|  |
| --- |
| null\_counts = df3.isnull().sum()  print(null\_counts) |
| from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn.metrics import accuracy\_score  # Select the features and target variable  features = ['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure','PhoneService',                          'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',                          'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',                          'Contract', 'PaperlessBilling','PaymentMethod', 'MonthlyCharges','TotalCharges']  target = 'Churn'  # Convert categorical variables to numeric using label encoding or one-hot encoding  # Split the dataset into train and test sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(df3[features], df[target], test\_size=0.2, random\_state=42)  # Initialize the Logistic Regression model  model = LogisticRegression()  # Fit the model to the training data  model.fit(X\_train, y\_train)  # Make predictions on the test data  y\_pred = model.predict(X\_test)  # Calculate the accuracy of the model  accuracy = accuracy\_score(y\_test, y\_pred)  print("Accuracy:", accuracy) |
| ##Stacked bar chart but getting ERRORS  df1= df[(df['MultipleLines'] == 'Yes')|(df['OnlineSecurity']=='No')|(df['OnlineBackup']=='No') | (df['DeviceProtection']=='No')| (df['TechSupport']=='No')| (df['Churn'] =='Yes')]  df1\_Yes\_No\_Yes = df1.groupby(["gender","SeniorCitizen"]).size().reset\_index(name= 'count')  df1\_Yes\_No\_Yes  # Prepare data for stacked bar chart  labels = df1\_Yes\_No\_Yes.apply(lambda x: f'{x["gender"]}\n({x["SeniorCitizen"]})', axis=1)  count\_yes = df1\_Yes\_No\_Yes[df1\_Yes\_No\_Yes['gender'] == 'Male']["count"]  count\_no = df1\_Yes\_No\_Yes[df1\_Yes\_No\_Yes['gender'] == 'Female']["count"]  # Plot stacked column chart  plt.figure(figsize=(8, 6))  plt.bar(labels, count\_yes, label="SeniorCitizen: 0")  plt.bar(labels, count\_no, bottom=count\_yes, label="SeniorCitizen: 1")  # Add labels and format the chart  plt.xlabel("Gender (Senior Citizen)")  plt.ylabel("Count")  plt.title("Stacked Column Chart")  plt.legend()  # Display values on top of bars  for i, (val\_yes, val\_no) in enumerate(zip(count\_yes, count\_no)):  plt.text(i, val\_yes / 2, str(val\_yes), ha="center", va="bottom")  plt.text(i, val\_yes + val\_no / 2, str(val\_no), ha="center", va="bottom")  # Rotate x-axis labels if needed  plt.xticks(rotation=45)  # Show the chart  plt.tight\_layout()  plt.show() |
| ## Working  import matplotlib.pyplot as plt  # Filter and count based on the following conditions  df1 = df[(df['MultipleLines'] == 'Yes') | (df['OnlineSecurity'] == 'No') | (df['OnlineBackup'] == 'No') |  (df['DeviceProtection'] == 'No') | (df['TechSupport'] == 'No') | (df['Churn'] == 'Yes')]  df1\_grouped = df1.groupby(["gender", "SeniorCitizen"]).size().reset\_index(name='count')  # Prepare data for stacked column chart  labels = df1\_grouped.apply(lambda x: f'{x["gender"]}\n({x["SeniorCitizen"]})', axis=1)  count = df1\_grouped["count"]  # Plot stacked column chart  plt.figure(figsize=(8, 6))  plt.bar(labels, count)  # Add labels and format the chart  plt.xlabel("Gender (Senior Citizen)")  plt.ylabel("Count")  plt.title("Stacked Column Chart")  # Display values on top of bars  for i, val in enumerate(count):  plt.text(i, val + 10, str(val), ha="center", va="top")  # Rotate x-axis labels if needed  plt.xticks(rotation=45)  # Show the chart  plt.tight\_layout()  plt.show() |
| ##Other possibilities  import matplotlib.pyplot as plt  # Plot stacked column chart  plt.figure(figsize=(8, 6))  plt.bar(df1\_Yes\_No\_Yes.index, df1\_Yes\_No\_Yes['count'], color='skyblue')  # Add labels and format the chart  plt.xlabel("Index")  plt.ylabel("Count")  plt.title("Stacked Column Chart")  plt.xticks(df1\_Yes\_No\_Yes.index, df1\_Yes\_No\_Yes["gender"] + " (Senior Citizen: " + df1\_Yes\_No\_Yes["SeniorCitizen"].astype(str) + ")")  # Display values on top of bars  for i, count in enumerate(df1\_Yes\_No\_Yes["count"]):  plt.text(i, count, str(count), ha="center", va="bottom")  # Show the chart  plt.tight\_layout()  plt.show() |
