Proyecto de Machine Learning: Predicción de Cancelación de Clientes

1. Introducción

Al operador de telecomunicaciones **Interconnect** le gustaría poder pronosticar su tasa de cancelación de clientes. Si se descubre que un usuario o usuaria planea irse, se le ofrecerán códigos promocionales y opciones de planes especiales. El equipo de marketing de Interconnect ha recopilado algunos de los datos personales de sus clientes, incluyendo información sobre sus planes y contratos.

En este proyecto se realizara un anlisis exploratorio de los datos proporcionados, se manipularan y procesaran de manera que se pueda crear al final de una etapa de preprocesamiento, un dataset para poder entrenar diferentes modelos, la métrica clave será 'AUC-ROC' y se buscara un modelo que cumpla con el siguiento objetivo:

Criterios de evaluación:

- **AUC-ROC < 0.75** — 0 SP
- **0.75 ≤ AUC-ROC < 0.81** — 4 SP
- **0.81 ≤ AUC-ROC < 0.85** — 4.5 SP
- **0.85 ≤ AUC-ROC < 0.87** — 5 SP
- **0.87 ≤ AUC-ROC < 0.88** — 5.5 SP
- **AUC-ROC ≥ 0.88** — 6 SP

Usare:

Clasificación: Random Forest, XGBoost, Logistic Regression, Redes Neuronales.

```
# Importar librerías para el preprocesamiento de datos
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Importar librerías para la modelización
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score, accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import numpy as np
import pandas as pd
import re
# Importar librerías para visualización
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots as sp


# Importar librerías adicionales (si es necesario)
import datetime
```

2. Análisis Exploratorio de Datos (EDA)

Comenzaremos con el analisis de los datasets en este orden:

1. contract.csv
2. internet.csv
3. personal.csv
4. phone.csv

```
# Contract.csv

df_contract = pd.read_csv('datasets/contract.csv')
```

```
df_contract.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   BeginDate         7043 non-null   object 
 2   EndDate           7043 non-null   object 
 3   Type              7043 non-null   object 
 4   PaperlessBilling  7043 non-null   object 
 5   PaymentMethod     7043 non-null   object 
 6   MonthlyCharges    7043 non-null   float64
 7   TotalCharges      7043 non-null   object 
dtypes: float64(1), object(7)
memory usage: 440.3+ KB
```

```
df_contract.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
```

```
 1   BeginDate         7043 non-null   object
 2   EndDate           7043 non-null   object
 3   Type              7043 non-null   object
 4   PaperlessBilling  7043 non-null   object
 5   PaymentMethod     7043 non-null   object
 6   MonthlyCharges    7043 non-null   float64
 7   TotalCharges      7043 non-null   object
dtypes: float64(1), object(7)
memory usage: 440.3+ KB
```

La primera impresión es positiva, no hay columnas con entradas nulas, solo hay que cambiar algunos tipos de dato como por ejemplo, ***BeginDate*** a tipo **datetime**, ***TotalCharges*** a tipo **float64** y modificar los nombres

```
def ColumName_change(df):
    new_cols = []

    for col_name in df.columns:
        # Paso 1: Insertar guion bajo antes de mayúsculas, pero manejar adecuadamente secuencias 
        new_col_name = re.sub(r'(?<=[a-z0-9])([A-Z])', r'_\1', col_name)  # Insertar _ antes de ma

        # Paso 2: Convertir todo a minúsculas
        new_col_name = new_col_name.lower()

        # Paso 3: Eliminar cualquier guion bajo inicial o duplicado
        new_col_name = re.sub(r'_+', '_', new_col_name).strip('_')

        new_cols.append(new_col_name)

    df.columns = new_cols
    return df
```

```
# Aplicación de la función a un DataFrame
df_contract = ColumName_change(df_contract)
df_contract
```

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charg
0	7590-VHVEG	2020-01-01	No	Month-to-month	Yes	Electronic check	29.85	29.85
1	5575-GNVDE	2017-04-01	No	One year	No	Mailed check	56.95	1889.5
2	3668-QPYBK	2019-10-01	2019-12-01 00:00:00	Month-to-month	Yes	Mailed check	53.85	108.15
3	7795-	2016-05-01	No	One	No	Bank transfer	42.30	1840.75

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charg
	CFOCW			year		(automatic)		
4	9237-HQITU	2019-09-01	2019-11-01 00:00:00	Month-to-month	Yes	Electronic check	70.70	151.65
...	...	...	...	...	...	...	...	...
7038	6840-RESVB	2018-02-01	No	One year	Yes	Mailed check	84.80	1990.5
7039	2234-XADUH	2014-02-01	No	One year	Yes	Credit card (automatic)	103.20	7362.9
7040	4801-JZAZL	2019-03-01	No	Month-to-month	Yes	Electronic check	29.60	346.45
7041	8361-LTMKD	2019-07-01	2019-11-01 00:00:00	Month-to-month	Yes	Mailed check	74.40	306.6
7042	3186-AJIEK	2014-08-01	No	Two year	Yes	Bank transfer (automatic)	105.65	6844.5

7043 rows × 8 columns

```
# Cambios de tipo de dato en la columnas

df_contract['begin_date'] = pd.to_datetime(df_contract['begin_date'])
df_contract['total_charges'] = pd.to_numeric(df_contract['total_charges'], errors='coerce')

df_contract.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   customer_id        7043 non-null   object
 1   begin_date         7043 non-null   datetime64[ns]
 2   end_date           7043 non-null   object
 3   type               7043 non-null   object
 4   paperless_billing  7043 non-null   object
 5   payment_method     7043 non-null   object
 6   monthly_charges    7043 non-null   float64
 7   total_charges      7032 non-null   float64
dtypes: datetime64[ns](1), float64(2), object(5)
memory usage: 440.3+ KB
```

En la linea modificada de ***total_charges*** lo converti a numerico y los datos que no puediesen convertirse a float se hicieron 'Nan', por lo tanto hay que rellenar esos datos, como son pocos lo haré a continuación

pues se puede buscar la posibilidad de calcularlo mediante el tiempo que llevan y su cargo mensual

```
# Observando los valores nulos de de la columna total_charges

df_contract[df_contract['total_charges'].isnull()]
```

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charges
488	4472-LVYGI	2020-02-01	No	Two year	Yes	Bank transfer (automatic)	52.55	NaN
753	3115-CZMZD	2020-02-01	No	Two year	No	Mailed check	20.25	NaN
936	5709-LVOEQ	2020-02-01	No	Two year	No	Mailed check	80.85	NaN
1082	4367-NUYAO	2020-02-01	No	Two year	No	Mailed check	25.75	NaN
1340	1371-DWPAZ	2020-02-01	No	Two year	No	Credit card (automatic)	56.05	NaN
3331	7644-OMVMY	2020-02-01	No	Two year	No	Mailed check	19.85	NaN
3826	3213-VVOLG	2020-02-01	No	Two year	No	Mailed check	25.35	NaN
4380	2520-SGTTA	2020-02-01	No	Two year	No	Mailed check	20.00	NaN
5218	2923-ARZLG	2020-02-01	No	One year	Yes	Mailed check	19.70	NaN
6670	4075-WKNIU	2020-02-01	No	Two year	No	Mailed check	73.35	NaN
6754	2775-SEFEE	2020-02-01	No	Two year	Yes	Bank transfer (automatic)	61.90	NaN

```
df_contract[df_contract['type'] == 'Two year']
```

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charges
11	7469-LKBCI	2018-10-01	No	Two year	No	Credit card (automatic)	18.95	326.80
15	3655-SNQYZ	2014-05-01	No	Two year	No	Credit card (automatic)	113.25	7895.15
17	9959-WOFKT	2014-03-01	No	Two year	No	Bank transfer (automatic)	106.70	7382.25

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charges
23	3638-WEABW	2015-04-01	No	Two year	Yes	Credit card (automatic)	59.90	3505.10
28	5248-YGIJN	2014-02-01	No	Two year	Yes	Credit card (automatic)	90.25	6369.45
...	...	...	...	...	...	...	...	...
7017	4807-IZYOZ	2015-11-01	No	Two year	No	Bank transfer (automatic)	20.65	1020.75
7019	9710-NJERN	2016-11-01	No	Two year	No	Mailed check	20.15	826.00
7028	9281-CEDRU	2014-06-01	No	Two year	No	Bank transfer (automatic)	64.10	4326.25
7037	2569-WGERO	2014-02-01	No	Two year	Yes	Bank transfer (automatic)	21.15	1419.40
7042	3186-AJIEK	2014-08-01	No	Two year	Yes	Bank transfer (automatic)	105.65	6844.50

1695 rows × 8 columns

```
df_contract[df_contract['begin_date'] == '2020-02-01']
```

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charges
488	4472-LVYGI	2020-02-01	No	Two year	Yes	Bank transfer (automatic)	52.55	NaN
753	3115-CZMZD	2020-02-01	No	Two year	No	Mailed check	20.25	NaN
936	5709-LVOEQ	2020-02-01	No	Two year	No	Mailed check	80.85	NaN
1082	4367-NUYAO	2020-02-01	No	Two year	No	Mailed check	25.75	NaN
1340	1371-DWPAZ	2020-02-01	No	Two year	No	Credit card (automatic)	56.05	NaN
3331	7644-OMVMY	2020-02-01	No	Two year	No	Mailed check	19.85	NaN
3826	3213-VVOLG	2020-02-01	No	Two year	No	Mailed check	25.35	NaN
4380	2520-SGTTA	2020-02-01	No	Two year	No	Mailed check	20.00	NaN
5218	2923-ARZLG	2020-02-01	No	One year	Yes	Mailed check	19.70	NaN

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charges
6670	4075-WKNIU	2020-02-01	No	Two year	No	Mailed check	73.35	NaN
6754	2775-SEFEE	2020-02-01	No	Two year	Yes	Bank transfer (automatic)	61.90	NaN

```
# Encontrar la fecha más actual
fecha_mas_actual = df_contract['begin_date'].max()

print("La fecha más actual es:", fecha_mas_actual)
```

```
La fecha más actual es: 2020-02-01 00:00:00
```

Explorando los datos veo que las peronas que tienen el registro más actual son de la fecha : 2020-02-01

También observo que los clientes que iniciaron en esa fecha no tienen **total_charge**, por lo tanto para calcular el pago hay que multiplicar la carga mensual que si viene registrada por los 24 meses del contrato

```
def Total_chargeCalculate(df):
    # Convertir 'type' a minúsculas para consistencia
    df['type'] = df['type'].str.lower()

    # Crear máscara para filas con 'total_charges' nulo y tipo de contrato
    mask_two_year = (df['total_charges'].isnull()) & (df['type'] == 'two year')
    mask_one_year = (df['total_charges'].isnull()) & (df['type'] == 'one year')

    # Actualizar 'total_charges' para 'two year'
    df.loc[mask_two_year, 'total_charges'] = df.loc[mask_two_year, 'monthly_charges'] * 24

    # Actualizar 'total_charges' para 'one year'
    df.loc[mask_one_year, 'total_charges'] = df.loc[mask_one_year, 'monthly_charges'] * 12

    return df
```

```
df_contract = Total_chargeCalculate(df_contract)

df_contract[df_contract['total_charges'].isnull()]
```

customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charges

Con esto confirmamos que se calculo correctamente todos lo valores, los cargos que no habían sido registrados ya se calcularon

```
# Entradas unicas en cada fila

df_contract.nunique()
```

```
customer_id          7043
begin_date             77
end_date                5
type                    3
paperless_billing       2
payment_method          4
monthly_charges      1585
total_charges        6541
dtype: int64
```

```
# Imprimiento entradas unicas de columnas
cat_cols = ['end_date',  'type', 'paperless_billing','payment_method']

for col in cat_cols:
    print(df_contract[col].unique())
```

```
['No' '2019-12-01 00:00:00' '2019-11-01 00:00:00' '2019-10-01 00:00:00'
 '2020-01-01 00:00:00']
['month-to-month' 'one year' 'two year']
['Yes' 'No']
['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
```

```
for i in df_contract['end_date'].unique():
    print("Hay {} filas que tienen a '{}' como entrada en 'end_date".format(len(df_contract[df_co
```

