PowerCo Customer Churn: EDA Report

Date: August 3, 2025
Author: Costas Pinto
Status: Initial Analysis Complete

Part 1: Project Setup and Configuration

The analysis was structured using an industry-grade methodology to ensure reproducibility and maintainability.

- **Configuration-Driven:**
  A centralized `Config` class was used to manage all parameters such as file paths and column names. This allows for easier updates and eliminates hardcoding.
- **Organized Outputs:**
  A directory structure (`/plots`, `/logs`) was established to systematically store generated artifacts like visualizations and logs.
- **Environment:**
  The analysis was conducted using the standard Python data science stack:
  `pandas`, `seaborn`, `matplotlib`.

In [1]:

```python
# --- Required Libraries ---
import logging
import os
from typing import List

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

# --- Configuration ---
class Config:
    """Centralized configuration for the EDA notebook."""
    # Directories
    DATA_DIR = "datasets"
    PLOTS_DIR = "plots"
    LOGS_DIR = "logs"
```

```python
    # Filenames
    CLIENT_DATA = "client_data.csv"
    PRICE_DATA = "price_data.csv"
    LOG_FILE = "eda_detailed.log"

    # Key Columns
    DATE_COLS = ['date_activ', 'date_end', 'date_modif_prod', 'date_renewal']
    TARGET_COL = 'churn'

# --- Initialization & Environment Setup ---
config = Config()
os.makedirs(config.PLOTS_DIR, exist_ok=True)
sns.set_theme(style="whitegrid", palette="viridis")
plt.rcParams['figure.figsize'] = (12, 8)

print("Setup Complete. Configuration is loaded and directories are ready.")
```

```
Setup Complete. Configuration is loaded and directories are ready.
```

Part 2: Data Loading and Cleaning

Initial steps involved preparing the customer dataset for further analysis.

- **Data Loading:**
  Loaded `client_data.csv`, which contains **14,606 customer records** and **26 features**.

- **Data Cleaning:**
  Converted key date columns (e.g., `date_activ`, `date_end`) to datetime format.

- **Validation:**
  Assertions were implemented to ensure successful data loading and cleaning. This confirmed dataset integrity before analysis.

In [2]:

```python
def load_data(path: str) -> pd.DataFrame:
    """Loads data from a specified CSV file path with robust error handling.""
    assert os.path.exists(path), f"Data file not found at: {path}"
    print(f"Loading data from: {path}")
    return pd.read_csv(path)

def clean_data(df: pd.DataFrame, date_cols: List[str]) -> pd.DataFrame:
    """Cleans the dataframe by converting specified columns to datetime object
    print("Converting date columns...")
    for col in date_cols:
        df[col] = pd.to_datetime(df[col], errors='coerce')
    return df

# --- Pipeline Execution ---
```

```
client_df = load_data(os.path.join(config.DATA_DIR, config.CLIENT_DATA))
client_df = clean_data(client_df, config.DATE_COLS)

# --- Validation ---
assert not client_df.empty, "Client data failed to load or is empty."
assert pd.api.types.is_datetime64_any_dtype(client_df['date_activ']), "Date cl

print("\n--- Data successfully loaded, cleaned, and validated. ---")
display(client_df.head())
```

```
Loading data from: datasets\client_data.csv
Converting date columns...

--- Data successfully loaded, cleaned, and validated. ---
```

	id	channel_sales	cons_12n
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	466
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	54
3	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	158
4	149d57cf92fc41cf94415803a877cb4b	MISSING	442

5 rows × 26 columns

Part 3: Data Inspection Summary

A high-level overview revealed the following characteristics of the dataset:

- **Data Structure:**
  The dataset is structurally sound with no widespread missing values in critical columns.

- **Key Features Identified:**
  - `cons_12m` (Consumption) and `net_margin` (Profitability) are both **right-skewed**, showing a wide range of values. Most customers are low-usage/low-margin, while a few have very high values.
  - `channel_sales` contains a **'MISSING'** category for a significant number of customers. This needs to be addressed before modeling.
  - `churn` is **imbalanced** — only **9.7% of customers** are marked as churned.

