In [2]:
```
import os
current_directory = os.getcwd()
print("Current working directory:", current_directory)
```

```
Current working directory: C:\Users\Prashant Sharma\PycharmProjects
```

Exploratory Data Analysis Starter

Import packages

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

# Shows plots in jupyter notebook
%matplotlib inline

# Set plot style
sns.set(color_codes=True)
```

Loading data with Pandas

In [6]:
```
client_df = pd.read_csv('client_data.csv')
price_df = pd.read_csv('price_data.csv')
client_df.head(3)
```

Out[6]:

	id	channel_sales	cons_12m	cons_gas_12m	co
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	

3 rows × 26 columns

In [7]:
```
price_df.head(3)
```

Out[7]:

	id	price_date	price_off_peak_var	price_peak_var	price_mid_peak_
0	038af19179925da21a25619c5a24b745	2015-01-01	0.151367	0.0	
1	038af19179925da21a25619c5a24b745	2015-02-01	0.151367	0.0	
2	038af19179925da21a25619c5a24b745	2015-03-01	0.151367	0.0	

Descriptive statistics of data

Data types It is useful to first understand the data that you're dealing with along with the data types of each column. The data types may dictate how you transform and engineer features.

To get an overview of the data types within a data frame, use the info() method.

In [9]:

```
price_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193002 entries, 0 to 193001
Data columns (total 8 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   id                  193002 non-null  object
 1   price_date          193002 non-null  object
 2   price_off_peak_var  193002 non-null  float64
 3   price_peak_var      193002 non-null  float64
 4   price_mid_peak_var  193002 non-null  float64
 5   price_off_peak_fix  193002 non-null  float64
 6   price_peak_fix      193002 non-null  float64
 7   price_mid_peak_fix  193002 non-null  float64
dtypes: float64(6), object(2)
memory usage: 11.8+ MB
```

In [10]:

```
client_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14606 entries, 0 to 14605
Data columns (total 26 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              14606 non-null  object
 1   channel_sales                   14606 non-null  object
 2   cons_12m                        14606 non-null  int64
 3   cons_gas_12m                    14606 non-null  int64
 4   cons_last_month                 14606 non-null  int64
 5   date_activ                      14606 non-null  object
 6   date_end                        14606 non-null  object
 7   date_modif_prod                 14606 non-null  object
 8   date_renewal                    14606 non-null  object
 9   forecast_cons_12m               14606 non-null  float64
 10  forecast_cons_year              14606 non-null  int64
 11  forecast_discount_energy        14606 non-null  float64
 12  forecast_meter_rent_12m         14606 non-null  float64
 13  forecast_price_energy_off_peak  14606 non-null  float64
 14  forecast_price_energy_peak      14606 non-null  float64
 15  forecast_price_pow_off_peak     14606 non-null  float64
 16  has_gas                         14606 non-null  object
 17  imp_cons                        14606 non-null  float64
 18  margin_gross_pow_ele            14606 non-null  float64
 19  margin_net_pow_ele              14606 non-null  float64
 20  nb_prod_act                     14606 non-null  int64
 21  net_margin                      14606 non-null  float64
 22  num_years_antig                 14606 non-null  int64
 23  origin_up                       14606 non-null  object
 24  pow_max                         14606 non-null  float64
 25  churn                           14606 non-null  int64
dtypes: float64(11), int64(7), object(8)
memory usage: 2.9+ MB
```

Statistics Now let's look at some statistics about the datasets. We can do this by using the describe() method.

In [11]:

```
client_df.describe()
```

Out[11]:

	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_year	forecast
count	1.460600e+04	1.460600e+04	14606.000000	14606.000000	14606.000000	
mean	1.592203e+05	2.809238e+04	16090.269752	1868.614880	1399.762906	
std	5.734653e+05	1.629731e+05	64364.196422	2387.571531	3247.786255	
min	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	
25%	5.674750e+03	0.000000e+00	0.000000	494.995000	0.000000	
50%	1.411550e+04	0.000000e+00	792.500000	1112.875000	314.000000	
75%	4.076375e+04	0.000000e+00	3383.000000	2401.790000	1745.750000	
max	6.207104e+06	4.154590e+06	771203.000000	82902.830000	175375.000000	

In [12]:

```
price_df.describe()
```

Out[12]:

	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix	price_peak_fix	price
count	193002.000000	193002.000000	193002.000000	193002.000000	193002.000000	
mean	0.141027	0.054630	0.030496	43.334477	10.622875	
std	0.025032	0.049924	0.036298	5.410297	12.841895	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.125976	0.000000	0.000000	40.728885	0.000000	
50%	0.146033	0.085483	0.000000	44.266930	0.000000	
75%	0.151635	0.101673	0.072558	44.444710	24.339581	
max	0.280700	0.229788	0.114102	59.444710	36.490692	

Data visualization

If you're working in Python, two of the most popular packages for visualization are matplotlib and seaborn. We highly recommend you use these, or at least be familiar with them because they are ubiquitous!

Below are some functions that you can use to get started with visualizations.

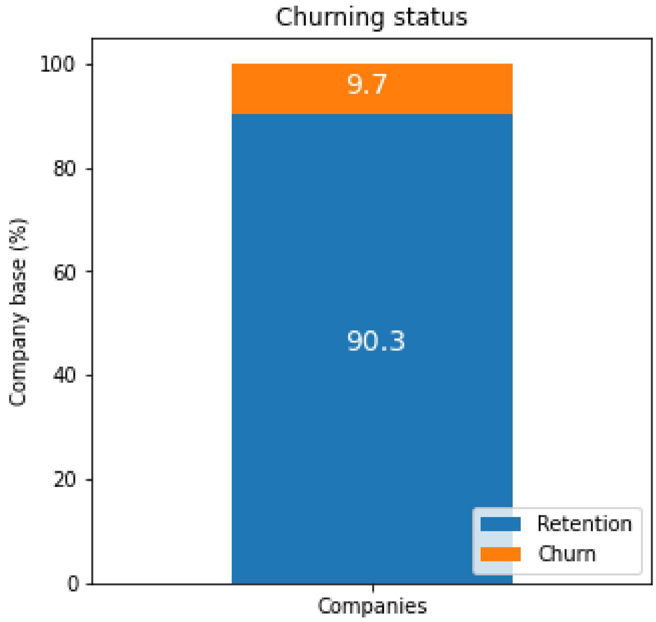
In [14]:

```
def plot_stacked_bars(dataframe, title_, size_=(18, 10), rot_=0, legend_="upper righ
    """
    Plot stacked bars with annotations
    """
    ax = dataframe.plot(
        kind="bar",
        stacked=True,
        figsize=size_,
        rot=rot_,
```

```
        title=title_)

    # Annotate bars
    annotate_stacked_bars(ax, textsize=14)
    # Rename legend
    plt.legend(["Retention", "Churn"], loc=legend_)
    # Labels
    plt.ylabel("Company base (%)")
    plt.show()

def annotate_stacked_bars(ax, pad=0.99, colour="white", textsize=13):
    """
    Add value annotations to the bars
    """

    # Iterate over the plotted rectanges/bars
    for p in ax.patches:

        # Calculate annotation
        value = str(round(p.get_height(),1))
        # If value is 0 do not annotate
        if value == '0.0':
            continue
        ax.annotate(
            value,
            ((p.get_x()+ p.get_width()/2)*pad-0.05, (p.get_y()+p.get_height()/2)*pad
            color=colour,
            size=textsize)

def plot_distribution(dataframe, column, ax, bins_=50):
    """
    Plot variable distirbution in a stacked histogram of churned or retained company
    """
    # Create a temporal dataframe with the data to be plot
    temp = pd.DataFrame({"Retention": dataframe[dataframe["churn"]==0][column],
    "Churn":dataframe[dataframe["churn"]==1][column]})
    # Plot the histogram
    temp[["Retention","Churn"]].plot(kind='hist', bins=bins_, ax=ax, stacked=True)
    # X-axis label
    ax.set_xlabel(column)
    # Change the x-axis to plain style
    ax.ticklabel_format(style='plain', axis='x')
```

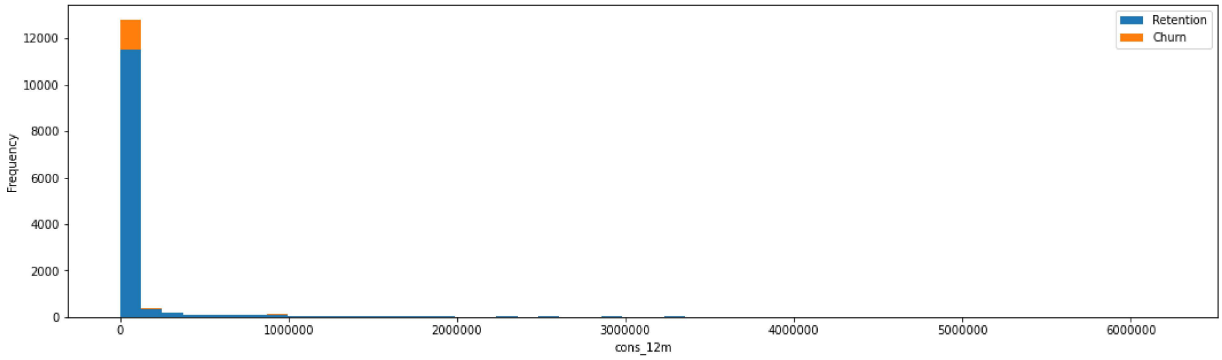
Thhe first function plot_stacked_bars is used to plot a stacked bar chart. An example of how you could use this is shown below:

In [15]:
```
churn = client_df[['id', 'churn']]
churn.columns = ['Companies', 'churn']
churn_total = churn.groupby(churn['churn']).count()
churn_percentage = churn_total / churn_total.sum() * 100
plot_stacked_bars(churn_percentage.transpose(), "Churning status", (5, 5), legend_="
```



The second function annotate_bars is used by the first function, but the third function plot_distribution helps you to plot the distribution of a numeric column. An example of how it can be used is given below:

In [16]:
```
consumption = client_df[['id', 'cons_12m', 'cons_gas_12m', 'cons_last_month', 'imp_c

fig, axs = plt.subplots(nrows=1, figsize=(18, 5))

plot_distribution(consumption, 'cons_12m', axs)
```



The process of building a predictive model for customer churn based on client data and pricing data. Below are the steps you can follow, along with Python code snippets for each step. We'll be using Python and popular libraries like Pandas, Scikit-Learn, and Matplotlib for data processing, modeling, and visualization.

Step 1: Data Preprocessing Load and preprocess your data. This may involve handling missing values, encoding categorical variables, and splitting the data into training and testing sets.

In [ ]:
```
# Import necessary libraries
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
```

```
# Load your client data and pricing data into a DataFrame
client_data = pd.read_csv('client_data.csv')
price_data = pd.read_csv('price_data.csv')

# Merge the datasets on a common key (e.g., customer ID)
merged_data = pd.merge(client_data, price_data, on='customer_id', how='inner')

# Encode categorical variables if needed (e.g., industry)
encoder = LabelEncoder()
merged_data['industry_encoded'] = encoder.fit_transform(merged_data['industry'])

# Split the data into training and testing sets
X = merged_data[['price', 'industry_encoded', 'contract_duration']]
y = merged_data['churn']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

# Standardize numerical features (price and contract_duration)
scaler = StandardScaler()
X_train[['price', 'contract_duration']] = scaler.fit_transform(X_train[['price', 'co
X_test[['price', 'contract_duration']] = scaler.transform(X_test[['price', 'contract

# Create and train a Logistic Regression model
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Generate a classification report
report = classification_report(y_test, y_pred)
print(report)

# Generate a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xticks([0, 1], ['Not Churn', 'Churn'])
plt.yticks([0, 1], ['Not Churn', 'Churn'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Name-Prashant Sharma email-iamprashant8008@gmail.com DATA Associate @NATIONAL AUSTRALIA BANK