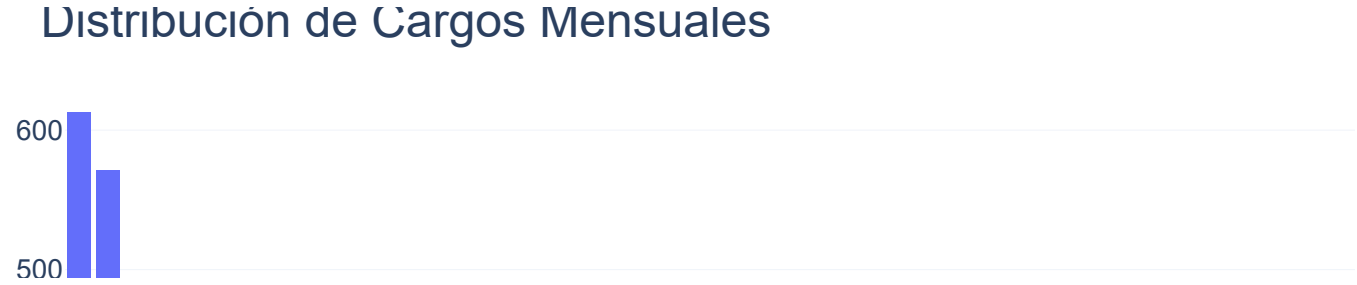
```
Hay 5174 filas que tienen a 'No' como entrada en 'end_date
Hay 466 filas que tienen a '2019-12-01 00:00:00' como entrada en 'end_date
Hay 485 filas que tienen a '2019-11-01 00:00:00' como entrada en 'end_date
Hay 458 filas que tienen a '2019-10-01 00:00:00' como entrada en 'end_date
Hay 460 filas que tienen a '2020-01-01 00:00:00' como entrada en 'end_date
```

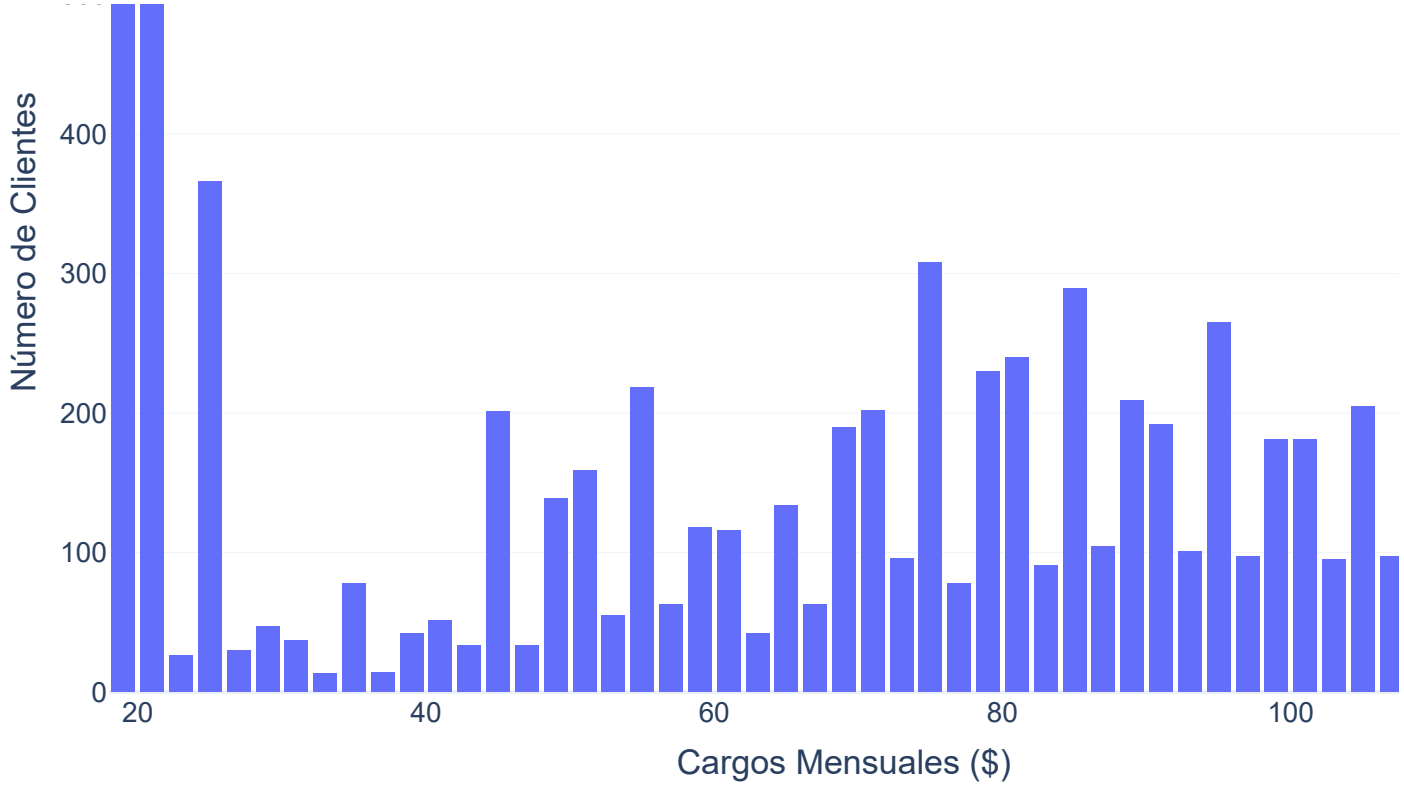
```
df_contract = df_contract.sort_values(by='begin_date')
df_contract.head(10)
```

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charges
4513	8580-QVLOC	2013-10-01	2019-10-01 00:00:00	two year	No	Credit card (automatic)	92.45	6440.25
4610	2889-FPWRM	2013-10-01	2019-10-01 00:00:00	one year	Yes	Bank transfer (automatic)	117.80	8684.80
3439	0917-EZOLA	2013-10-01	2019-10-01 00:00:00	two year	Yes	Bank transfer (automatic)	104.15	7689.95

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charges
975	2834-JRTUA	2013-11-01	2019-10-01 00:00:00	two year	Yes	Electronic check	108.05	7532.15
3040	7317-GGVPB	2013-11-01	2019-10-01 00:00:00	two year	Yes	Credit card (automatic)	108.60	7690.90
6038	1555-DJEQW	2013-12-01	2019-10-01 00:00:00	two year	Yes	Bank transfer (automatic)	114.20	7723.90
6290	2530-ENDWQ	2013-12-01	2019-11-01 00:00:00	two year	Yes	Bank transfer (automatic)	93.70	6585.35
5441	3512-IZIKN	2013-12-01	2019-10-01 00:00:00	two year	No	Credit card (automatic)	65.30	4759.75
6424	6034-ZRYCV	2014-01-01	2020-01-01 00:00:00	two year	Yes	Electronic check	54.20	3937.45
4684	6305-YLBMM	2014-01-01	2019-10-01 00:00:00	one year	Yes	Bank transfer (automatic)	104.05	7262.00

```
fig = px.histogram(
    df_contract,
    x='monthly_charges',
    title='Distribución de Cargos Mensuales',
    color_discrete_sequence=['#636EFA'],  # Color moderno
    template='plotly_white'  # Diseño limpio
)
fig.update_layout(
    title_font_size=24,
    xaxis_title='Cargos Mensuales ($)',
    yaxis_title='Número de Clientes',
    font=dict(family="Arial", size=14),
    bargap=0.2
)
fig.show()
```

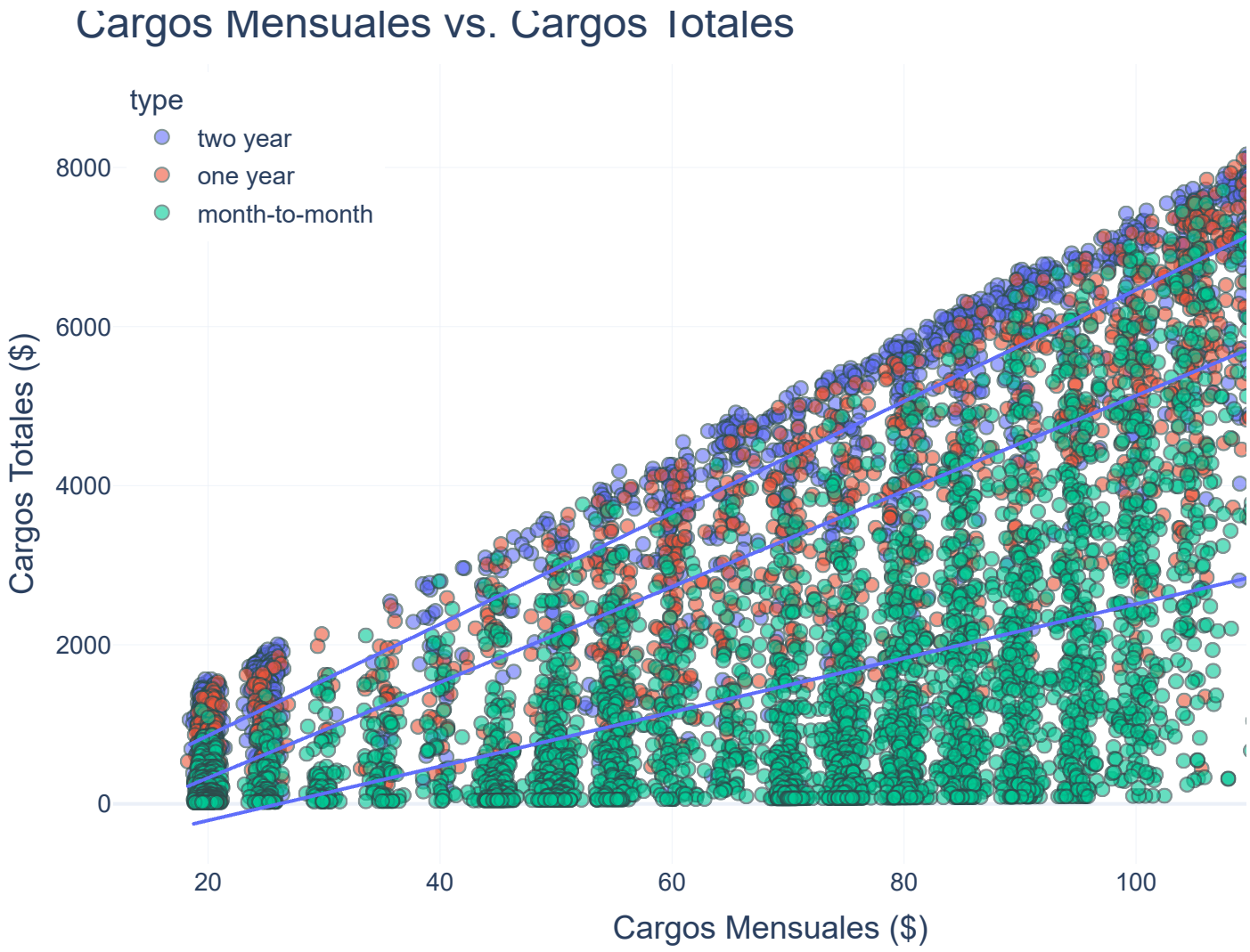




Se observa que la distribución entre el cargo mensual y el número de clientes es predominante en los 18-22 dolares, existe a partir de los 48 dolares un cargo más estable

```
fig = px.scatter(
    df_contract,
    x='monthly_charges',
    y='total_charges',
    title='Cargos Mensuales vs. Cargos Totales',
    labels={'monthly_charges': 'Cargos Mensuales ($)', 'total_charges': 'Cargos Totales ($)'},
    color='type',
    color_discrete_sequence=px.colors.qualitative.Plotly,
    template='plotly_white',
    opacity=0.6  # Transparencia para reducir sobreposición
)

# Ajuste del tamaño de los puntos y línea de tendencia
fig.update_traces(marker=dict(size=8, line=dict(width=1, color='DarkSlateGrey')))

# Mejora del layout
fig.update_layout(
    title_font_size=24,
    xaxis_title_font=dict(size=18),
    yaxis_title_font=dict(size=18),
    legend_title_font=dict(size=16),
    font=dict(family="Arial", size=14),
    legend=dict(yanchor="top", y=0.99, xanchor="left", x=0.01),  # Mover la leyenda
)

# Agregar línea de tendencia para cada tipo de contrato
```

