GitHub URL:

https://github.com/salmashamboul/UCDPA_Salmashamboul

Abstract:

This project is based on the analysis of the AdventureWorks dataset. This dataset contains information about the company's operations, including various aspects such as sales, customers, products, and employees. In this project, I imported various tables from this dataset into a Jupyter notebook. The tables I used for analysis include the Sales tables, Territory data, and customer data.

I utilized data importing functions from pandas to load the dataset into data frames. Once the datasets were loaded, I combined them using concatenation and merge functions. Then, I sorted the dataset and reset its index. Subsequently, I performed data cleansing by removing nulls and duplicates from the dataset. I also fixed data types and corrected various values in the data columns to ensure data consistency.

After data cleansing, I conducted an initial analysis of the dataset, which involved checking the number of rows and columns in the dataset and performing descriptive analysis. To answer the analysis questions, I employed loops, conditional statements, and functions to create new columns within the dataset.

Lastly, I carried out data visualization to extract insights and identify patterns from the dataset.

Introduction:

I chose this project because I believe it’s important for companies to be able to analyze their data and understand which products are in demand. They should be able to answer questions such as what are the demographics of their customers and how is it performing in different regions of the world? This use case provides a wide range of operations which we can perform on a dataset to find out significant insights. For this analysis I chose to use an adventure works dataset. AdventureWork’s Park is a recreational destination for thrill-seekers and outdoor enthusiasts of all ages. The dataset contains data for sales, products, customers, territories and other entities. This dataset contains many tables that are very helpful to understand how to combine data for sales from different years. For example using this dataset I used the concat function in Pandas to combine three years sales. Similarly I used the merge function in Pandas to combine customer data with sales and then combine territory data to it. The main aim of using this dataset is to perform concatenation and merge functions in Pandas.

Dataset:

The AdventureWorks dataset is a collection of sample databases. The dataset simulates the operations of a company and contains data related to sales, products, customers, employees, and more. This dataset is divided into several categories, each representing different types of data:

- Sales: Information about sales orders, order details, and related data.
- Products: Details about the products the company sells, including categories, subcategories, and product specifications.
- Customers: Data about customers, including contact information, demographics, and purchasing history.
- Employees: Information about company employees, such as their roles, departments, and employment history.
- Territories: Geographic regions and territories where the company operates.
- Vendors: Data about the suppliers and vendors that the company interacts with.

This dataset is collected from kaggle.com. For analysis purposes I utilized sales, customers and territories tables from the dataset to perform various data transformation functions in Pandas.

This dataset can be downloaded from:
https://www.kaggle.com/datasets/ukveteran/adventure-works

Implementation process:

Importing Required Modules

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

I started the process by loading all the required modules in Jupyter notebook. The modules I used for the data transformation, analysis and visualization includes pandas, numpy, matplotlib and seaborn. In short, Pandas is for data manipulation, NumPy for numerical operations, Matplotlib for general plotting, and Seaborn for statistical data visualization.

1. Data Loading:

Loading Sales Dataset

```
Sales_15=pd.read_csv("AdventureWorks_Sales_2015.csv")
Sales_16=pd.read_csv("AdventureWorks_Sales_2016.csv")
Sales_17=pd.read_csv("AdventureWorks_Sales_2017.csv")
```

Once I imported all the modules I imported the sales dataset into pandas. I created three data frames objects and used the .read_csv () method from pandas to read sales csv files for three years and store them into pandas data frames.

Loading customers data

```
customers_df=pd.read_csv("AdventureWorks_Customers.csv", encoding='latin-1')
customers_df.head()
```

	CustomerKey	Prefix	FirstName	LastName	BirthDate	MaritalStatus	Gender	EmailAddress	AnnualIncome	TotalChildren	EducationLevel	Occupatic
0	11000	MR.	JON	YANG	4/8/1966	M	M	jon24@adventure-works.com	$90,000	2	Bachelors	Professior
1	11001	MR.	EUGENE	HUANG	5/14/1965	S	M	eugene10@adventure-works.com	$60,000	3	Bachelors	Professior
2	11002	MR.	RUBEN	TORRES	8/12/1965	M	M	ruben35@adventure-works.com	$60,000	3	Bachelors	Professior
3	11003	MS.	CHRISTY	ZHU	2/15/1968	S	F	christy12@adventure-works.com	$70,000	0	Bachelors	Professior
4	11004	MRS.	ELIZABETH	JOHNSON	8/8/1968	S	F	elizabeth5@adventure-works.com	$80,000	5	Bachelors	Professior

Loading Territory data

```
Territory_df=pd.read_csv("AdventureWorks_Territories.csv")
Territory_df.head()
```

	TerritoryKey	Region	Country	Continent
0	1	Northwest	United States	North America
1	2	Northeast	United States	North America
2	3	Central	United States	North America
3	4	Southwest	United States	North America
4	5	Southeast	United States	North America

Similarly I used the pd.read_csv() method to load customers and territories dataset into pandas data frame.