In [3]:
```python
# Display data types and non-null counts
print("--- Data Types and Non-Null Counts ---")
client_df.info()
```

```
--- Data Types and Non-Null Counts ---
<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  datetime64[ns]
 6   date_end                        14606 non-null  datetime64[ns]
 7   date_modif_prod                 14606 non-null  datetime64[ns]
 8   date_renewal                    14606 non-null  datetime64[ns]
 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: datetime64[ns](4), float64(11), int64(7), object(4)
memory usage: 2.9+ MB
```

Part 4: Visual Analysis & Key Findings

Visualizations were used to identify relationships between customer attributes and churn.

4.1 Customer Seniority and Churn

- **Finding:**
  Churn is not uniform across tenure.
  Customers with **4–5 years** of service show a slightly higher churn rate.
  → Middle-tenure may be a critical period for retention strategies.

4.2 Consumption and Churn

- **Finding:**
  Annual consumption distributions for churned and non-churned customers are **heavily overlapping**.
  Median consumption is **slightly lower** for churned customers.
  → Consumption alone is **not a strong churn indicator**.

4.3 Net Margin and Churn

- **Finding:**
  Lower net_margin is associated with churn.
  Customers who churned had **visibly lower profitability**.
  → Profitability is a **strong churn signal**.

4.4 Churn Rate by Sales Channel

- **Finding:**
  Churn varies significantly by **acquisition channel**.
  Some channels show **much higher churn rates**.
  → Indicates **low loyalty** or **misaligned expectations** in certain channels.

In [4]:

```
# Display descriptive statistics for all columns
print("\n--- Descriptive Statistics ---")
display(client_df.describe(include='all').transpose())
```

```
--- Descriptive Statistics ---
```

	count	unique	top
id	14606	14606	563dde550fd624d7352f3de77c0cdfcd
channel_sales	14606	8	foosdfpfkusacimwkcsosbicdxkicaua
cons_12m	14606.0	NaN	NaN
cons_gas_12m	14606.0	NaN	NaN
cons_last_month	14606.0	NaN	NaN
date_activ	14606	NaN	NaN
date_end	14606	NaN	NaN
date_modif_prod	14606	NaN	NaN
date_renewal	14606	NaN	NaN
forecast_cons_12m	14606.0	NaN	NaN
forecast_cons_year	14606.0	NaN	NaN
forecast_discount_energy	14606.0	NaN	NaN
forecast_meter_rent_12m	14606.0	NaN	NaN
forecast_price_energy_off_peak	14606.0	NaN	NaN
forecast_price_energy_peak	14606.0	NaN	NaN
forecast_price_pow_off_peak	14606.0	NaN	NaN
has_gas	14606	2	f
imp_cons	14606.0	NaN	NaN
margin_gross_pow_ele	14606.0	NaN	NaN
margin_net_pow_ele	14606.0	NaN	NaN
nb_prod_act	14606.0	NaN	NaN
net_margin	14606.0	NaN	NaN
num_years_antig	14606.0	NaN	NaN
origin_up	14606	6	lxidpiddsbxsbosboudacockeimpuepw
pow_max	14606.0	NaN	NaN
churn	14606.0	NaN	NaN

In [5]:
```
def save_and_display_plot(fig: plt.Figure, filename: str, directory: str):
    """Saves a matplotlib figure and then displays it."""
    path = os.path.join(directory, filename)
    fig.savefig(path, bbox_inches='tight', dpi=150)
```