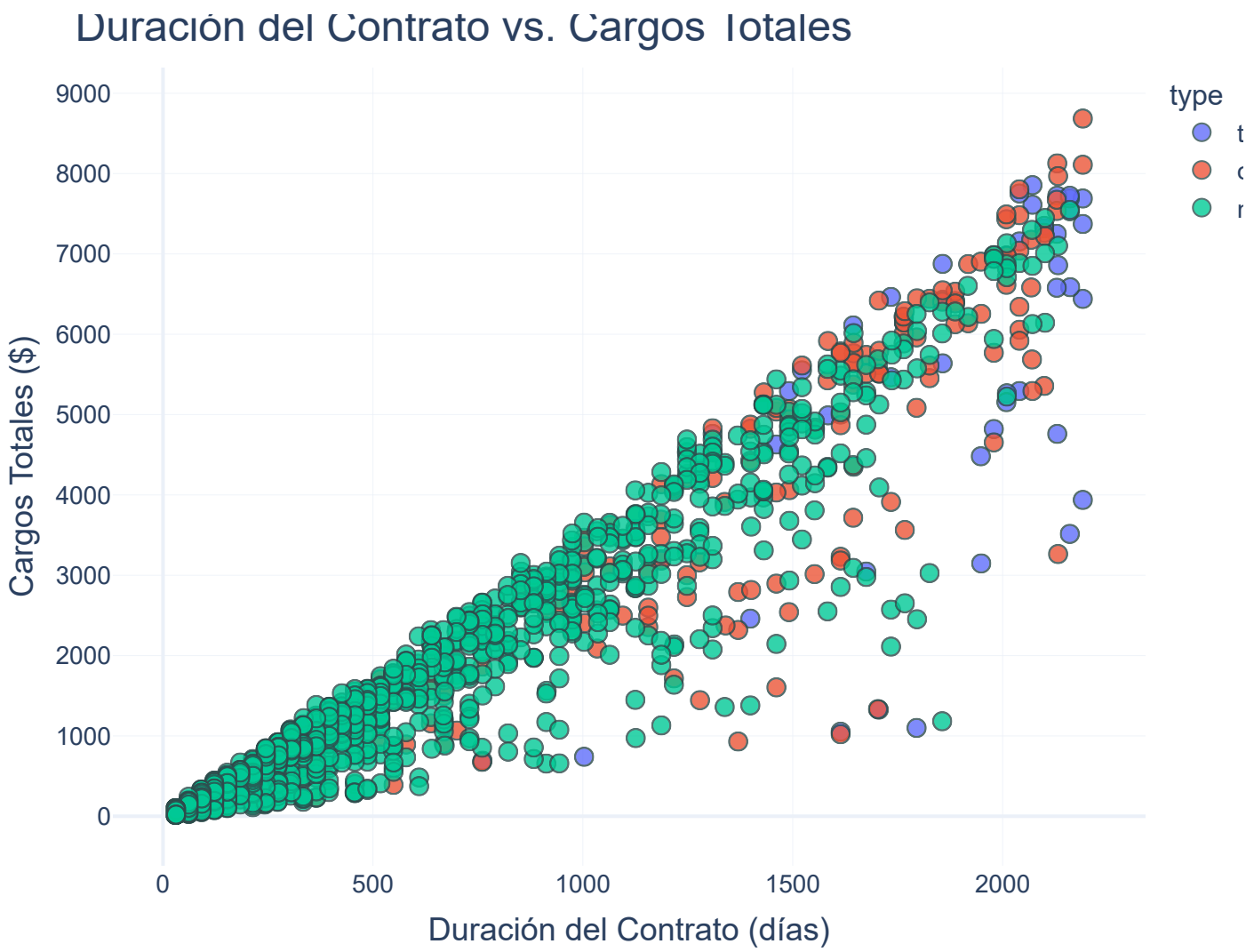
```
for contract_type in df_contract['type'].unique():
    df_subset = df_contract[df_contract['type'] == contract_type]
    fig.add_traces(px.scatter(df_subset, x='monthly_charges', y='total_charges',
                              trendline="ols").data[1])

fig.show()
```



Existe una clara relación entre cargos mensuales y totales

```
df_contract_copy_1 = df_contract.copy()
df_contract_copy_1['begin_date'] = pd.to_datetime(df_contract_copy_1['begin_date'])
df_contract_copy_1['end_date'] = pd.to_datetime(df_contract_copy_1['end_date'], errors='coerce')
df_contract_copy_1['contract_duration'] = (df_contract_copy_1['end_date'] - df_contract['begin_da

fig = px.scatter(
    df_contract_copy_1,
    x='contract_duration',
    y='total_charges',
    title='Duración del Contrato vs. Cargos Totales',
    labels={'contract_duration': 'Duración del Contrato (días)', 'total_charges': 'Cargos Totales
    color='type',  # Diferenciar por tipo de contrato
    color_continuous_scale=px.colors.sequential.Viridis,
    template='plotly_white'
```

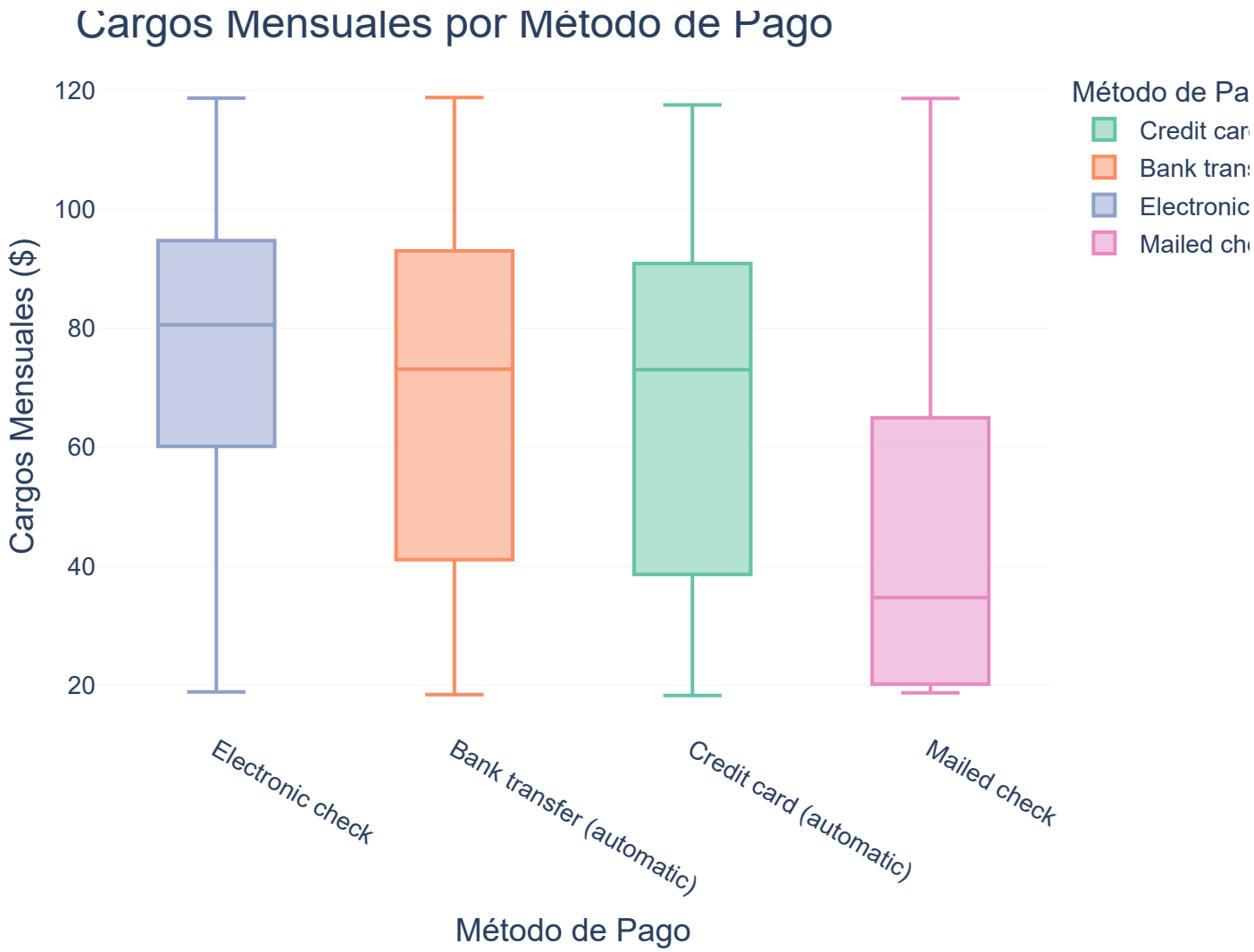
```
)
fig.update_traces(marker=dict(size=10, opacity=0.8, line=dict(width=1, color='DarkSlateGrey')))
fig.update_layout(
    title_font_size=24,
    xaxis_title_font=dict(size=18),
    yaxis_title_font=dict(size=18),
    legend_title_font=dict(size=16),
    font=dict(family="Arial", size=14)
)
fig.show()
```



Vemos que hay una reación entre la duración del contrato y los cargos totales, parece un grosor de datos como linea recta tendencial

```
fig = px.box(
    df_contract,
    x='payment_method',
    y='monthly_charges',
    title='Cargos Mensuales por Método de Pago',
    labels={'payment_method': 'Método de Pago', 'monthly_charges': 'Cargos Mensuales ($)'},
    color='payment_method',
```

```
    color_discrete_sequence=px.colors.qualitative.Set2,  # Colores suaves y modernos
    template='plotly_white'
)
fig.update_layout(
    title_font_size=24,
    xaxis_title_font=dict(size=18),
    yaxis_title_font=dict(size=18),
    legend_title_font=dict(size=16),
    font=dict(family="Arial", size=14),
    xaxis={'categoryorder':'total descending'}  # Ordenar categorías
)
fig.show()
```

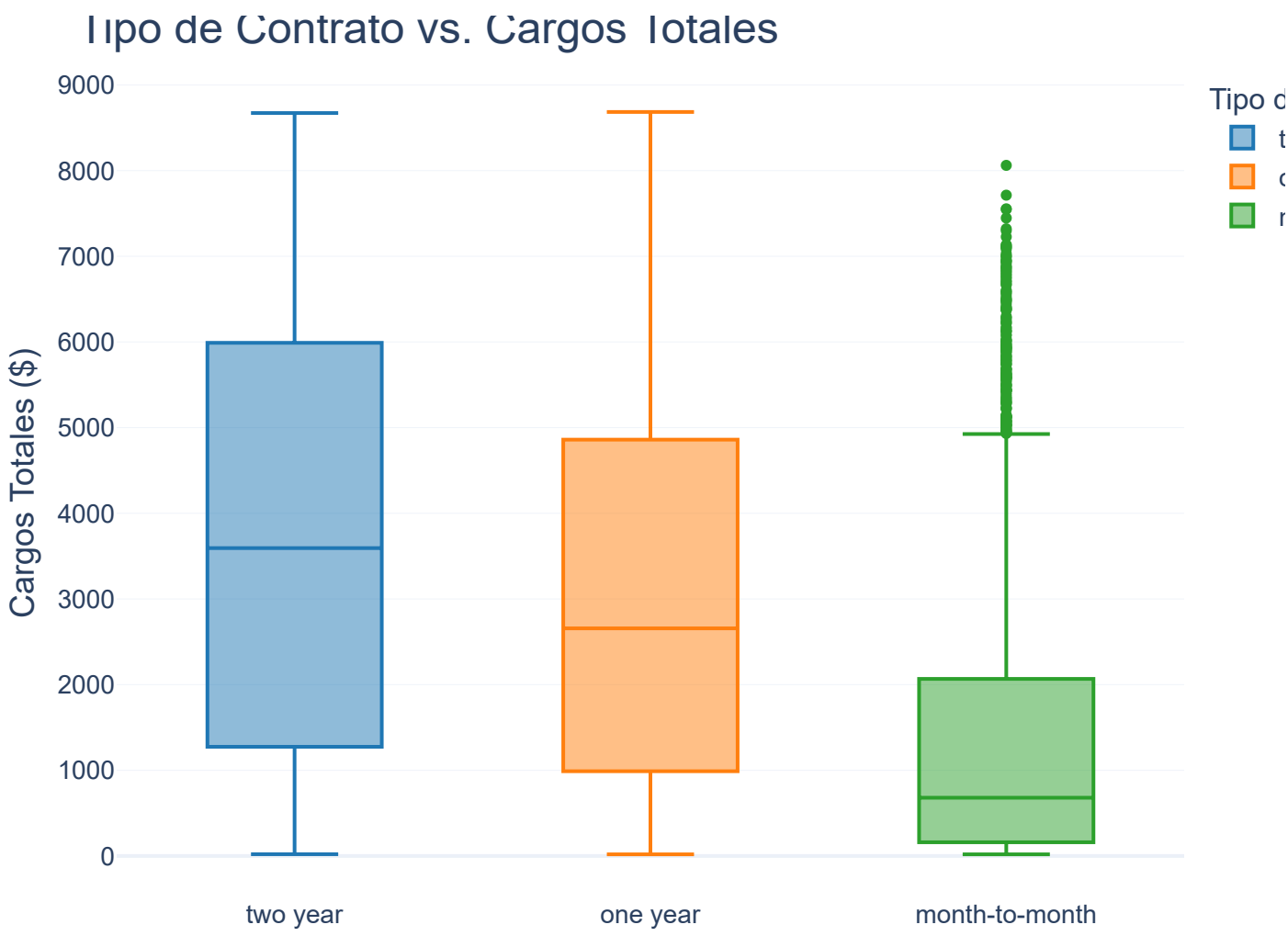


Se observa el los boxplot entre metodo de pago y cargos mensuales que 'Bank transfer' y 'Credit card' tienen cajas muy similares, puede ser debido a el metodo de pago que es algo similar, las otras 2 difieren bastante.