| import matplotlib.pyplot as plt  # Count the occurrences of each category  gender\_counts = df['gender'].value\_counts()  senior\_citizen\_counts = df['SeniorCitizen'].value\_counts()  # Prepare the data for the column chart  labels = gender\_counts.index  gender\_count = gender\_counts.values  senior\_citizen\_count = senior\_citizen\_counts.values  # Set the width of each column  bar\_width = 0.35  # Set the positions of the x-axis ticks  x = range(len(labels))  # Plot the column chart  plt.figure(figsize=(8, 6))  plt.bar(x, gender\_count, width=bar\_width, label='Gender')  plt.bar([i + bar\_width for i in x], senior\_citizen\_count, width=bar\_width, label='Senior Citizen')  # Add labels and format the chart  plt.xlabel("Category")  plt.ylabel("Count")  plt.title("Gender and Senior Citizen Counts")  plt.xticks([i + bar\_width / 2 for i in x], labels)  # Display the value on top of each column  for i, count in enumerate(gender\_count):  plt.text(i, count + 10, str(count), ha='center', va='bottom')  for i, count in enumerate(senior\_citizen\_count):  plt.text(i + bar\_width, count + 10, str(count), ha='center', va='bottom')  # Add legend  plt.legend()  # Show the chart  plt.tight\_layout()  plt.show() |
| # Not working  df1['SaleDate'] = pd.to\_datetime(df1['SaleDate'])  df1.set\_index('SaleDate', inplace=True)  yearly\_sales = df1['SalePrice'].resample('Y').sum()  fig, axes = plt.subplots(nrows=len(yearly\_sales), ncols=1, figsize=(10, 6 \* len(yearly\_sales)))  for i, (year, sales) in enumerate(yearly\_sales.iteritems()):  ax = axes[i]  ax.plot(sales.index, sales.values, linewidth=2, color='blue')  ax.set\_title(f"Year {year} Sales")  ax.set\_xlabel('Sale Date')  ax.set\_ylabel('Sale Price')  plt.tight\_layout()  plt.show() |
| #Not working  yearly\_sales = df1['SalePrice'].resample('Y').sum()  fig, axes = plt.subplots(nrows=len(yearly\_sales), ncols=1, figsize=(10, 6 \* len(yearly\_sales)))  for i, (year, sales) in enumerate(yearly\_sales.iteritems()):  ax = axes[i]  ax.plot(sales.index, sales.values, linewidth=2, color='blue')  ax.set\_title(f"Year {year} Sales")  ax.set\_xlabel('Sale Date')  ax.set\_ylabel('Sale Price')  plt.tight\_layout()  plt.show() |
| df.nlargest(5,['SalePrice']) |
| df1['month'].loc[(df1['SalePrice'].max())] |
| df1.reset\_index(drop=True) |
| !pip install plotly==5.15.0  from plotly.offline import init\_notebook\_mode, iplot  from plotly.graph\_objs import \*  import plotly.express as px  data\_ACTON ST = px.df1().query("StreetNameAndWay == 'ACTON ST'") |
| Dataset: Trade Commodity   1. import matplotlib.pyplot as plt 2. # Assuming you have already executed the code you provided and obtained `commodity\_highest\_price` dataframe 3. # Creating subplots 4. fig, axes = plt.subplots(nrows=len(commodity\_highest\_price), figsize=(5, 6 \* len(commodity\_highest\_price))) 5. # Plotting each year's highest price in a separate subplot 6. for i, (year, data) in enumerate(commodity\_highest\_price.iterrows()): 7. commodity = data['Commodity Name'] 8. value = data['Value'] 10. ax = axes[i] 11. ax.bar(commodity, value) 12. ax.set\_xlabel('Commodity') 13. ax.set\_ylabel('Highest Price') 14. ax.set\_title(f'Highest Price in {year}') 15. ax.tick\_params(axis='x', rotation=45) 16. plt.tight\_layout() 17. plt.show() |
| import matplotlib.pyplot as plt  # Assuming you have already executed the code you provided and obtained `commodity\_highest\_price` dataframe  # Creating subplots with reduced size  fig, axes = plt.subplots(nrows=len(commodity\_highest\_price), figsize=(10, 6 \* len(commodity\_highest\_price) // 10))  # Plotting each year's highest price in a separate subplot  for i, (year, data) in enumerate(commodity\_highest\_price.iterrows()):  commodity = data['Commodity Name']  value = data['Value']    ax = axes[i]  ax.bar(commodity, value)  ax.set\_xlabel('Commodity')  ax.set\_ylabel('Highest Price')  ax.set\_title(f'Highest Price in {year}')  ax.tick\_params(axis='x', rotation=45)  plt.tight\_layout()  plt.show() |
| import matplotlib.pyplot as plt  # Assuming you have already executed the code you provided and obtained `commodity\_highest\_price` dataframe  # Creating subplots with increased spacing  fig, axes = plt.subplots(nrows=len(commodity\_highest\_price), figsize=(10, 6 \* len(commodity\_highest\_price) // 10))  # Plotting each year's highest price in a separate subplot  for i, (year, data) in enumerate(commodity\_highest\_price.iterrows()):  commodity = data['Commodity Name']  value = data['Value']    ax = axes[i]  ax.bar(commodity, value)  ax.set\_xlabel('Commodity')  ax.set\_ylabel('Highest Price')  ax.set\_title(f'Highest Price in {year}')  ax.tick\_params(axis='x', rotation=45)  plt.tight\_layout(pad=2.0)  plt.show() |
