# Detailed Report on the Adventure Works ETL Assignment

-- Sayan Das

## 1. Data Extraction

I began by loading the data from the provided Excel files. The data consisted of six dimension tables (DimCustomer, DimEmployee, DimGeography, DimProduct, DimReseller, and DimSalesTerritory) and two fact tables (FactInternetSales and FactResellerSales). In this phase, I ensured that the correct data types were applied to each column by specifying the dtype argument and using parse\_dates for date columns.

**Code Snippet: Loading Dimension Tables**

print("Loading dimension tables...")  
  
dim\_customer\_df = pd.read\_excel(  
 'DimTables.xlsx',  
 sheet\_name='DimCustomer',  
 dtype={  
 'CustomerKey': 'Int64',  
 'GeographyKey': 'Int64',  
 'CustomerName': 'string',  
 'BirthDate': 'string',  
 'MaritalStatus': 'string',  
 'Gender': 'string',  
 'EmailAddress': 'string',  
 'YearlyIncome': 'float',  
 'Education': 'string',  
 'Occupation': 'string',  
 'HouseOwnerFlag': 'string',  
 'Address': 'string',  
 'FirstPurchaseDate': 'string'  
 }  
)  
  
#Similar for all other tables

**Code Snippet: Loading Fact Tables**

print("Loading fact tables...")  
  
fact\_internet\_sales\_df = pd.read\_excel(  
 'FactInternetSales.xlsx',  
 dtype={  
 'ProductKey': 'Int64',  
 'CustomerKey': 'Int64',  
 'SalesTerritoryKey': 'Int64',  
 'SalesOrderNumber': 'string',  
 'SalesOrderLineNumber': 'Int64',  
 'DiscountAmount': 'float',  
 'TotalProductCost': 'float',  
 'SalesAmount': 'float',  
 'Freight': 'float',  
 'CarrierTrackingNumber': 'string', # Will remove  
 },  
 parse\_dates=['OrderDate', 'DueDate', 'ShipDate']  
)  
  
fact\_reseller\_sales\_df = pd.read\_excel(  
 'FactResellerSales.xlsx',  
 dtype={  
 'ProductKey': 'Int64',  
 'ResellerKey': 'Int64',  
 'EmployeeKey': 'Int64',  
 'SalesTerritoryKey': 'Int64',  
 'SalesOrderNumber': 'string',  
 'SalesOrderLineNumber': 'Int64',  
 'DiscountAmount': 'float',  
 'TotalProductCost': 'float',  
 'SalesAmount': 'float',  
 'Freight': 'float',  
 'CarrierTrackingNumber': 'string',  
 },  
 parse\_dates=['OrderDate', 'DueDate', 'ShipDate']  
)

## 2. Data Transformation

After extraction, I performed several key transformations:

* **Removal of Unnecessary Columns:**  
  I removed the CarrierTrackingNumber column from the FactInternetSales table since it was not fully empty.
* if 'CarrierTrackingNumber' in fact\_internet\_sales\_df.columns:  
   fact\_internet\_sales\_df.drop(columns=['CarrierTrackingNumber'], inplace=True)  
   print("Removed 'CarrierTrackingNumber' column from FactInternetSales.")
* **Handling Missing Data:**  
  I applied specific rules to handle missing values:
  + For **DimProduct**, missing values in the Color column were replaced with 'NA'.
  + For **DimSalesTerritory**, any row with missing data was dropped.
  + For the fact tables (**FactInternetSales** and **FactResellerSales**), rows with any missing values were dropped.
* # DimProduct: fill missing 'Color' with 'NA'  
  if 'Color' in dim\_product\_df.columns:  
   missing\_color = dim\_product\_df['Color'].isnull().sum()  
   if missing\_color > 0:  
   print(f"Filling {missing\_color} missing 'Color' cells in DimProduct with 'NA'.")  
   dim\_product\_df['Color'] = dim\_product\_df['Color'].fillna('NA')  
    
  # DimSalesTerritory: drop rows with any missing values  
  before\_dst = dim\_salesterritory\_df.shape[0]  
  dim\_salesterritory\_df.dropna(inplace=True)  
  after\_dst = dim\_salesterritory\_df.shape[0]  
  print(f"Dropped {before\_dst - after\_dst} row(s) from DimSalesTerritory due to missing data.")  
    
