Pre-processing Global-Orders Dataset for Tableau

1. This Dataset covers Global Orders for a broad range of companies. It contains catagorical features as well as numerical. This dataset was chosen for its complexity, it seemed like a challenging dataset that also had enough information to be fruitful.

2. **Questions I will answer include:**

- What Regions Produce the most goods?
- Why might these regions produce their good(s)?
- Is there a correlation between priority level in orders and the profit earned?
- Is there a correlation for priority level of an orders and the associated shipping cost?
- What American Tech companies produce the most?
- What products do the American Tech companies specialize in?
- What/where is Canon's target market?
- How can Canon capitalize on this market?

3. ETL is listed below.

Loading Imports, Data, and Display

In [ ]:
```
import numpy as np
import pandas as pd
import re

data = pd.read_csv('/Users/alexandergursky/Local_Repository/Datasets/CSV/Global-Orders.csv', encoding='ISO-8859-1')

print(data.columns)
```

```
Index(['Row_ID', 'Order_ID', 'Order_Date', 'Ship_Date', 'Ship_Mode',
       'Customer_ID', 'Customer_Name', 'Segment', 'City', 'State', 'Country',
       'Postal_Code', 'Market', 'Region', 'Product_ID', 'Category',
       'Sub-Category', 'Product_Name', 'Sales', 'Quantity', 'Discount',
       'Profit', 'Shipping_Cost', 'Order_Priority'],
      dtype='object')
```

In [ ]:
```
data.columns = data.columns.str.replace(' ', '_')

print(data.columns)
```

```
Index(['Row_ID', 'Order_ID', 'Order_Date', 'Ship_Date', 'Ship_Mode',
       'Customer_ID', 'Customer_Name', 'Segment', 'City', 'State', 'Country',
       'Postal_Code', 'Market', 'Region', 'Product_ID', 'Category',
       'Sub-Category', 'Product_Name', 'Sales', 'Quantity', 'Discount',
       'Profit', 'Shipping_Cost', 'Order_Priority'],
      dtype='object')
```

Data Cleaning: Regular Expressions for Products

In [ ]:
```
# Lets start cleaning up the product names
print(data['Product_Name'].head())

# selects the column, splits the column once on specified delmiter, takes the first element from this split list.
data.Product_Name = data.Product_Name.str.split(',', n=1).str[0]
```

```
0    Plantronics CS510 - Over-the-Head monaural Wir...
1              Novimex Executive Leather Armchair, Black
2                      Nokia Smart Phone, with Caller ID
3                      Motorola Smart Phone, Cordless
4                      Sharp Wireless Fax, High-Speed
Name: Product_Name, dtype: object
```

In [ ]:
```
# Observing that we got rid of the color atributes that followed the ,
print(data['Product_Name'].head())
```

```
0    Plantronics CS510 - Over-the-Head monaural Wir...
1                  Novimex Executive Leather Armchair
2                                Nokia Smart Phone
3                             Motorola Smart Phone
4                             Sharp Wireless Fax
Name: Product_Name, dtype: object
```

In [ ]:
```
# Creating a new subset df to get the company names
prod_list = pd.DataFrame({'Product_Name' : data.Product_Name.unique()})


   .str.replace(r'[^\w\s]+', '')
```

This removes all special characters. The regular expression [^\w\s]+ matches any non-alphanumeric character that is not a space or underscore, and replaces it with an empty string. The + indicates that one or more occurrences should be replaced. The \w matches any word character (letter, digit, or underscore), and \s matches any whitespace character (space, tab, or newline).

In [ ]:
```
# Removing special characters
prod_list['Product_Name'] = prod_list['Product_Name'].str.replace(r'[^\w\s]+', '', regex=True)


   .str.replace(pattern, '', regex=True)
```

This would remove all special characters from the column_name column. However, in this approach, you are calling the replace() method for each row in the column, which can be slower for larger datasets.

```
   .apply(lambda x: re.sub(pattern, '', x))
```

Using a lambda function with apply() allows you to apply the pattern to the entire column at once, resulting in a faster execution time. Additionally, it allows for more flexibility in terms of modifying the input before applying the regular expression.

In [ ]:
```
import re

# Define regex pattern to remove non-ASCII characters and special characters
pattern = r'[^ -~]'

# Remove non-ASCII and special characters from the 'Text' column
# The regex pattern [^ -~] will match any characters that are not in the printable ASCII range.
prod_list['Product_Name'] = prod_list['Product_Name'].apply(lambda x: re.sub(pattern, '', x))
```

In [ ]:
```
# Get the first word before a space
companies = prod_list['Product_Name'].str.split(' ', n=1).str[0]

# Get the unique values
companies = pd.Series(sorted(companies.unique()))
```

Data Cleaning: Gathering Company Names

In [ ]:
```
# drop company names containing numbers
companies = companies[~companies.str.contains('\d')]

companies = pd.DataFrame({'Company_Names' : companies})
```

In [ ]:
```
# import spacy

# # load the spaCy English language model
# nlp = spacy.load('en_core_web_sm')

# # define a function to extract company names from text
# def extract_company_names(text):
#     doc = nlp(text)
#     company_names = []
#     for ent in doc.ents:
#         if ent.label_ == 'ORG':
#             company_names.append(ent.text)
#     return company_names

# # apply the function to your column of words
# companies['Company_Names1'] = companies['Company_Names'].apply(extract_company_names)
```

```
2023-04-18 01:34:51.503879: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available C
PU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Data Cleaning: Some Manual Work

In [ ]:
```
# export the column to a text file
col_export = companies['Company_Names']

col_export.to_csv('output_file1.txt', index=False, header=False)
```

In [ ]:
```
# import text file into a pandas dataframe
comp_txt = pd.read_csv('/Users/alexandergursky/Local_Repository/output_file.txt', header=None, names=['Column1'])
```

Data Cleaning: Reformatting, Cleaning, and Parsing for Companies/Products

In [ ]:
```
# convert values in 'Name' column to proper case
comp_txt['Column1'] = comp_txt['Column1'].str.title()

# Appending a new company I can see needs to be added
comp_txt.loc[len(comp_txt)] = ['Officestar']
comp_txt.loc[len(comp_txt)] = ['Ki']

# Drop duplicates
comp_txt.drop_duplicates(subset=['Column1'], inplace=True)
```

In [ ]:
```
# Remaining company names I need to fix
comp_dict = {
    'Motorla' : 'Motorola',
    'Peel' : 'Peeloff',  # later, will make a new dict for these
    'Neat' : 'Neatdesk',  # later
    'Permanent' : 'Perma',
    'Poly' : 'Polycom',  # later
    'Star' : 'Startechcom',  # later
    'Startech' : 'Startechcom',  # later
    'Jhome' : 'Apple',
    'Euro' : 'Europro',  # later
    'High' : 'Highback',  # later
    'Hewlettpackard' : 'Hewlett',
    'Martinyale' : 'Martin',
    'Memo' : 'Memorex',  # later
    'Maxellito' : 'Maxwell',
    'Maxellivdr' : 'Maxwell',
    'Ki': 'Kitchenaid',
}

comp_txt['Column1'].replace(comp_dict, inplace=True)

# replace any text after 'Logitech' with 'Logitech'
comp_txt['Column1'] = comp_txt['Column1'].str.replace(r'Logitech.+', 'Logitech', regex=True)

# replace any text after 'Imation' with 'Imation'
comp_txt['Column1'] = comp_txt['Column1'].str.replace(r'Imation.+', 'Imation', regex=True)

# Drop duplicates
comp_txt.drop_duplicates(subset=['Column1'], inplace=True)
```

In [ ]:
```
# Removing special characters
data['Product_Name'] = data['Product_Name'].str.replace(r'[^\w\s]+', '', regex=True)

# Define regex pattern to remove non-ASCII characters and special characters
pattern = r'[^ -~]'

# Remove non-ASCII and special characters from the 'Product_Name' column
# The regex pattern [^ -~] will match any characters that are not in the printable ASCII range.
data['Product_Name'] = data['Product_Name'].apply(lambda x: re.sub(pattern, '', x))

# Fixing company name in product
data['Product_Name'] = data['Product_Name'].str.lower()
data['Product_Name'] = data['Product_Name'].replace({'office star' : 'Officestar'}, regex=True)

# convert values in 'Product_Name' column to proper case
data['Product_Name'] = data['Product_Name'].str.title()
```

In [ ]:
```
fix_names_dict = {
    'Peel' : 'Peeloff',
    'Neat' : 'Neatdesk',
    'Poly' : 'Polycom',
    'Star' : 'Startechcom',
    'Startech' : 'Startechcom',
    'Euro' : 'Europro',
    'High' : 'Highback',
    'Memo' : 'Memorex',
}

# apply replacement using regex
data['Product_Name'] = data['Product_Name'].replace(fix_names_dict, regex=True)
```

In [ ]:
```
# function to match product names with company names
def match_company(product_name):
    for company in comp_txt['Column1']:
        if product_name.startswith(company):
            return company
    return None

# apply the match_company function to the product names and create a new column with the matching company names
data['Company_Name'] = data['Product_Name'].apply(match_company)

replace_exces = {
    'Memorexrex' : 'Memorex',
    'Polycomcom' : 'Polycom',
}
# apply replacement using regex
data['Product_Name'] = data['Product_Name'].replace(replace_exces, regex=True)
```

Data Cleaning: Profits

In [ ]:
```
data.Profit.dtypes
```

Out[ ]:
```
dtype('float64')
```

In [ ]:
```
# I noticed the profit values were incorrect by handcalculating so im fixing it here
data['Profit'] = abs(data['Profit'])
```

Data Cleaning: Parsing/Removal of Company Names from Products

In [ ]:
```
# Define a list of company names from the 'Value' column
companies = data['Company_Name'].tolist()

# Defining a custom function to remove the company name from the 'Product_Name' column if it appears in the 'Company_Name' column
def remove_company_name(row, companies):
    words = row['Product_Name'].split()
    for company in companies:
        if company in words:
            words.remove(company)
            break
    return ' '.join(words)

# Apply the custom function to the DataFrame to remove the company name from the 'Product_Name' column if it appears in the 'Company_Na
data['Product_Name'] = data.apply(lambda row: remove_company_name(row, companies), axis=1)
```

Data Cleaning: Removing Unwanted Values (NaN's and NPC's)

In [ ]:
```
# Drop naN values
data = data.dropna(subset=['Company_Name'])
```

In [ ]:
```
city, state = pd.DataFrame({'City' : data.City.unique()}), pd.DataFrame({'State' : data.State.unique()})
```

In [ ]:
```
# Count the number of spaces in the "value" column
city['num_spaces'] = city['City'].apply(lambda x: x.count(' '))
```

In [ ]:
```
print(city.iloc[2887])
```

```
City          SÃ£o Jos dos Campos
num_spaces                      3
Name: 2887, dtype: object
```

In [ ]:
```
# Define a function to check if a string contains non-printable or non-ASCII characters
def has_non_printable_characters(string):
    return bool(re.search(r'[^\x00-\x7F]', string))

# Apply the function to each row of the dataframe and store the result in a new column
city['has_non_printable_characters'] = city['City'].apply(has_non_printable_characters)
```

In [ ]:
```
# create a Boolean mask based on the function
mask = city['City'].apply(has_non_printable_characters)

# drop rows where the mask is True
df = city.drop(index=city[mask].index)
```

In [ ]:
```
# Applying this to the dataframe now

# create a Boolean mask based on the function
mask = data['City'].apply(has_non_printable_characters)
mask1 = data['State'].apply(has_non_printable_characters)
mask2 = data['Country'].apply(has_non_printable_characters)

# drop rows where the mask is True
data = data.drop(index=data[mask].index)
data = data.drop(index=data[mask1].index)
data = data.drop(index=data[mask2].index)
```

Exporting Preprocessed Dataset

In [ ]:
```
data = data.drop('Row_ID', axis=1)
data.to_csv('preprocessed-Global-Orders.csv', index=False)
```