Universidad Nacional de Córdoba - Facultad de Matemática, Astronomía, Física y Computación

Diplomatura en Ciencia de Datos, Aprendizaje Automático y sus Aplicaciones

Práctico I - Estadística

Análisis y Visualización de Datos - 2019 - Mentoría

Integrante:

- Tarletta Juan

In [1]:

```
#Importamos las librerías necesarias para trabajar en el práctico
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import numpy as np
import pandas as pd
import seaborn
import scipy.stats as stats
import statsmodels.api as sm
import random
from IPython.display import display, Markdown
```

In [2]:

```
#Equipo: Indica si corresponde al mismo camion
#Componente: Indica a que componente pertenece la muestra
#Horas Funcionamiento: Indica la cantidad de horas de funcionamiento del camión (sería
#Horas del Aceite: Representa la cantidad de horas de utilización del aceite (este dat
#Resultado: (El laboratorio indica si la muestra de aceite está Bien = 1, Regular=2, M
#St: Presencia de Hollin en el Aceite
#Al: Presencia de Aluminio en el Aceite
#Fe Presencia de Hierro en el Aceite
#Si Presencia de Silicio en el Aceite
#Na Presencia de Sodio en el Aceite
#Visco: Viscosidad del aceite
```

1. Análisis General de las variables

1. Realizar un análisis de las variables que presenta el dataset (al margen de analizar los features mas importantes que comentamos al inicio del notebook, revisar el resto de los features)

In [3]:

```
dataset = pd.read_csv('OilDataSet.csv')
important_cols= ['Equipo', 'Componente','Resultado','Horas Funcionamiento','Horas del A

dataset[important_cols]

#De la función "describe" localizamos los valores mínimos y máximos(filas), para todas
min_max = dataset[important_cols].describe().loc[['min','max'],:]

mini = min_max.loc['min']
maxi = min_max.loc['max']

#Obtenemos el rango
rango = maxi - mini


#Convertimos la "Serie" del rango en "DataFrame", y agregamos la columna Rango
rango = rango.to_frame(name='Rango')

#Aplicamos la función lambda para establecer los valores con dos decimales de coma flo
rango = rango.applymap(lambda x: float("%.2f" %x))
rango

```

Out[3]:

	Rango
Equipo	706.0
Resultado	2.0
Horas Funcionamiento	79884.0
Horas del Aceite	20208.0
VISCO	341.7
St	131.0
Al	37.1
Fe	709.0
Si	1721.0
Na	35352.0

In [4]:

```
#Vemos una descripción general del dataset
describe_dataset = dataset.describe().applymap(lambda x: float("%.2f" %x))
describe_dataset
```

Out[4]:

	Equipo	Id	Resultado	Horas Funcionamiento	Horas del Aceite	B	Nit	Oxi
count	18469.00	18469.00	18469.00	18469.00	17707.00	17802.00	3099.00	3430.00
mean	1446.55	235618.10	1.41	37053.59	1000.69	60.80	5.82	13.67
std	156.74	31734.06	0.66	10765.69	1270.91	50.55	1.85	4.15
min	1355.00	155972.00	1.00	106.00	0.00	0.00	0.00	0.80
25%	1381.00	207515.00	1.00	32502.00	308.00	4.00	4.00	9.00
50%	1398.00	238977.00	1.00	40116.00	585.00	85.90	6.00	15.00
75%	1424.00	263301.00	2.00	44704.00	1226.00	102.00	7.00	17.00
max	2061.00	285436.00	3.00	79990.00	20208.00	492.00	17.00	23.00

8 rows × 37 columns

2. Obtendremos la media, mediana y desviación estándar de algunos Features. Agrupando esta información por Componente

In [5]:

```
#Creamos una función que nos da la lista de componentes (Sin repetir)
def componentes_ciclo():
    componentes=[]
    np.array(componentes)
    for x in dataset['Componente']:
        if x not in componentes:
            componentes.append(x)
    return componentes
compo = []
sta_compos = []

#Realizamos una iteración que nos devuelva los estadísticos que deseamos sobre cada cor
for componente in componentes_ciclo():
    #Obtenemos las estadísticas del componente que nos interesa
    sta_compo = dataset[dataset['Componente'] == componente].describe()
    #Localizamos las estadísticas y features de nuestro interes (No seleccionamos Visc
    sta_compo = sta_compo.loc[["mean",'50%','std'],['Horas Funcionamiento','Horas del A
    #Aplicamos solo dos digitos de coma flotante para una mejor visualización
    sta_compo = sta_compo.applymap(lambda x: float("%.2f" %x))


    sta_compos +=[sta_compo]
    compo += [dataset[dataset['Componente'] == componente]]

    #imprimimos
    display("Para el componente '{}':".format(componente),sta_compo,)


```

"Para el componente 'Masa Izquierda':"

	Horas Funcionamiento	Horas del Aceite	Fe	Al	Si	Na
mean	37001.24	443.21	30.23	0.09	5.02	5.26
50%	40085.00	422.00	16.00	0.00	3.00	4.00
std	10735.39	187.12	37.90	0.30	6.95	6.31

"Para el componente 'Sist. Hidráulico':"

	Horas Funcionamiento	Horas del Aceite	Fe	Al	Si	Na
mean	37030.67	1319.71	14.31	2.16	12.68	5.39
50%	40310.00	1108.00	8.00	2.00	11.00	4.00
std	10831.99	1052.49	37.07	1.58	6.01	5.66

"Para el componente 'Convertidor':"

	Horas Funcionamiento	Horas del Aceite	Fe	Al	Si	Na
mean	36998.07	636.53	5.51	0.96	3.64	2.51
50%	40199.00	559.00	4.30	1.00	3.00	2.00
std	10851.48	345.06	8.56	1.04	2.52	1.64

"Para el componente 'Diferencial Trasero':"

	Horas Funcionamiento	Horas del Aceite	Fe	Al	Si	Na
mean	36896.70	1277.05	22.34	0.10	2.91	5.77
50%	40031.00	1127.00	19.00	0.00	2.00	5.10
std	10777.15	852.98	13.81	0.41	2.24	1.74

"Para el componente 'Mando Final TD':"

	Horas Funcionamiento	Horas del Aceite	Fe	Al	Si	Na
mean	36901.28	1314.65	22.79	0.07	2.89	5.79
50%	40007.00	1181.00	19.00	0.00	2.00	5.10
std	10831.75	883.85	13.96	0.25	2.18	1.73

"Para el componente 'Masa Derecha':"

	Horas Funcionamiento	Horas del Aceite	Fe	Al	Si	Na
mean	36976.09	445.36	23.96	0.13	4.68	4.59
50%	40048.00	428.00	11.00	0.00	3.00	4.00
std	10750.99	187.72	38.40	1.16	7.56	1.39

"Para el componente 'Motor':"

	Horas Funcionamiento	Horas del Aceite	Fe	Al	Si	Na
mean	36878.52	296.9	22.52	1.77	6.23	17.47
50%	39896.00	279.0	18.70	2.00	3.00	5.00
std	10836.95	127.1	15.47	1.36	68.48	103.29

"Para el componente 'Sist de Dirección':"

	Horas Funcionamiento	Horas del Aceite	Fe	Al	Si	Na
mean	36935.95	1337.91	1.87	1.00	4.00	3.07
50%	40151.00	1096.00	2.00	1.00	4.00	3.00
std	10876.22	938.27	1.27	0.69	2.25	2.17

"Para el componente 'Mando Final TI':"

	Horas Funcionamiento	Horas del Aceite	Fe	Al	Si	Na
mean	36946.68	1304.60	22.89	0.08	2.9	5.79
50%	40037.50	1161.50	19.00	0.00	2.0	5.00
std	10740.83	864.51	13.98	0.35	2.2	2.13

"Para el componente 'Transmisión':"

	Horas Funcionamiento	Horas del Aceite	Fe	Al	Si	Na
mean	37091.76	638.57	5.68	0.97	3.67	2.58

	Horas Funcionamiento	Horas del Aceite	Fe	Al	Si	Na
50%	40097.00	561.00	4.80	1.00	3.00	2.00
std	10650.21	358.50	8.86	0.87	2.43	1.74

```
"Para el componente 'Radiador':"
```

	Horas Funcionamiento	Horas del Aceite	Fe	Al	Si	Na
mean	39824.98	5637.16	12.31	10.72	39.52	14610.33
50%	43387.00	5151.00	4.40	10.40	25.20	13944.00
std	9687.96	4534.46	16.18	2.32	39.19	9434.97

3.¿Cómo pueden sanearse los valores faltantes?

De los vaores faltantes hay que tener en cuenta si su distribución es aleatoria o no, ya que si no lo son, éstos pueden disminuir la representatividad de la muestra. La forma de sanear los datos faltantes puede ser:

1. Mediante la eliminación de los mismos (Preferentemente si los faltantes son aleatorios).
2. Mediante la sustitución de esos datos(En estas sustituciones se pueden determinar los valores a través de Media, Moda, Regresión, Knn, entre otras)

A continuación analizaremos la presencia de Outliers para cada componente

In [6]:

```
#Realizamos un analisis de los Outliers pertenecientes a cada componente, con los Featu

#Creamos un DataFrame en el que almacenaremos los datos relevantes al respecto
out_index = np.array(['Outliers Inferiores','Outliers Superiores','Outliers Totales','T
elementos = np.array(['Fe', 'Si', 'Na','Al'])
dataliers = pd.DataFrame(index=out_index, columns=elementos)


total_analizados=0
#Realizamos una iteración que nos seleccione el componente deseado(Motor,Sist. Hidrául
for componente in componentes_ciclo():

    #Realizamos otra iteración pero esta vez para obtener los datos de cada elemento(Fe
    for elemento in elementos:

        #Seleccionamos los datos de interes (Componente con el elemento asociado)
        data_compo_elemen = dataset[dataset['Componente'] == componente].loc[:,[elemen


        #Obtenemos sus estadísticos.
        sta_elemento = data_compo_elemen.describe()



        #Primer Cuartil o percentil 25th Q1=25% de los datos.
        q1 = sta_elemento[4:5][elemento].values #Utilizamos el valor del elemento con


        #Tercer Cuartil o percentil 75th Q3=75%
        q3 = sta_elemento[6:7][elemento].values


        #Intercuartil (Q3-Q1)
        iqr = q3-q1


        #Límites inferior y superior para la busqueda de outliers
        lower_limit = float(q1 -(1.5 * iqr)) #Convertimos el elemento array en float pa
        upper_limit = float(q3 +(1.5 * iqr)) #Convertimos el elemento array en float pa


        #Obtenemos la cantidad de outliers inferiores y sus índices
        low_element_cant = data_compo_elemen[data_compo_elemen[elemento] < lower_limit
        low_indice = data_compo_elemen[data_compo_elemen[elemento] < lower_limit].index

        #Almacenamos el porcentaje en el DataFrame
        i = 0
        dataliers.loc[out_index[i],elemento] = str(round((low_element_cant/data_compo_

        #Obtenemos la cantidad de outliers Superiores y sus índices
        high_element_cant = data_compo_elemen[data_compo_elemen[elemento] > upper_limit
        high_indice = data_compo_elemen[data_compo_elemen[elemento] > upper_limit].inde

        #Almacenamos el porcentaje en el DataFrame
        i=1
        dataliers.loc[out_index[i],elemento] = str(round((high_element_cant/data_compo_

        #Almacenamos el porcentaje total outliers
        i = 2
```

```
        dataliers.loc[out_index[i],elemento] = str(round(((low_element_cant+high_eleme
        #Almacenamos el TOTAL de datos
        i = 3
        dataliers.loc[out_index[i],elemento] = data_compo_elemen.shape[0]

        #Almacenamos el porcentaje TOTAL de datos que quedaría "REMOVIENDO" los outlie
        i = 4
        dataliers.loc[out_index[i],elemento] =str(round(((data_compo_elemen.shape[0]-(

        #Armamos una lista de todos con los índices de Outliers (Inferior y Superior)
        total_out_indice = low_indice + high_indice

        #Otra lista con los índices totales
        total_indice = data_compo_elemen.index.tolist()

        #Creamos una lista de índices sin outliers
        total_less_outliers_index = set(total_indice) - (set(total_out_indice)) #Utili
        total_less_outliers_index = list(total_less_outliers_index) #Transformamos en
        total_less_outliers_index = sorted(total_less_outliers_index) #La ordenamos


    #Imprimimos por pantalla el DataFrame correspondiente a cada Componente
    display(componente,dataliers)
```