  # Fact tables: drop rows with any missing values  
  before\_fis = fact\_internet\_sales\_df.shape[0]  
  fact\_internet\_sales\_df.dropna(inplace=True)  
  print(f"Dropped {before\_fis - fact\_internet\_sales\_df.shape[0]} row(s) from FactInternetSales due to missing data.")  
    
  before\_frs = fact\_reseller\_sales\_df.shape[0]  
  fact\_reseller\_sales\_df.dropna(inplace=True)  
  print(f"Dropped {before\_frs - fact\_reseller\_sales\_df.shape[0]} row(s) from FactResellerSales due to missing data.")

## 3. Data Quality Checks

To ensure high-quality data, I implemented two key checks:

* **Primary Key Uniqueness:**  
  I verified that each dimension table had unique primary keys, dropping duplicate rows when necessary.
* def check\_uniqueness(df, key\_column, table\_name):  
   before = df.shape[0]  
   df\_clean = df.drop\_duplicates(subset=[key\_column])  
   after = df\_clean.shape[0]  
   dropped = before - after  
   if dropped > 0:  
   print(f"{dropped} duplicate row(s) dropped from {table\_name} based on primary key '{key\_column}'.")  
   else:  
   print(f"All rows in {table\_name} have a unique '{key\_column}'.")  
   return df\_clean  
    
  print("\n-- Checking PK uniqueness in dimension tables --")  
  dim\_customer\_df = check\_uniqueness(dim\_customer\_df, 'CustomerKey', 'DimCustomer')  
  dim\_employee\_df = check\_uniqueness(dim\_employee\_df, 'EmployeeKey', 'DimEmployee')  
  dim\_geography\_df = check\_uniqueness(dim\_geography\_df, 'GeographyKey', 'DimGeography')  
  dim\_product\_df = check\_uniqueness(dim\_product\_df, 'ProductKey', 'DimProduct')  
  dim\_reseller\_df = check\_uniqueness(dim\_reseller\_df, 'ResellerKey', 'DimReseller')  
  dim\_salesterritory\_df = check\_uniqueness(dim\_salesterritory\_df, 'SalesTerritoryKey', 'DimSalesTerritory')
* **Foreign Key Validation:**  
  I validated that each foreign key in the fact tables matched a primary key in the corresponding dimension table. Any fact row with an invalid reference was dropped.
* def validate\_fk(fact\_df, dim\_df, fact\_fk, dim\_pk, fact\_table\_name, dim\_table\_name):  
   valid\_ids = set(dim\_df[dim\_pk].unique())  
   before = fact\_df.shape[0]  
   fact\_df\_clean = fact\_df[fact\_df[fact\_fk].isin(valid\_ids)]  
   after = fact\_df\_clean.shape[0]  
   dropped = before - after  
   if dropped > 0:  
   print(f"{dropped} row(s) dropped from {fact\_table\_name} due to invalid '{fact\_fk}' not found in {dim\_table\_name}.")  
   else:  
   print(f"All rows in {fact\_table\_name} have a valid foreign key '{fact\_fk}'.")  
   return fact\_df\_clean  
    
  print("\n-- Validating foreign keys in FactInternetSales --")  
  fact\_internet\_sales\_df = validate\_fk(  
   fact\_internet\_sales\_df,  
   dim\_customer\_df,  
   'CustomerKey',  
   'CustomerKey',  
   'FactInternetSales',  
   'DimCustomer'  
  )  
  fact\_internet\_sales\_df = validate\_fk(  
   fact\_internet\_sales\_df,  
   dim\_product\_df,  
   'ProductKey',  
   'ProductKey',  
   'FactInternetSales',  
   'DimProduct'  
  )  
  fact\_internet\_sales\_df = validate\_fk(  
   fact\_internet\_sales\_df,  
   dim\_salesterritory\_df,  
   'SalesTerritoryKey',  
   'SalesTerritoryKey',  
   'FactInternetSales',  
   'DimSalesTerritory'  
  )  
    
