
-8419.6
No. Observations:                4076   AIC:
1.684e+04
Df Residuals:                    4075   BIC:
1.685e+04
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
x1             0.0516      0.004     11.885      0.000       0.043       0.060
================================================================================
Omnibus:                     2682.567   Durbin-Watson:                   1.633
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            37578.309
Skew:                           2.981   Prob(JB):                         0.00
Kurtosis:                      16.628   Cond. No.                         1.00
================================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not
```

contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[ ]:
```
####################################
######### SECCION COMENTADA ########

####################################

# Calcular residuos del modelo Poisson
#aux = ((df_grouped["Failure_today"] - poisson_model.mu) ** 2 - poisson_model.
 ↪mu) / poisson_model.mu

# Ajustar un modelo OLS a estos residuos
#auxr = sm.OLS(aux, poisson_model.mu).fit()

# Imprimir el resumen del modelo OLS para ver los resultados
#print(auxr.summary())
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[103], line 2
      1 # Calcular residuos del modelo Poisson
----> 2 aux = ((df_grouped["Failure_today"] - poisson_model.mu) ** 2 -␣
 ↪poisson_model.mu) / poisson_model.mu
      4 # Ajustar un modelo OLS a estos residuos
      5 auxr = sm.OLS(aux, poisson_model.mu).fit()

File d:
 ↪\benja\Universidad\PYTHON(LENGUAJE)\Lib\site-packages\pandas\core\ops\common.
 ↪py:76, in _unpack_zerodim_and_defer.<locals>.new_method(self, other)
     72             return NotImplemented
     74 other = item_from_zerodim(other)
---> 76 return method(self, other)

File d:
 ↪\benja\Universidad\PYTHON(LENGUAJE)\Lib\site-packages\pandas\core\arraylike.p
 ↪194, in OpsMixin.__sub__(self, other)
    192 @unpack_zerodim_and_defer("__sub__")
    193 def __sub__(self, other):
--> 194     return self._arith_method(other, operator.sub)

File d:\benja\Universidad\PYTHON(LENGUAJE)\Lib\site-packages\pandas\core\series
 ↪py:6135, in Series._arith_method(self, other, op)
   6133 def _arith_method(self, other, op):
   6134     self, other = self._align_for_op(other)
-> 6135     return base.IndexOpsMixin._arith_method(self, other, op)
```

```
File d:\benja\Universidad\PYTHON(LENGUAJE)\Lib\site-packages\pandas\core\base.py:
 ↪1382, in IndexOpsMixin._arith_method(self, other, op)
   1379     rvalues = np.arange(rvalues.start, rvalues.stop, rvalues.step)
   1381 with np.errstate(all="ignore"):
-> 1382     result = ops.arithmetic_op(lvalues, rvalues, op)
   1384 return self._construct_result(result, name=res_name)

File d:
 ↪\benja\Universidad\PYTHON(LENGUAJE)\Lib\site-packages\pandas\core\ops\array_ops.
 ↪py:283, in arithmetic_op(left, right, op)
    279     _bool_arith_check(op, left, right)  # type: ignore[arg-type]
    281     # error: Argument 1 to "_na_arithmetic_op" has incompatible type
    282     # "Union[ExtensionArray, ndarray[Any, Any]]"; expected "ndarray[Any ␣
 ↪Any]"
--> 283     res_values = _na_arithmetic_op(left, right, op)  # type:␣
 ↪ignore[arg-type]
    285 return res_values

File d:
 ↪\benja\Universidad\PYTHON(LENGUAJE)\Lib\site-packages\pandas\core\ops\array_ops.
 ↪py:218, in _na_arithmetic_op(left, right, op, is_cmp)
    215     func = partial(expressions.evaluate, op)
    217 try:
--> 218     result = func(left, right)
    219 except TypeError:
    220     if not is_cmp and (
    221         left.dtype == object or getattr(right, "dtype", None) == object
    222     ):
   (…)
    225         # Don't do this for comparisons, as that will handle complex␣
 ↪numbers
    226         #  incorrectly, see GH#32047

File d:
 ↪\benja\Universidad\PYTHON(LENGUAJE)\Lib\site-packages\pandas\core\computation\expressions.
 ↪py:242, in evaluate(op, a, b, use_numexpr)
    239 if op_str is not None:
    240     if use_numexpr:
    241         # error: "None" not callable
--> 242         return _evaluate(op, op_str, a, b)  # type: ignore[misc]
    243 return _evaluate_standard(op, op_str, a, b)

File d:
 ↪\benja\Universidad\PYTHON(LENGUAJE)\Lib\site-packages\pandas\core\computation\expressions.
 ↪py:73, in _evaluate_standard(op, op_str, a, b)
     71 if _TEST_MODE:
     72     _store_test_result(False)
---> 73 return op(a, b)
```