'Masa Izquierda'

	Fe	Si	Na	Al
Outliers Inferiores	0%	0%	1%	0%
Outliers Superiores	7%	9%	8%	19%
Outliers Totales	7%	9%	9%	19%
Total de Datos	1688	1688	1688	1688
Total de Datos Sin Outliers	93%	91%	91%	81%

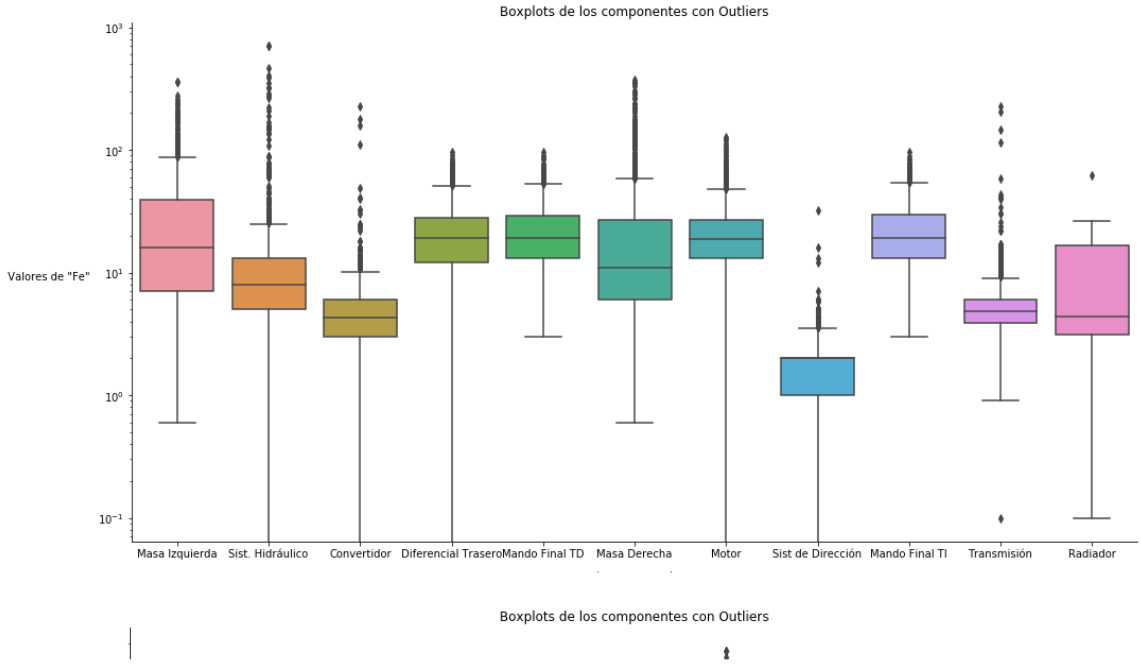
'Sist. Hidráulico'

	Fe	Si	Na	Al
Outliers Inferiores	0%	0%	0%	0%
Outliers Superiores	7%	5%	3%	2%

A continuación visualizaremos cada componente con los features más importantes, y veremos como están distribuidos los outliers

In [7]:

```
def graficar_boxplot():

    for elemento in elementos:
        plt.figure(figsize=(17,9))
        score_box_grap =seaborn.boxplot(x=dataset['Componente'], y=dataset[elemento], (
        score_box_grap.set_title("Boxplots de los componentes con Outliers")
        score_box_grap.set_ylabel(' Valores de "{}"                                    '.forma
        score_box_grap.set_xlabel('.        '+'             .')
        score_box_grap.set_yscale('log') #Utilizamos una escala logaritmica para visua
        seaborn.despine()
        plt.show()

graficar_boxplot()
```



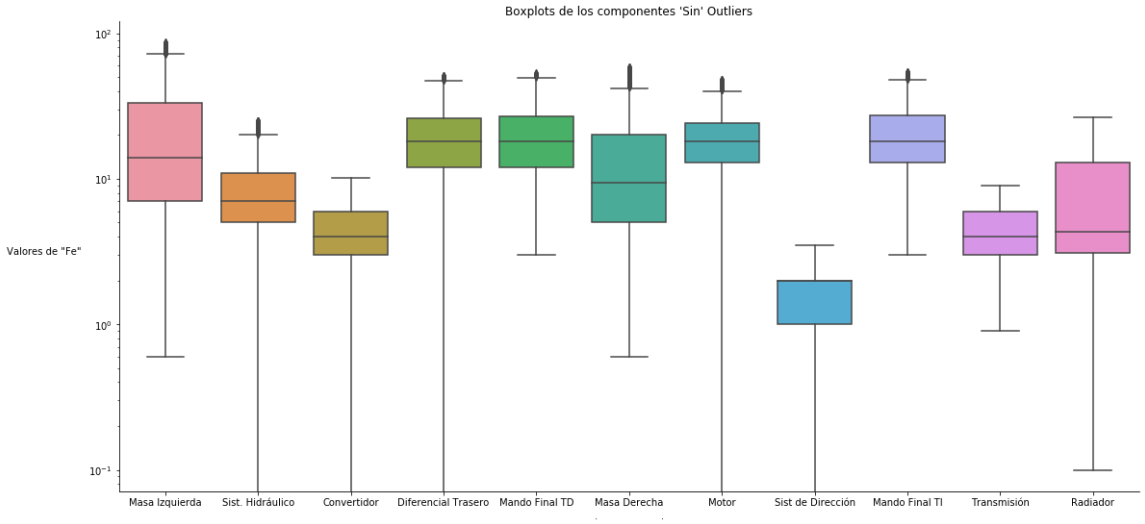
Boxplots de los componentes con Outliers

A contrinuación realizaremos los mismos gráficos, descartando algunos outliers analíticamente mediante el 'Test de Tukey'

In [8]:

```
#Realizamos un analisis de los Outliers pertenecientes a cada componente, con los Featu

#Creamos un DataFrame en el que almacenaremos los datos relevantes al respecto
out_index = np.array(['Outliers Inferiores','Outliers Superiores','Outliers Totales','T
elementos = np.array(['Fe', 'Si', 'Na','Al'])
dataliers = pd.DataFrame(index=out_index, columns=elementos)

dataset_clean = dataset

tota=[]
total_analizados=0
#Realizamos una iteración que nos seleccione el componente deseado(Motor,Sist. Hidrául
for elemento in elementos:
    tota=[]
    #Realizamos otra iteración pero esta vez para obtener los datos de cada elemento(Fe
    for componente in componentes_ciclo():


        #Seleccionamos los datos de interes (Componente con el elemento asociado)
        data_compo_elemen = dataset[dataset['Componente'] == componente].loc[:,[elemen


        #Obtenemos sus estadísticos.
        sta_elemento = data_compo_elemen.describe()



        #Primer Cuartil o percentil 25th Q1=25% de los datos.
        q1 = sta_elemento[4:5][elemento].values #Utilizamos el valor del elemento con


        #Tercer Cuartil o percentil 75th Q3=75%
        q3 = sta_elemento[6:7][elemento].values


        #Intercuartil (Q3-Q1)
        iqr = q3-q1


        #Límites inferior y superior para la busqueda de outliers
        lower_limit = float(q1 -(1.5 * iqr)) #Convertimos el elemento array en float pa
        upper_limit = float(q3 +(1.5 * iqr)) #Convertimos el elemento array en float pa


        #Obtenemos la cantidad de outliers inferiores y sus índices
        low_element_cant = data_compo_elemen[data_compo_elemen[elemento] < lower_limit
        low_indice = data_compo_elemen[data_compo_elemen[elemento] < lower_limit].index



        #Obtenemos la cantidad de outliers Superiores y sus índices
        high_element_cant = data_compo_elemen[data_compo_elemen[elemento] > upper_limi
        high_indice = data_compo_elemen[data_compo_elemen[elemento] > upper_limit].inde



        #Armamos una lista de todos con los índices de Outliers (Inferior y Superior)
        total_out_indice = low_indice + high_indice

```

```
60
61          #Otra lista con los índices totales
62          total_indice = data_compo_elemen.index.tolist()
63
64
65
66          #Creamos una lista de índices sin outliers
67          total_less_outliers_index = set(total_indice) - (set(total_out_indice)) #Utili
68          total_less_outliers_index = list(total_less_outliers_index) #Transformamos en
69          tota += total_less_outliers_index
70
71          #print(componente, len(tota))
72          #total_less_outliers_index = sorted(total_less_outliers_index) #La ordenamos
73
74          #Datos sin Outliers
75          #total_less_outliers_data = dataset_clean.reindex(total_less_outliers_index)
76
77      plt.figure(figsize=(19,9))
78      score_box_grap1 = seaborn.boxplot(x=dataset['Componente'], y=dataset.iloc[tota][el
79      score_box_grap1.set_title("Boxplots de los componentes 'Sin' Outliers")
80      score_box_grap1.set_ylabel(' Valores de "{}"                                    '.format(e
81      score_box_grap1.set_xlabel('.          '+'              .')
82      score_box_grap1.set_yscale('log')#Utilizamos una escala logaritmica para visualiza
83
84      seaborn.despine()
85      plt.show()
86
87
```



2. Graficar

1. Realizaremos el gráfico de motor de 3 equipos al azar y veremos como evolucionan algunos features a lo largo del tiempo

In [9]:

```
# Fijamos una semilla para obtener siempre el mismo resultado
random.seed(47)
#Seleccionamos el índice de los 3 Equipos al azar (tenemos que convertir el "diccionar
equipos = random.sample(list(dataset[dataset['Componente']=='Motor']['Equipo'].values)
equipos
```