Hablando de el metodo de pago por chequeo electronico vemos que los cargos mensuales entre el quartil 1 y 3 esta más comprimido, la mediana es la más alta de todas($80.55), el 75% de las cargas mensuales por ese metodo son inferiores a $94.7

En cuanto a los cheques por correo es todo lo contrario, el 25% de los datos es menor a $18.7 que de hecho coincide con el dato minimo por este metodo, el 75% tienen cargos mensuales menores a los $65 por los que puede ser que haya una tendencia a pagar por este metodo cuando las caragas mensuales son inferiores.

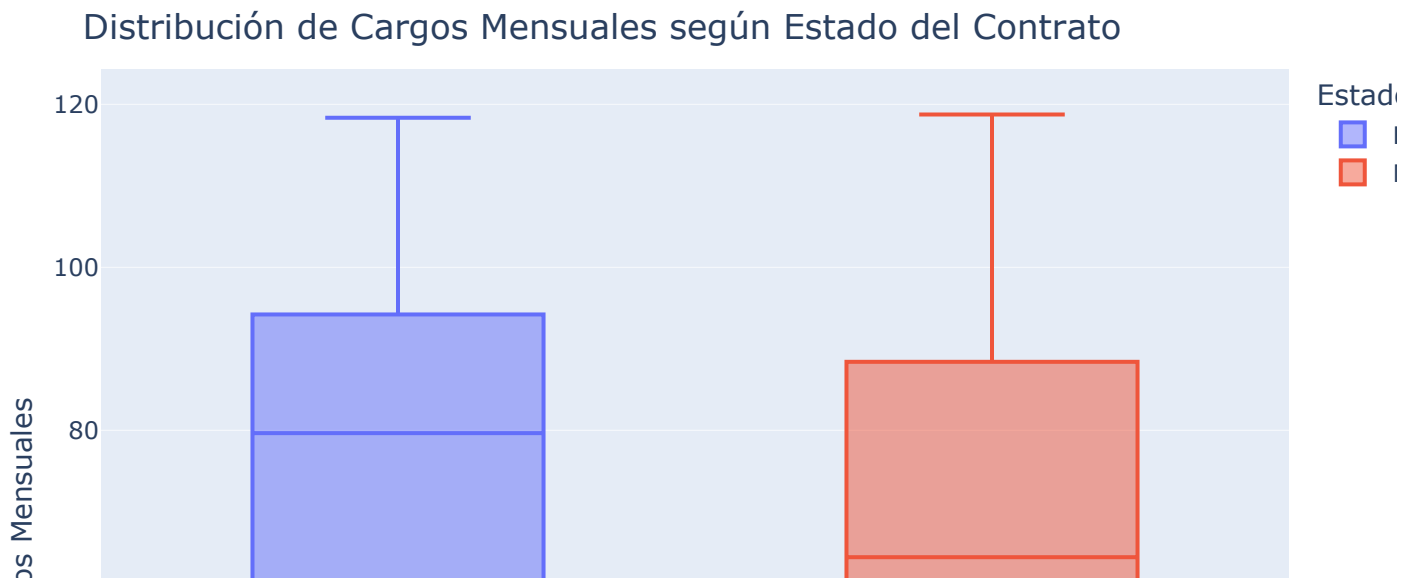
```
fig = px.box(
    df_contract,
    x='type',
    y='total_charges',
    title='Tipo de Contrato vs. Cargos Totales',
    labels={'type': 'Tipo de Contrato', 'total_charges': 'Cargos Totales ($)'},
    color='type',
    color_discrete_sequence=px.colors.qualitative.D3,  # Colores vibrantes
    template='plotly_white'
)
fig.update_layout(
    title_font_size=24,
    xaxis_title_font=dict(size=18),
    yaxis_title_font=dict(size=18),
    legend_title_font=dict(size=16),
    font=dict(family="Arial", size=14)
)
fig.show()
```

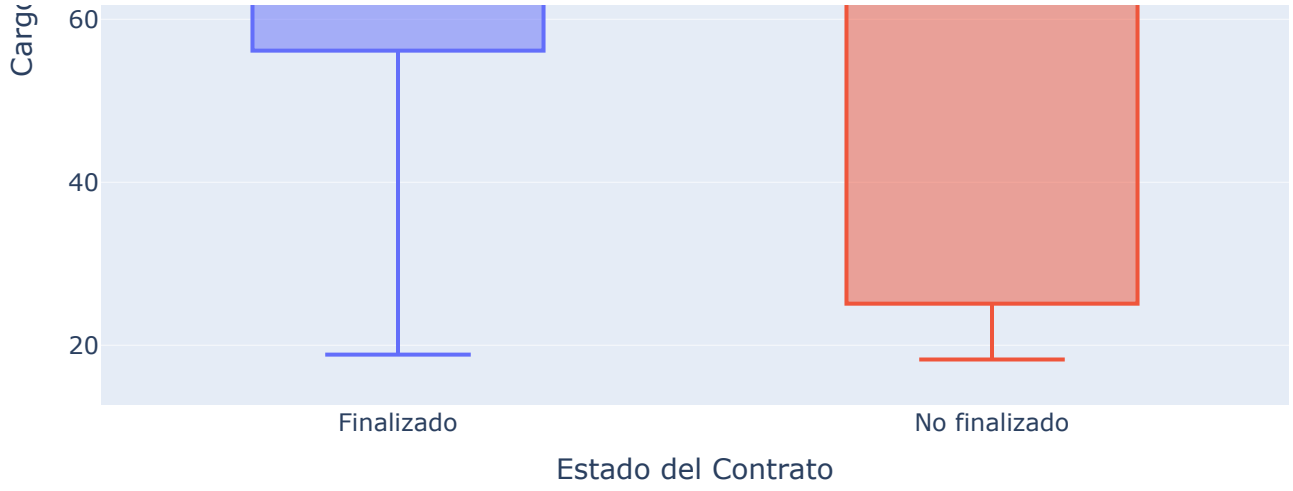


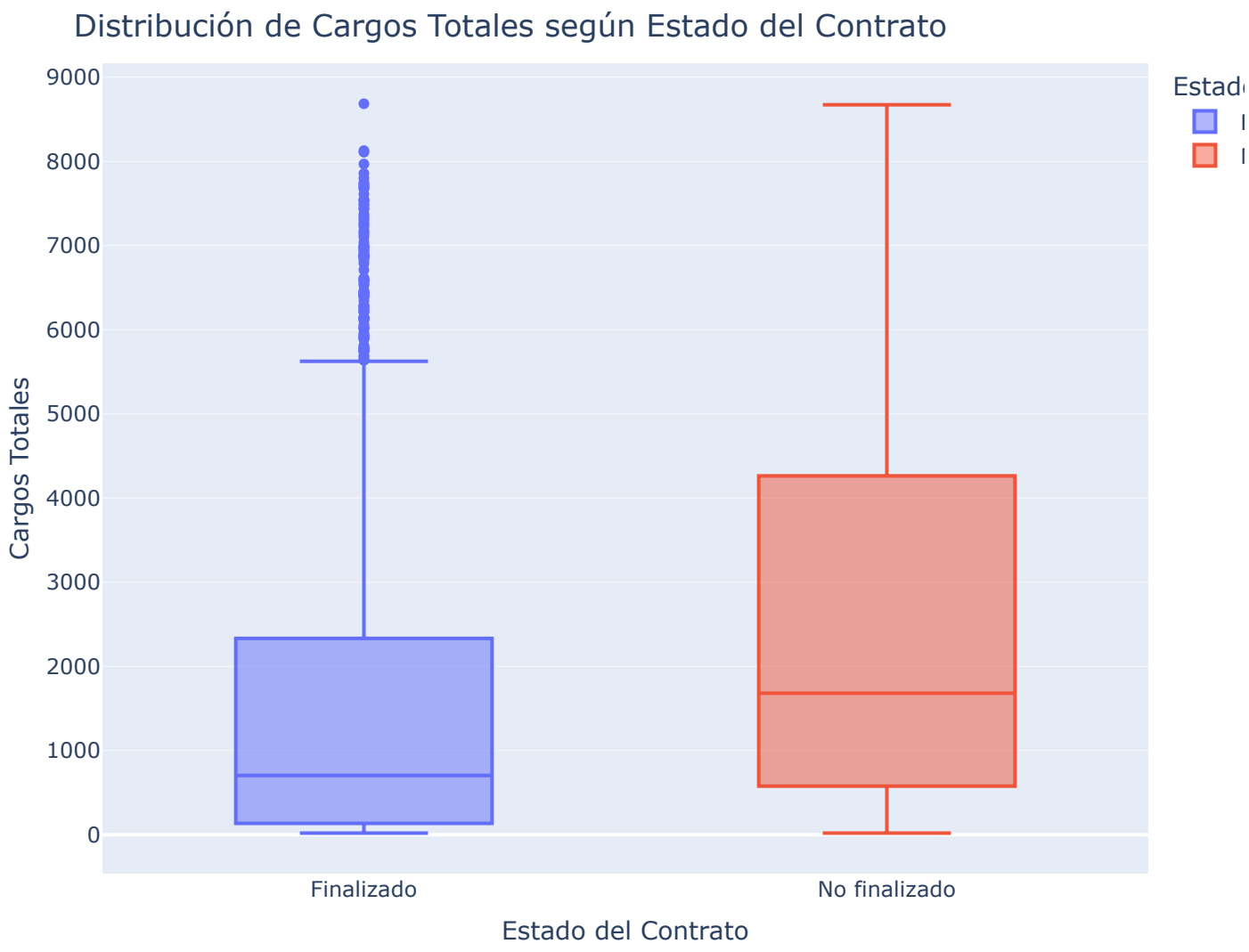
Tipo de Contrato

A perimera vista podemos ver que la caja de los que tienen su contrato mes a mes esta comprimida a cargos totales inferiores, tamabién tiene una cantidad considerable de datos anomalos como es de esperarse,

```
# Copia del dataset original
df_contract_copy = df_contract.copy()

# Crear la columna 'end_status'
df_contract_copy['end_status'] = df_contract_copy['end_date'].apply(lambda x: 'No finalizado' if x

# Gráfico de distribución de cargos mensuales en función del estado del contrato
fig_monthly_charges = px.box(df_contract_copy,
                             x='end_status',
                             y='monthly_charges',
                             title='Distribución de Cargos Mensuales según Estado del Contrato',
                             color='end_status',
                             labels={'end_status': 'Estado del Contrato', 'monthly_charges': 'Carg
                             color_discrete_sequence=['#636EFA', '#EF553B'])

# Gráfico de distribución de cargos totales en función del estado del contrato
fig_total_charges = px.box(df_contract_copy,
                           x='end_status',
                           y='total_charges',
                           title='Distribución de Cargos Totales según Estado del Contrato',
                           color='end_status',
                           labels={'end_status': 'Estado del Contrato', 'total_charges': 'Cargos
                           color_discrete_sequence=['#636EFA', '#EF553B'])

# Mostrar los gráficos
fig_monthly_charges.show()
fig_total_charges.show()
```







Vemos 2 graficos que comparan a las personas que no han finalizado su contrato y las que ya lo hicieron

En el primer gradico observamos estos 2 tipos clientes por Cargos mensuales observamos que los que no han finalizado su contrato tienen cargos mensuales más variados y los que ya lo finalizaron suelen tener cargos más altos, esto se puede observar en las diferencias del primer quartil de ambos tipos de clientes, los que ya lo finalizaron es de $56 y los que no $25.1, el 3er quartil también existe una diferencia que puede hacer pensar que los que finalizas su contrato suelen tener caragas mensuales más altas.