  print("\n-- Validating foreign keys in FactResellerSales --")  
  fact\_reseller\_sales\_df = validate\_fk(  
   fact\_reseller\_sales\_df,  
   dim\_reseller\_df,  
   'ResellerKey',  
   'ResellerKey',  
   'FactResellerSales',  
   'DimReseller'  
  )  
  fact\_reseller\_sales\_df = validate\_fk(  
   fact\_reseller\_sales\_df,  
   dim\_employee\_df,  
   'EmployeeKey',  
   'EmployeeKey',  
   'FactResellerSales',  
   'DimEmployee'  
  )  
  fact\_reseller\_sales\_df = validate\_fk(  
   fact\_reseller\_sales\_df,  
   dim\_product\_df,  
   'ProductKey',  
   'ProductKey',  
   'FactResellerSales',  
   'DimProduct'  
  )  
  fact\_reseller\_sales\_df = validate\_fk(  
   fact\_reseller\_sales\_df,  
   dim\_salesterritory\_df,  
   'SalesTerritoryKey',  
   'SalesTerritoryKey',  
   'FactResellerSales',  
   'DimSalesTerritory'  
  )

## 4. Loading the Transformed Data to MySQL

After completing the transformations and quality checks, I loaded the cleaned data into a MySQL database. The database is structured with six dimension tables and two fact tables, forming a star schema optimized for reporting and analytics.

**Code Snippet: Loading Data to MySQL**

print("\n-- Loading data into MySQL --")  
dim\_customer\_df.to\_sql('dimcustomer', con=engine, if\_exists='replace', index=False)  
dim\_employee\_df.to\_sql('dimemployee', con=engine, if\_exists='replace', index=False)  
dim\_geography\_df.to\_sql('dimgeography', con=engine, if\_exists='replace', index=False)  
dim\_product\_df.to\_sql('dimproduct', con=engine, if\_exists='replace', index=False)  
dim\_reseller\_df.to\_sql('dimreseller', con=engine, if\_exists='replace', index=False)  
dim\_salesterritory\_df.to\_sql('dimsalesterritory', con=engine, if\_exists='replace', index=False)  
  
fact\_internet\_sales\_df.to\_sql('fact\_internetsales', con=engine, if\_exists='replace', index=False)  
fact\_reseller\_sales\_df.to\_sql('fact\_resellersales', con=engine, if\_exists='replace', index=False)

## Output

Loading dimension tables...

Loading fact tables...

Removed 'CarrierTrackingNumber' column from FactInternetSales.

-- Checking PK uniqueness in dimension tables --

All rows in DimCustomer have a unique 'CustomerKey'.

All rows in DimEmployee have a unique 'EmployeeKey'.

All rows in DimGeography have a unique 'GeographyKey'.

All rows in DimProduct have a unique 'ProductKey'.

All rows in DimReseller have a unique 'ResellerKey'.

All rows in DimSalesTerritory have a unique 'SalesTerritoryKey'.

Filling 56 missing 'Color' cells in DimProduct with 'NA'.

Dropped 1 row(s) from DimSalesTerritory due to missing data.

Dropped 44 row(s) from FactResellerSales due to missing data.

-- Validating foreign keys in FactInternetSales --

All rows in FactInternetSales have a valid foreign key 'CustomerKey'.

All rows in FactInternetSales have a valid foreign key 'ProductKey'.

All rows in FactInternetSales have a valid foreign key 'SalesTerritoryKey'.

-- Validating foreign keys in FactResellerSales --

All rows in FactResellerSales have a valid foreign key 'ResellerKey'.

All rows in FactResellerSales have a valid foreign key 'EmployeeKey'.

All rows in FactResellerSales have a valid foreign key 'ProductKey'.