Out[9]:

```
[1402, 1371, 1932]
```

In [10]:

```
#Seteamos los datos del equipo (1) a utilizar
data_mot = dataset[(dataset['Componente']=='Motor') & (dataset['Equipo']==1402)]

#Añadimos los features VISCOSIDAD Y Resultado
elementos = np.concatenate((elementos,['VISCO'],['Resultado']))
```

In [11]:

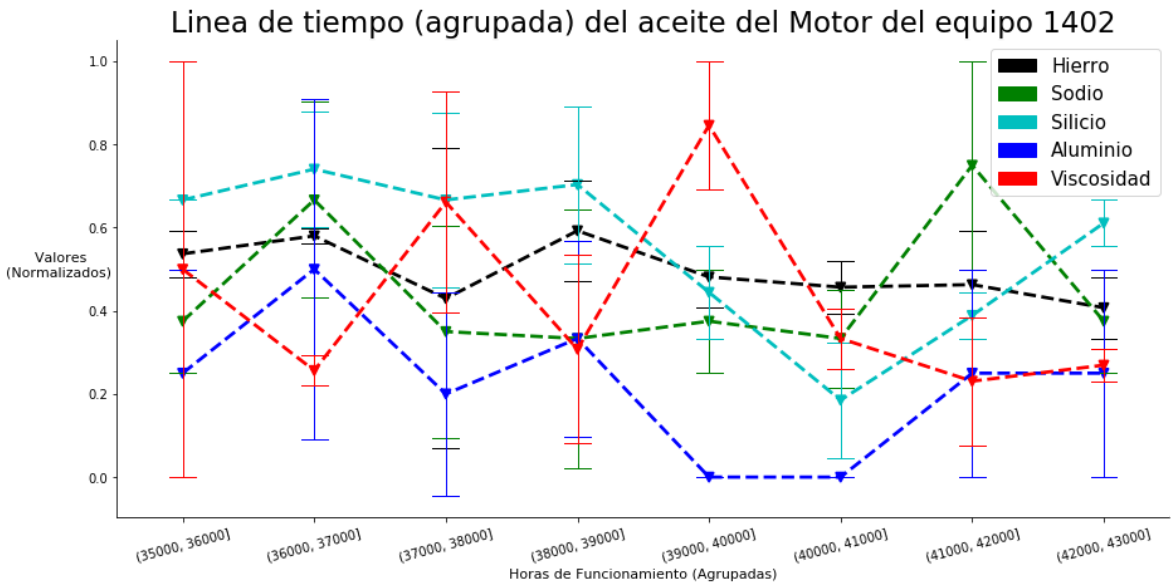
```
#Utilizamos una función de agrupación de las "Horas Funcionamiento" para una mejor visu
def to_categorical(column, bin_size=5, min_cut=15, max_cut=50):
    if min_cut is None:
        min_cut = int(round(column.min())) - 1
    value_max = int(np.ceil(column.max()))
    max_cut = min(max_cut, value_max)
    intervals = [(x, x+bin_size) for x in range(min_cut, max_cut, bin_size)]
    if max_cut != value_max:
        intervals.append((max_cut, value_max))
    return pd.cut(column, pd.IntervalIndex.from_tuples(intervals))


#seaborn.countplot(to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,35000,
#plt.xticks(rotation=45)

intervalo_horas = to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,35000,4
```

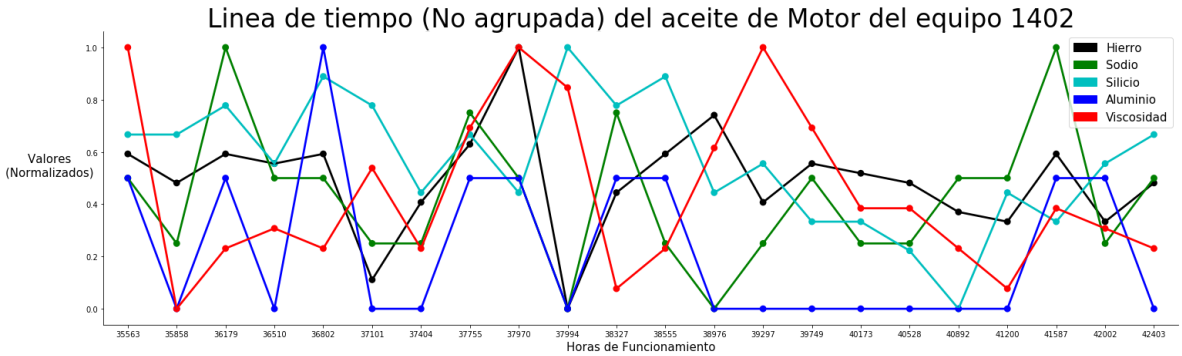
In [12]:

```
#Normalizamos los datos para poder mostrar los features en un mismo gráfico
data_motor = data_mot.copy()
#Creamos una función z-minmax para normalizar los features entre [0,1]
def z(data, maxi, mini):#datos,maximo del feature, minimo del feature
    return (data-mini)/(maxi-mini)


#Obtenemos los datos necesarios de cada feature para realizar la normalización.
for elemento in elementos:
    maxi = data_motor[elemento].dropna().max() #Seleccionamos el máximo del feature
    mini = data_motor[elemento].dropna().min() #Seleccionamos el mínimo del feature

    data_motor[elemento] = z(data_motor[elemento],maxi,mini) #Asignamos al dataset los
```

In [13]:

```
#Realizamos la gráfica del Equipo con los valores normalizados
plt.figure(figsize=(15,7))
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Fe'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Hierro
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Na'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Sodio'
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Si'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Silici
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Al'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Alumin
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['VISCO'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Viscos

plt.xticks(rotation=15)
#Seteamos el label de los componentes
label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

#Ponemos los labels en la imagen, y seteamos el tamaño de las etiquetas
plt.legend(handles=[label_Fe,label_Na,label_Si,label_Al,label_VI],prop={'size': 15})
plt.title('Linea de tiempo (agrupada) del aceite del Motor del equipo {}'.format(equip
plt.xlabel('Horas de Funcionamiento (Agrupadas)',size=11)
plt.ylabel('Valores          \n(Normalizados)           ',size=11,rotation=360)
seaborn.despine()
plt.show()

plt.figure(figsize=(25,7))

seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Fe'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Na'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Si'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Al'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['VISCO']

plt.title('Linea de tiempo (No agrupada) del aceite de Motor del equipo {}'.format(equ
plt.xlabel('Horas de Funcionamiento',size=15)
plt.ylabel('Valores                \n(Normalizados)                ',size=15,rotat



label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

plt.legend(handles=[label_Fe,label_Na,label_Si,label_Al,label_VI],prop={'size': 15})
seaborn.despine()
plt.show()

```





In [14]:

```
# Fijamos una semilla para obtener siempre el mismo resultado
random.seed(47)
#Seleccionamos el índice de los 3 Equipos al azar (tenemos que convertir el "diccionar
equipos = random.sample(list(dataset[dataset['Componente']=='Motor']['Equipo'].values)
equipos
```

Out[14]:

```
[1402, 1371, 1932]
```

In [15]:

```
#Seteamos los datos del equipo (2) a utilizar
data_mot = dataset[(dataset['Componente']=='Motor') & (dataset['Equipo']==1371)]

```

In [16]:

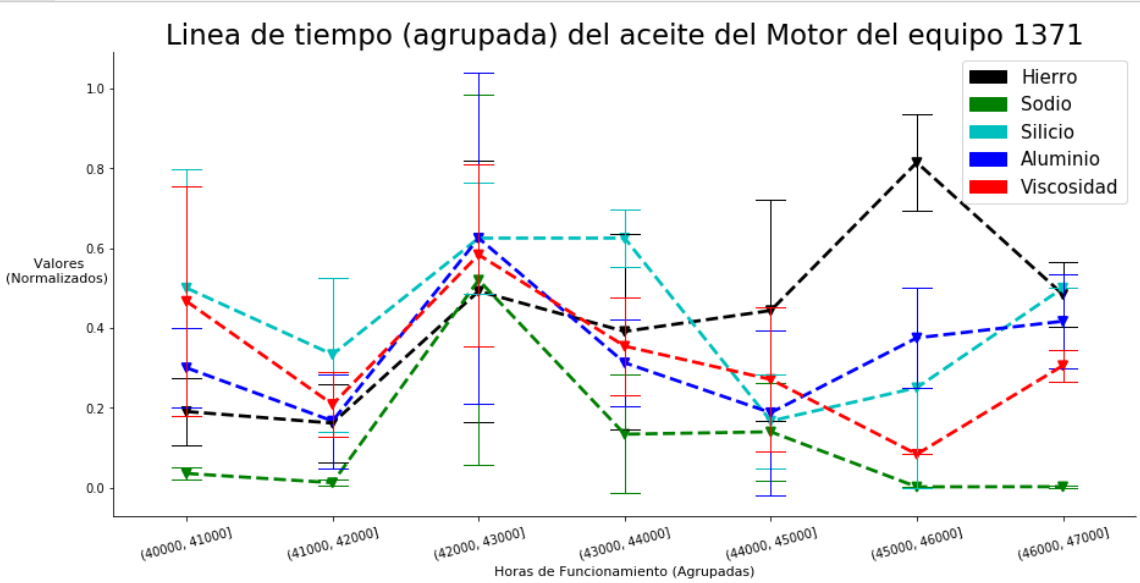
```
#Utilizamos una función de agrupación de las "Horas Funcionamiento" para una mejor visu
def to_categorical(column, bin_size=5, min_cut=15, max_cut=50):
    if min_cut is None:
        min_cut = int(round(column.min())) - 1
    value_max = int(np.ceil(column.max()))
    max_cut = min(max_cut, value_max)
    intervals = [(x, x+bin_size) for x in range(min_cut, max_cut, bin_size)]
    if max_cut != value_max:
        intervals.append((max_cut, value_max))
    return pd.cut(column, pd.IntervalIndex.from_tuples(intervals))


#seaborn.countplot(to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,40000,
#plt.xticks(rotation=45)

intervalo_horas = to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,40000,
```

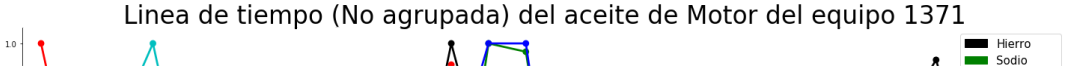
In [17]:

```
#Normalizamos los datos para poder mostrar los 4 features en un mismo gráfico
data_motor = data_mot.copy()
#Creamos una función z-minmax para normalizar los features entre [0,1]
def z(data, maxi, mini):#datos,maximo del feature, minimo del feature
    return (data-mini)/(maxi-mini)

#Obtenemos los datos necesarios de cada feature para realizar la normalización.
for elemento in elementos:
    maxi = data_motor[elemento].dropna().max() #Seleccionamos el máximo del feature
    mini = data_motor[elemento].dropna().min() #Seleccionamos el mínimo del feature

    data_motor[elemento] = z(data_motor[elemento],maxi,mini) #Asignamos al dataset los
```