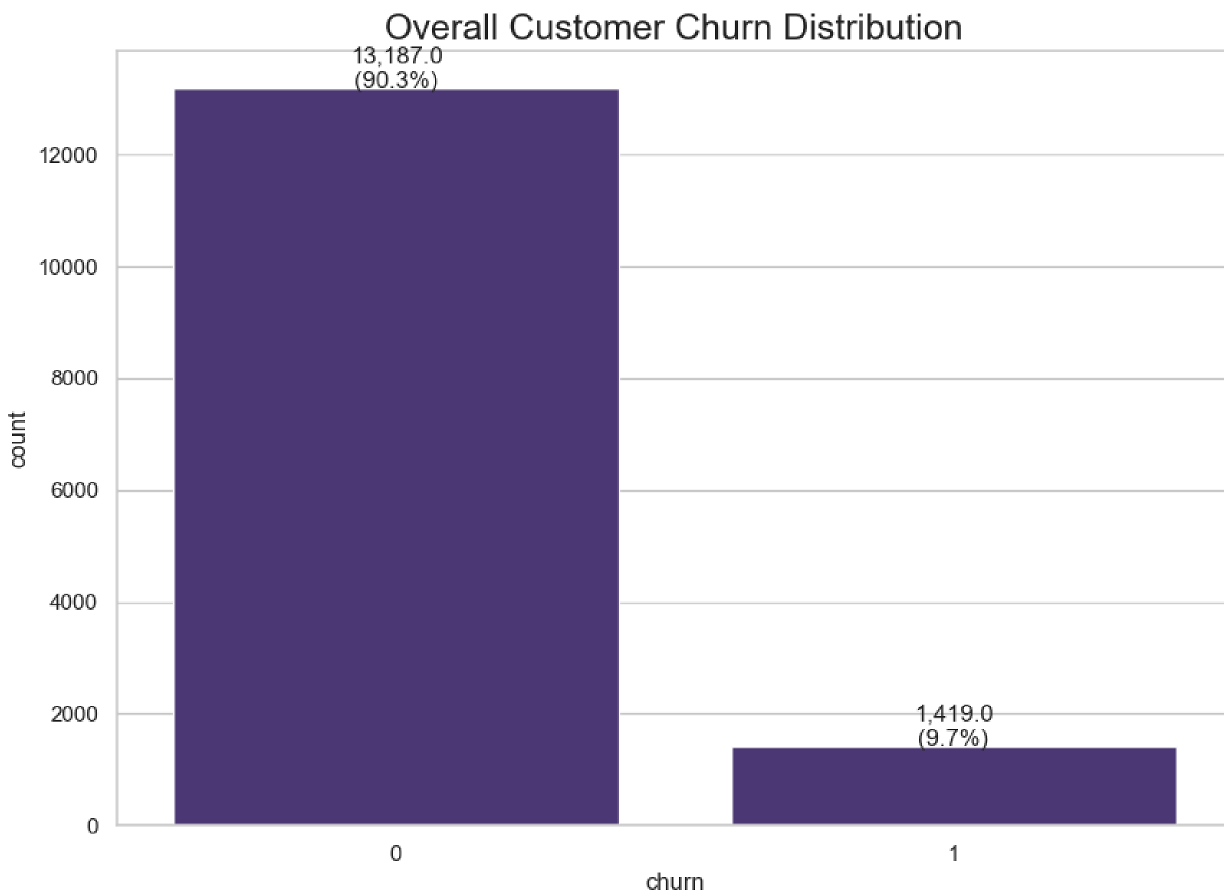
```
    print(f"Plot saved to: {path}")
    plt.show()
```

In [6]:
```
fig, ax = plt.subplots(figsize=(10, 7))
sns.countplot(x=config.TARGET_COL, data=client_df, ax=ax)
ax.set_title('Overall Customer Churn Distribution', fontsize=18)

total = len(client_df)
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2., height + 3, f'{height:,}\n({100 *

save_and_display_plot(fig, "1_churn_distribution.png", config.PLOTS_DIR)
```