2. Data Preparation:

Once I was done with data loading, I performed operations like concatenation and merge to combine all the data sources into one data frame to perform analysis on it.

Concatinating 3 years sales data

```
Sales_df = pd.concat([Sales_15, Sales_16, Sales_17], ignore_index=True)
Sales_df
```

	OrderDate	StockDate	OrderNumber	ProductKey	CustomerKey	TerritoryKey	OrderLineItem	OrderQuantity
0	1/1/2015	9/21/2001	SO45080	332	14657	1	1	1
1	1/1/2015	12/5/2001	SO45079	312	29255	4	1	1
2	1/1/2015	10/29/2001	SO45082	350	11455	9	1	1
3	1/1/2015	11/16/2001	SO45081	338	26782	6	1	1
4	1/2/2015	12/15/2001	SO45083	312	14947	10	1	1
...	...	...	...	...	...	...	...	...
56041	6/30/2017	3/22/2004	SO74143	477	28517	10	3	2
56042	6/30/2017	3/15/2004	SO74143	479	28517	10	2	1
56043	6/30/2017	4/8/2004	SO74143	606	28517	10	1	1
56044	6/30/2017	5/15/2004	SO74124	480	21676	7	2	2
56045	6/30/2017	5/4/2004	SO74124	538	21676	7	1	2

56046 rows × 8 columns

I used the concat function in pandas to combine 2015, 2016 and 2017 sales into one data frame and named that as sales_df.

Merging sales and customers data ¶

```
merged_df = pd.merge(Sales_df, customers_df, on='CustomerKey')
merged_df.head()
```

	OrderDate	StockDate	OrderNumber	ProductKey	CustomerKey	TerritoryKey	OrderLineItem	OrderQuantity	Prefix	FirstName	LastName	BirthDate	Ma
0	1/1/2015	9/21/2001	SO45080	332	14657	1	1	1	MR.	JOHN	THOMAS	11/11/1958	
1	1/9/2017	11/22/2003	SO61768	485	14657	1	2	2	MR.	JOHN	THOMAS	11/11/1958	
2	1/9/2017	12/20/2003	SO61768	215	14657	1	3	1	MR.	JOHN	THOMAS	11/11/1958	
3	1/9/2017	10/1/2003	SO61768	352	14657	1	1	1	MR.	JOHN	THOMAS	11/11/1958	
4	1/1/2015	12/5/2001	SO45079	312	29255	4	1	1	MR.	KYLE	WASHINGTON	4/11/1955	

Then I combined the customer dataset with sales_df using CustomerKey as a common field among both the tables.

Merging merged dataframe and territory data

```
merged_df = pd.merge(merged_df, Territory_df, on='TerritoryKey')
merged_df
```

	OrderDate	StockDate	OrderNumber	ProductKey	CustomerKey	TerritoryKey	OrderLineItem	OrderQuantity	Prefix	FirstName	LastName	BirthDate
0	1/1/2015	9/21/2001	SO45080	332	14657	1	1	1	MR.	JOHN	THOMAS	11/11/1958
1	1/9/2017	11/22/2003	SO61768	485	14657	1	2	2	MR.	JOHN	THOMAS	11/11/1958
2	1/9/2017	12/20/2003	SO61768	215	14657	1	3	1	MR.	JOHN	THOMAS	11/11/1958
3	1/9/2017	10/1/2003	SO61768	352	14657	1	1	1	MR.	JOHN	THOMAS	11/11/1958
4	1/4/2015	9/15/2001	SO45098	310	29167	1	1	1	MRS.	DAWN	SHEN	3/12/1959

Finally I combined sales and customer dataset with territory dataset and named the final combined dataset as merged_df. This final merged_df contains data from sales, customers and territory tables.

Sorting dataframe based on CustomerKey and reindexing it

```
df = merged_df.sort_values(by='CustomerKey', ascending=False)
df.head()
```

	OrderDate	StockDate	OrderNumber	ProductKey	CustomerKey	TerritoryKey	OrderLineItem	OrderQuantity	Prefix	FirstName	LastName	BirthDate	M
46652	3/13/2016	12/23/2002	SO49665	360	29483	7	1	1	MR.	JÉSUS	NAVARRO	12/8/1959	
46716	3/22/2016	12/10/2002	SO49746	358	29482	7	1	1	MR.	CLAYTON	ZHANG	3/5/1959	
50716	2/13/2015	11/15/2001	SO45427	349	29481	8	1	1	MR.	IVAN	SURI	1/5/1960	

```
df = df.reset_index(drop=True)
df.head()
```