In [18]:

```
#Realizamos la gráfica del Equipo con los valores normalizados
plt.figure(figsize=(15,7))
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Fe'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Hierro
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Na'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Sodio'
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Si'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Silici
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Al'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Alumin
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['VISCO'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Viscos
plt.xticks(rotation=15)

#Seteamos el label de los componentes
label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

#Ponemos los labels en la imagen, y seteamos el tamaño de las etiquetas
plt.legend(handles=[label_Fe,label_Na,label_Si,label_Al,label_VI],prop={'size': 15})
plt.title('Linea de tiempo (agrupada) del aceite del Motor del equipo {}'.format(equip
plt.xlabel('Horas de Funcionamiento (Agrupadas)',size=11)
plt.ylabel('Valores          \n(Normalizados)           ',size=11,rotation=360)
seaborn.despine()
plt.show()

plt.figure(figsize=(25,7))

seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Fe'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Na'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Si'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Al'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['VISCO']

plt.title('Linea de tiempo (No agrupada) del aceite de Motor del equipo {}'.format(equ
plt.xlabel('Horas de Funcionamiento',size=15)
plt.ylabel('Valores                \n(Normalizados)                ',size=15,rotat



label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

plt.legend(handles=[label_Fe,label_Na,label_Si,label_Al,label_VI],prop={'size': 15})
seaborn.despine()
plt.show()

```





In [19]:

```
# Fijamos una semilla para obtener siempre el mismo resultado
random.seed(47)
#Seleccionamos el índice de los 3 Equipos al azar (tenemos que convertir el "diccionar
equipos = random.sample(list(dataset[dataset['Componente']=='Motor']['Equipo'].values)
equipos
```

Out[19]:

```
[1402, 1371, 1932]
```

In [20]:

```
#Seteamos los datos del equipo (3) a utilizar
data_mot = dataset[(dataset['Componente']=='Motor') & (dataset['Equipo']==1932)]
```

In [21]:

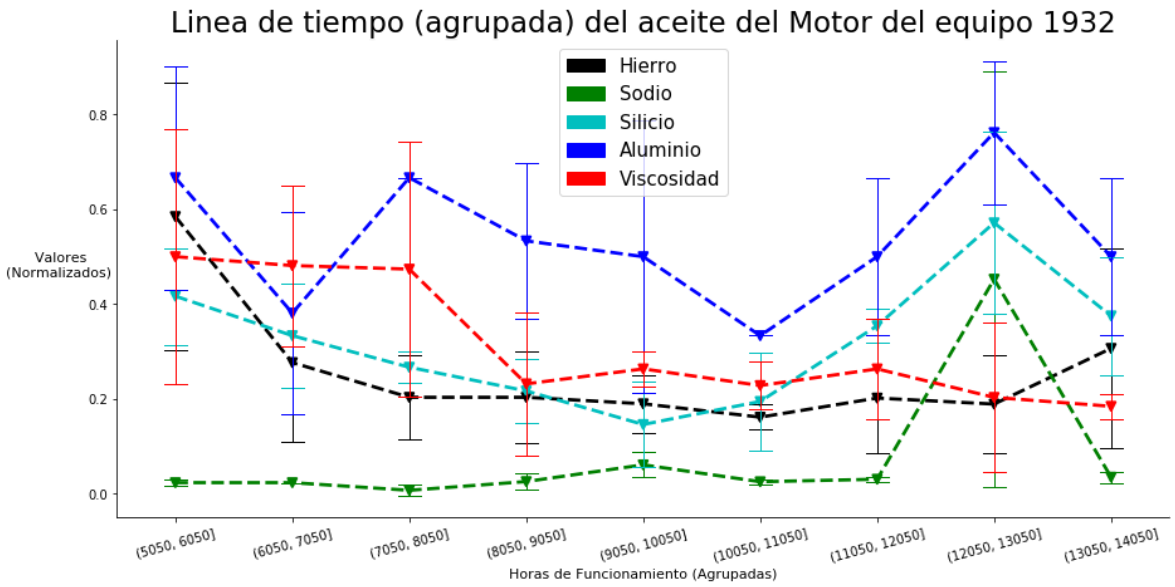
```
#Utilizamos una función de agrupación de las "Horas Funcionamiento" para una mejor visu
def to_categorical(column, bin_size=5, min_cut=15, max_cut=50):
    if min_cut is None:
        min_cut = int(round(column.min())) - 1
    value_max = int(np.ceil(column.max()))
    max_cut = min(max_cut, value_max)
    intervals = [(x, x+bin_size) for x in range(min_cut, max_cut, bin_size)]
    if max_cut != value_max:
        intervals.append((max_cut, value_max))
    return pd.cut(column, pd.IntervalIndex.from_tuples(intervals))


#seaborn.countplot(to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,5050,1
#plt.xticks(rotation=45)

intervalo_horas = to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,5050,1
```

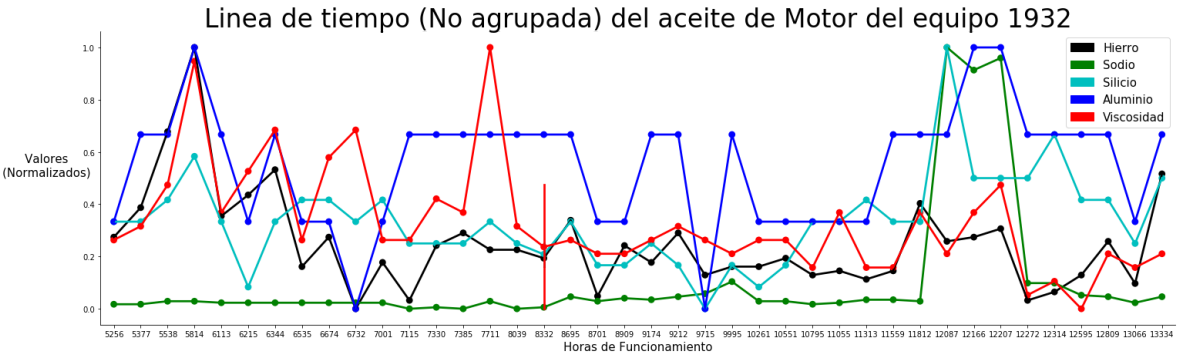
In [22]:

```
#Normalizamos los datos para poder mostrar los 4 features en un mismo gráfico
data_motor = data_mot.copy()
#Creamos una función z-minmax para normalizar los features entre [0,1]
def z(data, maxi, mini):#datos,maximo del feature, minimo del feature
    return (data-mini)/(maxi-mini)


#Obtenemos los datos necesarios de cada feature para realizar la normalización.
for elemento in elementos:
    maxi = data_motor[elemento].dropna().max() #Seleccionamos el máximo del feature
    mini = data_motor[elemento].dropna().min() #Seleccionamos el mínimo del feature

    data_motor[elemento] = z(data_motor[elemento],maxi,mini) #Asignamos al dataset los
```

In [23]:

```
#Realizamos la gráfica del Equipo con los valores normalizados
plt.figure(figsize=(15,7))
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Fe'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Hierro
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Na'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Sodio'
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Si'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Silici
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Al'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Alumin
plt.xticks(rotation=15)
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['VISCO'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Viscos
plt.xticks(rotation=15)
#Seteamos el label de los componentes
label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

#Ponemos los labels en la imagen, y seteamos el tamaño de las etiquetas
plt.legend(handles=[label_Fe,label_Na,label_Si,label_Al,label_VI],prop={'size': 15})
plt.title('Linea de tiempo (agrupada) del aceite del Motor del equipo {}'.format(equip
plt.xlabel('Horas de Funcionamiento (Agrupadas)',size=11)
plt.ylabel('Valores          \n(Normalizados)           ',size=11,rotation=360)
seaborn.despine()
plt.show()

plt.figure(figsize=(25,7))

seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Fe'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Na'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Si'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Al'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['VISCO']

plt.title('Linea de tiempo (No agrupada) del aceite de Motor del equipo {}'.format(equ
plt.xlabel('Horas de Funcionamiento',size=15)
plt.ylabel('Valores                \n(Normalizados)                ',size=15,rotat



label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

plt.legend(handles=[label_Fe,label_Na,label_Si,label_Al,label_VI],prop={'size': 15})
seaborn.despine()
plt.show()

```





2. Ahora realizaremos el gráfico de un equipo y 3 de sus componentes componentes

In [24]:

```
# Fijamos una semilla para obtener siempre el mismo resultado
random.seed(47)
#Seleccionamos el índice de los 3 Equipos al azar (tenemos que convertir el "diccionar
componentes = random.sample(list(dataset[dataset['Equipo']==1402]['Componente'].values
componentes
```

Out[24]:

```
['Transmisión', 'Masa Izquierda', 'Masa Derecha']
```

In [25]:

```
#Seteamos los datos del componente a utilizar (Transmisión)
data_mot = dataset[(dataset['Componente']==componentes[0]) & (dataset['Equipo']==1402)
```

In [26]:

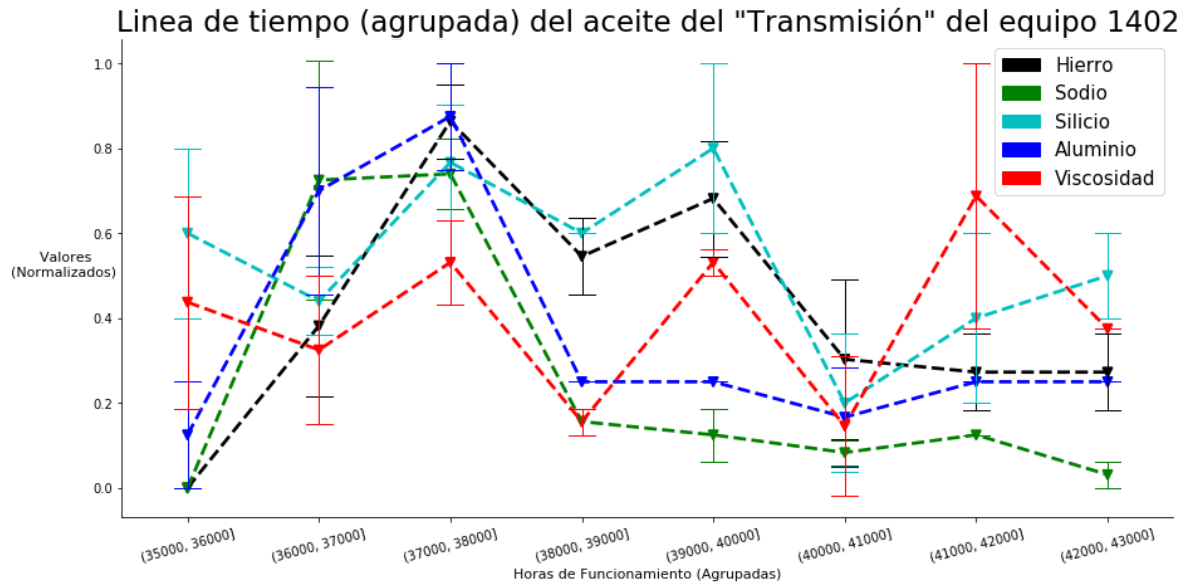
```
#Utilizamos una función de agrupación de las "Horas Funcionamiento" para una mejor visu
def to_categorical(column, bin_size=5, min_cut=15, max_cut=50):
    if min_cut is None:
        min_cut = int(round(column.min())) - 1
    value_max = int(np.ceil(column.max()))
    max_cut = min(max_cut, value_max)
    intervals = [(x, x+bin_size) for x in range(min_cut, max_cut, bin_size)]
    if max_cut != value_max:
        intervals.append((max_cut, value_max))
    return pd.cut(column, pd.IntervalIndex.from_tuples(intervals))


#seaborn.countplot(to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,35000,
#plt.xticks(rotation=45)

intervalo_horas = to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,35000,4
```

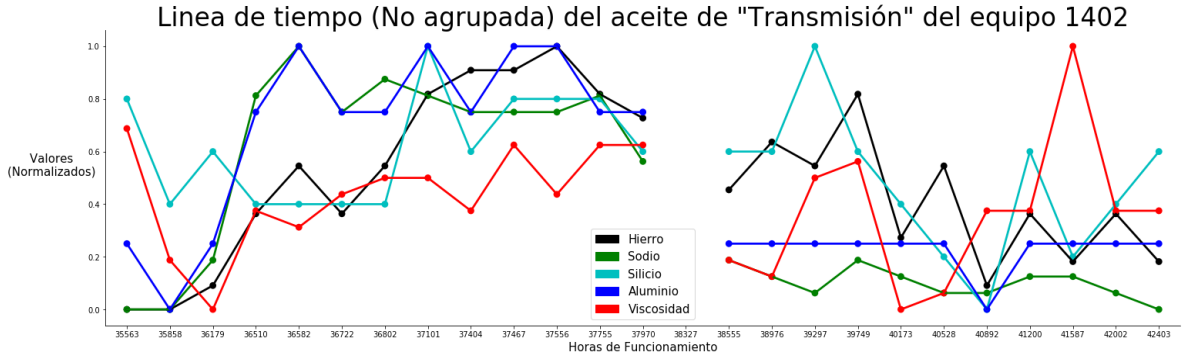
In [27]:

```
#Normalizamos los datos para poder mostrar los 4 features en un mismo gráfico
data_motor = data_mot.copy()
#Creamos una función z-minmax para normalizar los features entre [0,1]
def z(data, maxi, mini):#datos,maximo del feature, minimo del feature
    return (data-mini)/(maxi-mini)

#Obtenemos los datos necesarios de cada feature para realizar la normalización.
for elemento in elementos:
    maxi = data_motor[elemento].dropna().max() #Seleccionamos el máximo del feature
    mini = data_motor[elemento].dropna().min() #Seleccionamos el mínimo del feature

    data_motor[elemento] = z(data_motor[elemento],maxi,mini) #Asignamos al dataset los
```

In [28]:

```
#Realizamos la gráfica del Equipo con los valores normalizados
plt.figure(figsize=(15,7))
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Fe'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Hierro
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Na'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Sodio'
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Si'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Silici
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Al'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Alumin
plt.xticks(rotation=15)
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['VISCO'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Viscos
plt.xticks(rotation=15)
#Seteamos el label de los componentes
label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

#Ponemos los labels en la imagen, y seteamos el tamaño de las etiquetas
plt.legend(handles=[label_Fe,label_Na,label_Si,label_Al,label_VI],prop={'size': 15})
plt.title('Linea de tiempo (agrupada) del aceite del "{}" del equipo 1402'.format(compo
plt.xlabel('Horas de Funcionamiento (Agrupadas)',size=11)
plt.ylabel('Valores          \n(Normalizados)           ',size=11,rotation=360)
seaborn.despine()
plt.show()

plt.figure(figsize=(25,7))

seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Fe'].dro
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Na'].dro
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Si'].dro
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Al'].dro
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['VISCO']

plt.title('Linea de tiempo (No agrupada) del aceite de "{}" del equipo 1402'.format(co
plt.xlabel('Horas de Funcionamiento',size=15)
plt.ylabel('Valores                \n(Normalizados)                ',size=15,rotat



label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')


plt.legend(handles=[label_Fe,label_Na,label_Si,label_Al,label_VI],prop={'size': 15})
seaborn.despine()
plt.show()

```





In [29]:

```
# Fijamos una semilla para obtener siempre el mismo resultado
random.seed(47)
#Seleccionamos el índice de los 3 Equipos al azar (tenemos que convertir el "diccionar
componentes = random.sample(list(dataset[dataset['Equipo']==1402]['Componente'].values
componentes
```

Out[29]:

```
['Transmisión', 'Masa Izquierda', 'Masa Derecha']
```

In [30]:

```
#Seteamos los datos del componente a utilizar (Masa Izquierda)
data_mot = dataset[(dataset['Componente']==componentes[1]) & (dataset['Equipo']==1402)
```

In [31]:

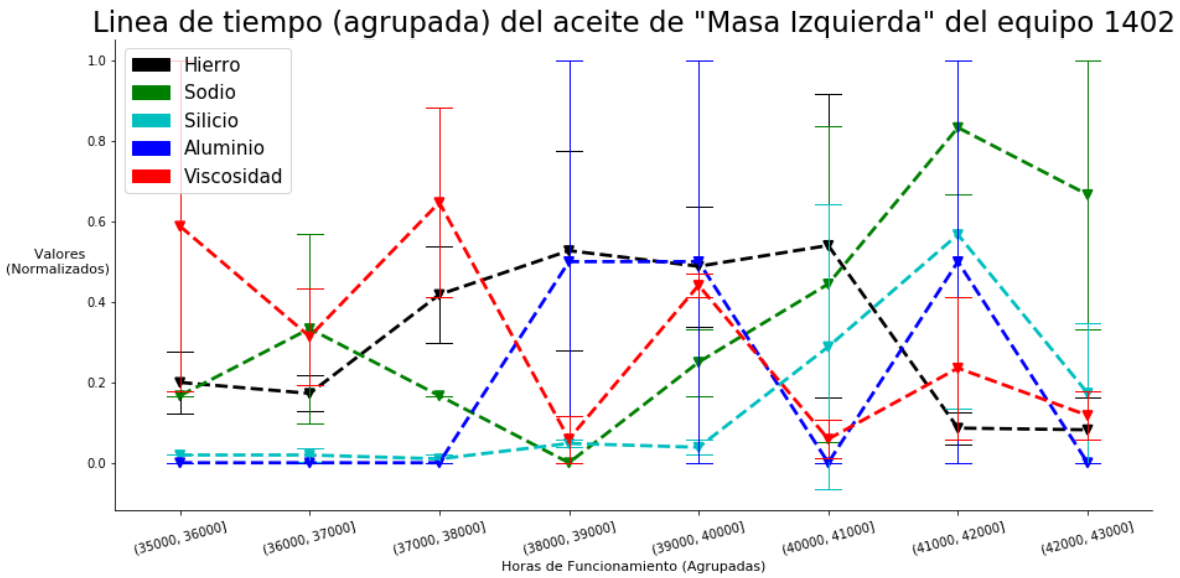
```
#Utilizamos una función de agrupación de las "Horas Funcionamiento" para una mejor visu
def to_categorical(column, bin_size=5, min_cut=15, max_cut=50):
    if min_cut is None:
        min_cut = int(round(column.min())) - 1
    value_max = int(np.ceil(column.max()))
    max_cut = min(max_cut, value_max)
    intervals = [(x, x+bin_size) for x in range(min_cut, max_cut, bin_size)]
    if max_cut != value_max:
        intervals.append((max_cut, value_max))
    return pd.cut(column, pd.IntervalIndex.from_tuples(intervals))


#seaborn.countplot(to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,35000,
#plt.xticks(rotation=45)

intervalo_horas = to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,35000,4
```

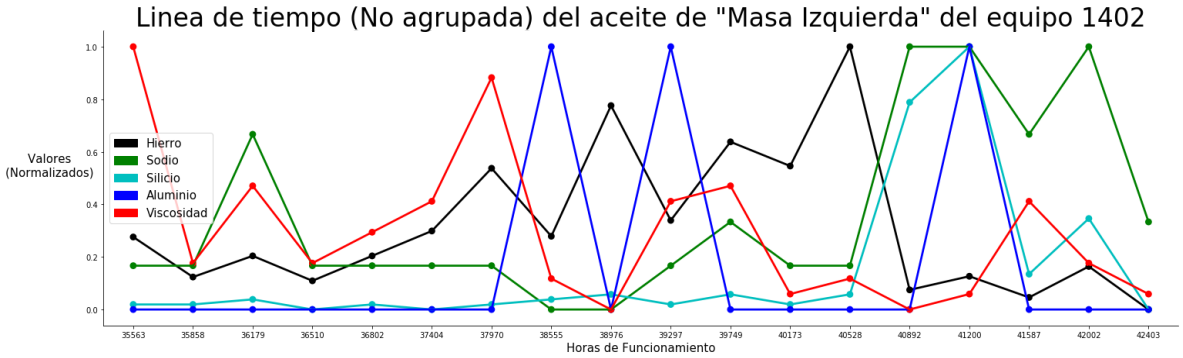
In [32]:

```
#Normalizamos los datos para poder mostrar los 4 features en un mismo gráfico
data_motor = data_mot.copy()
#Creamos una función z-minmax para normalizar los features entre [0,1]
def z(data, maxi, mini):#datos,maximo del feature, minimo del feature
    return (data-mini)/(maxi-mini)

#Obtenemos los datos necesarios de cada feature para realizar la normalización.
for elemento in elementos:
    maxi = data_motor[elemento].dropna().max() #Seleccionamos el máximo del feature
    mini = data_motor[elemento].dropna().min() #Seleccionamos el mínimo del feature

    data_motor[elemento] = z(data_motor[elemento],maxi,mini) #Asignamos al dataset los
```

In [33]:

```
#Realizamos la gráfica del Equipo con los valores normalizados
plt.figure(figsize=(15,7))
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Fe'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Hierro
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Na'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Sodio'
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Si'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Silici
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Al'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Alumin
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['VISCO'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Alumin

plt.xticks(rotation=15)
#Seteamos el label de los componentes
label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

#Ponemos los labels en la imagen, y seteamos el tamaño de las etiquetas
plt.legend(handles=[label_Fe,label_Na,label_Si,label_Al,label_VI],prop={'size': 15})
plt.title('Linea de tiempo (agrupada) del aceite de "{}" del equipo 1402'.format(compo
plt.xlabel('Horas de Funcionamiento (Agrupadas)',size=11)
plt.ylabel('Valores          \n(Normalizados)           ',size=11,rotation=360)
seaborn.despine()
plt.show()

plt.figure(figsize=(25,7))

seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Fe'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Na'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Si'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Al'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['VISCO']


plt.title('Linea de tiempo (No agrupada) del aceite de "{}" del equipo 1402'.format(co
plt.xlabel('Horas de Funcionamiento',size=15)
plt.ylabel('Valores                \n(Normalizados)                ',size=15,rotat



label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

plt.legend(handles=[label_Fe,label_Na,label_Si,label_Al,label_VI],prop={'size': 15})
seaborn.despine()
plt.show()