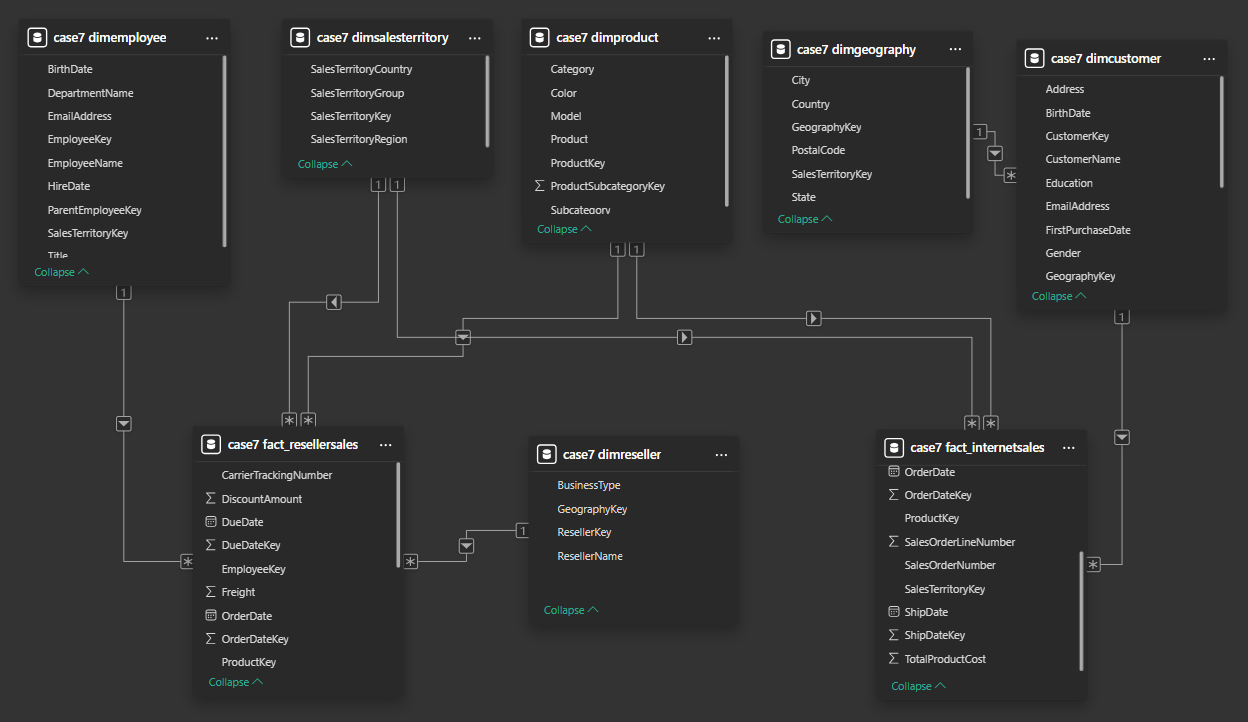
All rows in FactResellerSales have a valid foreign key 'SalesTerritoryKey'.

...

-- Loading data into MySQL --

ETL process completed successfully!

## Schema

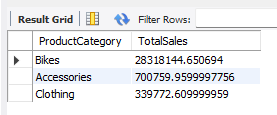


## 5. Analysis Using SQL

Once the data is in MySQL, I can run SQL queries to perform further analysis. For example, here are some SQL commands I used to analyze the data:

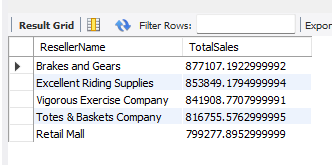
**Total Internet Sales by Product Category**

SELECT p.Category AS ProductCategory,  
 SUM(f.SalesAmount) AS TotalSales  
 FROM fact\_internetsales f  
 JOIN dimproduct p ON f.ProductKey = p.ProductKey  
 GROUP BY p.Category  
 ORDER BY TotalSales DESC;



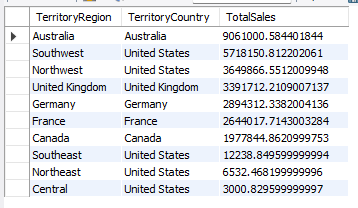
**Top 5 Resellers by Total Sales**

SELECT r.ResellerName,  
 SUM(f.SalesAmount) AS TotalSales  
 FROM fact\_resellersales f  
 JOIN dimreseller r ON f.ResellerKey = r.ResellerKey  
 GROUP BY r.ResellerName  
 ORDER BY TotalSales DESC  
 LIMIT 5;