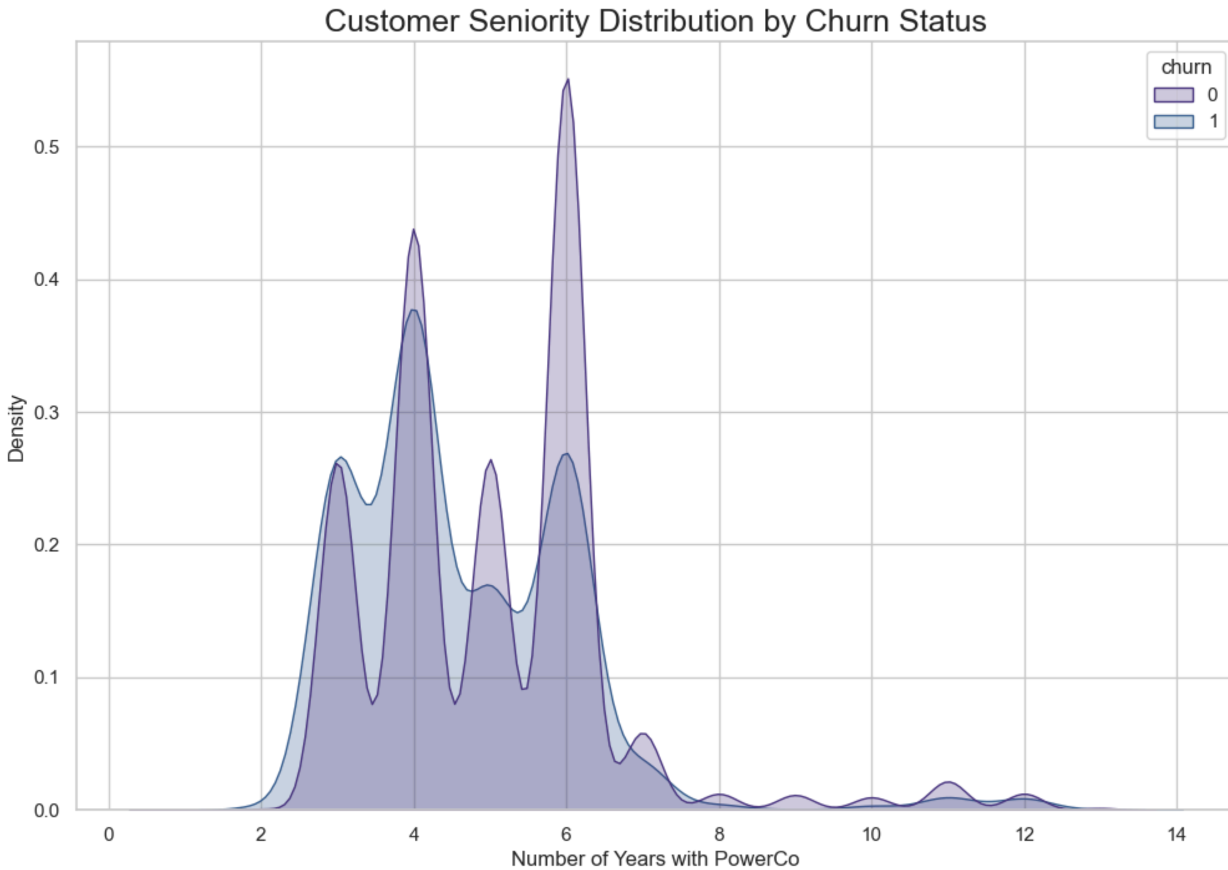
```
Plot saved to: plots\1_churn_distribution.png
```



In [7]:
```
fig, ax = plt.subplots(figsize=(12, 8))
sns.kdeplot(data=client_df, x='num_years_antig', hue=config.TARGET_COL, fill=T
ax.set_title('Customer Seniority Distribution by Churn Status', fontsize=18)
ax.set_xlabel('Number of Years with PowerCo')

save_and_display_plot(fig, "2_seniority_vs_churn.png", config.PLOTS_DIR)
```

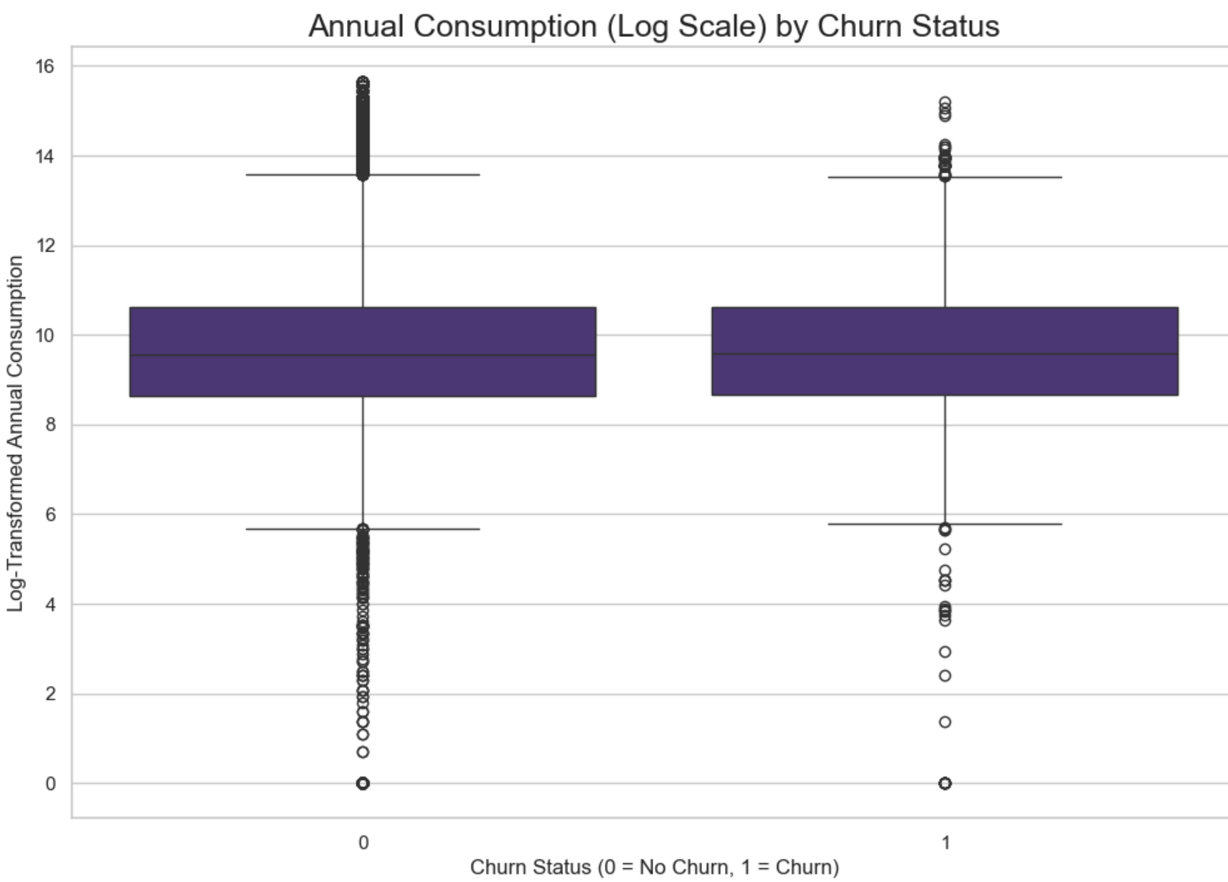
```
Plot saved to: plots\2_seniority_vs_churn.png
```



In [8]:
```
fig, ax = plt.subplots(figsize=(12, 8))
client_df['cons_12m_log'] = np.log1p(client_df['cons_12m'])
sns.boxplot(x=config.TARGET_COL, y='cons_12m_log', data=client_df)
ax.set_title('Annual Consumption (Log Scale) by Churn Status', fontsize=18)
ax.set_xlabel('Churn Status (0 = No Churn, 1 = Churn)')
ax.set_ylabel('Log-Transformed Annual Consumption')

save_and_display_plot(fig, "3_consumption_vs_churn.png", config.PLOTS_DIR)
```

