Tutorial: Limpieza de Datos y Modelado de Churn para una Empresa de Telecomunicaciones

1. Configuración del Entorno

Primero, importamos las bibliotecas necesarias para trabajar con los datos y realizar análisis:
- `pandas` y `numpy` para la manipulación de datos.
- `matplotlib` y `seaborn` para la visualización.
- Varios módulos de `sklearn` para modelado y selección de características.

```
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', None)
import seaborn as sns
import matplotlib.pyplot as plt
import json
```

2. Carga y Preprocesamiento de Datos

2.1 Cargando los Datos

Primero, cargamos los datos desde un archivo JSON. En este caso, el dataset contiene información de clientes de telecomunicaciones.

```
datos_churn = pd.read_json("base_clientes.json")
datos_churn.head()
```

Cuidado, estos datos aún no están normalizados, para esto necesitarás aplicar `json_normalize` de la siguiente forma:

```
def lectura_datos():
  with open("base_clientes.json") as f:
    json_bruto = json.load(f)
  datos_churn = pd.json_normalize(json_bruto)
lectura_datos()
```

2.2 Tratamiento de Datos

El siguiente paso es procesar los datos y eliminar registros con valores faltantes o duplicados, convertir algunas columnas al tipo correcto, etc. A continuación todos los pasos a ser realizados:

- Definición de la función `preprocesamiento()`:
Se declara la función que realizará el preprocesamiento de datos.

- Uso de variable global:
Se declara que `datos_churn` es una variable global que se utilizará dentro de la función.

- Identificación de índices con valores vacíos en `cuenta.cobros.Total`:
Se obtienen los índices de las filas donde la columna `cuenta.cobros.Total` tiene un valor vacío.

- Cálculo de `cuenta.cobros.Total` para los índices identificados:
Se calcula el valor de `cuenta.cobros.Total` multiplicando `cuenta.cobros.mensual` por 24.

- Asignación del tiempo de servicio:
Se establece que `cliente.tiempo_servicio` para esos índices es 24.

- Conversión de tipo de datos:
La columna `cuenta.cobros.Total` se convierte al tipo `float`.

- Selección de columnas de tipo objeto:
Se identifican las columnas del DataFrame que son de tipo `object`.

- Reemplazo de valores vacíos con `NaN`:
Se reemplazan los valores vacíos en las columnas de tipo `object` con `NaN`.

- Eliminación de filas con valores `NaN`:
Se eliminan las filas que contienen `NaN` en las columnas de tipo `object`.

- Eliminación de duplicados:
Se eliminan filas duplicadas del DataFrame.

- Relleno de valores faltantes en `cliente.tiempo_servicio`:
Se rellenan los valores `NaN` en `cliente.tiempo_servicio` usando el cálculo basado en `cuenta.cobros.Total` dividido por `cuenta.cobros.mensual`.

- Identificación de columnas para eliminar nulos:
Se especifican las columnas `cuenta.contrato`, `cuenta.facturacion_electronica`, y `cuenta.metodo_pago` para eliminar nulos.

- Eliminación de filas con valores `NaN` en columnas específicas:
Se eliminan las filas que contienen `NaN` en las columnas especificadas.

- Reinicio del índice del DataFrame:
Se reinicia el índice del DataFrame, eliminando el índice anterior.

Algunos ejemplos de esta primera parte de los procesamientos:

```
def preprocesamiento():
  idx = datos_churn[datos_churn['cuenta.cobros.Total'] == ' '].index
  datos_churn.loc[idx, "cuenta.cobros.Total"] = datos_churn.loc[idx, "cuenta.cobros.
  datos_churn['cuenta.cobros.Total'] = datos_churn['cuenta.cobros.Total'].astype(flo
  # Eliminamos valores nulos y duplicados
  columnas_object = datos_churn.select_dtypes(include='object').columns
  datos_churn[columnas_object] = datos_churn[columnas_object].replace('', np.nan)
  datos_churn.dropna(subset=columnas_object, inplace=True)
  datos_churn.drop_duplicates(inplace=True)
```

- Cálculo del rango intercuartílico (IQR):
Se calculan los cuartiles Q1 y Q3 para la columna `cliente.tiempo_servicio`.

- Determinación de límites para detectar outliers:
Se establecen límites inferior y superior para identificar outliers.

- Identificación de outliers en `cliente.tiempo_servicio`:
Se identifican los índices dónde `cliente.tiempo_servicio` es menor que el límite inferior o mayor que el límite superior.

- Corrección de outliers:
Se recalcula `cliente.tiempo_servicio` para los outliers identificados.

- Cálculo del IQR nuevamente:
Se recalculan los cuartiles Q1 y Q3 para `cliente.tiempo_servicio`.

- Determinación de nuevos límites para detectar outliers:
Se recalculan los límites inferior y superior.

- Eliminación de outliers del DataFrame:
Se eliminan las filas identificadas como outliers de `cliente.tiempo_servicio`.

- Reinicio del índice final del DataFrame:
Se reinicia el índice del DataFrame una vez más después de la eliminación de outliers.

2.3 Normalización de Datos
Para preparar los datos, eliminamos columnas irrelevantes y convertimos valores categóricos en numéricos para poder llevarlos al modelo. A continuación todos los pasos a ser realizados:

- Definición de la función `normalizacion()`:
Se declara la función que realizará la normalización de datos.

- Uso de variable global:
Se declara que `datos_churn` es una variable global que se utilizará dentro de la función.

- Eliminación de la columna `id_cliente`:
Se elimina la columna `id_cliente` del DataFrame.

- Definición de un diccionario de mapeo:
Se crea un diccionario que asigna valores numéricos a ciertas categorías: 'no' a 0, 'si' a 1, 'masculino' a 0 y 'femenino' a 1.

- Especificación de columnas a normalizar:
Se definen las columnas que serán normalizadas: `telefono.servicio_telefono`, `Churn`, `cliente.pareja`, `cliente.dependientes`, `cuenta.facturacion_electronica`, y `cliente.genero`.

- Reemplazo de categorías por valores numéricos:
Se reemplazan las categorías en las columnas especificadas utilizando el diccionario de mapeo.

- Creación de variables dummy:
Se crean variables dummy para las columnas categóricas restantes del DataFrame y se reinicia el índice, eliminando el índice anterior.

Algunos ejemplos de esta primera parte de los procesamientos:

```
def normalizacion():
  datos_churn = datos_churn.drop('id_cliente', axis=1)
  mapping = {
      'no': 0,
      'si': 1,
      'masculino': 0,
      'femenino': 1
  }
  columnas = ['telefono.servicio_telefono', 'Churn', 'cliente.pareja'
  datos_churn[columnas] = datos_churn[columnas].replace(mapping)
```

3. Entrenamiento del Modelo con Random Forest

Separamos los datos en conjuntos de entrenamiento y prueba, luego entrenamos un modelo de Random Forest para predecir la variable objetivo `Churn`.