```





In [34]:

```
# Fijamos una semilla para obtener siempre el mismo resultado
random.seed(47)
#Seleccionamos el índice de los 3 Equipos al azar (tenemos que convertir el "diccionar
componentes = random.sample(list(dataset[dataset['Equipo']==1402]['Componente'].values
componentes
```

Out[34]:

```
['Transmisión', 'Masa Izquierda', 'Masa Derecha']
```

In [35]:

```
#Seteamos los datos del componente a utilizar (Masa Derecha)
data_mot = dataset[(dataset['Componente']==componentes[2]) & (dataset['Equipo']==1402)
```

In [36]:

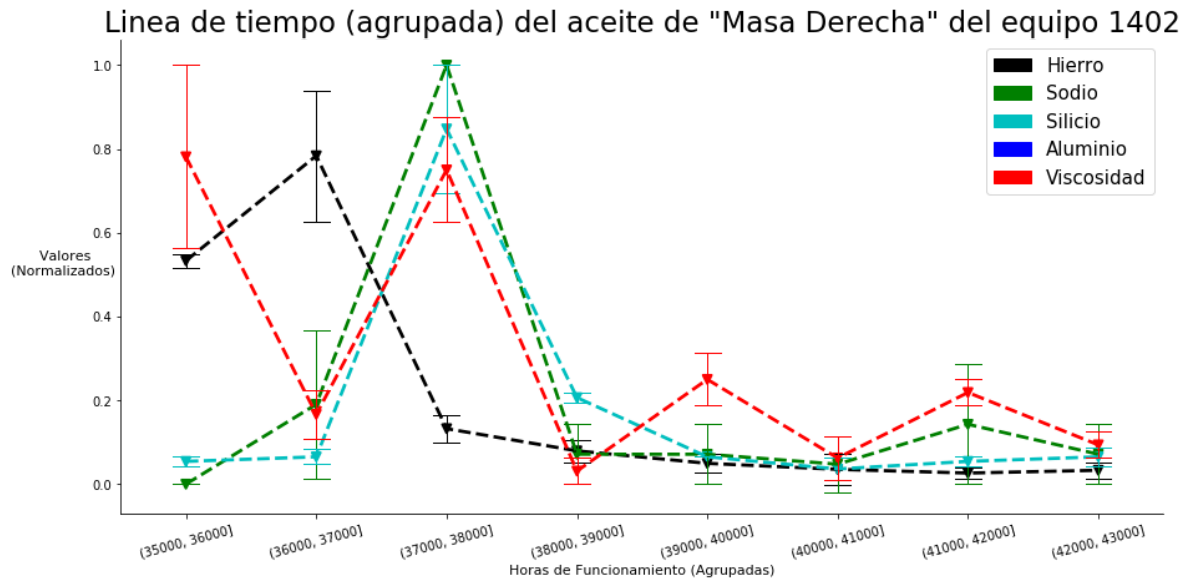
```
#Utilizamos una función de agrupación de las "Horas Funcionamiento" para una mejor visu
def to_categorical(column, bin_size=5, min_cut=15, max_cut=50):
    if min_cut is None:
        min_cut = int(round(column.min())) - 1
    value_max = int(np.ceil(column.max()))
    max_cut = min(max_cut, value_max)
    intervals = [(x, x+bin_size) for x in range(min_cut, max_cut, bin_size)]
    if max_cut != value_max:
        intervals.append((max_cut, value_max))
    return pd.cut(column, pd.IntervalIndex.from_tuples(intervals))


#seaborn.countplot(to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,35000,
#plt.xticks(rotation=45)

intervalo_horas = to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,35000,4
```

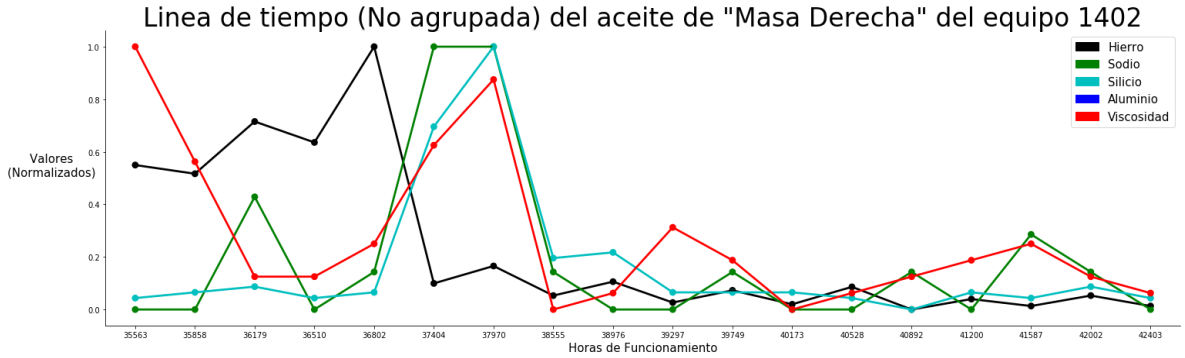
In [37]:

```
#Normalizamos los datos para poder mostrar los 5 features en un mismo gráfico
data_motor = data_mot.copy()
#Creamos una función z-minmax para normalizar los features entre [0,1]
def z(data, maxi, mini):#datos,maximo del feature, minimo del feature
    return (data-mini)/(maxi-mini)


#Obtenemos los datos necesarios de cada feature para realizar la normalización.
for elemento in elementos:
    maxi = data_motor[elemento].dropna().max() #Seleccionamos el máximo del feature
    mini = data_motor[elemento].dropna().min() #Seleccionamos el mínimo del feature

    data_motor[elemento] = z(data_motor[elemento],maxi,mini) #Asignamos al dataset los
```

In [38]:

```
#Realizamos la gráfica del Equipo con los valores normalizados
plt.figure(figsize=(15,7))
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Fe'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Hierro
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Na'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Sodio'
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Si'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Silici
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Al'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Alumin
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['VISCO'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Alumin

plt.xticks(rotation=15)
#Seteamos el label de los componentes
label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')


#Ponemos los labels en la imagen, y seteamos el tamaño de las etiquetas
plt.legend(handles=[label_Fe,label_Na,label_Si,label_Al,label_VI],prop={'size': 15})
plt.title('Linea de tiempo (agrupada) del aceite de "{}" del equipo 1402'.format(compo
plt.xlabel('Horas de Funcionamiento (Agrupadas)',size=11)
plt.ylabel('Valores          \n(Normalizados)           ',size=11,rotation=360)
seaborn.despine()
plt.show()

plt.figure(figsize=(25,7))

seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Fe'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Na'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Si'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Al'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['VISCO']

plt.title('Linea de tiempo (No agrupada) del aceite de "{}" del equipo 1402'.format(co
plt.xlabel('Horas de Funcionamiento',size=15)
plt.ylabel('Valores                  \n(Normalizados)                  ',size=15,rotat



label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

plt.legend(handles=[label_Fe,label_Na,label_Si,label_Al,label_VI],prop={'size': 15})
seaborn.despine()
plt.show()

```





3. Correlaciones

1. A continuación analizaremos la Viscosidad vs el Hierro (Fe), a medida que aumentan las Horas del Aceite

1er Ejemplo: Motor del equipo 1371

In [39]:

```
# Fijamos una semilla para obtener siempre el mismo resultado
random.seed(47)
#Seleccionamos el índice de los 3 Equipos al azar (tenemos que convertir el "diccionar
equipos = random.sample(list(dataset[dataset['Componente']=='Motor']['Equipo'].values)
equipos
```

Out[39]:

```
[1402, 1371, 1932]
```

In [40]:

```
#Seteamos los datos del equipo (2) a utilizar
data_mot = dataset[(dataset['Componente']=='Motor') & (dataset['Equipo']==1371)]

```

In [41]:

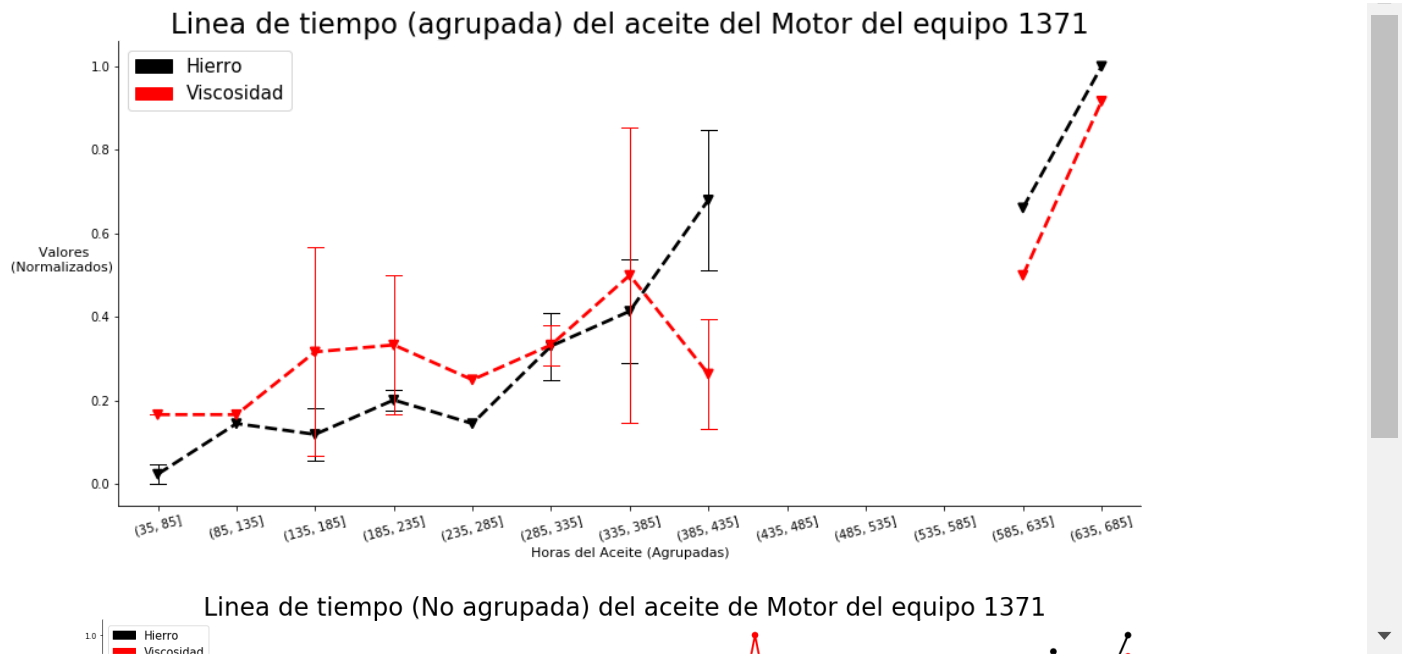
```
#Utilizamos una función de agrupación de las "Horas Funcionamiento" para una mejor visu
def to_categorical(column, bin_size=5, min_cut=15, max_cut=50):
    if min_cut is None:
        min_cut = int(round(column.min())) - 1
    value_max = int(np.ceil(column.max()))
    max_cut = min(max_cut, value_max)
    intervals = [(x, x+bin_size) for x in range(min_cut, max_cut, bin_size)]
    if max_cut != value_max:
        intervals.append((max_cut, value_max))
    return pd.cut(column, pd.IntervalIndex.from_tuples(intervals))


#seaborn.countplot(to_categorical(data_mot['Horas del Aceite'].dropna(),50,35,750), co
#plt.xticks(rotation=45)

intervalo_horas = to_categorical(data_mot['Horas del Aceite'].dropna(),50,35,750)
```

In [42]:

```
#Normalizamos los datos para poder mostrar los 4 features en un mismo gráfico
data_motor = data_mot.copy()
#Creamos una función z-minmax para normalizar los features entre [0,1]
def z(data, maxi, mini):#datos,maximo del feature, minimo del feature
    return (data-mini)/(maxi-mini)

#Obtenemos los datos necesarios de cada feature para realizar la normalización.
for elemento in elementos:
    maxi = data_motor[elemento].dropna().max() #Seleccionamos el máximo del feature
    mini = data_motor[elemento].dropna().min() #Seleccionamos el mínimo del feature

    data_motor[elemento] = z(data_motor[elemento],maxi,mini) #Asignamos al dataset los
```

In [43]:

```
#Realizamos la gráfica del Equipo con los valores normalizados
plt.figure(figsize=(15,7))
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Fe'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Hierro
#seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Na'],
#                   dodge=True, ci='sd',
#                   markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Sodio
#seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Si'],
#                   dodge=True, ci='sd',
#                   markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Silic
#seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Al'],
#                   dodge=True, ci='sd',
#                   markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Alumi
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['VISCO'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Viscos
plt.xticks(rotation=15)

#Seteamos el label de los componentes
label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

#Ponemos los labels en la imagen, y seteamos el tamaño de las etiquetas
plt.legend(handles=[label_Fe,label_VI],prop={'size': 15})
plt.title('Linea de tiempo (agrupada) del aceite del Motor del equipo {}'.format(equip
plt.xlabel('Horas del Aceite (Agrupadas)',size=11)
plt.ylabel('Valores          \n(Normalizados)           ',size=11,rotation=360)
seaborn.despine()
plt.show()

plt.figure(figsize=(25,7))

seaborn.pointplot(x=data_motor['Horas del Aceite'].dropna(), y=data_motor['Fe'].dropna
#seaborn.pointplot(x=data_motor['Horas del Aceite'].dropna(), y=data_motor['Na'].dropn
#seaborn.pointplot(x=data_motor['Horas del Aceite'].dropna(), y=data_motor['Si'].dropn
#seaborn.pointplot(x=data_motor['Horas del Aceite'].dropna(), y=data_motor['Al'].dropn
seaborn.pointplot(x=data_motor['Horas del Aceite'].dropna(), y=data_motor['VISCO'].dro

plt.title('Linea de tiempo (No agrupada) del aceite de Motor del equipo {}'.format(equ
plt.xlabel('Horas del Aceite',size=15)
plt.ylabel('Valores                  \n(Normalizados)                  ',size=15,rotat



label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

plt.legend(handles=[label_Fe,label_VI],prop={'size': 15})
seaborn.despine()
plt.show()

```



En el gráfico anterior visualmente no parece haber una correlación directa entre el Hierro y la viscosidad a medida que aumentan las horas del aceite

Analizamos la posibilidad de correlación analiticamente mediante el **Coeficiente de Spearman**

In [44]:

```
1 data_motor[['Fe','VISCO']].corr(method='spearman')
```

Out[44]:

	Fe	VISCO
Fe	1.000000	0.232281
VISCO	0.232281	1.000000

Observamos que el coeficiente es muy bajo, por lo que podríamos descartar la correlación entre los features

2do Ejemplo: Motor del equipo 1402

In [45]:

```
1 # Fijamos una semilla para obtener siempre el mismo resultado
2 random.seed(47)
3 #Seleccionamos el índice de los 3 Equipos al azar (tenemos que convertir el "diccionar
4 equipos = random.sample(list(dataset[dataset['Componente']=='Motor']['Equipo'].values)
5 equipos
```

Out[45]:

```
[1402, 1371, 1932]
```

In [46]:

```
#Seteamos los datos del equipo (3) a utilizar
data_mot = dataset[(dataset['Componente']=='Motor') & (dataset['Equipo']==1402)]

```

In [47]:

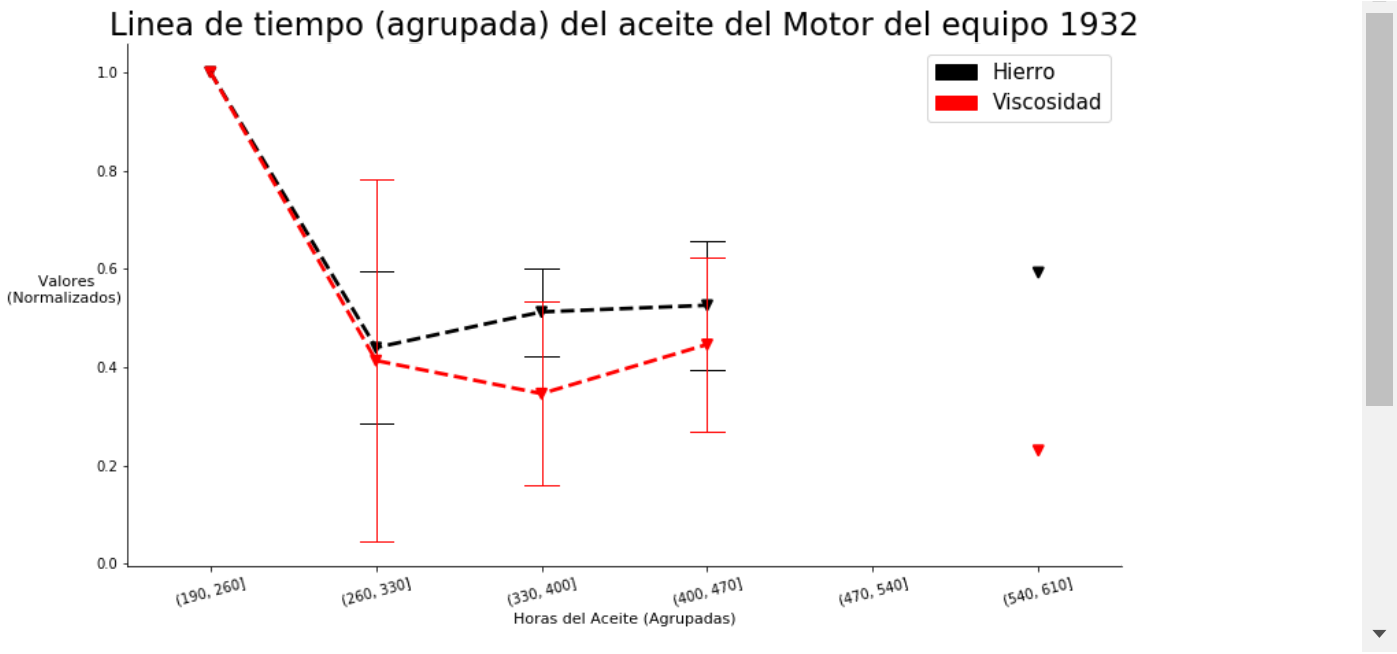
```
#Utilizamos una función de agrupación de las "Horas Funcionamiento" para una mejor visu
def to_categorical(column, bin_size=5, min_cut=15, max_cut=50):
    if min_cut is None:
        min_cut = int(round(column.min())) - 1
    value_max = int(np.ceil(column.max()))
    max_cut = min(max_cut, value_max)
    intervals = [(x, x+bin_size) for x in range(min_cut, max_cut, bin_size)]
    if max_cut != value_max:
        intervals.append((max_cut, value_max))
    return pd.cut(column, pd.IntervalIndex.from_tuples(intervals))


#seaborn.countplot(to_categorical(data_mot['Horas del Aceite'].dropna(),70,190,700), co
#plt.xticks(rotation=45)

intervalo_horas = to_categorical(data_mot['Horas del Aceite'].dropna(),70,190,700)
```

In [48]:

```
#Normalizamos los datos para poder mostrar los 4 features en un mismo gráfico
data_motor = data_mot.copy()
#Creamos una función z-minmax para normalizar los features entre [0,1]
def z(data, maxi, mini):#datos,maximo del feature, minimo del feature
    return (data-mini)/(maxi-mini)

#Obtenemos los datos necesarios de cada feature para realizar la normalización.
for elemento in elementos:
    maxi = data_motor[elemento].dropna().max() #Seleccionamos el máximo del feature
    mini = data_motor[elemento].dropna().min() #Seleccionamos el mínimo del feature

    data_motor[elemento] = z(data_motor[elemento],maxi,mini) #Asignamos al dataset los
```

In [49]:

```
#Realizamos la gráfica del Equipo con los valores normalizados
plt.figure(figsize=(13,7))
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Fe'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Hierro
#seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Na'],
#                   dodge=True, ci='sd',
#                   markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Sodio
#seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Si'],
#                   dodge=True, ci='sd',
#                   markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Silic
#seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['Al'],
#                   dodge=True, ci='sd',
#                   markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Alumi
seaborn.pointplot(data=data_motor, x=intervalo_horas, y=data_motor['VISCO'],
                  dodge=True, ci='sd',
                  markers='v', linestyles='--', errwidth=1, capsize=0.2, label='Viscos
plt.xticks(rotation=15)

#Seteamos el label de los componentes
label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

#Ponemos los labels en la imagen, y seteamos el tamaño de las etiquetas
plt.legend(handles=[label_Fe,label_VI],prop={'size': 15})
plt.title('Linea de tiempo (agrupada) del aceite del Motor del equipo {}'.format(equip
plt.xlabel('Horas del Aceite (Agrupadas)',size=11)
plt.ylabel('Valores          \n(Normalizados)           ',size=11,rotation=360)
seaborn.despine()
plt.show()

plt.figure(figsize=(25,7))

seaborn.pointplot(x=data_motor['Horas del Aceite'].dropna(), y=data_motor['Fe'].dropna
#seaborn.pointplot(x=data_motor['Horas del Aceite'].dropna(), y=data_motor['Na'].dropn
#seaborn.pointplot(x=data_motor['Horas del Aceite'].dropna(), y=data_motor['Si'].dropn
#seaborn.pointplot(x=data_motor['Horas del Aceite'].dropna(), y=data_motor['Al'].dropn
seaborn.pointplot(x=data_motor['Horas del Aceite'].dropna(), y=data_motor['VISCO'].dro

plt.title('Linea de tiempo (No agrupada) del aceite de Motor del equipo {}'.format(equ
plt.xlabel('Horas del Aceite',size=15)
plt.ylabel('Valores                  \n(Normalizados)                  ',size=15,rotat



label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')

plt.legend(handles=[label_Fe,label_VI],prop={'size': 15})
seaborn.despine()
plt.show()