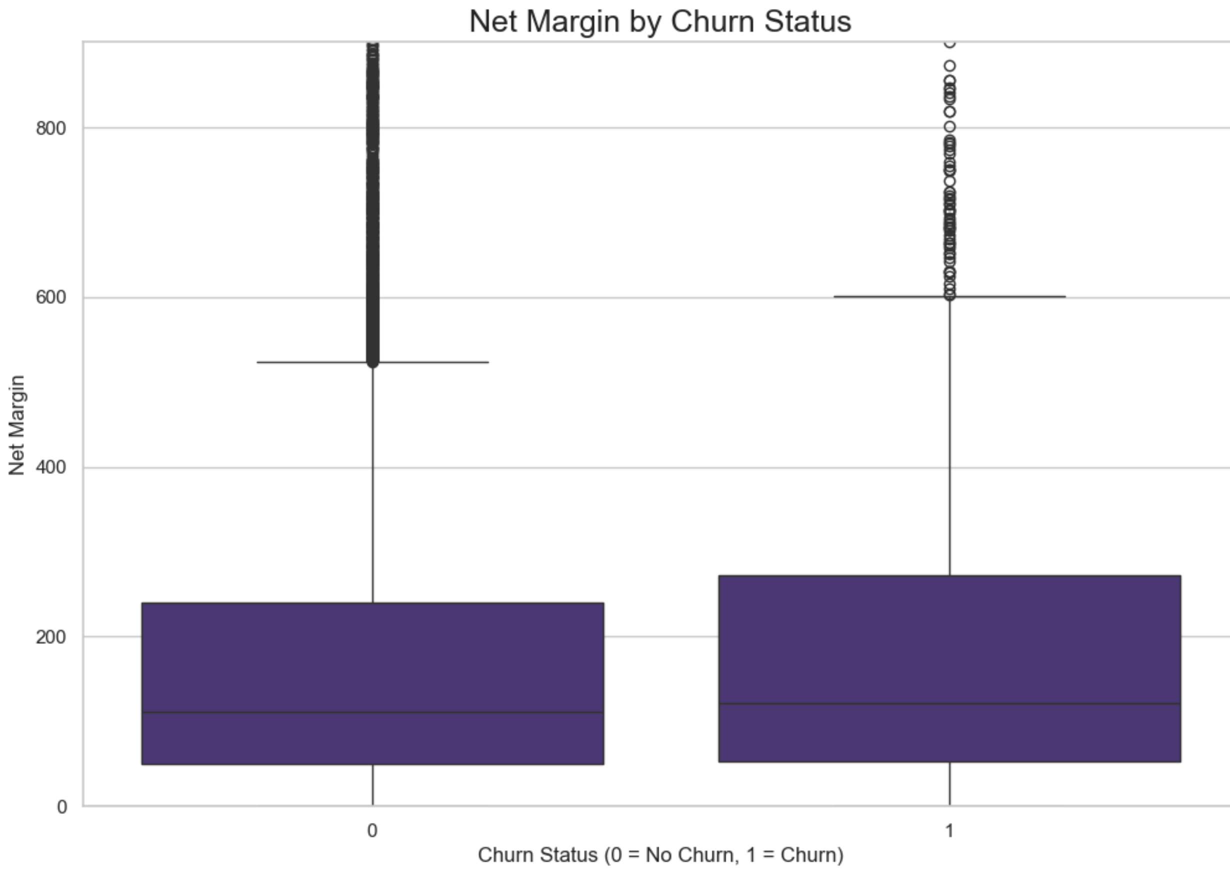
```
Plot saved to: plots\3_consumption_vs_churn.png
```



In [9]:
```
fig, ax = plt.subplots(figsize=(12, 8))
sns.boxplot(x=config.TARGET_COL, y='net_margin', data=client_df)
ax.set_title('Net Margin by Churn Status', fontsize=18)
ax.set_xlabel('Churn Status (0 = No Churn, 1 = Churn)')
ax.set_ylabel('Net Margin')
# Zoom in on the main distribution by limiting the y-axis
ax.set_ylim(0, client_df['net_margin'].quantile(0.99))

save_and_display_plot(fig, "4_margin_vs_churn.png", config.PLOTS_DIR)
```

```
Plot saved to: plots\4_margin_vs_churn.png
```