	OrderDate	StockDate	OrderNumber	ProductKey	CustomerKey	TerritoryKey	OrderLineItem	OrderQuantity	Prefix	FirstName	LastName	BirthDate	Marita
0	3/13/2016	12/23/2002	SO49665	360	29483	7	1	1	MR.	JÉSUS	NAVARRO	12/8/1959	
1	3/22/2016	12/10/2002	SO49746	358	29482	7	1	1	MR.	CLAYTON	ZHANG	3/5/1959	
2	2/13/2015	11/15/2001	SO45427	349	29481	8	1	1	MR.	IVAN	SURI	1/5/1960	

Then I sorted the final dataset based on the CustomerKey field and reset its index.

Data Cleansing and fixing column values:

```
df.isna().sum()
```

```
OrderDate           0
StockDate           0
OrderNumber         0
ProductKey          0
CustomerKey         0
TerritoryKey        0
OrderLineItem       0
OrderQuantity       0
Prefix            380
FirstName           0
LastName            0
BirthDate           0
MaritalStatus       0
Gender            380
```

Once I combined all of the datasets, I looked for nulls in the dataset and prefix and found that genders contained 380 nulls in it.

```
df.dropna(inplace=True)
df.isna().sum()
```

```
OrderDate         0
StockDate         0
OrderNumber       0
ProductKey        0
CustomerKey       0
TerritoryKey      0
OrderLineItem     0
OrderQuantity     0
Prefix            0
```

Since we have very few rows in the dataset that have nulls in them, I used the df.dropna () method to remove nulls from the dataset.

Dropping Duplicates

```
df = df.drop_duplicates()
```

Once the nulls are removed, I also removed duplicates from the dataset.

Converting datacolumns in datatype

```
df['OrderDate'] = pd.to_datetime(df['OrderDate'])
df['StockDate'] = pd.to_datetime(df['StockDate'])
df['BirthDate'] = pd.to_datetime(df['BirthDate'])
```

I changed the datatype of order Date, Stock Date and Birthdate to date time field. These fields were recognized as strings in pandas by default but they are date time fields.

```
df['AnnualIncome'] = df['AnnualIncome'].str.replace(r'\D', '', regex=True).astype(int)
df.head()
```

Then I removed the str.replace function to remove $ sign from the income column in Pandas data frame.

```
from datetime import datetime
current_date = datetime.now()
df['CustomerAge'] = (current_date - df['BirthDate']).astype('<m8[Y]')
df.head()
```

I added the age column to the dataset by using the Birth date field and current date. This customer age field shows the ages of their customers.

Creating Fullname column

```
: df['Full Name'] = df['FirstName'] + ' ' + df['LastName']
```

Finally the full name field combines the first and last name of the customers.

3. Data Analysis:

To perform the analysis on the dataset. I formulated the question that I wanted to analyze using this dataset. I started the analysis by performing a basic overview of the dataset and then answered the business questions using analysis techniques.

```
df.shape
```

```
(55666, 25)
```

Here I looked for the number of columns and rows in this dataset. This dataset contains 55666 rows and 25 columns in total.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 55666 entries, 0 to 56045
Data columns (total 25 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   OrderDate       55666 non-null  datetime64[ns]
 1   StockDate       55666 non-null  datetime64[ns]
 2   OrderNumber     55666 non-null  object
 3   ProductKey      55666 non-null  int64
 4   CustomerKey     55666 non-null  int64
 5   TerritoryKey    55666 non-null  int64
 6   OrderLineItem   55666 non-null  int64
```

Then I used the df.info () method to find out the data type of every column in the dataset and the total size of this dataset.

```
df.describe()
```

	ProductKey	CustomerKey	TerritoryKey	OrderLineItem	OrderQuantity	AnnualIncome	TotalChildren	CustomerAge
count	55666.000000	55666.000000	55666.000000	55666.000000	55666.000000	55666.000000	55666.000000	55666.000000
mean	438.917274	18843.083929	6.250835	1.903083	1.501545	59789.817842	1.842777	60.979377
std	118.642511	5411.401983	2.959660	1.020988	0.612372	33064.869016	1.619084	11.019229
min	214.000000	11000.000000	1.000000	1.000000	1.000000	10000.000000	0.000000	42.000000
25%	360.000000	14018.000000	4.000000	1.000000	1.000000	30000.000000	0.000000	53.000000
50%	479.000000	18155.000000	7.000000	2.000000	1.000000	60000.000000	2.000000	59.000000
75%	529.000000	23422.000000	9.000000	2.000000	2.000000	80000.000000	3.000000	68.000000
max	606.000000	29483.000000	10.000000	8.000000	3.000000	170000.000000	5.000000	113.000000

Following up from this, I used the df.describe () method on the dataset. I used this to provide a concise overview of the basic statistics for each numerical column to provide a summary of the data's distribution (W3Schools. (n.d.)).

Average Age of customers based on gender and marital status

```
# Replacing M with Male and F with female in Gender column
gender_mapping = {'F': 'Female', 'M': 'Male'}
df['Gender'] = df['Gender'].replace(gender_mapping)
# Replacing S with Single and M with Married in MaritalStatus column
marital_status_mapping = {'S': 'Single', 'M': 'Married'}
# Replace values using the dictionary
df['MaritalStatus'] = df['MaritalStatus'].replace(marital_status_mapping)
```

```
df.groupby("Gender").mean()['CustomerAge']
```

```
Gender
Female    60.924697
Male      61.033130
Name: CustomerAge, dtype: float64
```

```
df.groupby("MaritalStatus").mean()['CustomerAge']
```

```
MaritalStatus
Married    63.011052
Single     58.485976
Name: CustomerAge, dtype: float64
```

This very first analysis I wanted to perform on this dataset included finding out the age of customers based on gender and marital status. I started this by replacing M in the Gender column with Male, F in Gender column with Female, S in marital status with Single and M with Married using a dictionary mapping technique. I chose to use a dictionary over a list as it enhances code clarity and provides a seamless way of mapping gender and marital status values while also allowing easy additions in case I need to modify in future. By also using a key for the code with the use of the dictionary, lookups will be faster than if i was to use a list (Dubois, 1996).

Average Age of customers based on gender and marital status

```
In [21]: # Replacing M with Male and F with female in Gender column
         gender_mapping = {'F': 'Female', 'M': 'Male'}
         df['Gender'] = df['Gender'].replace(gender_mapping)
         # Replacing S with Single and M with Married in MaritalStatus column
         marital_status_mapping = {'S': 'Single', 'M': 'Married'}
         # Replace values using the dictionary
         df['MaritalStatus'] = df['MaritalStatus'].replace(marital_status_mapping)

In [22]: df.groupby("Gender").mean()['CustomerAge']

Out[22]: Gender
         Female    60.928103
         Male      61.036728
         Name: CustomerAge, dtype: float64

In [23]: df.groupby("MaritalStatus").mean()['CustomerAge']

Out[23]: MaritalStatus
         Married    63.014377
         Single     58.489697
         Name: CustomerAge, dtype: float64
```

Then I used group by operations to find out the average age. I found that the average age of their female customers is 60 and male is 61. The average age for married customers is 63 and single is 58.

```
def categorize_children(total_children):
    if total_children == 0:
        return 'No Children'
    else:
        return 'Children'

df['Children Group'] = df['TotalChildren'].apply(categorize_children)
df.head()
```

	OrderDate	StockDate	OrderNumber	ProductKey	CustomerKey	TerritoryKey	OrderLineItem	OrderQuantity	Prefix	FirstName	LastName	BirthDate	MaritalS
0	2016-03-13	2002-12-23	SO49665	360	29483	7	1	1	MR.	JÉSUS	NAVARRO	1959-12-08	M
1	2016-03-22	2002-12-10	SO49746	358	29482	7	1	1	MR.	CLAYTON	ZHANG	1959-03-05	M
2	2015-02-13	2001-11-15	SO45427	349	29481	8	1	1	MR.	IVAN	SURI	1960-01-05	
3	2017-01-18	2003-11-11	SO62341	223	29480	10	5	3	MRS.	NINA	RAJI	1960-11-10	
4	2017-01-18	2003-11-25	SO62341	562	29480	10	1	1	MRS.	NINA	RAJI	1960-11-10	

```
df['Children Group'].value_counts()
```

```
Children       39866
No Children    15800
Name: Children Group, dtype: int64
```

The second analysis I wanted to perform on this dataset includes finding how many customers have children and how many don't have children. To achieve this I created a function in python that takes the total children column as input and returns no children if the total children is 0 and returns children if the total children is greater than 0. Based on the analysis we can conclude 39866 customers have children and 15800 customers don't have children.

How many customers have low, medium, high and very high income

```
income_groups = []

for income in df['AnnualIncome']:
    if income < 50000:
        income_groups.append('Low')
    elif 50000 <= income < 100000:
        income_groups.append('Medium')
    elif 100000 <= income < 150000:
        income_groups.append('High')
    else:
        income_groups.append('Very High')

# Add the income groups as a new column in the DataFrame
df['Income Group'] = income_groups
```

```
df['Income Group'].value_counts()
```

```
Medium       25637
Low          22618
High          6292
Very High     1119
Name: Income Group, dtype: int64
```

The third analysis I performed on this dataset includes finding out how many customers have low, medium, high and very high income. To perform this analysis I used loop and control flow statements in python to create a new column that categorizes the income column based on the conditions and returns a new column.

From the analysis we can observe most of the customers belong to the medium income category, followed by low income customers. We have 6292 customers who have an annual income between 100-150k and 1119 customers have an income above 150k.

```
df.groupby(['Full Name']).sum()['OrderQuantity'].sort_values(ascending=False)
```

```
Full Name
JENNIFER SIMMONS        106
FERNANDO BARNES         106
SAMANTHA JENKINS        102
ASHLEY HENDERSON        100
APRIL SHAN               99
                       ...
OSCAR BUTLER              1
SAMUEL ALEXANDER          1
DANNY RUBIO               1
ALEXANDRIA HENDERSON      1
ELIJAH CAMPBELL           1
Name: OrderQuantity, Length: 17221, dtype: int64
```

Here I wanted to analyze the customers based on order quantity. We can observe that Jennifer Simmons, Fernando Barnes and Samantha Jenkins are the top 3 customers based on order quantity.

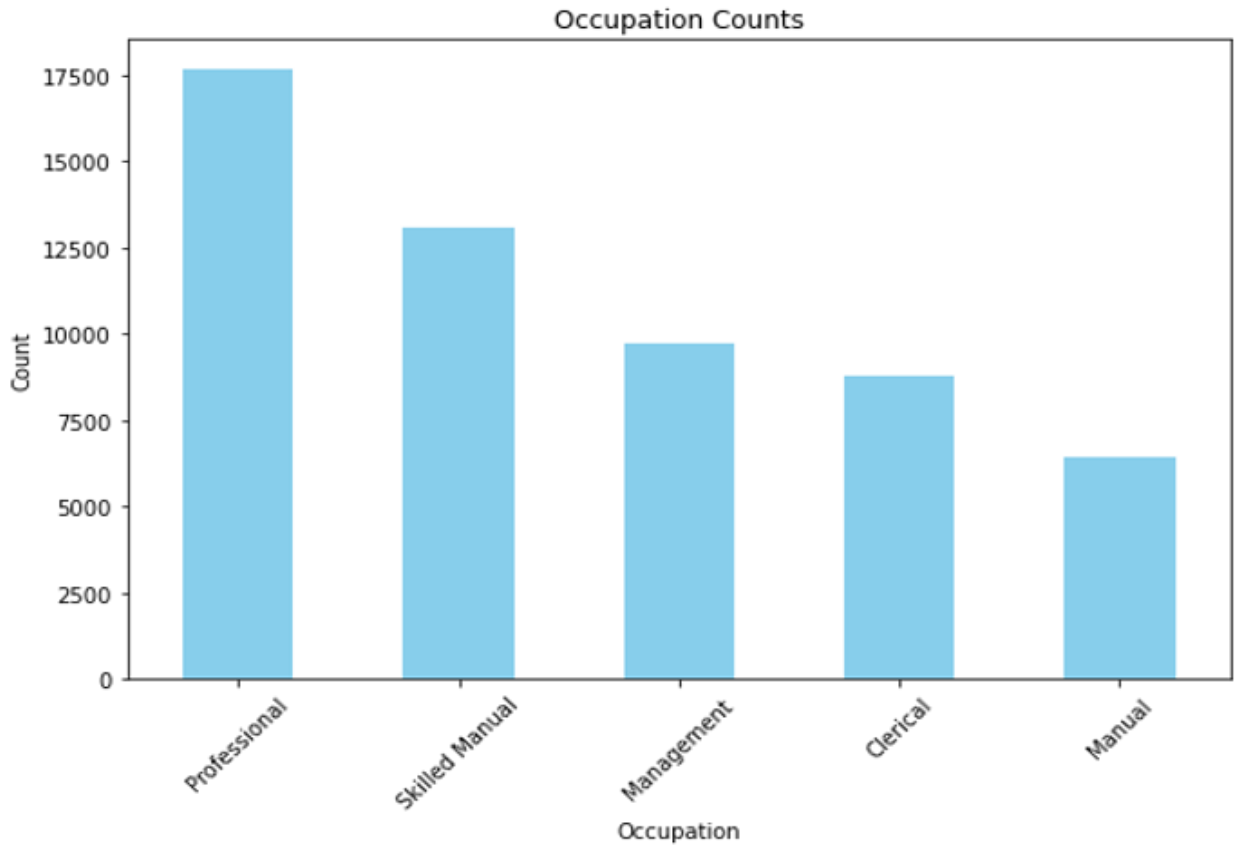
```
owns_house_count = df['HomeOwner'].value_counts()['Y']
print("Number of people who own a house:", owns_house_count)
```