```
ValueError: operands could not be broadcast together with shapes (4356,) (4076,
```

[93]:
```
#aux=((df_grouped["Failure_today"]-poisson_model.mu)**2-poisson_model.mu)/
 ↪poisson_model.mu
#auxr=sm.OLS(aux,poisson_model.mu).fit()
#print(auxr.summary())
```

[104]:
```
#alpha de poisson
alpha_poisson= np.exp(0.0516)
alpha_poisson
```

[104]:
```
np.float64(1.052954476475184)
```

RESPUESTA PREGUNTA 7: EL ALPHA PARA LA POISSON NOS DA UN VALOR DE 1.05(significativo) APROXIMADAMENTE. - Luego, esto indica que hay una sobredispersion considerable, lo que indica que el modelo poisson podria no ser del todo adecuado, ya que este propone que la media sea igual a la varianza, mas este valor de alfa nos indica que la varianza suele aumentar en una magnitud mayor a la media, por lo que no armoniza con lo que propone el modelo de poisson, es decir, en realidad los datos estan mas dispersos de lo que "cree poisson"(sobredispersion).

2.12 Pregunta 8 Binomial negativa

[109]:
```
#resultado modelo binomial negativa

nbin = smf.negativebinomial("Failure_today ~ Parameter1_Speed + Parameter4_9am␣
 ↪+ Min_Temp + Parameter7_3pm", data=df_grouped).fit()
print(nbin.summary())
```

```
Optimization terminated successfully.
         Current function value: 2.377483
         Iterations: 26
         Function evaluations: 37
         Gradient evaluations: 37
                     NegativeBinomial Regression Results
==============================================================================
Dep. Variable:          Failure_today   No. Observations:                 4076
Model:               NegativeBinomial   Df Residuals:                     4071
Method:                          MLE   Df Model:                            4
Date:                Fri, 25 Apr 2025   Pseudo R-squ.:                 0.1414
Time:                        09:51:56   Log-Likelihood:                -9690.6
converged:                       True   LL-Null:                       -11287.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
====
                       coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
----
```

```
Intercept            0.0749      0.132       0.566      0.571      -0.185
0.334
Parameter1_Speed     0.0187      0.001      14.167      0.000       0.016
0.021
Parameter4_9am       0.0255      0.001      27.257      0.000       0.024
0.027
Min_Temp             0.1067      0.003      34.509      0.000       0.101
0.113
Parameter7_3pm      -0.1017      0.004     -28.562      0.000      -0.109
-0.095
alpha                0.0580      0.005      11.376      0.000       0.048
0.068
==============================================================================
====
```

[106]:
```
#el valor de la binomial negativa es exp(valor alpha)

alpha_bn= np.exp(0.0580)

#comparamos valores de alpha, el estimado por la regresion auxiliar y el de la␣
 ↪binomial
print("alpha poisson: ", alpha_poisson, "alpha BN: ", alpha_bn)

print("diferencia: ", (alpha_bn-alpha_poisson)*100, ", diferencia de menos del␣
 ↪1%")
```

```
alpha poisson:  1.052954476475184 alpha BN:  1.0597149957102876
diferencia:  0.6760519235103679 , diferencia de menos del 1%
```

Respuesta pregunta 8: el valor de alfa estimado por la regresion auxiliar es muy cercano a el real que nos indica la binomial negativa, esto indica que el modelo BN es mas adecuado que el poisson para modelar los datos, puesto que este capta de mejor manera la varianza de ellos.En particular siendo algo mas preciso en numeros de fallas altos, lo que poisson no hace del todo bien, dado que los valores de concentran mas en cantidades de fallas bajas y al no tratar bien la dispersion de los datos, provoca imprecisiones por parte de poisson.

Luego los coeficientes de las variables en cuestion: - Parameter1_Speed => exp(0.0187) = 1.018 - Parameter4_9am => exp(0.0255) = 1.025 - Min_Temp => exp(0.1067) = 1.112 - Parameter7_3pm => exp(-0.1017) = 0.903

luego se tiene que un aumento de una unidad en los parametros, afecta(en su respectivo valor "x" de cada variable) en x veces a la variable failure_today, es decir, para la velocidad del viento por ejemplo, por cada unidad que aumente este, la cantidad de fallos mensuales, en promedio, subira 1.018 veces.

[107]:
```
print("Parameter1_Speed =>", np.exp(0.0187),"\n"
      "Parameter4_9am =>", np.exp(0.0255), "\n"
      "Min_Temp =>", np.exp(0.1067),"\n"
      "Parameter7_3pm => ", np.exp(-0.1017))
```

```
Parameter1_Speed => 1.018875939981411
Parameter4_9am => 1.0258279062704445
Min_Temp => 1.1126004242800154
Parameter7_3pm =>  0.9033005011747712
```

2.13 Pregunta 9

respuesta 9: Existen diferencias en los resultados porque se hacen supuestos en los modelos que en realidad no son del todo asi, como el que realiza poisson con la varianza igual a la media, donde esto en realidad (como podemos ver en la regresion auxiliar) no es asi, sino que en realidad la varianza es mayor, por lo que el modelo poisson no termina de capturar bien toda la informacion, en particular la que esta mas dispersa, tales como un numero de fallas alto por ejemplo. Por lo que, el modelo de la binomial negativa si captura esta dispersion de los datos y ajusta mejor los coeficientes, siendo estos mas representativos.

Por otro lado podemos señalar cierta robustes en los coeficientes, sobre todo en min_temp y parametro 7, ya que estos a pesar del cambio de modelos y metricas calculadas siguen teniendo una participacion no muy variada, por lo que su contribucion es de tal manera que no se ve afectada en gran medida por estos factores(aunque su magnitud si, pero no la forma)(poseen cierta robustez). Luego, por lo dicho anteriormente, se concluye que el modelo mas adecuado en este caso seria el de la binomial negativa.

[108]:
```
df_grouped.shape
```

[108]:
```
(4356, 15)
```

[ ]:
```

```