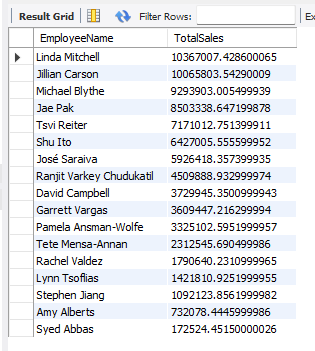
**Total Sales by Territory (Internet Sales)**

SELECT t.SalesTerritoryRegion AS TerritoryRegion,  
 t.SalesTerritoryCountry AS TerritoryCountry,  
 SUM(f.SalesAmount) AS TotalSales  
 FROM fact\_internetsales AS f  
 JOIN dimsalesterritory AS t ON f.SalesTerritoryKey = t.SalesTerritoryKey  
 GROUP BY t.SalesTerritoryRegion, t.SalesTerritoryCountry  
 ORDER BY TotalSales DESC;



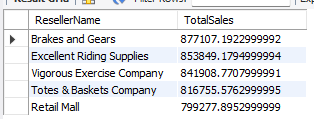
**Reseller Sales by Employee**

SELECT e.EmployeeName,  
 SUM(f.SalesAmount) AS TotalSales  
 FROM fact\_resellersales AS f  
 JOIN dimemployee AS e ON f.EmployeeKey = e.EmployeeKey  
 GROUP BY e.EmployeeName  
 ORDER BY TotalSales DESC;



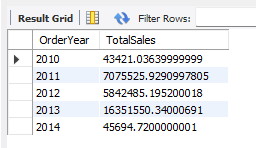
**Top 5 Resellers by Total Sales**

SELECT r.ResellerName,  
 SUM(f.SalesAmount) AS TotalSales  
 FROM fact\_resellersales AS f  
 JOIN dimreseller AS r ON f.ResellerKey = r.ResellerKey  
 GROUP BY r.ResellerName  
 ORDER BY TotalSales DESC  
 LIMIT 5;



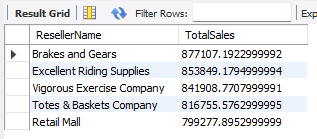
**Yearly Internet Sales Trend**

SELECT YEAR(OrderDate) AS OrderYear,  
 SUM(SalesAmount) AS TotalSales  
 FROM fact\_internetsales  
 GROUP BY YEAR(OrderDate)  
 ORDER BY OrderYear;



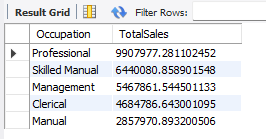
**Top 5 Resellers by Total Sales**

SELECT r.ResellerName,  
 SUM(f.SalesAmount) AS TotalSales  
 FROM fact\_resellersales AS f  
 JOIN dimreseller AS r ON f.ResellerKey = r.ResellerKey  
 GROUP BY r.ResellerName  
 ORDER BY TotalSales DESC  
 LIMIT 5;



**Internet Sales by Customer Occupation**

SELECT c.Occupation,  
 SUM(f.SalesAmount) AS TotalSales  
 FROM fact\_internetsales AS f  
 JOIN dimcustomer AS c ON f.CustomerKey = c.CustomerKey  
 GROUP BY c.Occupation  
 ORDER BY TotalSales DESC;



**Compare Discount vs. Sales (Reseller Channel)**

SELECT SUM(DiscountAmount) AS TotalDiscount,  
 SUM(SalesAmount) AS TotalSales  
 FROM fact\_resellersales;



## 6. KPI Tracking & Monitoring Results

## 

## 

## Conclusion

**Bikes** are the top-selling product category.

**June** is a peak month for both Internet and Reseller sales.

**Age 30–50** is the most active buying demographic.

**Higher-educated** customers (Bachelor’s, Graduate, Partial College) account for a large share of purchases.

**Black, Red, Silver** colors are most popular.