```
Number of people who own a house: 38412
```

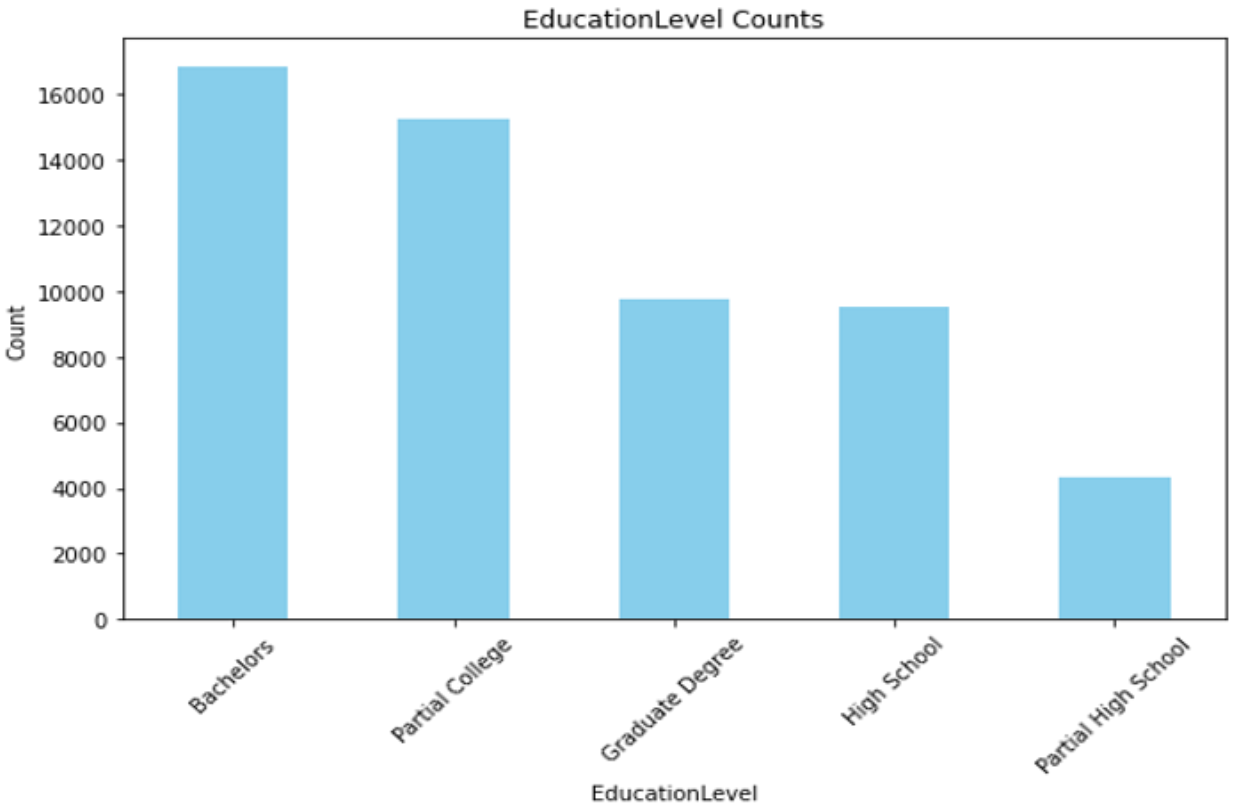
Finally I wanted to analyze how many customers owned homes. To compute how many customers owned their houses I used the value counts () function.

Results:

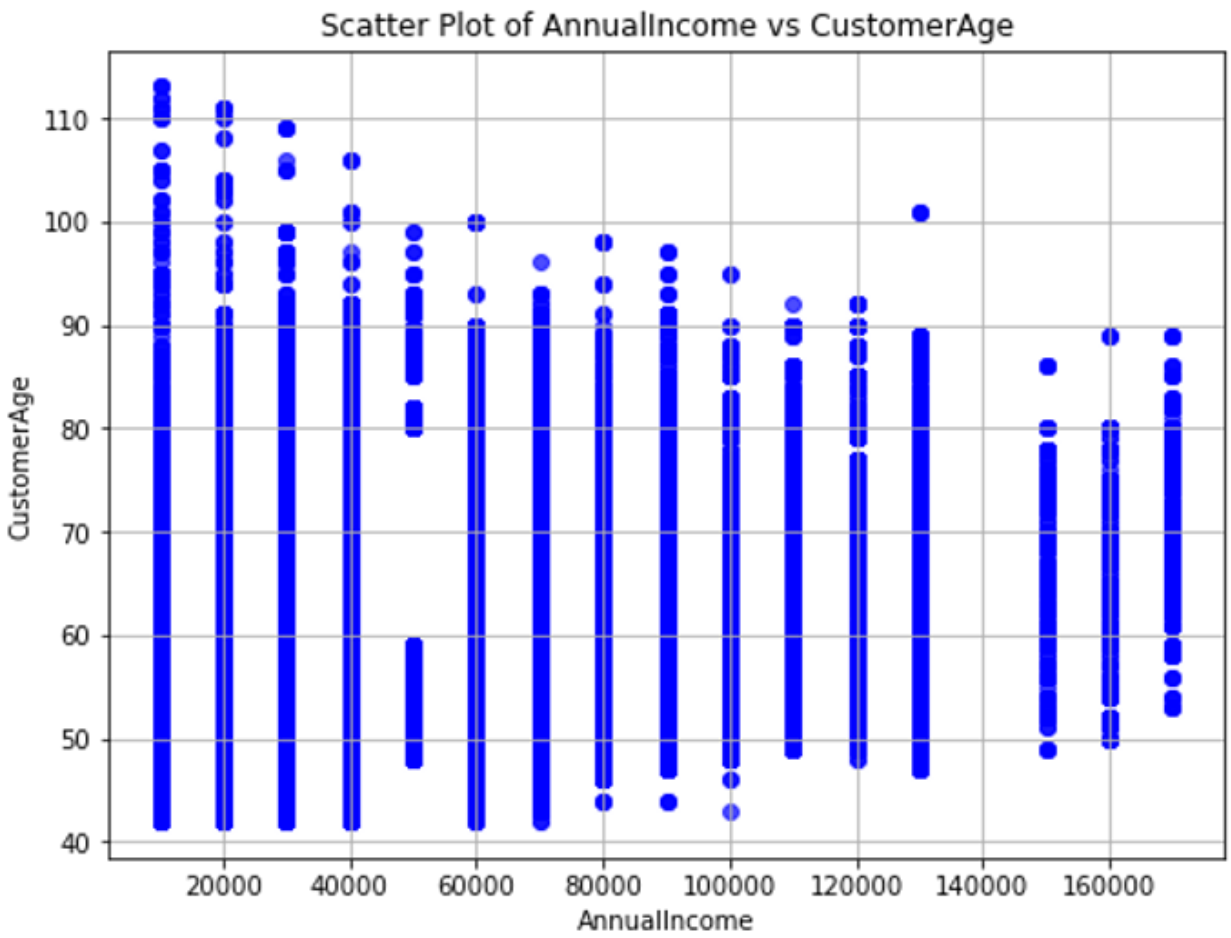
Once I completed the data analysis I created visualization to find out further insights from the dataset.



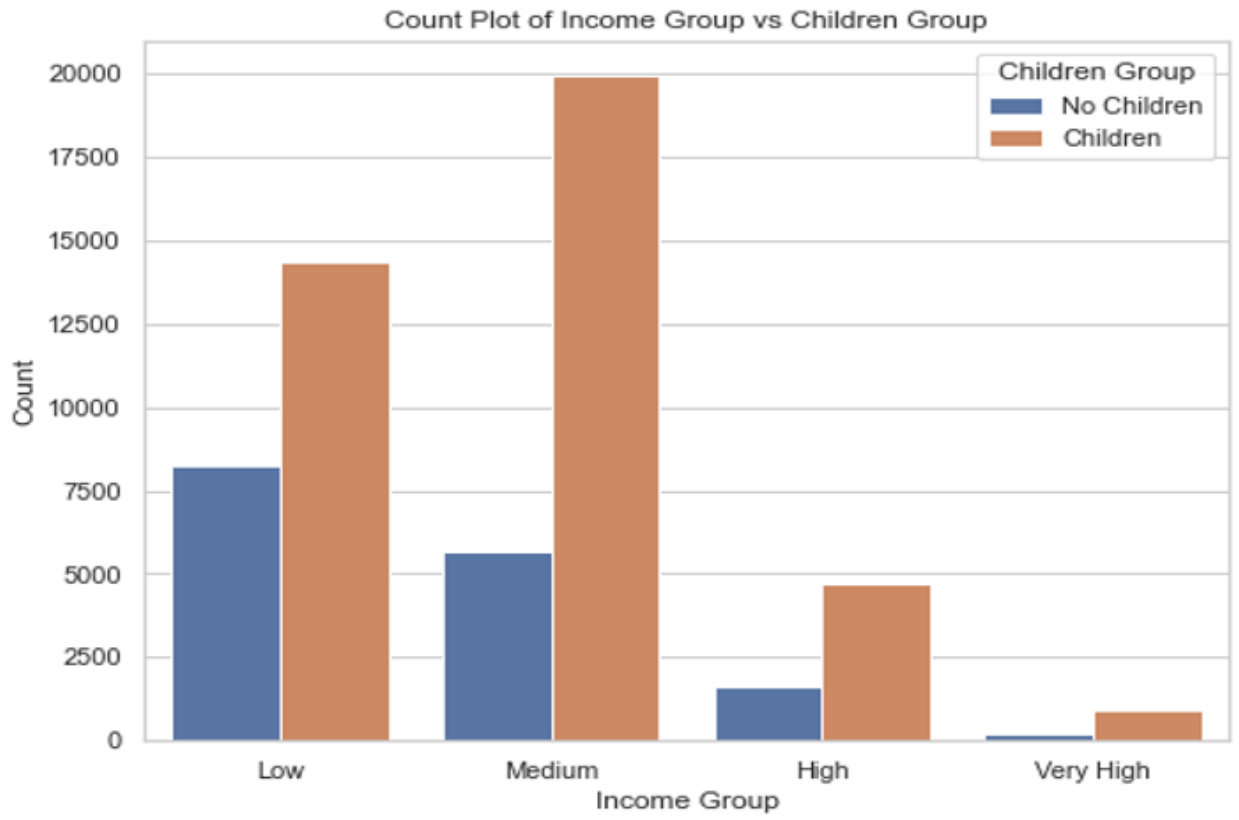
The first visualization I created counts the number of customers based on their occupation. We can observe most of the customers are professionals followed by skilled manual and management. We can see that the customers who have clerical and manual labor as their occupation are our least visited types of guests.



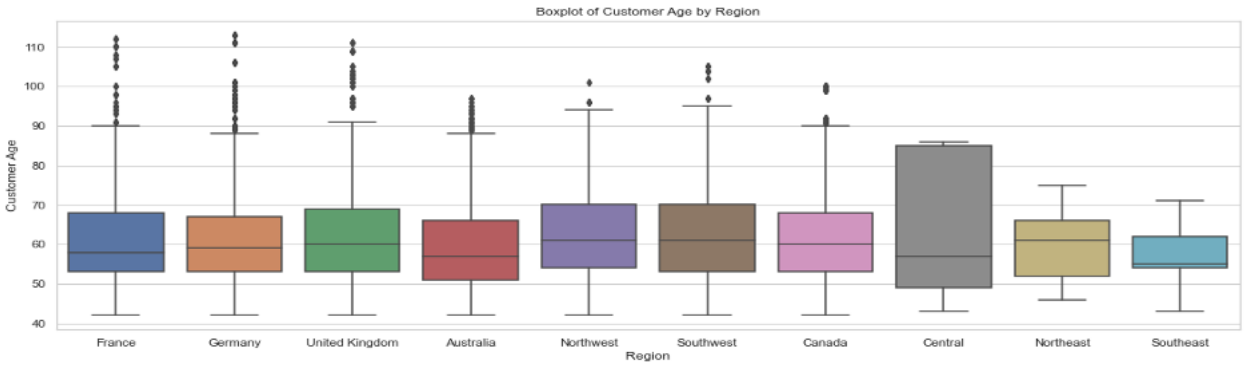
Based on the education level most customers have bachelors as their education level followed by partial college. Graduate degrees and high school seem to be tied in the middle, with the least amount of customers having partial high school as their highest education level.