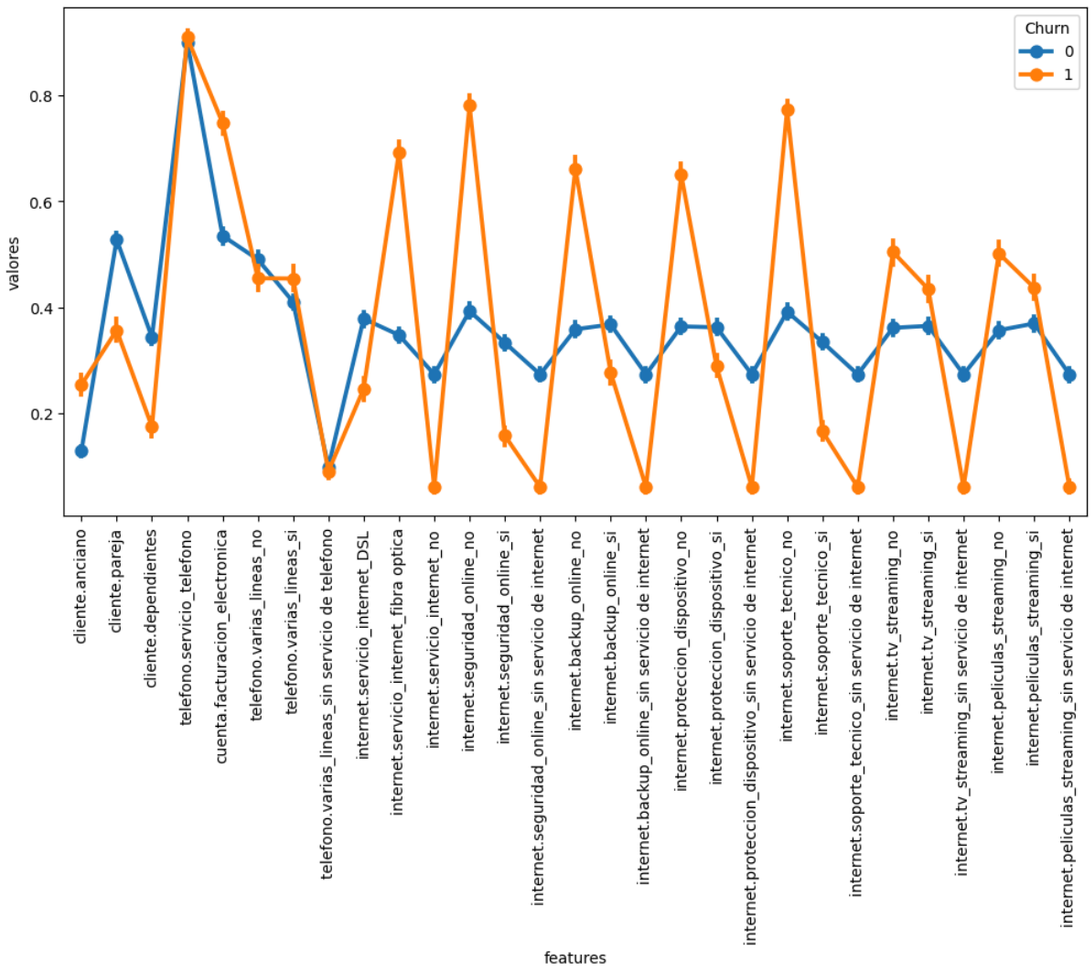
```
y = datos_churn['Churn']
x = datos_churn.drop(columns='Churn')
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.3, random_st

def pronosticar(train_x, train_y):
  model = RandomForestClassifier(random_state=50)
  model.fit(train_x, train_y)
  return model

model = pronosticar(train_x, train_y)
model.score(test_x, test_y)
```

4. Reducción de Dimensionalidad

4.1 Métodos Gráficos

Generamos diagramas de violín y de puntos para analizar la distribución de las características con respecto a `Churn`.

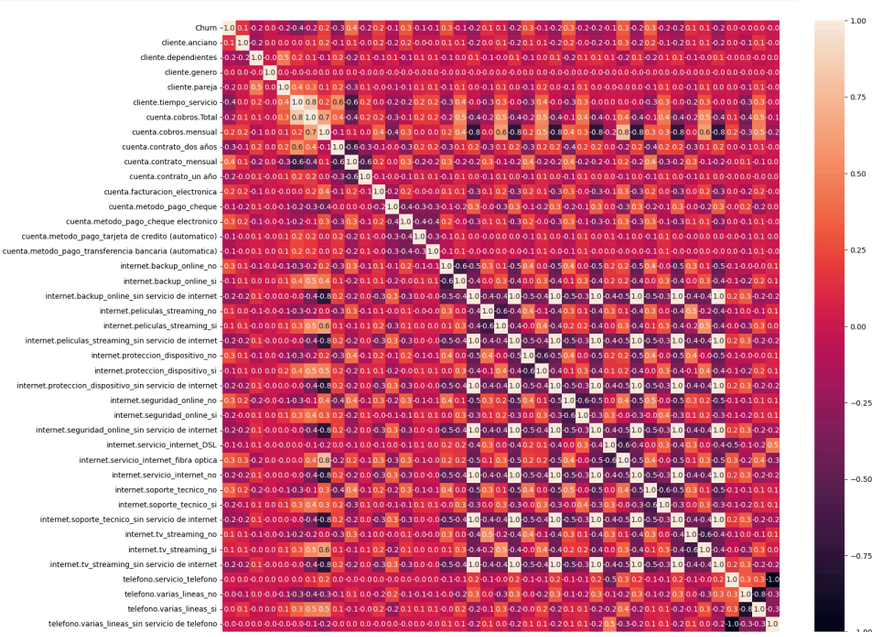
```
def diagrama_puntos(df, inicio, fin):
    df = df[df.select_dtypes(include='int').columns]
    y = df['Churn']
    x = df.drop(columns='Churn')
    df = pd.concat([y, x.iloc[:, inicio:fin]], axis=1)
    df_melted = pd.melt(df, id_vars="Churn", var_name="features", value_name='valores')
    plt.figure(figsize=(12, 6))
    sns.pointplot(x="features", y="valores", hue="Churn", data=df_melted)
    plt.xticks(rotation=90)
    plt.show()
```



4.2 Mapa de Calor

Creamos un mapa de calor para visualizar las correlaciones entre las variables y encontrar relaciones que podrían no ser evidentes.

```
def mapa_calor(df):
    grafico = df.corr()
    plt.figure(figsize=(17, 15))
    sns.heatmap(grafico, annot=True, fmt=".1f")
    return grafico
mapa_calor(datos_churn)
```



4.3 Feature Importances

Implementamos métodos de selección de características usando SelectKBest, RFE y PCA para reducir la dimensionalidad del conjunto de datos.

SelectKBest:

Este método selecciona las mejores características según su relevancia estadística usando la prueba **chi-cuadrado**. Es útil cuando se trabaja con datos categóricos y selecciona las características más relevantes para el modelo.

```
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.ensemble import RandomForestClassifier

def pronosticar_SelectKBest(train_x, test_x, train_y, test_y):
    selector = SelectKBest(score_func=chi2, k=10)
    train_x_selected = selector.fit_transform(train_x, train_y)
    test_x_selected = selector.transform(test_x)
    model = RandomForestClassifier()
    model.fit(train_x_selected, train_y)
    test_score = model.score(test_x_selected, test_y)
    print(f'Accuracy en prueba con SelectKBest y chi2: {test_score}')
```

RFE (Recursive Feature Elimination):

Este algoritmo selecciona características eliminando recursivamente las menos importantes según el modelo. Usa un clasificador (en este caso, **RandomForest**) para identificar y eliminar las características menos relevantes, dejando solo las más útiles.

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

def pronosticar_RFE(train_x, test_x, train_y, test_y):
    model = RandomForestClassifier()
    selector = RFE(estimator=model, n_features_to_select=10, step=1)
    train_x_selected = selector.fit_transform(train_x, train_y)
    test_x_selected = selector.transform(test_x)
    model.fit(train_x_selected, train_y)
    test_score = model.score(test_x_selected, test_y)
    print(f'Accuracy en prueba con RFE: {test_score}')
```

PCA (Principal Component Analysis):

Este método transforma las características originales en un conjunto de nuevas variables no correlacionadas llamadas **componentes principales**, que capturan la mayor parte de la variabilidad de los datos, reduciendo la dimensionalidad sin perder mucha información.

```
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier

def pronosticar_PCA(train_x, test_x, train_y, test_y):
    pca = PCA(n_components=10)
    train_x_pca = pca.fit_transform(train_x)
    test_x_pca = pca.transform(test_x)
    model = RandomForestClassifier()
    model.fit(train_x_pca, train_y)
    test_score = model.score(test_x_pca, test_y)
    print(f'Accuracy en prueba con PCA: {test_score}')
```

Con este tutorial, estarás preparado para desarrollar tu proyecto. ¡Mucha Suerte! 🚀