En el segundo grafico se observa que los clientes que finalizan suelen tener cargas totales más bajas, las cargas altas suelen ser anomalas, se podría decir que son clientes de corto plazo, mientras que los que no han finalizado tienen cargas más altas, quiza clientes que estan contentos con el servicio a largo plazo no suelen finalizar

```
# internet.csv

df_internet = pd.read_csv('datasets/internet.csv')
```

```
df_internet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5517 entries, 0 to 5516
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   customerID        5517 non-null   object
 1   InternetService   5517 non-null   object
 2   OnlineSecurity    5517 non-null   object
 3   OnlineBackup      5517 non-null   object
 4   DeviceProtection  5517 non-null   object
 5   TechSupport       5517 non-null   object
 6   StreamingTV       5517 non-null   object
 7   StreamingMovies   5517 non-null   object
dtypes: object(8)
memory usage: 344.9+ KB
```

Este dataset da información sobre los servicios de internet, los tipos de datos parecen estar bien unicamente seria modificar los nombres de las columnas

```
df_internet = ColumName_change(df_internet)

df_internet
```

	customer_id	internet_service	online_security	online_backup	device_protection	tech_support	streaming_tv
0	7590-VHVEG	DSL	No	Yes	No	No	No
1	5575-GNVDE	DSL	Yes	No	Yes	No	No
2	3668-QPYBK	DSL	Yes	Yes	No	No	No
3	7795-CFOCW	DSL	Yes	No	Yes	Yes	No
4	9237-HQITU	Fiber optic	No	No	No	No	No
...	...	...	...	...	...	...	...
5512	6840-RESVB	DSL	Yes	No	Yes	Yes	Yes

	customer_id	internet_service	online_security	online_backup	device_protection	tech_support	streaming_tv
5513	2234-XADUH	Fiber optic	No	Yes	Yes	No	Yes
5514	4801-JZAZL	DSL	Yes	No	No	No	No
5515	8361-LTMKD	Fiber optic	No	No	No	No	No
5516	3186-AJIEK	Fiber optic	Yes	No	Yes	Yes	Yes

5517 rows × 8 columns

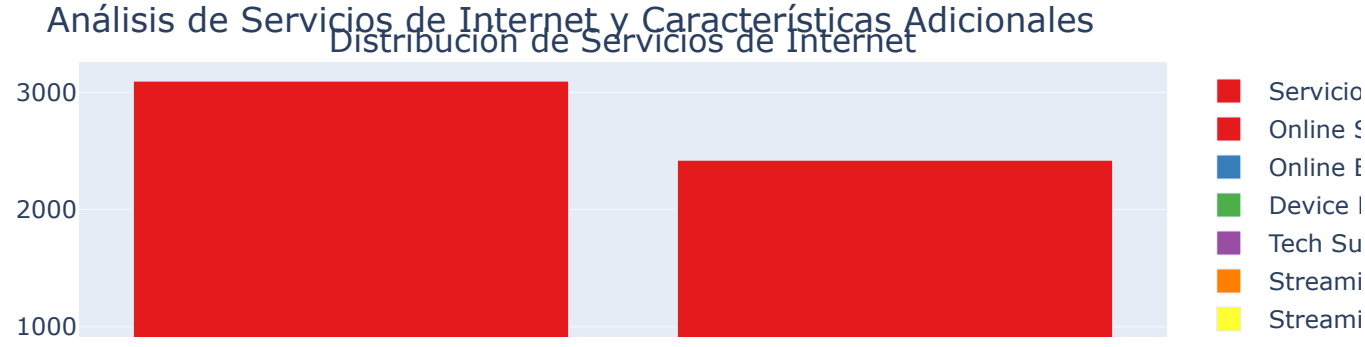
```
df_internet.nunique()
```

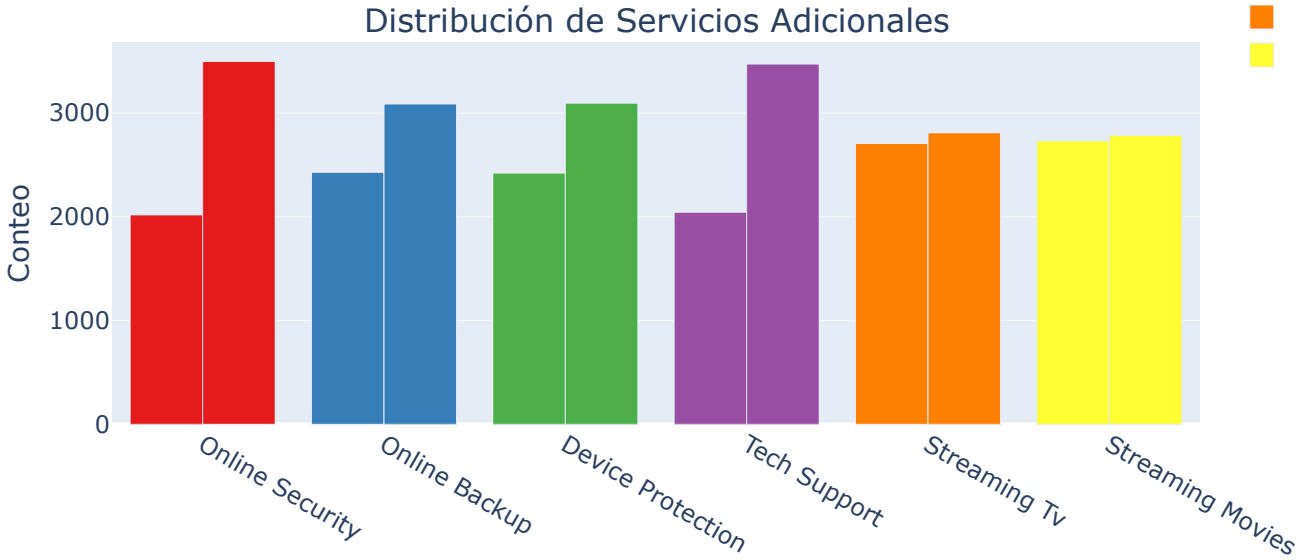
```
customer_id          5517
internet_service        2
online_security         2
online_backup           2
device_protection       2
tech_support            2
streaming_tv            2
streaming_movies        2
dtype: int64
```

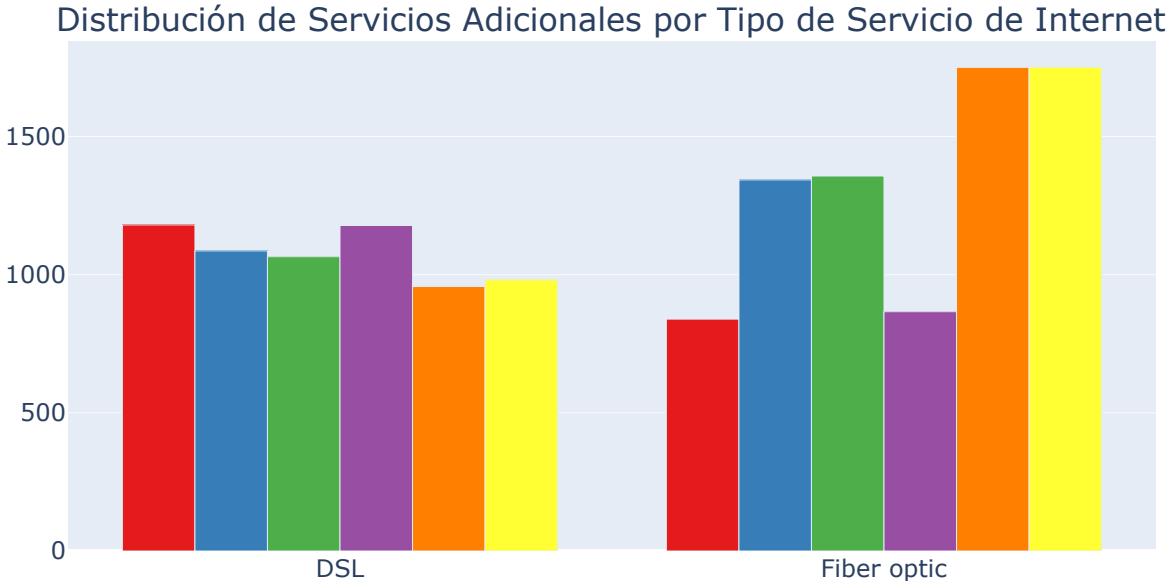
Se observa que principalmente se contrata el servicio de fribra optica

```
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots

# Crear el DataFrame para contar las características por tipo de servicio de internet
df_counts = pd.DataFrame()
services = ['online_security', 'online_backup', 'device_protection', 'tech_support', 'streaming_t

for service in services:
    service_counts = df_internet.groupby(['internet_service', service]).size().reset_index(name='
    df_counts = pd.concat([df_counts, service_counts], axis=0)

# Crear los subplots
fig = make_subplots(rows=3, cols=1,
                    subplot_titles=('Distribución de Servicios de Internet',
                                    'Distribución de Servicios Adicionales',
                                    'Distribución de Servicios Adicionales por Tipo de Servicio d
                    row_heights=[0.3, 0.3, 0.4])

# Gráfico 1: Distribución de Servicios de Internet
internet_service_counts = df_internet['internet_service'].value_counts()
fig.add_trace(go.Bar(
    x=internet_service_counts.index,
```

```
    y=internet_service_counts.values,
    name='Servicios de Internet',
    marker_color=px.colors.qualitative.Set1[0]
), row=1, col=1)

# Gráfico 2: Distribución de Características Adicionales agrupadas por servicio
for response in ['Yes', 'No']:
    for service in services:
        service_counts = df_internet[df_internet[service] == response][service].value_counts()
        fig.add_trace(go.Bar(
            x=[service.replace("_", " ").title()],
            y=service_counts.values,
            name=f'{service.replace("_", " ").title()} ({response})',
            marker_color=px.colors.qualitative.Set1[services.index(service)],
            offsetgroup=response  # Agrupa "Yes" y "No" juntas
        ), row=2, col=1)

# Cambiar el modo de barras a 'group' para no apilar en el segundo gráfico
fig.update_xaxes(type='category', row=2, col=1)
fig.update_yaxes(title_text="Conteo", row=2, col=1)
fig.update_layout(barmode='group')  # Configura el modo de barras a 'group' para este gráfico

# Gráfico 3: Distribución de Servicios Adicionales por Tipo de Servicio de Internet
for service in services:
    service_data = df_counts[df_counts[service] == 'Yes']
    fig.add_trace(go.Bar(
        x=service_data['internet_service'],
        y=service_data['count'],
        name=service.replace("_", " ").title(),
        marker_color=px.colors.qualitative.Set1[services.index(service)]
    ), row=3, col=1)

# Actualizar la disposición de los gráficos y aumentar la altura
fig.update_layout(
    title_text='Análisis de Servicios de Internet y Características Adicionales',
    showlegend=True,
    height=1000  # Ajusta esta altura según tus necesidades
)

fig.show()
```







En estos 3 graficos podemos ver infomación valiosa, en el primero se puede observar que hay una preferencia de contratos de internet para la fibra optica

En el segundo grafico se observa que para seguridad online, apoyo online y protección del hardware es más usual que no se contraten, sin embargo para servicios de streaming de tv y peliculas hay un igualdad entre los que si piden y los que no

En el ultimo grafico se suele pedir bastante la seguridad online y bastante parecido para los demás servicios, en la fibra ooptoca los servicios de streaming de tv y peliculas son los mas solicitados y en segundo lugarla protección del hardware y el apoyo online

```
# personal.csv

df_personal = pd.read_csv('datasets/personal.csv')
```

```
df_personal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   customerID     7043 non-null   object
 1   gender         7043 non-null   object
 2   SeniorCitizen  7043 non-null   int64 
 3   Partner        7043 non-null   object
 4   Dependents     7043 non-null   object
dtypes: int64(1), object(4)
memory usage: 275.2+ KB
```

```
df_personal.nunique()
```

```
customerID       7043
gender              2
SeniorCitizen       2
Partner             2
Dependents          2
dtype: int64
```

Viendo el dataset, son datos personales se hara un cambio de nombre de columnas y algunas visualizaciones de los datos

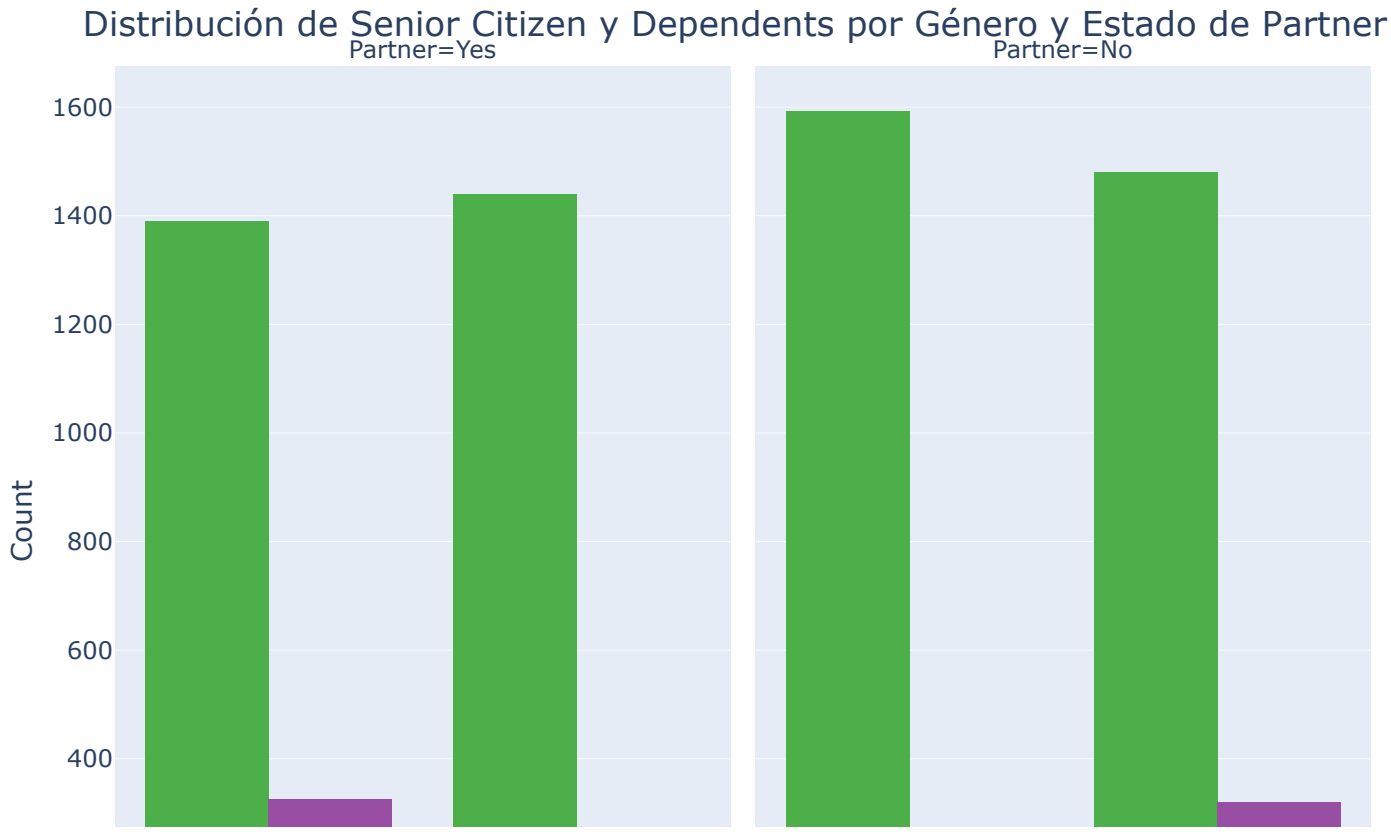
```
df_personal = ColumName_change(df_personal)
df_personal
```

	customer_id	gender	senior_citizen	partner	dependents
0	7590-VHVEG	Female	0	Yes	No
1	5575-GNVDE	Male	0	No	No
2	3668-QPYBK	Male	0	No	No
3	7795-CFOCW	Male	0	No	No
4	9237-HQITU	Female	0	No	No
...	...	...	...	...	...

	customer_id	gender	senior_citizen	partner	dependents
7038	6840-RESVB	Male	0	Yes	Yes
7039	2234-XADUH	Female	0	Yes	Yes
7040	4801-JZAZL	Female	0	Yes	Yes
7041	8361-LTMKD	Male	1	Yes	No
7042	3186-AJIEK	Male	0	No	No

7043 rows × 5 columns

```
# Crear un gráfico de barras apiladas
fig = px.histogram(df_personal,
                   x='gender',
                   color='senior_citizen',
                   barmode='group',
                   category_orders={"gender": ["Male", "Female"], "senior_citizen": ["0", "1"]},
                   facet_col='partner',
                   title='Distribución de Senior Citizen y Dependents por Género y Estado de Partı
                   labels={'senior_citizen': 'Senior Citizen', 'dependents': 'Dependents', 'partn
                   color_discrete_sequence=px.colors.qualitative.Set1)

fig.update_layout(barmode='group',
                  xaxis_title='Gender',
                  yaxis_title='Count')

fig.show()
```



La mayoría de los clientes no son senior citizens (`Senior Citizen = 0`), sin importar género o estado de pareja.

El estado de pareja no influye significativamente en ser o no un senior citizen.

No hay diferencias notables entre géneros en la proporción de senior citizens.

Posible menor cantidad de dependents entre senior citizens, pero requiere análisis adicional.

```
# phone.csv

df_phone = pd.read_csv('datasets/phone.csv')
```

```
df_phone.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6361 entries, 0 to 6360
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   customerID     6361 non-null   object
 1   MultipleLines  6361 non-null   object
dtypes: object(2)
memory usage: 99.5+ KB
```

Es un simple df que contiene si los clientes cuentan con lineas multiples y su id

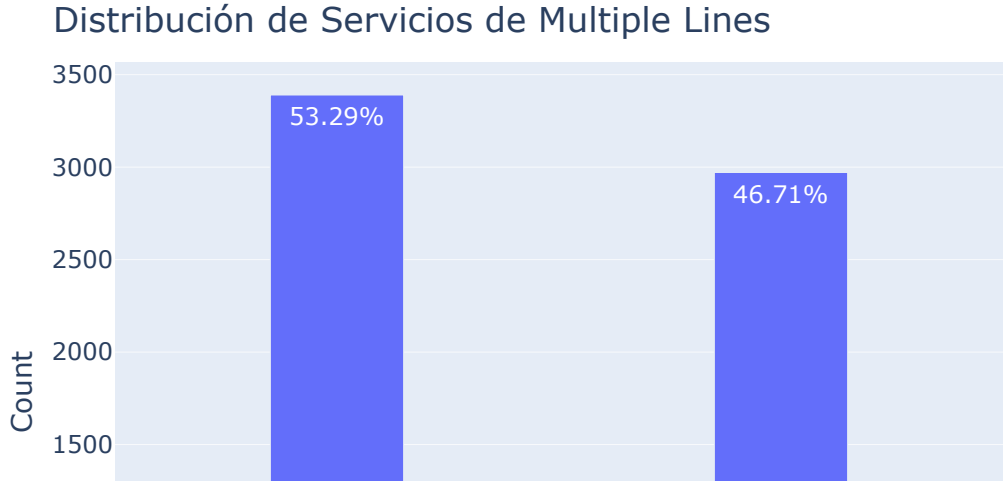
```
df_phone = ColumName_change(df_phone)

df_phone
```

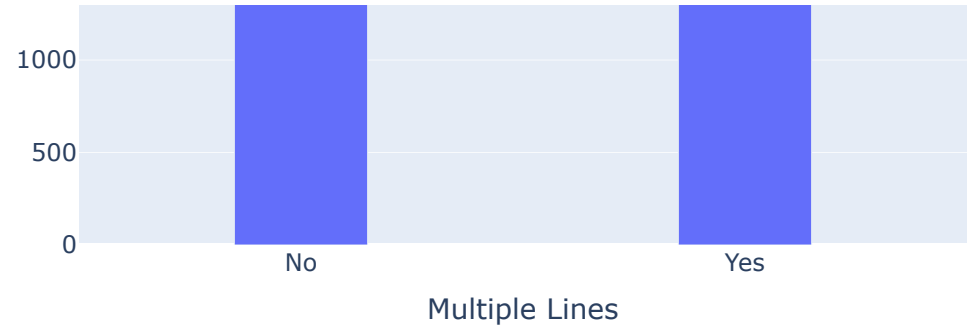
	customer_id	multiple_lines
0	5575-GNVDE	No
1	3668-QPYBK	No
2	9237-HQITU	No
3	9305-CDSKC	Yes
4	1452-KIOVK	Yes
...	...	...

	customer_id	multiple_lines
6356	2569-WGERO	No
6357	6840-RESVB	Yes
6358	2234-XADUH	Yes
6359	8361-LTMKD	Yes
6360	3186-AJIEK	No

6361 rows × 2 columns

```
# Contar la distribución de los servicios de "multiple_lines"
multiple_lines_service_counts = df_phone['multiple_lines'].value_counts()

# Calcular los porcentajes
percentages = (multiple_lines_service_counts / multiple_lines_service_counts.sum()) * 100

# Crear el gráfico con barras más delgadas y un tamaño ajustado
fig = px.bar(
    x=multiple_lines_service_counts.index,
    y=multiple_lines_service_counts.values,
    labels={'x': 'Multiple Lines', 'y': 'Count'},
    title='Distribución de Servicios de Multiple Lines'
)

# Agregar los porcentajes como texto en las barras
fig.update_traces(
    text=percentages.round(2).astype(str) + '%',
    textposition='auto',
    width=0.3  # Hacer las barras más delgadas
)

# Ajustar el tamaño del gráfico para hacerlo menos horizontal
fig.update_layout(width=500, height=400)

# Mostrar el gráfico
fig.show()
```





Se observa en el grafico que la mayoría no cuenta con multiples lineas aunque no por mucho, el 53% no tiene y el 46.7% si tiene

3. Preprocesamiento de Datos

Para esta sección preparare el dataset final para poder entrenar el modelo, tomare las variables mas importantes de cada dataset y usare ingeniería de caracteristicas para datos categoricos, datos numericos, para obtener un dataset limpio que se dividira en entrenamiento y validación para comenzar a entrenar los modelos.

```
# Crear un diccionario para almacenar los nombres de las columnas
data = {
    'df_contract': df_contract.columns,
    'df_internet': df_internet.columns,
    'df_phone': df_phone.columns,
    'df_personal': df_personal.columns
}

# Convertir el diccionario en un DataFrame para visualizarlo como una tabla
columns_df = pd.DataFrame.from_dict(data, orient='index').transpose()

# Mostrar la tabla
columns_df
```

	df_contract	df_internet	df_phone	df_personal
0	customer_id	customer_id	customer_id	customer_id
1	begin_date	internet_service	multiple_lines	gender
2	end_date	online_security	None	senior_citizen
3	type	online_backup	None	partner
4	paperless_billing	device_protection	None	dependents
5	payment_method	tech_support	None	None
6	monthly_charges	streaming_tv	None	None
7	total_charges	streaming_movies	None	None

- En el dataset **contract**, las columnas importantes son:

begin_date, end_date(*target*), paperless_billing, type, paymenth_method, monthly_charge, total_charge

- En el dataset **internet**, las columnas importantes son:

  internet_service, online_security, online_backup device_protection, tech_support, streaming_tv, streaming_movies

- En el dataset **phone**, las columnas importantes son:

  multiple_lines

- En el dataset **personal**, las columnas importantes son:

  senior_citizen, partner, dependents

El Genero no lo inclui por un tema de que en el analisis del dataset personal no fue muy relevante costumer_id servira para cuando se necsiten unir los datasets pero no para un fin predictivo.

Construcción del dataset para los modelos

```
from functools import reduce

df_personal_model = df_personal.drop('gender', axis=1)
# Lista de DataFrames
dfs = [df_contract, df_internet, df_phone, df_personal_model]

# Realizar merge en cadena usando 'outer' para no perder filas
df_merged = reduce(lambda left, right: pd.merge(left, right, on='customer_id', how='outer'), dfs)
df_merged
```

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charg
0	0002-ORFBO	2019-05-01	No	one year	Yes	Mailed check	65.60	593.30
1	0003-MKNFE	2019-05-01	No	month-to-month	No	Mailed check	59.90	542.40
2	0004-TLHLJ	2019-09-01	2020-01-01 00:00:00	month-to-month	Yes	Electronic check	73.90	280.85
3	0011-IGKFF	2018-12-01	2020-01-01 00:00:00	month-to-month	Yes	Electronic check	98.00	1237.85
4	0013-EXCHZ	2019-09-01	2019-12-01 00:00:00	month-to-month	Yes	Mailed check	83.90	267.40
...	...	...	...	...	...	...	...	...

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charg
7038	9987-LUTYD	2019-01-01	No	one year	No	Mailed check	55.15	742.90
7039	9992-RRAMN	2018-02-01	2019-12-01 00:00:00	month-to-month	Yes	Electronic check	85.10	1873.70
7040	9992-UJOEL	2019-12-01	No	month-to-month	Yes	Mailed check	50.30	92.75
7041	9993-LHIEB	2014-07-01	No	two year	No	Mailed check	67.85	4627.65
7042	9995-HOTOH	2014-11-01	No	two year	No	Electronic check	59.00	3707.60

7043 rows × 19 columns

```
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 19 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   customer_id        7043 non-null   object        
 1   begin_date         7043 non-null   datetime64[ns]
 2   end_date           7043 non-null   object        
 3   type               7043 non-null   object        
 4   paperless_billing  7043 non-null   object        
 5   payment_method     7043 non-null   object        
 6   monthly_charges    7043 non-null   float64       
 7   total_charges      7043 non-null   float64       
 8   internet_service   5517 non-null   object        
 9   online_security    5517 non-null   object        
 10  online_backup      5517 non-null   object        
 11  device_protection  5517 non-null   object        
 12  tech_support       5517 non-null   object        
 13  streaming_tv       5517 non-null   object        
 14  streaming_movies   5517 non-null   object        
 15  multiple_lines     6361 non-null   object        
 16  senior_citizen     7043 non-null   int64         
 17  partner            7043 non-null   object        
 18  dependents         7043 non-null   object        
dtypes: datetime64[ns](1), float64(2), int64(1), object(15)
memory usage: 1.0+ MB
```

Vamos a rellenar las columnas primeramente mulltiple_lines y posteriormente con las demás

```
columns_full = ['internet_service', 'online_security', 'online_backup', 'device_protection', 'tecl
```

```
for df in dfs:
    print(df.shape)
    print(df['customer_id'].nunique())
```

```
(7043, 8)
7043
(5517, 8)
5517
(6361, 2)
6361
(7043, 4)
7043
```

Vemos que solo df_personal y df_contract son los unicos datasets con todos los ids rellenare los datos faltamtes con 'unknown' para pasarlos por labelencoder posteriormente a las columnas categoricas y finalmente aplicar un StandarScaler

```
df_merged[columns_full] = df_merged[columns_full].fillna('unknown')
df_merged
```

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charg
0	0002-ORFBO	2019-05-01	No	one year	Yes	Mailed check	65.60	593.30
1	0003-MKNFE	2019-05-01	No	month-to-month	No	Mailed check	59.90	542.40
2	0004-TLHLJ	2019-09-01	2020-01-01 00:00:00	month-to-month	Yes	Electronic check	73.90	280.85
3	0011-IGKFF	2018-12-01	2020-01-01 00:00:00	month-to-month	Yes	Electronic check	98.00	1237.85
4	0013-EXCHZ	2019-09-01	2019-12-01 00:00:00	month-to-month	Yes	Mailed check	83.90	267.40
...	...	...	...	...	...	...	...	...
7038	9987-LUTYD	2019-01-01	No	one year	No	Mailed check	55.15	742.90
7039	9992-RRAMN	2018-02-01	2019-12-01 00:00:00	month-to-month	Yes	Electronic check	85.10	1873.70

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charg
7040	9992-UJOEL	2019-12-01	No	month-to-month	Yes	Mailed check	50.30	92.75
7041	9993-LHIEB	2014-07-01	No	two year	No	Mailed check	67.85	4627.65
7042	9995-HOTOH	2014-11-01	No	two year	No	Electronic check	59.00	3707.60

7043 rows × 19 columns

```
df_merged.nunique()
```

```
customer_id          7043
begin_date             77
end_date                5
type                    3
paperless_billing       2
payment_method          4
monthly_charges      1585
total_charges        6541
internet_service        3
online_security         3
online_backup           3
device_protection       3
tech_support            3
streaming_tv            3
streaming_movies        3
multiple_lines          3
senior_citizen          2
partner                 2
dependents              2
dtype: int64
```

```
fecha_actual = df_merged['begin_date'].max()
print(f"La fecha más reciente en el dataset es: {fecha_actual}")
```

```
La fecha más reciente en el dataset es: 2020-02-01 00:00:00
```

```
# Agrego el tiempo en días que ha estado el cleinte desde el inicio de su contrato
df_merged['time_in_company'] = (fecha_actual - df_merged['begin_date']).dt.days
df_merged.describe()
```

	begin_date	monthly_charges	total_charges	senior_citizen	time_in_company
count	7043	7043.000000	7043.000000	7043.000000	7043.000000

	begin_date	monthly_charges	total_charges	senior_citizen	time_in_company
mean	2017-04-30 13:01:50.918642688	64.761692	2281.253259	0.162147	1006.457050
min	2013-10-01 00:00:00	18.250000	18.800000	0.000000	0.000000
25%	2015-06-01 00:00:00	35.500000	401.900000	0.000000	306.000000
50%	2017-09-01 00:00:00	70.350000	1396.250000	0.000000	883.000000
75%	2019-04-01 00:00:00	89.850000	3786.600000	0.000000	1706.000000
max	2020-02-01 00:00:00	118.750000	8684.800000	1.000000	2314.000000
std	NaN	30.090047	2265.703526	0.368612	736.596428

De media vemos que los clientes tienen 1000 días en la compañía

De media los cargo totales son de $2281.25

De media los cargos mensuales son de $64.76

```
df_merged['begin_year'] = df_merged['begin_date'].dt.year
df_merged['begin_month'] = df_merged['begin_date'].dt.month
df_merged['begin_day'] = df_merged['begin_date'].dt.day
```

```
df_merged['end_date'] = np.where(df_merged['end_date'] == 'No', 0, 1)
```

```
print(list(df_merged.columns))
```

```
['customer_id', 'begin_date', 'end_date', 'type', 'paperless_billing', 'payment_method',
'monthly_charges', 'total_charges', 'internet_service', 'online_security', 'online_backup',
'device_protection', 'tech_support', 'streaming_tv', 'streaming_movies', 'multiple_lines',
'senior_citizen', 'partner', 'dependents', 'time_in_company', 'begin_year', 'begin_month',
'begin_day']
```

Eliminamos las columnas que ya no son necesarias

```
df_merged.head()
```

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charges
0	0002-ORFBO	2019-05-01	0	one year	Yes	Mailed check	65.6	593.30
1	0003-MKNFE	2019-05-01	0	month-to-month	No	Mailed check	59.9	542.40
2	0004-TLHLJ	2019-09-01	1	month-to-month	Yes	Electronic check	73.9	280.85
3	0011-IGKFF	2018-12-01	1	month-to-	Yes	Electronic check	98.0	1237.85

	customer_id	begin_date	end_date	type	paperless_billing	payment_method	monthly_charges	total_charges
				month				
4	0013-EXCHZ	2019-09-01	1	month-to-month	Yes	Mailed check	83.9	267.40

5 rows × 23 columns

```
# Elimino begin_day puesgto que todos son 1, entonces no tiene mucha importancia
df_merged = df_merged.drop(['customer_id', 'begin_date', 'begin_day'], axis=1)
df_merged
```

	end_date	type	paperless_billing	payment_method	monthly_charges	total_charges	internet_service	onlin
0	0	one year	Yes	Mailed check	65.60	593.30	DSL	No
1	0	month-to-month	No	Mailed check	59.90	542.40	DSL	No
2	1	month-to-month	Yes	Electronic check	73.90	280.85	Fiber optic	No
3	1	month-to-month	Yes	Electronic check	98.00	1237.85	Fiber optic	No
4	1	month-to-month	Yes	Mailed check	83.90	267.40	Fiber optic	No
...	...	...	...	...	...	...	...	...
7038	0	one year	No	Mailed check	55.15	742.90	DSL	Yes
7039	1	month-to-month	Yes	Electronic check	85.10	1873.70	Fiber optic	No
7040	0	month-to-month	Yes	Mailed check	50.30	92.75	DSL	No
7041	0	two year	No	Mailed check	67.85	4627.65	DSL	Yes
7042	0	two year	No	Electronic check	59.00	3707.60	DSL	Yes

7043 rows × 20 columns

```
features = df_merged.drop('end_date', axis=1)
targets = df_merged['end_date']

features_train, features_valid, targets_train, targets_valid = train_test_split(features, targets,
```

```
cat_cols = ['type', 'paperless_billing', 'payment_method', 'internet_service', 'online_security',
            'online_backup', 'device_protection', 'tech_support', 'streaming_tv', 'streaming_movi
            'multiple_lines', 'partner', 'dependents', 'begin_year', 'begin_month']


num_cols = ['monthly_charges', 'total_charges', 'time_in_company', 'senior_citizen']
```

```
# Haciendo un OneHotEncoder a las columnas categoricas

# Preprocesamiento para características categóricas
categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(drop='first', handle_unknown='ignore', sparse_output=False))])

# Preprocesamiento para características numéricas
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())])

# Combinación de transformadores
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, num_cols),
        ('cat', categorical_transformer, cat_cols)])
```

```
df_models = df_merged.drop('end_date', axis=1)

df_models= preprocessor.fit_transform(df_models)
```

```
print(df_models.shape)
```

```
(7043, 46)
```

Creado el pipeline nos queda un array de 7043 entradas y 45 columnas tomando begin_month y begin_year como categoricas haciendoles un Ohe