This scatterplot shows the relationship between the annual income of customers and their age. We can't observe a strong positive or negative relationship between customer age and annual income. We can see that customers with the highest annual income are aged between 50 and 80.



This side by side bar chart shows the number of customers based on income group and children status. We can see that in this dataset there is no direct correlation between income and having children. For example, consumers with very high income had the lowest amount of children in comparison with consumers with low income that had a significant amount of children.



The above box plots shows the distribution of customer age based on the Region.

Insights:

1. The majority of AdventureWorks clientele is individuals with children. Consequently, there exists an opportunity for the company to conceptualize and offer products tailored to the preferences of children.

2. Most of the customers have bachelors and partial college in their educational background.

3. A significant portion of the customer base is constituted by individuals from low and moderate income brackets. In light of this, there is a strategic avenue for the company to enhance its revenue by implementing initiatives such as discounts and coupons targeted specifically towards this demographic.

4. Most of the customers of the company are professionals followed by manual skilled individuals.

5. Majority of the high income customers are aged between 50 to 80. The company can offer them specific products to increase revenue.

Machine Learning:

I want to create a smart machine learning model that predicts how old customers are based on different things like where they're from and what they do. This will help me show the right ads to the right people, depending on their age. To do this, I'll collect data about customers and use it to build a model that guesses their age. Since age is a numeric value, I will develop a regression model.

References:

Dubois, P.F., Hinsen, K. and Hugunin, J., 1996. Numerical python. *Computers in Physics*, *10*(3), pp.262-267.

Pandas DataFrame describe() Method. Retrieved from https://www.w3schools.com/python/pandas/ref_df_describe.asp#:~:text=The%20describe()%20method%20returns,The%20average%20(mean)%20value.