```



En el gráfico anterior no se observa claramente que haya una correlación directa, aunque hay tendencias en comun

Analizamos la posibilidad de correlación analiticamente mediante el **Coeficiente de Spearman**

In [50]:

```
1 data_motor[['Fe','VISCO']].corr(method='spearman')
```

Out[50]:

	Fe	VISCO
Fe	1.000000	0.245319
VISCO	0.245319	1.000000

Observamos que el coeficiente es muy bajo, por lo que podríamos descartar la correlación entre los features

2. A continuación analizaremos el Sodio (Na) respecto a los Resultados de las muestras

In [51]:

```
1 # Fijamos una semilla para obtener siempre el mismo resultado
2 random.seed(47)
3 #Seleccionamos el índice de los 3 Equipos al azar (tenemos que convertir el "diccionar
4 componentes = random.sample(list(dataset[dataset['Equipo']==1402]['Componente'].values
5 componentes
```

Out[51]:

```
['Transmisión', 'Masa Izquierda', 'Masa Derecha']
```

In [52]:

```
#Seteamos los datos del componente a utilizar (Masa Derecha)
data_mot = dataset[(dataset['Componente']==componentes[2]) & (dataset['Equipo']==1402)
```

In [53]:

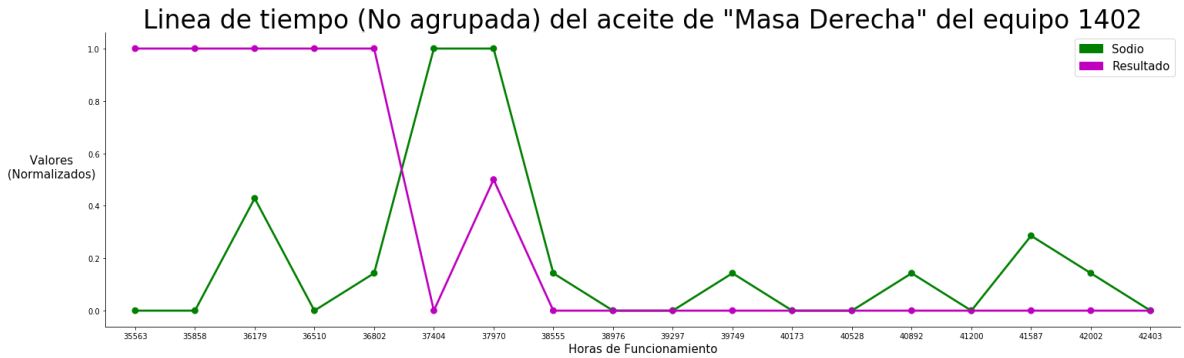
```
#Utilizamos una función de agrupación de las "Horas Funcionamiento" para una mejor visu
def to_categorical(column, bin_size=5, min_cut=15, max_cut=50):
    if min_cut is None:
        min_cut = int(round(column.min())) - 1
    value_max = int(np.ceil(column.max()))
    max_cut = min(max_cut, value_max)
    intervals = [(x, x+bin_size) for x in range(min_cut, max_cut, bin_size)]
    if max_cut != value_max:
        intervals.append((max_cut, value_max))
    return pd.cut(column, pd.IntervalIndex.from_tuples(intervals))


#seaborn.countplot(to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,35000,
#plt.xticks(rotation=45)

intervalo_horas = to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,35000,4
```

In [54]:

```
#Normalizamos los datos para poder mostrar los 5 features en un mismo gráfico
data_motor = data_mot.copy()
#Creamos una función z-minmax para normalizar los features entre [0,1]
def z(data, maxi, mini):#datos,maximo del feature, minimo del feature
    return (data-mini)/(maxi-mini)

#Obtenemos los datos necesarios de cada feature para realizar la normalización.
for elemento in elementos:
    maxi = data_motor[elemento].dropna().max() #Seleccionamos el máximo del feature
    mini = data_motor[elemento].dropna().min() #Seleccionamos el mínimo del feature

    data_motor[elemento] = z(data_motor[elemento],maxi,mini) #Asignamos al dataset los
```

In [55]:

```
#Realizamos la gráfica del Equipo con los valores normalizados

plt.figure(figsize=(25,7))

#seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Fe'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Na'].dr
#seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Si'].dr
#seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Al'].dr
#seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['VISCO'
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Resulta


plt.title('Linea de tiempo (No agrupada) del aceite de "{}" del equipo 1402'.format(co
plt.xlabel('Horas de Funcionamiento',size=15)
plt.ylabel('Valores                \n(Normalizados)                ',size=15,rotat


#label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
#label_Si = mpatches.Patch(color='c', label='Silicio')
#label_Al = mpatches.Patch(color='b', label='Aluminio')
#label_VI = mpatches.Patch(color='r', label='Viscosidad')
label_Re = mpatches.Patch(color='m', label='Resultado')

plt.legend(handles=[label_Na,label_Re],prop={'size': 15})
seaborn.despine()
plt.show()

```



Como vemos en el gráfico anterior, no se aprecia una correllación visible entre el resultado y el aumento del valor del Sodio

Analizamos la posibilidad de correlación analiticamente mediante el **Coeficiente de Spearman**

In [56]:

```
data_motor[['Fe','VISCO']].corr(method='spearman')
```

Out[56]:

	Fe	VISCO
Fe	1.00000	0.33925
VISCO	0.33925	1.00000

Observamos que el coeficiente es muy bajo, por lo que podríamos descartar la correlación entre los features

In [58]:

```
# Fijamos una semilla para obtener siempre el mismo resultado
random.seed(47)
#Seleccionamos el índice de los 3 Equipos al azar (tenemos que convertir el "diccionar
equipos = random.sample(list(dataset[dataset['Componente']=='Motor']['Equipo'].values)
equipos
```

Out[58]:

```
[1402, 1371, 1932]
```

In [59]:

```
#Seteamos los datos del equipo (1) a utilizar
data_mot = dataset[(dataset['Componente']=='Motor') & (dataset['Equipo']==1371)]

```

In [60]:

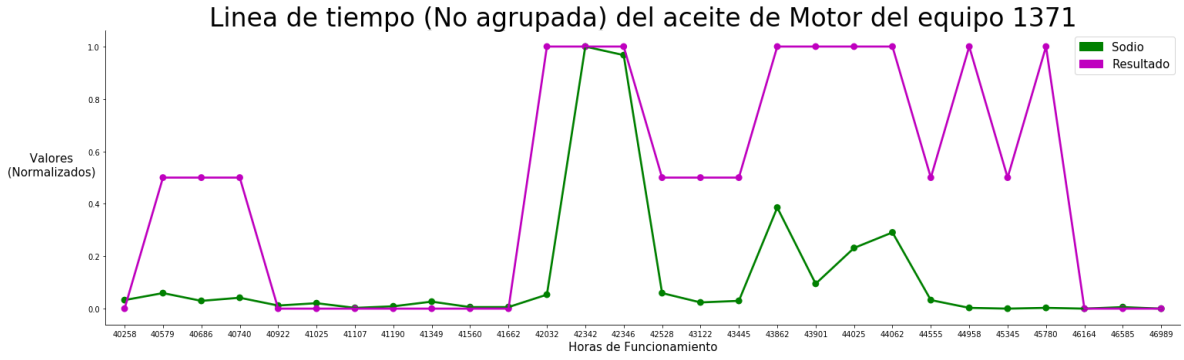
```
#Utilizamos una función de agrupación de las "Horas Funcionamiento" para una mejor visu
def to_categorical(column, bin_size=5, min_cut=15, max_cut=50):
    if min_cut is None:
        min_cut = int(round(column.min())) - 1
    value_max = int(np.ceil(column.max()))
    max_cut = min(max_cut, value_max)
    intervals = [(x, x+bin_size) for x in range(min_cut, max_cut, bin_size)]
    if max_cut != value_max:
        intervals.append((max_cut, value_max))
    return pd.cut(column, pd.IntervalIndex.from_tuples(intervals))


#seaborn.countplot(to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,35000,
#plt.xticks(rotation=45)

intervalo_horas = to_categorical(data_mot['Horas Funcionamiento'].dropna(),1000,35000,
```

In [61]:

```
#Normalizamos los datos para poder mostrar los features en un mismo gráfico
data_motor = data_mot.copy()
#Creamos una función z-minmax para normalizar los features entre [0,1]
def z(data, maxi, mini):#datos,maximo del feature, minimo del feature
    return (data-mini)/(maxi-mini)


#Obtenemos los datos necesarios de cada feature para realizar la normalización.
for elemento in elementos:
    maxi = data_motor[elemento].dropna().max() #Seleccionamos el máximo del feature
    mini = data_motor[elemento].dropna().min() #Seleccionamos el mínimo del feature

    data_motor[elemento] = z(data_motor[elemento],maxi,mini) #Asignamos al dataset los
```

In [62]:

```
#Realizamos la gráfica del Equipo con los valores normalizados

plt.figure(figsize=(25,7))

#seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Fe'].dr
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Na'].dr
#seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Si'].dr
#seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Al'].dr
#seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['VISCO'
seaborn.pointplot(x=data_motor['Horas Funcionamiento'].dropna(), y=data_motor['Resulta


plt.title('Linea de tiempo (No agrupada) del aceite de Motor del equipo {}'.format(equ
plt.xlabel('Horas de Funcionamiento',size=15)
plt.ylabel('Valores                    \n(Normalizados)                    ',size=15,rotat


label_Fe = mpatches.Patch(color='k', label='Hierro')
label_Na = mpatches.Patch(color='g', label='Sodio')
label_Si = mpatches.Patch(color='c', label='Silicio')
label_Al = mpatches.Patch(color='b', label='Aluminio')
label_VI = mpatches.Patch(color='r', label='Viscosidad')
label_Re = mpatches.Patch(color='m', label='Resultado')


plt.legend(handles=[label_Na,label_Re],prop={'size': 15})
seaborn.despine()
plt.show()

```



Aquí(arriba) si parecería haber una correlación visual entre el Resultado y el Valor de Sodio, pero no es tan evidente. **Se aprecia que cuando los valores de Sodio se elevan, la muestra tiende a ser mala o regular.**

Analizamos la posibilidad de correlación analiticamente mediante el **Coeficiente de Spearman**

In [63]:

```
1 data_motor[['Fe','VISCO']].corr(method='spearman')
```

Out[63]:

	Fe	VISCO
Fe	1.000000	0.232281
VISCO	0.232281	1.000000

Observamos que el coeficiente es muy bajo, por lo que podríamos descartar la correlación entre los features