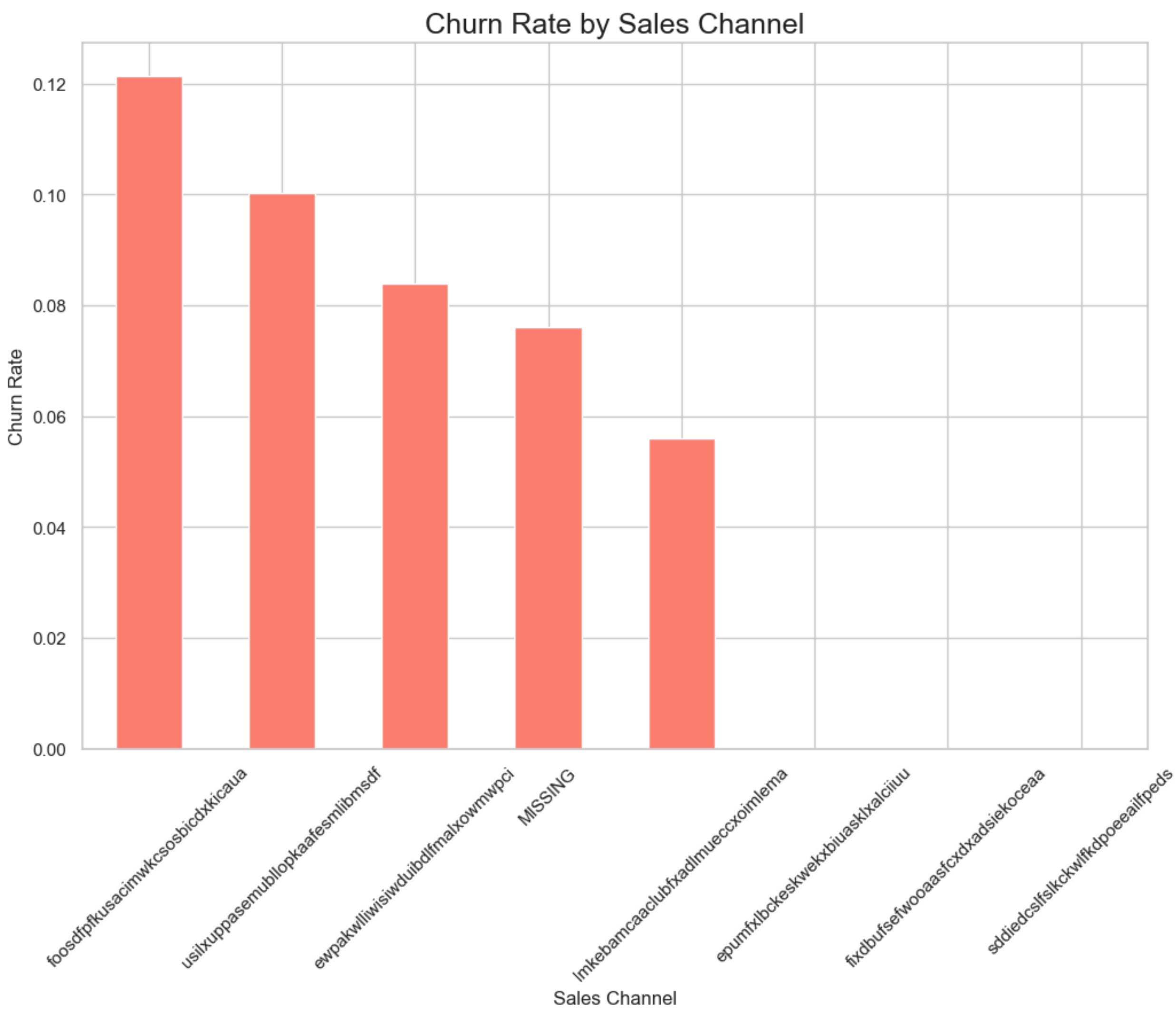


In [10]:
```
# Calculate churn rate by channel
churn_by_channel = client_df.groupby('channel_sales')[config.TARGET_COL].value
churn_by_channel = churn_by_channel.sort_values(by=1, ascending=False)

# Plot the results
fig, ax = plt.subplots(figsize=(12, 8))
churn_by_channel[1].plot(kind='bar', ax=ax, color='salmon')
ax.set_title('Churn Rate by Sales Channel', fontsize=18)
ax.set_ylabel('Churn Rate')
ax.set_xlabel('Sales Channel')
ax.tick_params(axis='x', rotation=45)

save_and_display_plot(fig, "5_churn_rate_by_channel.png", config.PLOTS_DIR)
```

```
Plot saved to: plots\5_churn_rate_by_channel.png
```



Part 5: Conclusion and Next Steps

The initial EDA has uncovered key insights into churn behavior:

- **Strong churn indicators:**
    - Net margin
    - Sales channel

Recommended Next Steps:

1. **Feature Engineering:**

    - Create a **tenure** feature from `date_end` – `date_activ`.
    - Join with **pricing data** to derive features like average price, price volatility, and price sensitivity.
2. **Handle Missing Values:**

- Develop a strategy for `'MISSING'` in `channel_sales` (e.g., categorize explicitly or impute).

3. **Predictive Modeling:**
    - Build a **classification model** to predict churn.
    - Use techniques to **handle class imbalance**.
    - Leverage **key features** from the analysis for optimal performance.