4. Modelado

Para esta tarea usaremos los siguientes modelos

Clasificación: Random Forest, XGBoost, Logistic Regression, Redes Neuronales.

```
# Transformacion de caracteristicas

features_train = preprocessor.fit_transform(features_train)
features_valid = preprocessor.transform(features_valid)
```

```
features_train.shape
```

```
(5282, 46)
```

```
# Regresión Logistica

log_reg = LogisticRegression()

param_grid = {
    'C': [0.01, 0.1, 1],
    'solver': ['liblinear', 'saga', 'lbfgs'],
    'max_iter': [100, 200, 300]
}



grid_search = GridSearchCV(
    estimator=log_reg,
    param_grid=param_grid,
    cv=5,
    scoring='roc_auc'
)

grid_search.fit(features_train, targets_train)

best_model = grid_search.best_estimator_


probabilities = best_model.predict_proba(features_valid)[:, 1]


predictions = best_model.predict(features_valid)


print(f"Mejores parámetros encontrados: {grid_search.best_params_}")

# Mejor puntuación en la búsqueda en rejilla
print(f"Mejor puntuación: {grid_search.best_score_:.4f}")
print()
```

```
# Calcular AUC-ROC usando las probabilidades
auc_roc = roc_auc_score(targets_valid, probabilities)
print(f'AUC-ROC: {auc_roc:.2f}')

# Calcular Accuracy usando las predicciones binarias
accuracy = accuracy_score(targets_valid, predictions)
print(f'Accuracy: {accuracy:.2f}')
```

```
C:\Users\evolu\AppData\Roaming\Python\Python312\site-packages\sklearn\linear_model\_sag.py:349:
ConvergenceWarning:

The max_iter was reached which means the coef_ did not converge

C:\Users\evolu\AppData\Roaming\Python\Python312\site-packages\sklearn\linear_model\_sag.py:349:
ConvergenceWarning:

The max_iter was reached which means the coef_ did not converge

Mejores parámetros encontrados: {'C': 1, 'max_iter': 100, 'solver': 'lbfgs'}
Mejor puntuación: 0.8529

AUC-ROC: 0.87
Accuracy: 0.81
```

```
# Random forest


Rand_forest_class = RandomForestClassifier(random_state=12345)

param_grid = {
    'n_estimators': [300, 500],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 4],
}


grid_search = GridSearchCV(
    estimator=Rand_forest_class,
    param_grid=param_grid,
    cv=5,
    n_jobs=-2,
    scoring='roc_auc'
)

grid_search.fit(features_train, targets_train)

best_model = grid_search.best_estimator_
```

```
probabilities = best_model.predict_proba(features_valid)[:, 1]


predictions = best_model.predict(features_valid)


print(f"Mejores parámetros encontrados: {grid_search.best_params_}")

# Mejor puntuación en la búsqueda en rejilla
print(f"Mejor puntuación: {grid_search.best_score_:.4f}")
print()

# Calcular AUC-ROC usando las probabilidades
auc_roc = roc_auc_score(targets_valid, probabilities)
print(f'AUC-ROC: {auc_roc:.2f}')

# Calcular Accuracy usando las predicciones binarias
accuracy = accuracy_score(targets_valid, predictions)
print(f'Accuracy: {accuracy:.2f}')
```

```
Mejores parámetros encontrados: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 500}
Mejor puntuación: 0.8886

AUC-ROC: 0.89
Accuracy: 0.86
```

```
# XGBoost


Gran_Boost_Class = GradientBoostingClassifier(random_state=12345)

param_grid = {
    'n_estimators': [300, 500],
    'max_depth': [6, 10],
    'learning_rate': [0.01, 0.1],
    'subsample': [0.7, 0.8, 1.0]
}


grid_search = GridSearchCV(
    estimator=Gran_Boost_Class,
    param_grid=param_grid,
    cv=5,
    n_jobs=-2,
    scoring='roc_auc'
)

grid_search.fit(features_train, targets_train)
```

```
best_model = grid_search.best_estimator_


probabilities = best_model.predict_proba(features_valid)[:, 1]


predictions = best_model.predict(features_valid)


print(f"Mejores parámetros encontrados: {grid_search.best_params_}")

# Mejor puntuación en la búsqueda en rejilla
print(f"Mejor puntuación: {grid_search.best_score_:.4f}")
print()

# Calcular AUC-ROC usando las probabilidades
auc_roc = roc_auc_score(targets_valid, probabilities)
print(f'AUC-ROC: {auc_roc:.2f}')

# Calcular Accuracy usando las predicciones binarias
accuracy = accuracy_score(targets_valid, predictions)
print(f'Accuracy: {accuracy:.2f}')
```

```
Mejores parámetros encontrados: {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 500,
'subsample': 0.8}
Mejor puntuación: 0.9272

AUC-ROC: 0.94
Accuracy: 0.90
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import AdamW
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import roc_auc_score, accuracy_score

# Crear el modelo
model = Sequential()

# Mantener la complejidad pero con ajustes en Dropout y regularización
model.add(Dense(units=512, activation='relu', input_dim=features_train.shape[1], kernel_regulariz
model.add(BatchNormalization())
model.add(Dropout(0.5))  # Aumentar Dropout

model.add(Dense(units=256, activation='relu', kernel_regularizer='l2'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(units=128, activation='relu', kernel_regularizer='l2'))
```

```
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(units=64, activation='relu', kernel_regularizer='l2'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Capa de salida para clasificación binaria
model.add(Dense(units=1, activation='sigmoid'))

# Compilar el modelo con una tasa de aprendizaje ajustada
optimizer = AdamW(learning_rate=0.0001)
model.compile(
    loss='binary_crossentropy',
    optimizer=optimizer,
    metrics=['AUC']
)

# Mostrar el resumen del modelo
model.summary()

# Restaurar EarlyStopping y agregar ModelCheckpoint para guardar los mejores pesos
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

# Guardar los mejores pesos del modelo
checkpoint = ModelCheckpoint(
    filepath='best_model.weights.h5',  # Cambiar el nombre del archivo a .weights.h5
    monitor='val_loss',
    save_best_only=True,
    save_weights_only=True
)

# Entrenar el modelo con EarlyStopping y ModelCheckpoint
history = model.fit(
    features_train,
    targets_train,
    epochs=1000,                # Aumentar significativamente el número de épocas
    batch_size=64,              # Mantener el tamaño del batch
    validation_data=(features_valid, targets_valid),
    verbose=1,
    callbacks=[early_stopping, checkpoint]  # Aplicar callbacks
)

# Cargar los mejores pesos del modelo
model.load_weights('best_model.weights.h5')

# Obtener predicciones
```

```
probabilities = model.predict(features_valid).flatten()

# Convertir probabilidades a predicciones binarias
predictions = (probabilities > 0.5).astype(int)

# Calcular AUC-ROC usando las probabilidades
auc_roc = roc_auc_score(targets_valid, probabilities)
print(f'AUC-ROC: {auc_roc:.2f}')

# Calcular Accuracy usando las predicciones binarias
accuracy = accuracy_score(targets_valid, predictions)
print(f'Accuracy: {accuracy:.2f}')
```

```
C:\Users\evolu\AppData\Roaming\Python\Python312\site-packages\keras\src\layers\core\dense.py:87:
UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	24,064
batch_normalization (BatchNormalization)	(None, 512)	2,048
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
batch_normalization_1 (BatchNormalization)	(None, 256)	1,024
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32,896
batch_normalization_2 (BatchNormalization)	(None, 128)	512
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 64)	8,256
batch_normalization_3 (BatchNormalization)	(None, 64)	256
dropout_3 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 1)	65

Total params: 200,449 (783.00 KB)

Trainable params: 198,529 (775.50 KB)

```
 Non-trainable params: 1,920 (7.50 KB)

Epoch 1/1000
 1/83 ━━━━━━━━━━━━━━━━━━━━ 5:26 4s/step - AUC: 0.4219 - loss:
7.9854 8/83
━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - AUC: 0.4341 - loss: 7.9545
15/83 ━━━━━━━━━━━━━━━━━━━━ 0s
8ms/step - AUC: 0.4469 - loss:
7.923425/83
━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - AUC: 0.4650 - loss:
7.892536/83
━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - AUC: 0.4747 - loss:
7.870948/83
━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - AUC: 0.4831 - loss:
7.850062/83
━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - AUC: 0.4907 - loss:
7.829975/83
━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - AUC: 0.4970 - loss:
7.814083/83
━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - AUC: 0.5006 - loss: 7.8042 - val_AUC: 0.6682 - val_loss:
7.2493
Epoch 2/1000
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - AUC: 0.6752 - loss:
7.400514/83
━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - AUC: 0.6020 - loss: 7.4841
27/83 ━━━━━━━━━━━━━━━━━━━━ 0s
4ms/step - AUC: 0.6032 - loss:
7.488140/83
━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - AUC: 0.6040 - loss:
7.485953/83
━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - AUC: 0.6056 - loss:
7.479567/83
━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - AUC: 0.6090 - loss:
7.469480/83
━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - AUC: 0.6129 - loss:
7.459283/83
━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - AUC: 0.6139 - loss: 7.4563 - val_AUC: 0.7954 - val_loss:
7.0402
Epoch 3/1000
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - AUC: 0.5250 - loss:
7.469112/83
━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - AUC: 0.5956 - loss: 7.3738
21/83 ━━━━━━━━━━━━━━━━━━━━ 0s
6ms/step - AUC: 0.6096 - loss:
7.354235/83
━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - AUC: 0.6254 - loss:
7.326647/83
━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - AUC: 0.6327 - loss:
7.309059/83
━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - AUC: 0.6367 - loss:
7.296372/83
```

```
0.2801▯▯▯
```

```
0.3206⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰54/83
━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - AUC: 0.9654 - loss:
0.3178⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰68/83
━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - AUC: 0.9657 - loss:
0.3167⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰81/83
━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - AUC: 0.9655 - loss:
0.3166⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰83/83
━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - AUC: 0.9655 - loss: 0.3165 - val_AUC: 0.9168 - val_loss:
0.4176
 1/56 ━━━━━━━━━━━━━━━━━━━━ 5s 98ms/step⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰56/56
━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step ⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰56/56 ━━━━━━━━━━━━━━━━━━━━
0s 2ms/step
AUC-ROC: 0.92
Accuracy: 0.88
```

5. Resultados

Como mencione al principio de la sección de modelado, hice 4 modelos, uno de regresión logistica, otro de boosting por descenso de gradiente, un clasificador por bosques aleatorios y por ultimo una red neuronal.

El resultado fue exitoso, se sobrepaso el objetivo más alto que era conseguir un 'AUC-ROC' de más de 0.89 y en 2 modelos se pudo sobrepasar que fueron: **El Claificador por descenso de Gradiente** y **La red neuronal**

Recordemos los critrerios:

- **Métrica principal:** AUC-ROC.
- **Métrica adicional:** Exactitud.

Criterios de evaluación:

- **AUC-ROC < 0.75** — 0 SP
- **0.75 ≤ AUC-ROC < 0.81** — 4 SP
- **0.81 ≤ AUC-ROC < 0.85** — 4.5 SP
- **0.85 ≤ AUC-ROC < 0.87** — 5 SP
- **0.87 ≤ AUC-ROC < 0.88** — 5.5 SP
- **AUC-ROC ≥ 0.88** — 6 SP

Regresión Logística:

- **Mejores parámetros encontrados**: `{'C': 10, 'l1_ratio': 1.0, 'max_iter': 300, 'penalty': 'l1', 'solver': 'liblinear'}`
- **Mejor puntuación**: 0.8537
- **AUC-ROC**: 0.87
- **Accuracy**: 0.82

Clasificador por Bosque Aleatorio:

- **Mejores parámetros encontrados**: `{'bootstrap': True, 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 1000}`
- **Mejor puntuación**: 0.8903
- **AUC-ROC**: 0.89
- **Accuracy**: 0.85

Clasificador por Descenso de Gradiente:

- **Mejores parámetros encontrados**: `{'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 500, 'subsample': 0.8}`
- **Mejor puntuación**: 0.9276
- **AUC-ROC**: 0.94
- **Accuracy**: 0.90

Red Neuronal:

- **Arquitectura**: | **Layer (type)** | **Output Shape** | **Param #** | |———————————-|————————|————-| | **Dense (dense_75)** | (None, 512) | 23,552 | | **BatchNormalization (batch_51)**| (None, 512) | 2,048 | | **Dropout (dropout_51)** | (None, 512) | 0 | | **Dense (dense_76)** | (None, 256) | 131,328 | | **BatchNormalization (batch_52)**| (None, 256) | 1,024 | | **Dropout (dropout_52)** | (None, 256) | 0 | | **Dense (dense_77)** | (None, 128) | 32,896 | | **BatchNormalization (batch_53)**| (None, 128) | 512 | | **Dropout (dropout_53)** | (None, 128) | 0 | | **Dense (dense_78)** | (None, 64) | 8,256 | | **BatchNormalization (batch_54)**| (None, 64) | 256 | | **Dropout (dropout_54)** | (None, 64) | 0 | | **Dense (dense_79)** | (None, 1) | 65 |

- **Entrenamiento**:

  - **Epoch 1/1000**: AUC: 0.5244, loss: 7.8016, val_AUC: 0.7564, val_loss: 7.4223
  - **Epoch 2/1000**: AUC: 0.6461, loss: 7.4874, val_AUC: 0.7904, val_loss: 7.2717
  - **Epoch 3/1000**: AUC: 0.6852, loss: 7.2402, val_AUC: 0.8100, val_loss: 7.0595
  - **Epoch 4/1000**: AUC: 0.7044, loss: 7.0418, val_AUC: 0.8184, val_loss: 6.8342
  - **Epoch 5/1000**: AUC: 0.7000, loss: 6.8556, val_AUC: 0.8299, val_loss: 6.6247 ...
  - **Última Epoch**: AUC: 0.9708, loss: 0.2998, val_AUC: 0.9294, val_loss: 0.3969

- **Resultados**:

  - **AUC-ROC**: 0.93
  - **Accuracy**: 0.89

Conslusión

En este proyecto se realizo un modelo capaz de pedecir si un cliente ha terminado su contrato o no además de un analisis exploratorio de datos, se llegaron a las siguientes conclusiones:

- La mayor cantidad de contratos son de pagos mensuales de entre $18 - $22 dolares
- Existe una relación entre cantidad de cargos mensuales y totales de los cleintes
- Los métodos de pago por transferencia bancaria y tarjeta de crédito tienen cargos mensuales similares, el pago por cheque electrónico tiene cargos concentrados con una mediana alta ($80.55), y los cheques por correo muestran una gran dispersión, sugiriendo que se usan más para cargos bajos.
- Los clientes con contrato finalizado muestran cargos mensuales más altos y menos variados, con mayores diferencias en los cuartiles comparados con los clientes activos.
- Los clientes que han finalizado su contrato tienden a tener cargos totales más bajos y presentan más anomalías, sugiriendo que podrían ser clientes de corto plazo, mientras que los activos muestran cargos más altos, indicando posible satisfacción a largo plazo.
- La seguridad online es bastante solicitada, y la fibra óptica junto con los servicios de streaming son los más requeridos, seguidos por protección del hardware y apoyo online.

Conslusiones de los modelos:

En conclusión, el claro ganador es la red neuronal, no solo por su métrica AUC-ROC, sino porque supera significativamente al único competidor, que es el modelo de descenso de gradiente. La red neuronal se entrena mucho más rápido. Sin embargo, es importante mencionar que el modelo de descenso de gradiente también es muy bueno y ambos logran cumplir el umbral objetivo del proyecto.