| Type error:  # Filter the dataframe for the year 2023  year\_2023 = commodity\_lowest\_price.loc[2023]  # Find the commodity with the lowest price index in 2023  min\_commodity\_2023 = year\_2023[year\_2023['Value'] == year\_2023['Value'].min()]  # Print the commodity with the lowest price index in 2023  print(min\_commodity\_2023) |
| ValueError: # Convert the index to datetime if it is not already in datetime format  commodity\_lowest\_price.index = pd.to\_datetime(commodity\_lowest\_price.index)  # Filter the dataframe for the year 2023  year\_2023 = commodity\_lowest\_price[commodity\_lowest\_price.index.year == 2023]  # Find the commodity with the lowest price index in 2023  min\_commodity\_2023 = year\_2023[year\_2023['Value'] == year\_2023['Value'].min()]  # Print the commodity with the lowest price index in 2023  print(min\_commodity\_2023) |
| ParserError:  # Convert the index to datetime if it is not already in datetime format  commodity\_lowest\_price.index = pd.to\_datetime(commodity\_lowest\_price.index)  # Filter the dataframe for the year 2023 and specific time periods  selected\_periods = ['2023', '2023M1', '2023M2'] # Specify the desired time periods  filtered\_data = commodity\_lowest\_price[commodity\_lowest\_price.index.isin(selected\_periods)]  # Print the filtered data  print(filtered\_data) |
| Correct code:  import re  # Convert the index to string if it is not already  commodity\_lowest\_price.index = commodity\_lowest\_price.index.astype(str)  # Specify the desired time periods using regular expressions  #selected\_patterns = ['2023', '23M1', '23M2'] # Specify the desired patterns  selected\_patterns = ['2023']  # Filter the dataframe for the specified patterns  filtered\_data = commodity\_lowest\_price[commodity\_lowest\_price.index.str.contains('|'.join(selected\_patterns))]  # Print the filtered data  print(filtered\_data) |
| Attribute Error:  #commodity\_highest\_price.index = commodity\_highest\_price.index.astype(str)  # Filter the dataframe for the time period from 2010 to 2020  filtered\_data = commodity\_highest\_price.index[(commodity\_highest\_price.index >= '2010') & (commodity\_highest\_price.index <= '2020')]  # Extract the necessary data for plotting  time\_periods = filtered\_data.index  values = filtered\_data['Value']  commodities = filtered\_data['Commodity Name']  # Plotting the graph  plt.figure(figsize=(10, 6))  plt.plot(time\_periods, values, marker='o', linestyle='-', color='b')  # Adding labels to the data points  for time\_period, value, commodity in zip(time\_periods, values, commodities):  plt.text(time\_period, value, f'{commodity}\n{value}', ha='center', va='bottom')  plt.xlabel('Time Period')  plt.ylabel('Lowest Price')  plt.title('Commodity with Lowest Price Index')  plt.xticks(rotation=45)  plt.tight\_layout()  plt.show() |
| import pandas as pd  # Filter the dataset for the most recent year  most\_recent\_year = pcpdf['Time Period'].max()  recent\_year\_data = pcpdf[pcpdf['Time Period'] == most\_recent\_year]  # Filter the data further for the type of crude oil  crude\_oil\_data = recent\_year\_data[recent\_year\_data['Commodity Name'].str.contains('Crude')]  # Check if there are any rows that match the filtering condition  if not crude\_oil\_data.empty:  # Find the type of crude oil with the second-highest price index  sorted\_crude\_oil\_data = crude\_oil\_data.sort\_values('Value', ascending=False)  second\_highest\_price\_index = sorted\_crude\_oil\_data['Value'].iloc[3]  crude\_oil\_second\_highest = sorted\_crude\_oil\_data[sorted\_crude\_oil\_data['Value'] == second\_highest\_price\_index]  # Extract the type of crude oil and the second-highest price index  crude\_oil\_type = crude\_oil\_second\_highest['Commodity Name'].iloc[0]  highest\_price\_index = crude\_oil\_second\_highest['Value'].iloc[0]  # Print the type of crude oil with the second-highest price index  print("Type of Crude Oil with Second-Highest Price Index:")  print("Crude Oil Type:", crude\_oil\_type)  print("Second-Highest Price Index:", highest\_price\_index)  else:  print("No data found for Crude Oil in the most recent year.") |
|  |
|  |
|  |
|  |
|  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |