Data Warehousing Project

Nathan Scott: 18913101

26 Apr 18

Task 1 - Business Analysis and Project Description

This project describes a Data Warehouse design done for a fictitious mining company called "ABC Mining". ABC Mining is a mining company that is in the business of extracting and selling high quality iron ore to customers across the globe. The company operates 4 open cut iron ore mines in the North West of Western Australia, a private railway network connecting the mines to Port Headland and a storage and ship loading facility in Port Headland. The company's corporate headquarters are located in Perth, WA.

Disclaimer: All statistics, locations, names and data used within this project is fictitious, any similarities to real-world companies are simply co-incidental or for the purposes of demonstration only. The modeling of the businesses operations is very simplistic and only a rough order of magnitude estimation of a real-world mining company operation.

Business Model

The company's business model is relatively simple:

Profit = Revenue from Selling Ore – Costs from Extracting/Transporting/Storing Ore

Obviously it is in the company's interests to maximize profit, which is a 2 part goal of maximizing Revenue while minimizing Costs.

Maximizing revenue not something that the company has complete control over as it is primarily determined by the global appetite for iron ore and the resultant market prices. However certain controllable factors do help to maximize the revenue such as differentiation of the company's product and services which can help the company compete with other producers and even to charge a premium for certain supply contracts. Product and service differentiation is one key area identified where the data warehousing project can offer some benefit to the business.

Minimizing costs, on the other hand, is an area that the company has much more control over. While they have to work within the laws of Australia, especially in regards to OHS and employee pay and management, they can still control who they hire and how they run their operations. These factors in particular have played into a technology drive as a way to further minimize human and operational costs and this will be another focus area for the data warehousing project.

Description of Operations

The iron ore that the company target's is located within the surface layer of the Earth's crust, typically to a depth not exceeding 200m. To extract this ore the company employs an open cut mining strategy. This involves first mapping of the in-ground deposits through an exploration process, then planning where and when to blast to loosen the material enough for it to be accessed by the heavy machinery, primarily heavy diggers which load the material onto the dump trucks. The scheduling of the pit expansions is a multi-factor puzzle that has to account for variables such as: ore demand in the future, existing and future supply contracts, structural integrity of the cut, the size and quality of the ore deposit and the cost of transporting the ore to the port from the particular mine.

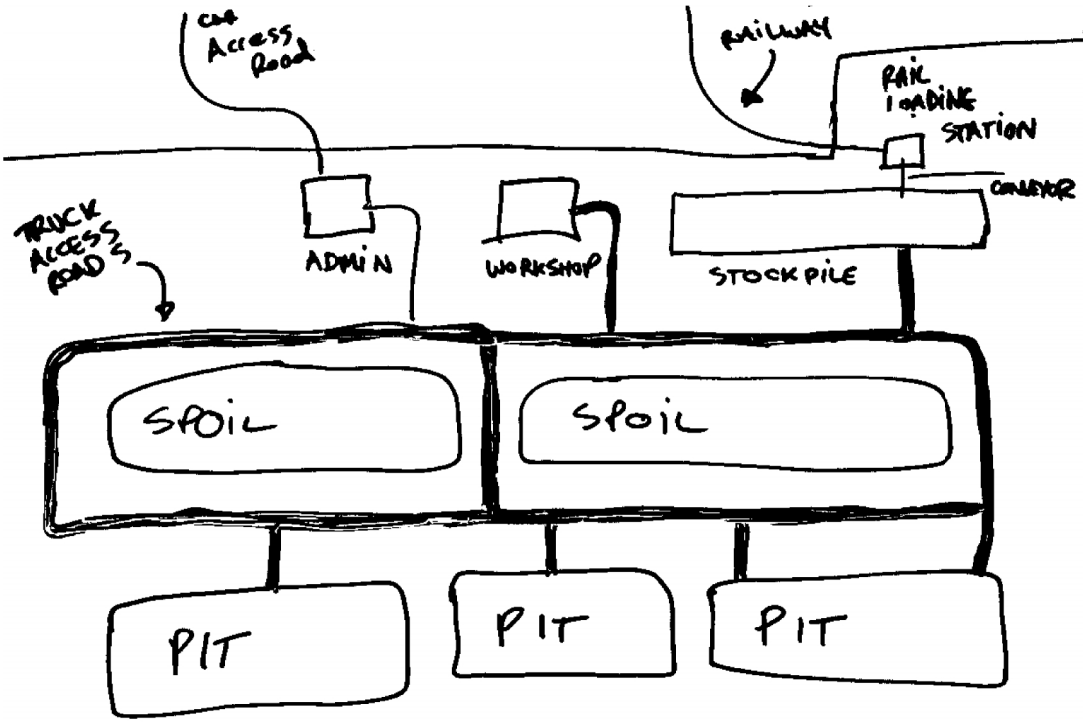


Fig.1 – Typical Mine Layout

Once a blast takes place, the diggers move in to clear access paths for the dump trucks and then start loading the trucks with either spoil or ore for the ore stockpiles. The spoil piles are piles of non-ore bearing earth located between the pits and the ore stockpiles, the spoil piles can reach several hundred meters in height. The ore stockpiles are located on the far side of the spoil piles adjacent to the railway loading station. Here the raw ore is filtered and loaded onto the ore stockpile in long piles reaching up to 50m high and over 1km long. Ore in the stockpile is ready for transport and can be loaded by a conveyor belt system onto trains up to 2400 carriages long. The length and frequency of the trains can be varied daily to meet the current stockpile status and shipping demand. All trains take ore to Port Headland, where another stockpile complex holds the ore before it is loaded onto bulk carriers via conveyor belt. The following diagram shows the relative location of the mines to Port Headland.

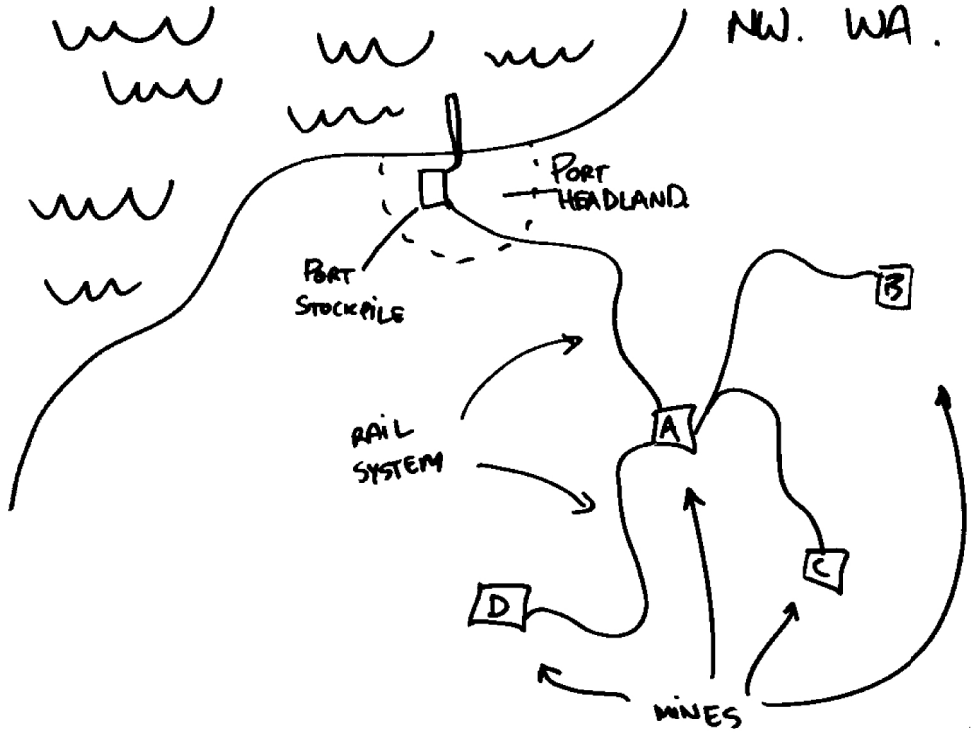


Fig.2 – Mine, Railway and Port Location

Some of the key factors that need to be managed due to their impact on ore production quality, latency and cost are:

- Access roads throughout the mines, these are typically dirt and need to be able to handle trucks weighing in excess of 400 tonnes. The quality of the access roads has been shown to have a high correlation to truck and tire wear and tear, which is a major cost to the business due to the extreme loading and size of the truck parts, eg. A single tire can cost over $50,000.
- Driver management. It has been shown that driver psychology is highly correlated to negative events while driving which cost the company through downtime and repairs.
- Automation. As driverless truck technology has advanced to the point where it is now feasible, this has become a major area of focus for cost savings as it can reduce the reliance on humans for the critical task of ore transport. This can have potentially major costs savings from a reduction in negative events while driving as well as a reduction in repair costs.
- Supply Chain Management. The status of the mine and port stockpiles needs to be better automated and managed and synchronized with the train capacities and mine production rate to minimize stockpile overflow or stockpile underflow conditions which will have an impact on meeting supply targets and can create significant costs. The demand for iron ore will always be somewhat unpredictable and the company needs to be able to ramp-up and down production within reasonable time-frames.

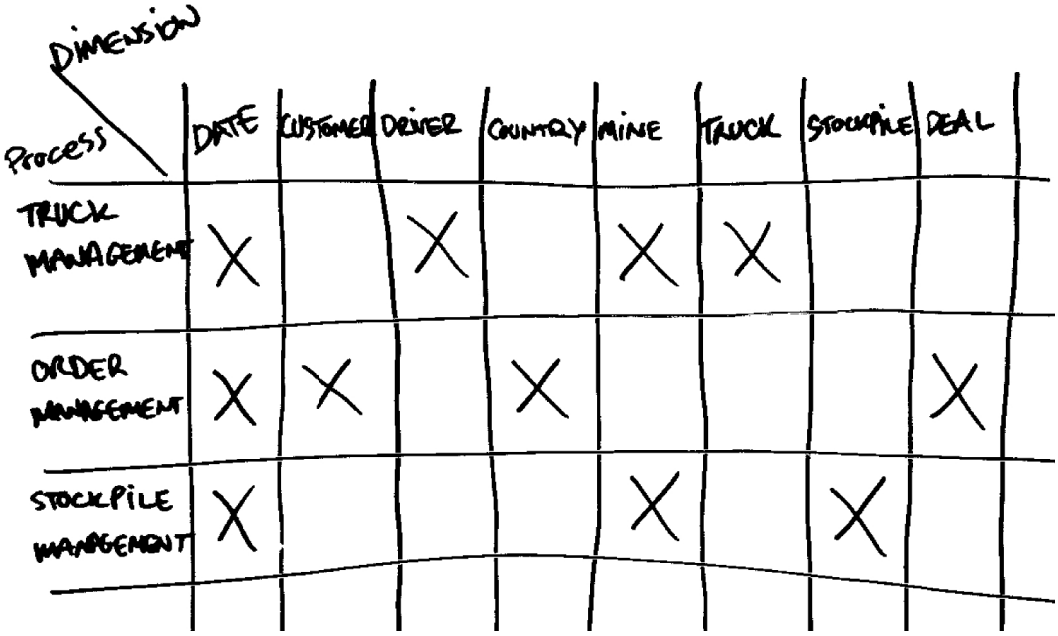
Description of Sales

The other side of the business equation is the Revenue aspect which is dominated by sales and contract negotiations. Demand for Iron Ore is driven by manufacturing and construction. Steel is used in many ways around the world and as such the market price for iron ore rises and falls along with economic activity, globally, as well as regionally. The company employs diversity strategies by developing relationships with customers in different countries, industries of different sizes in order to minimize the cyclical nature of the iron ore market. However, the market is still cyclical and the demand for iron ore will still rise and fall, at times unpredictably. One strategy the company uses successfully to compete and to even generate better than market prices, is to offer superior supply guarantees to competitors. These supply guarantees are primarily concerned with ore quality as well as shipment times. Ore quality is relatively easy to guarantee due to the high quality of the raw material which requires no concentration processing, typically, it only requires basic filtering, however, some input into the extraction scheduling is required as some mines and deposits have higher grades than others, so depending on various factors, the ore supply quality guarantees of a particular contract may affect which mine and which pit is selected for the next blast-extraction cycle. Timely shipments, on the other hand, are a function of the previously mentioned supply chain management system, which does require a close interworking between sales and operations to ensure supply meets demand.

When negotiating contracts with current or new customers, the company's representatives need to be able to estimate the available resources at hand as well as in the future so as to determine the parameters of the contract so that risks can be managed to minimize loss or damages to the company.

Which Business Processes to Model

A Bus Matrix was created for the company and 3 business processes were selected for dimensional modeling as they offer the most immediate and substantial benefit to the management and bottom line of the company.



The Truck Management Business Process

The massive dump trucks used in each mine to haul ore are one of the most critical components behind the company's successes. At the same time, the extreme loads these trucks carry result in extreme costs especially when they are mishandled. Discussions with the company's management highlighted the ongoing costs of running its fleet of trucks as being a serious concern to the bottom line of the business especially in lean times when contracts are being squeezed by lower market prices for iron ore.

The truck fleet of the company comprises 6 models of trucks broadly classed into 2 categories, the ultra class which can haul up to 350 tonnes and the large class which can haul up to 250 tonnes. The trucks are at the extreme end of vehicle manufacturing in terms of strength. The loads that the trucks are subjected to, have consequences for many highly specialized components of the trucks. The main maintenance and repair issues and costs arise from:

- Overloading of the trucks => this causes structural stress resulting in reduced maintenance intervals. It can also impact tire wear which is a major operational cost. This is a loader and driver problem.
- Over-speed, Engine Over-speed, Brake Temperature of the trucks => can seriously impact break, tire and gear wear as well as dramatically increase the risk of accidents. This is a driver problem.
- Tire wall deformations and Tire loss events => tire wall deformations are a general result of aggressive driving, poor road surface and design as well as truck overloading. A tire loss event is when a tire actually fails.

Other metrics that management flagged as important were:

- productivity, i.e. tones of ore delivered to the mine stockpile per day
- fuel burn => how much fuel was being used per tonne of ore delivered
- effectiveness of the truck automation program

Summarizing the discussions, a list of the most pressing questions that management desire to be able to answer and monitor was created, it is:

1. How effective is the truck automation program? What benefit to the bottom line (cost savings) are created by the automated trucks? Quantify this benefit with some metrics such as a reduced dollar cost per tonne of ore delivered. Which aspects of the cost are affected most by automation?
2. What types of human drivers are the best for business? Breakdown human driver performance by gender, age and relationship status. Performance in terms of negative events and cost as well as productivity.
3. Which brand of truck is best for business? Breakdown the productivity and negative events/costs by truck type/manufacturer.
4. What are the productivity, production capacity and costs at the different mines?

The Stockpile and Supply Chain Management Business Process

A major concern for management was to get visibility of the current and historical status of the stockpiled ore. This includes ore that has already been extracted and is stockpiled and ready to move and also in-ground resources that range from currently being extracted to being several tiers away from being blasted. The following diagram is the model of the company's stockpiles.

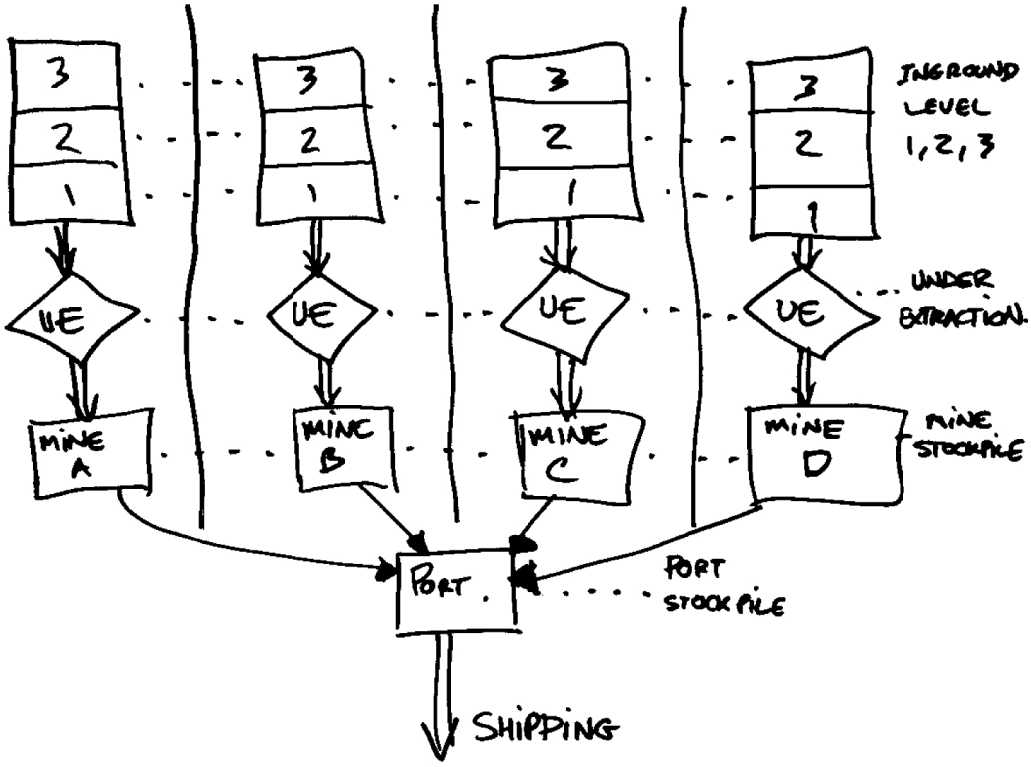


Fig.3 – Stockpile Map

Overtime, there will be incoming resources to the 'In-Ground' stockpiles from new discoveries driven by the exploration process and there will be outgoing resources from the port stockpile to the bulk carriers leaving Port Headland everyday to customers around the world. The in-ground resources do not have any ingress though, they simply move down the chain until they are scheduled for extraction then their level decreases as they ore is removed until they are depleted and disappear. The mine and port stockpiles are, however, standard stockpiles which are essentially buckets with a finite capacity that have constant ingress and egress of ore. The whole supply chain is a process that needs to adapt to constantly changing demand by adjusting the tuning knobs of train capacity, production capacity, scheduling of new blasting and exploration. As shown in the following diagram.

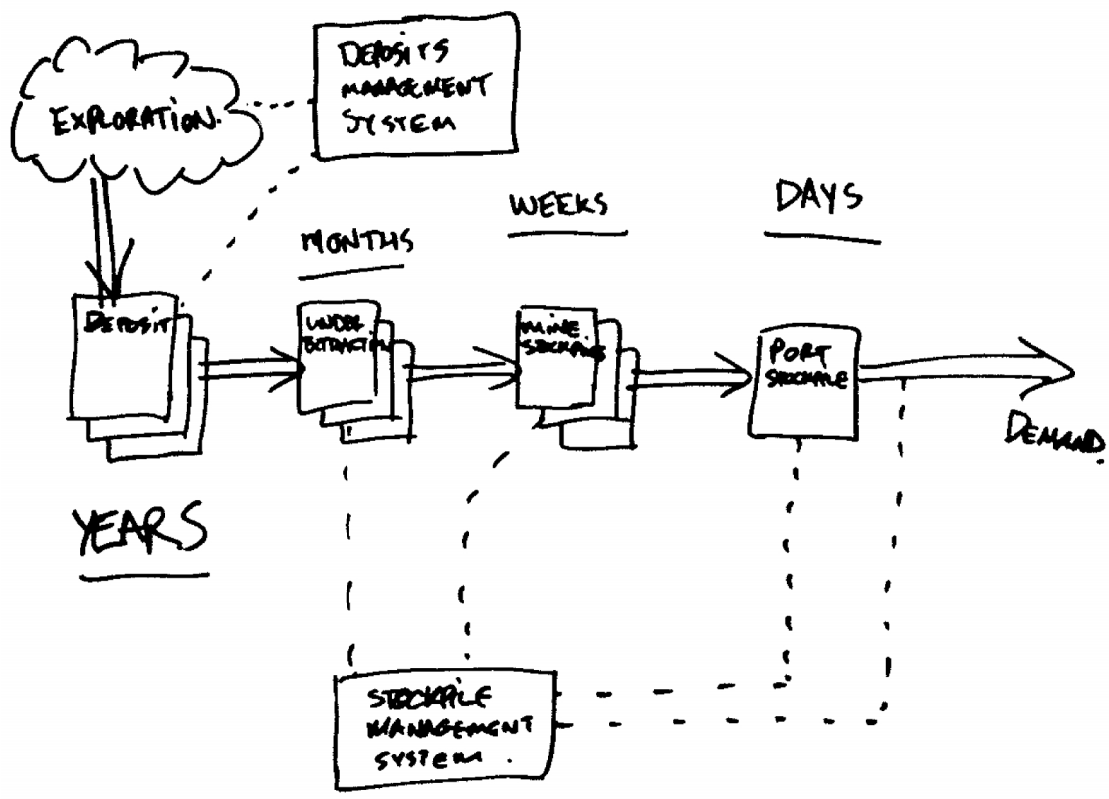


Fig.4 – Ore Supply Chain

The primary concern of management is to ensure that:

1. They know the status of the in-ground resources in terms of lead-time to extraction as well as capacity. This will help in resource planning as well as in contract negotiations.
2. They know the current and historical status of the mine and port stockpiles so as to avoid under-filling/over-filling as well as to feedback this information into the demand, train capacity, mine production capacity, blasting schedule process. This will help the management monitor what has become a significant problem in the past.

The Order Pipeline Business Process

The 3rd area of interest from management was to get analytics on the order pipeline. In the mining industry the ordering process is quite different to retail or any other industry. Customers will typically sign supply agreements with a mining company for an agreed quantity of ore delivered at an agreed price and frequency. One supply contract may last for 10 or more years, the shipment frequency and quantity is usually loosely fixed as well as the price. There is usually room to add to the supply contract and to reduce it. As a result, at a particular time, the order-book is comprised of a series of shipments at scheduled dates running 10-20 even 30 years into the future. This information is the aggregate demand, it is a key parameter in planning the supply chain as previously discussed.

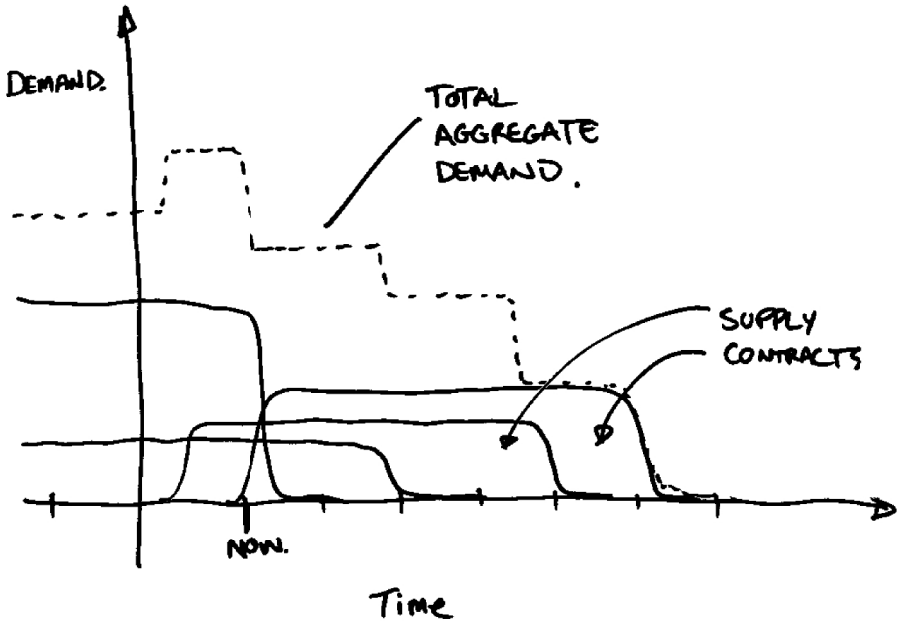


Fig.5 – Order Pipeline

Management highlighted several areas and questions they required visibility on:

1. What is our current demand for the next month, next year, next 10 years? This will help the business to plan production as well as to negotiate contracts.
2. Where is the demand coming from? Where has it been coming from? Where will it come from? This will help the business identify key regions and assign sufficient resources to developing new business.
3. Who are our best/worst customers in terms of profit generated, year on year. This will help the business identify key customers and to develop strategies to strengthen those relationships. Conversely it will enable the management of the more challenging customers in order to limit losses or downside.

Task 2 - Database Design and Creation

The source operational databases for this project where designed around the company's operations systems.

Trucks

3 source operational systems make up the data source for the Truck Management data cube, they are:

1. Truck Logging System – This is where the bulk of the data for the Truck data cube comes from. The truck Logging System comprises a central database and on-board units in every truck. The on-board units in each truck are connected to a multitude of sensors located around the truck, some of key sensors are: GPS location, Speed, Engine Revs and Temp, Gear, Tire Wall Deformity, Brake Temp, Loading and Fuel Level. This data is continuous and is collected with a per second granularity and uploaded to the Truck Logging System real time via the mines wireless network. This enables the support staff to monitor the trucks position and condition and make changes or diagnosis on the fly. Other event based data is also logged and uploaded such as: truck status, check out of depot event, arrival at loading zone event, departing loading zone event, arrival at stockpile event, unscheduled field event.
1. Driver Database – the driver database tracks the vital information, status and performance of the drivers, both autonomous and human. When a driver enters a truck, the log in, when they exit the truck, they log out. The autonomous drivers obviously are tied to the same truck.
2. Workshop Job Tracker – this system manages the truck maintenance, it tracks the performance of each truck via the Truck Logging System and schedules maintenance either periodically or as needed. Out of service times and costs are tracked here.

The outline of the schema for the Truck Logging System, Workshop Job Tracker and Driver DB is shown below:

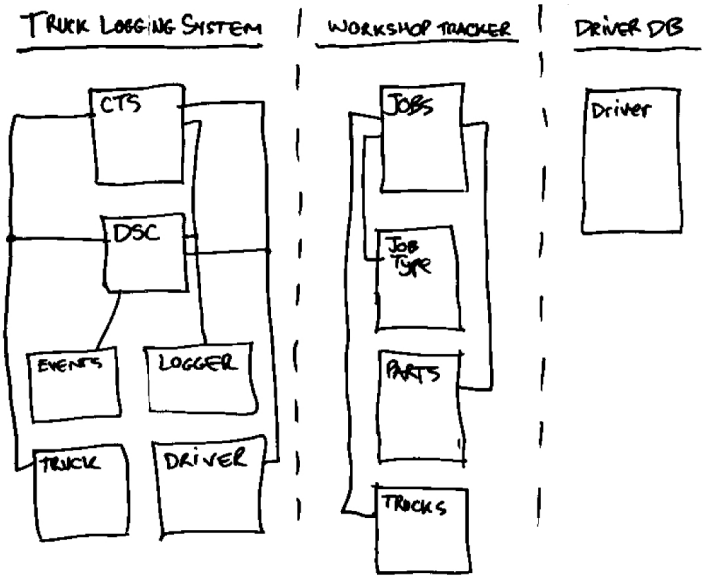


Fig.6 – Truck Data Cube Sources

The Schema for these 3 systems is shown below:

System	Table	Column	Data Type	Granularity
Truck Logging System	Continuous Metrics	Date	Date	per second
Truck Logging System	Continuous Metrics	Time	Time	per second
Truck Logging System	Continuous Metrics	Truck Id	integer	per second
Truck Logging System	Continuous Metrics	Driver Id	integer	per second
Truck Logging System	Continuous Metrics	Logger Id	integer	per second
Truck Logging System	Continuous Metrics	Fuel Level	Real	per second
Truck Logging System	Continuous Metrics	Engine Temperature	Real	per second
Truck Logging System	Continuous Metrics	Engine RPM	Real	per second
Truck Logging System	Continuous Metrics	Gear	Real	per second
Truck Logging System	Continuous Metrics	Brake Temperature - FL	Real	per second
Truck Logging System	Continuous Metrics	Brake Temperature - FR	Real	per second
Truck Logging System	Continuous Metrics	Brake Temperature - RL	Real	per second
Truck Logging System	Continuous Metrics	Brake Temperature - RR	Real	per second
Truck Logging System	Continuous Metrics	Tire Wall Deformity - FL	Real	per second
Truck Logging System	Continuous Metrics	Tire Wall Deformity - FR	Real	per second
Truck Logging System	Continuous Metrics	Tire Wall Deformity - RL	Real	per second
Truck Logging System	Continuous Metrics	Tire Wall Deformity - RR	Real	per second
Truck Logging System	Continuous Metrics	Loading - FL	Real	per second
Truck Logging System	Continuous Metrics	Loading - FR	Real	per second
Truck Logging System	Continuous Metrics	Loading - RL	Real	per second
Truck Logging System	Continuous Metrics	Loading - RR	Real	per second
Truck Logging System	Continuous Metrics	Speed kph	Real	per second
Truck Logging System	Continuous Metrics	Latitude	Real	per second
Truck Logging System	Continuous Metrics	Longitude	Real	per second
Truck Logging System	Continuous Metrics	RF RSRP dBm	Real	per second
Truck Logging System	Continuous Metrics	RF SINR dB	Real	per second
Truck Logging System	Continuous Metrics	Tx Pwr dBm	Real	per second
Truck Logging System	Continuous Metrics	UL Tput kbps	Real	per second
Truck Logging System	Event Metrics	Date	Date	per event
Truck Logging System	Event Metrics	Time	Time	per event
Truck Logging System	Event Metrics	Truck Id	integer	per event
Truck Logging System	Event Metrics	Driver Id	integer	per event
Truck Logging System	Event Metrics	Logger Id	integer	per event
Truck Logging System	Event Metrics	Event Id	integer	
Truck Logging System	Events	Event Id	Integer	per event type
Truck Logging System	Events	Event Name	Varchar(50)	per event type
Truck Logging System	Events	Event Description	Varchar(200)	per event type
Truck Logging System	Logger	Logger Id	Integer	per logger type
Truck Logging System	Logger	Logger Serial Number	Varchar(50)	per logger type
Truck Logging System	Logger	Manufacturer	Varchar(50)	per logger type
Truck Logging System	Truck	Truck Id	Integer	per truck
Truck Logging System	Truck	Truck Name	Varchar(50)	per truck
Truck Logging System	Truck	Truck Manufacturer	Varchar(50)	per truck
Truck Logging System	Truck	Truck Model	Varchar(50)	per truck
Truck Logging System	Truck	Truck Maintenance Code	Varchar(50)	per truck
Truck Logging System	Truck	Other Truck specific attributes	Varchar(50)	per truck

System	Table	Column	Data Type	Granularity
Workshop Tracker	Jobs	Date	Date	per job
Workshop Tracker	Jobs	Time	Time	per job
Workshop Tracker	Jobs	Job Id	Integer	per job
Workshop Tracker	Jobs	Job Type Id	Integer	per job
Workshop Tracker	Jobs	Truck Id	Integer	per job
Workshop Tracker	Jobs	Manhours Total Planned	Real	per job
Workshop Tracker	Jobs	Manhours Total Actual	Real	per job
Workshop Tracker	Jobs	Manhours Cost	Real	per job
Workshop Tracker	Jobs	Parts Cost Planned	Real	per job
Workshop Tracker	Jobs	Parts Cost Actual	Real	per job
Workshop Tracker	Jobs	Duration Planned	Real	per job
Workshop Tracker	Jobs	Duration Actual	Real	per job
Workshop Tracker	Job Type	Job Type Id	Integer	per job type
Workshop Tracker	Job Type	Job Name	Varchar(50)	per job type
Workshop Tracker	Job Type	Job Description	Varchar(50)	per job type
Workshop Tracker	Job Type	Part Id	Integer	per job type
Workshop Tracker	Parts	Part Id	Integer	per part
Workshop Tracker	Parts	Part Name	Varchar(50)	per part
Workshop Tracker	Parts	Manufacturer	Varchar(50)	per part
Workshop Tracker	Parts	Model	Varchar(50)	per part
Workshop Tracker	Parts	Part Cost	Real	per part
Workshop Tracker	Parts	Manhours	Real	per part
Workshop Tracker	Truck	Truck Id	Integer	per truck
Workshop Tracker	Truck	Truck Name	Varchar(50)	per truck
Workshop Tracker	Truck	Truck Manufacturer	Varchar(50)	per truck
Workshop Tracker	Truck	Truck Model	Varchar(50)	per truck
Workshop Tracker	Truck	Truck Maintenance Code	Varchar(50)	per truck
Workshop Tracker	Truck	Other Truck specific attrib	Varchar(50)	per truck

System	Table	Column	Data Type	Granularity
Driver Database	Driver	Driver Id	Integer	per driver
Driver Database	Driver	Driver Type	Varchar(50)	per driver
Driver Database	Driver	Driver Name	Varchar(50)	per driver
Driver Database	Driver	Driver DOB	Date	per driver
Driver Database	Driver	Driver Gender	Varchar(50)	per driver
Driver Database	Driver	Driver Relationship Status	Varchar(50)	per driver
Driver Database	Driver	Driver Serial Number	Varchar(50)	per driver
Driver Database	Driver	Driver Manufactuer	Varchar(50)	per driver
Driver Database	Driver	Driver Model	Varchar(50)	per driver

Stockpiles

The systems relating to the stockpiles and the flow of ore between them are spread out throughout the system. The systems that supply the data for the stockpile datamart are:

1. The Ore Flow Control System
2. The Ore Deposits Database
3. The Stockpiles Management System

The schema connections between these systems is shown in the following diagram.

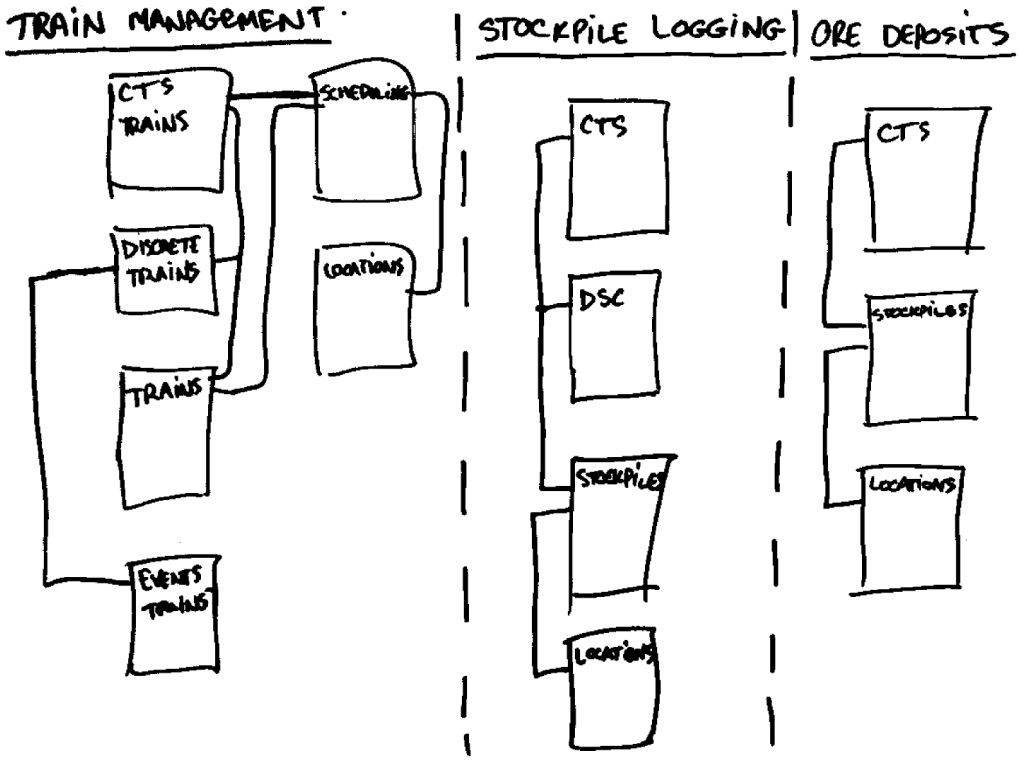


Fig.7 – Stockpile Data Cube Sources

The Trains/Flow Control System collects real time data on the status of the trains and the railway infrastructure. The data is both continuous for things like train position, speed and length as well as discrete for events such as leaving/arriving and passing route milestones as well as for breakdown and stoppage events. We will be mainly concerned with getting data on the flow rate of ore from the mines to the port from this system. The other systems are the mine and port stockpile management systems, these have daily estimates as to the current loading of the concerned stockpile as well. Lastly the Ore Deposits system has estimates of the in-ground resource quantities as well as the level. This database is updated weekly.

The schema for these systems is shown below:

System	Table	Column	Data Type	Granularity
Train Management	Continuous Metrics	Date	Date	per second
Train Management	Continuous Metrics	Time	Time	per second
Train Management	Continuous Metrics	Trip Id	integer	per second
Train Management	Continuous Metrics	Train Id	integer	per second
Train Management	Continuous Metrics	Speed	Real	per second
Train Management	Continuous Metrics	Latitude	Real	per second
Train Management	Continuous Metrics	Longitude	Real	per second
Train Management	Continuous Metrics	Size	Real	per second
Train Management	Continuous Metrics	Other Metrics	Real	per second
Train Management	Discrete Metrics	Date	Date	per event
Train Management	Discrete Metrics	Time	Time	per event
Train Management	Discrete Metrics	Train Id	integer	per event
Train Management	Discrete Metrics	Event Id	integer	per event
Train Management	Trains	Train Id	integer	per train
Train Management	Trains	Train Name	Varchar(50)	per train
Train Management	Trains	Manufacturer	Varchar(50)	per train
Train Management	Trains	Model	Varchar(50)	per train
Train Management	Trains	Max Size	Real	per train
Train Management	Trains	Max Speed	Real	per train
Train Management	Trains	Fuel Burn Data	Real	per train
Train Management	Events	Event Id	integer	per event type
Train Management	Trains	Event Name	Varchar(50)	per event type
Train Management	Trains	Event Code	Varchar(50)	per event type
Train Management	Trains	Event Description	Varchar(50)	per event type
Train Management	Trains	Event Level	Varchar(50)	per event type
Train Management	Scheduling	Departure Date	Date	per trip
Train Management	Scheduling	Departure Time	Time	per trip
Train Management	Scheduling	Trip Id	Time	per trip
Train Management	Scheduling	Train Id	integer	per trip
Train Management	Scheduling	Departure Location	Varchar(50)	per trip
Train Management	Scheduling	Arrival Location	Varchar(50)	per trip
Train Management	Scheduling	Expected Size	Real	per trip
Train Management	Scheduling	Journey Status	Varchar(50)	per trip
Train Management	Locations	Location Id	integer	per location
Train Management	Locations	Location Name	Varchar(50)	per location
Train Management	Locations	Other Attributes	Varchar(50)	per location

System	Table	Column	Data Type	Granularity
Stockpile Logging	Continuous Metrics	Date	Date	per Day
Stockpile Logging	Continuous Metrics	Stockpile Id	integer	per Day
Stockpile Logging	Continuous Metrics	Loading pct	Real	per Day
Stockpile Logging	Continuous Metrics	Loading kt	Real	per Day
Stockpile Logging	Continuous Metrics	Other Metrics	Real	per Day
Stockpile Logging	Discrete Metrics	Date	Date	per Event
Stockpile Logging	Discrete Metrics	Time	Time	per Event
Stockpile Logging	Discrete Metrics	Stockpile Id	Real	per Event
Stockpile Logging	Discrete Metrics	0% Alarm	Real	per Event
Stockpile Logging	Discrete Metrics	10% Alarm	Real	per Event
Stockpile Logging	Discrete Metrics	25% Alarm	Real	per Event
Stockpile Logging	Discrete Metrics	75% Alarm	Real	per Event
Stockpile Logging	Discrete Metrics	90% Alarm	Real	per Event
Stockpile Logging	Discrete Metrics	100% Alarm	Real	per Event
Stockpile Logging	Discrete Metrics	Addition Event	Real	per Event
Stockpile Logging	Discrete Metrics	Removal Event	Real	per Event
Stockpile Logging	Discrete Metrics	Other alarms	Real	per Event
Stockpile Logging	Stockpiles	Stockpile Id	integer	per stockpile
Stockpile Logging	Stockpiles	Location Id	integer	per stockpile
Stockpile Logging	Stockpiles	Max Capacity	real	per stockpile
Stockpile Logging	Stockpiles	Other Attributes	Varchar(50)	per stockpile
Stockpile Logging	Locations	Location Id	integer	per location
Stockpile Logging	Locations	Location Name	Varchar(50)	per location
Stockpile Logging	Locations	Other Attributes	Varchar(50)	per location

System	Table	Column	Data Type	Granularity
Ore Deposits	Continuous Metrics	Date	Date	per month
Ore Deposits	Continuous Metrics	Stockpile Id	integer	per month
Ore Deposits	Continuous Metrics	Loading pct	real	per month
Ore Deposits	Continuous Metrics	Loading kt	real	per month
Ore Deposits	Continuous Metrics	Other Metrics	Varchar(50)	per month
Ore Deposits	Stockpiles	Stockpile Id	integer	per stockpile
Ore Deposits	Stockpiles	Location Id	integer	per stockpile
Ore Deposits	Stockpiles	Max Capacity	real	per stockpile
Ore Deposits	Stockpiles	Current Tier	integer	per stockpile
Ore Deposits	Stockpiles	Current Status	Varchar(50)	per stockpile
Ore Deposits	Stockpiles	Other Attributes	Varchar(50)	per stockpile
Ore Deposits	Locations	Location Id	integer	per location
Ore Deposits	Locations	Location Name	Varchar(50)	per location
Ore Deposits	Locations	Other Attributes	Varchar(50)	per location

Orders

The orders data comes primarily from one a single source system, the Order Management System. This is a one stop shop for order information, where orders are tracked from initial signing to the final shipment, sometimes covering up to 20 years. The primary data in the order system is an accumulating snapshot order table and an associated shipment table.

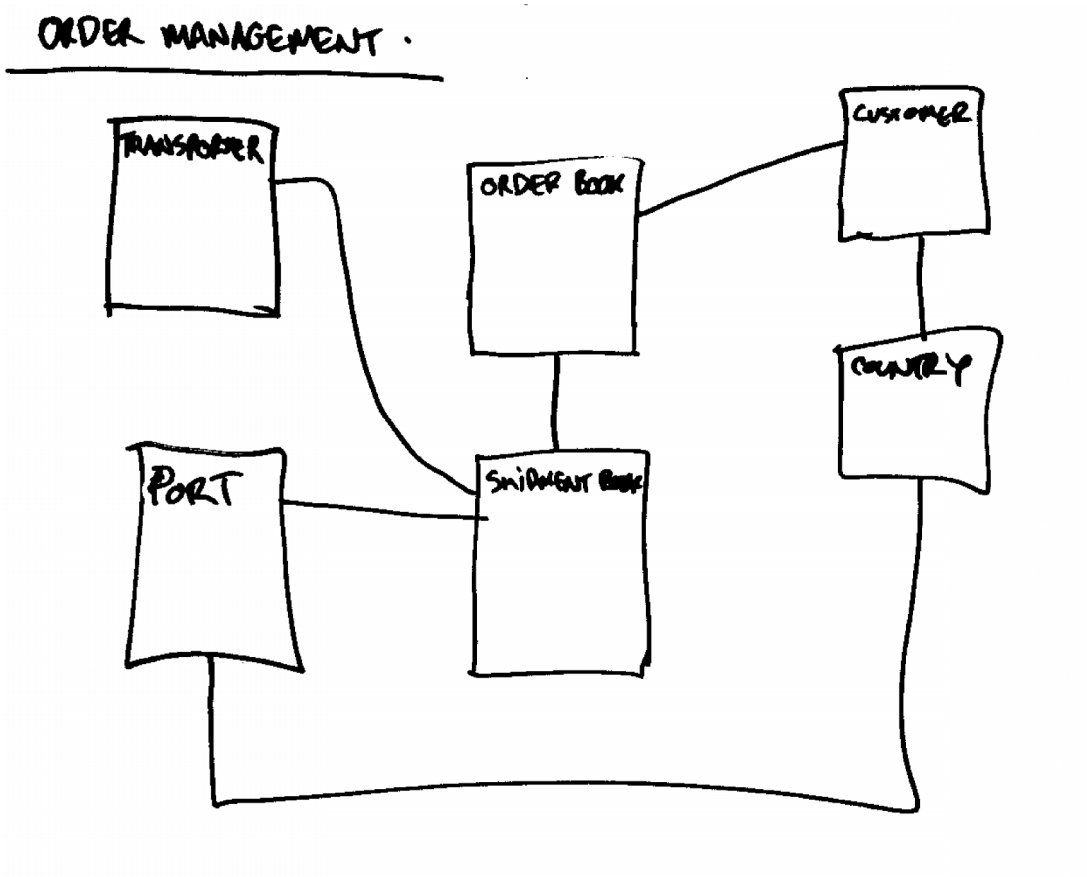


Fig.8 – Orders Data Cube Sources

The schema for the order management system is shown below:

System	Table	Column	Data Type	Granularity
Order Management	Order Book	Date	Date	per order
Order Management	Order Book	Order Id	integer	per order
Order Management	Order Book	Customer Id	integer	per order
Order Management	Order Book	Price	Real	per order
Order Management	Order Book	Quantity	Real	per order
Order Management	Order Book	Shipment Interval	Real	per order
Order Management	Order Book	Start Date	Real	per order
Order Management	Order Book	Other Metrics	Real	per order
Order Management	Shipment Book	Shipment Id	integer	per shipment
Order Management	Shipment Book	Shipment Date	Date	per shipment
Order Management	Shipment Book	Order Id	integer	per shipment
Order Management	Shipment Book	Transporter Id	integer	per shipment
Order Management	Shipment Book	Destination Port Id	integer	per shipment
Order Management	Shipment Book	Shipment Quantity	Real	per shipment
Order Management	Shipment Book	Other Metrics	Real	per shipment
Order Management	Customers	Customer Id	integer	per customer
Order Management	Customers	Country Id	integer	per customer
Order Management	Customers	Other Metrics	Varchar(50)	per customer
Order Management	Country	Country Id	integer	per country
Order Management	Country	Country Name	Varchar(50)	per country
Order Management	Country	Other Attributes	Varchar(50)	per country
Order Management	Port	Port Id	integer	per port
Order Management	Port	Country Id	Varchar(50)	per port
Order Management	Port	Latitude	real	per port
Order Management	Port	Longitude	real	per port
Order Management	Port	Other Metrics	Varchar(50)	per port
Order Management	Transporter	Transporter Id	integer	per transporter
Order Management	Transporter	Transporter Name	Varchar(50)	per transporter
Order Management	Transporter	Other Attributes	Varchar(50)	per transporter

Data Generation Process

Note that the actual data for the data cubes was generated by building simulations in Python. In order to speed up the process, the data was actually generated directly for the final data cubes as opposed to creating the source database data first and then doing ETL on it. As such the ETL process did not have to be carried out on the source data.

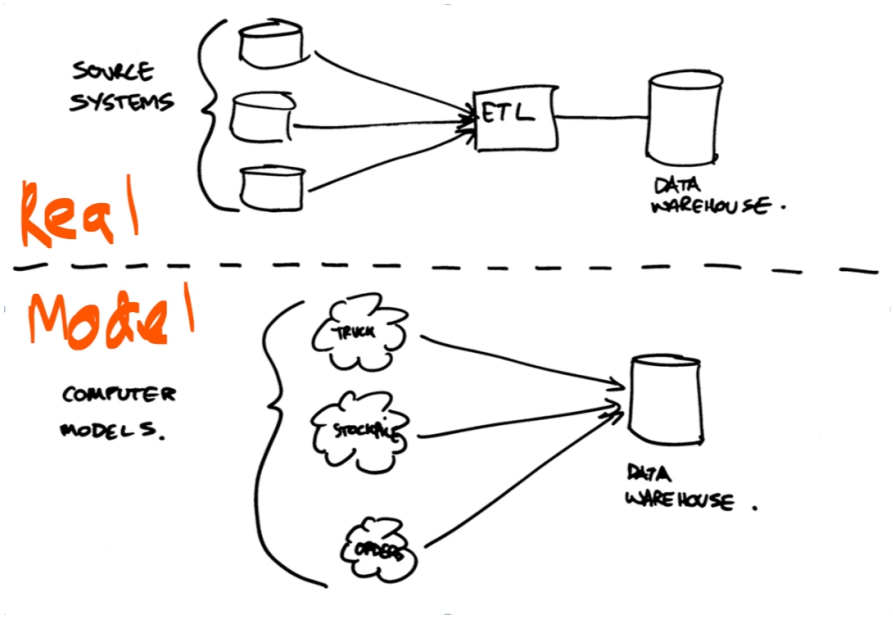


Fig.9 – Data Generation Process

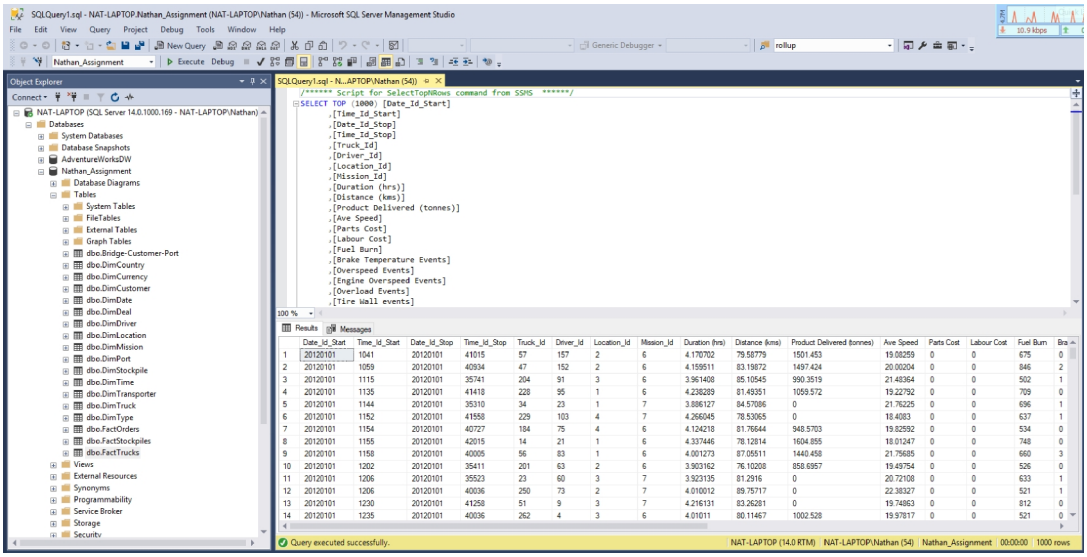


Fig.9.1 – DB viewed from MSSMS

Please ask for the data cube data. The python code for the 3 simulations is attached in an appendix to this document.

Task 3 – Data Cube Design and Creation

Truck Management Data Cube

The Truck management data cube was designed to enable management to quickly see the cost and production performance of the truck fleet and to get meaningful data on what factors are affecting it.

The truck management data cube was designed with the following parameters:

- Grain: the truck facts table has a grain of one round trip from the depot, thus the fact table is a transactional fact table.
- Dimensions:
    - Date – this is a role play dimension as it appears several times in the Fact Table
    - Time – again a role playing dimension as it appears several times in the Fact Table
    - Location – has data for the various mines
    - Mission – has data for the various missions that the trucks leave the depot with
    - Truck – has data for the individual trucks
    - Driver – has data for the individual drivers (both human and autonomous)
- Facts: the Truck Fact Table has metrics on truck performance during each mission. Such as negative events, fuel use, production quantity, parts/labour/driver cost and duration for each mission.

From a business perspective, the most critical metric is the cost/tonne of ore delivered to the mine stockpile. This metric can be derived from the fact table in the 'BI Layer' (Business Intelligence Layer) and analysed in various ways, such as:

1. looking at the difference in cost/tonne between autonomous drivers vs humans
2. between human gender
3. between human driver ages

Broken down by mine, truck model, manufacturer (or over time).

Other analysis services could be:

1. Breaking down serious events (like accidents and tire losses) by mine, gender, human/auto, age
2. Looking at correlations between events and age, gender, mine etc.
3. Looking at the fuel burn to product delivered and driver relationship
4. Looking at the breakdown of mission volume over time (or by mine) by driver type and gender to see how the driver mix is changing over time

The Schema and dependency diagram for the trucks management data cube is shown below.

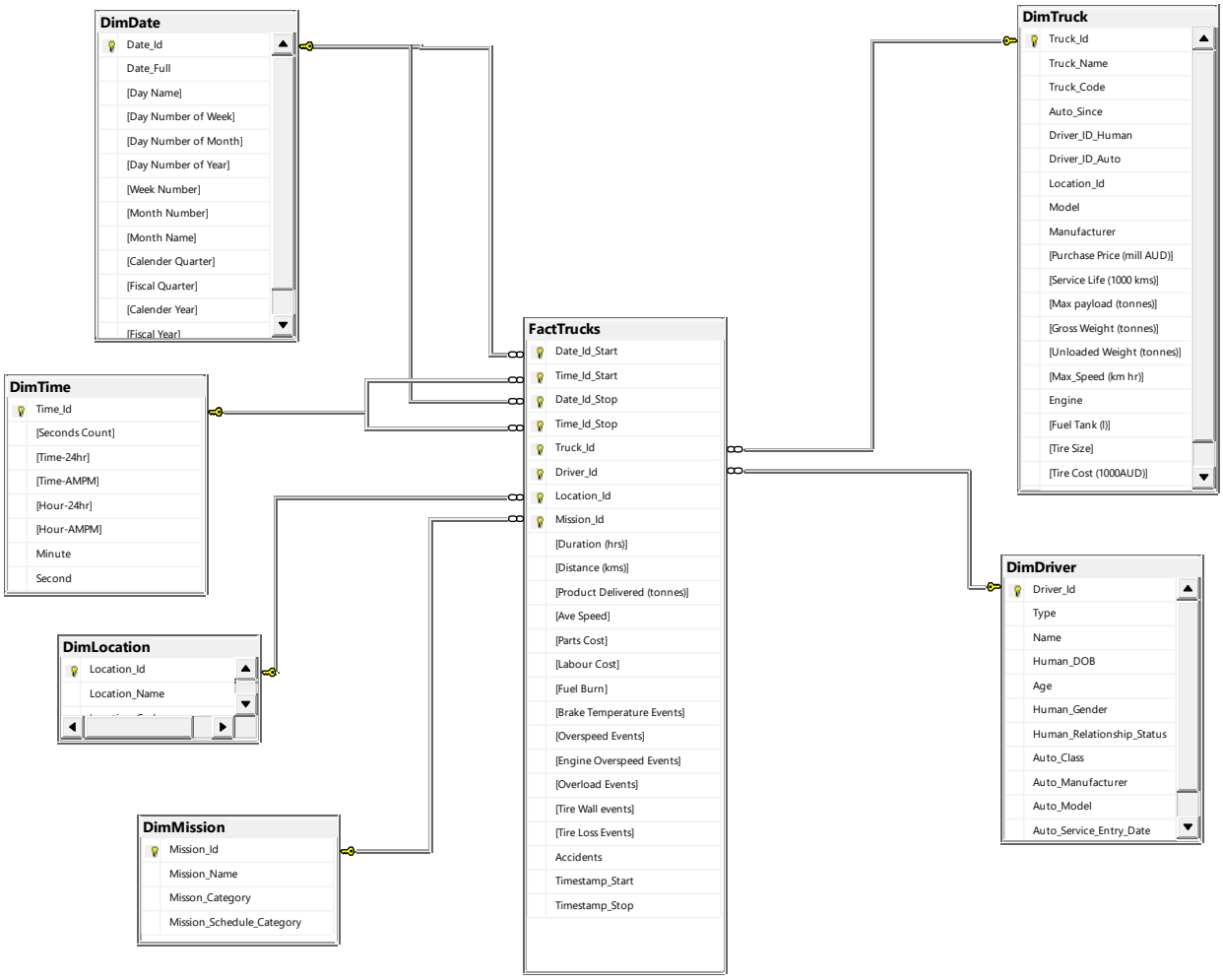


Fig.10 – Truck Data Cube Diagram and Dependencies

The schema for this data cube is shown below.

FactTrucks		
Column Name	**Data Type**	**Allow Nulls**
Truck_Id	smallint	Unchecked
Truck_Name	varchar(50)	Checked
Truck_Code	varchar(50)	Checked
Auto_Since	date	Checked
Driver_ID_Human	varchar(50)	Checked
Driver_ID_Auto	varchar(50)	Checked
Location_Id	smallint	Checked
Model	varchar(50)	Checked
Manufacturer	varchar(50)	Checked
[Purchase Price (mill AUD)]	real	Checked
[Service Life (1000 kms)]	real	Checked
[Max payload (tonnes)]	real	Checked
[Gross Weight (tonnes)]	real	Checked
[Unloaded Weight (tonnes)]	real	Checked
[Max_Speed (km hr)]	real	Checked
Engine	varchar(50)	Checked
[Fuel Tank (l)]	real	Checked
[Tire Size]	varchar(50)	Checked
[Tire Cost (1000AUD)]	real	Checked
[Tire Quantity]	smallint	Checked

DimDate		
Column Name	**Data Type**	**Allow Nulls**
Date_Id	int	Unchecked
Date_Full	date	Checked
[Day Name]	varchar(50)	Checked
[Day Number of Week]	smallint	Checked
[Day Number of Month]	smallint	Checked
[Day Number of Year]	smallint	Checked
[Week Number]	smallint	Checked
[Month Number]	smallint	Checked
[Month Name]	varchar(50)	Checked
[Calender Quarter]	smallint	Checked
[Fiscal Quarter]	smallint	Checked
[Calender Year]	smallint	Checked
[Fiscal Year]	smallint	Checked

DimTime		
Column Name	**Data Type**	**Allow Nulls**
Time_Id	int	Unchecked
[Seconds Count]	int	Checked
[Time-24hr]	time(7)	Checked
[Time-AMPM]	varchar(50)	Checked
[Hour-24hr]	smallint	Checked
[Hour-AMPM]	smallint	Checked
Minute	smallint	Checked
Second	smallint	Checked

DimTruck		
Column Name	**Data Type**	**Allow Nulls**
Truck_Id	smallint	Unchecked
Truck_Name	varchar(50)	Checked
Truck_Code	varchar(50)	Checked
Auto_Since	date	Checked
Driver_ID_Human	varchar(50)	Checked
Driver_ID_Auto	varchar(50)	Checked
Location_Id	smallint	Checked
Model	varchar(50)	Checked
Manufacturer	varchar(50)	Checked
[Purchase Price (mill AUD)]	real	Checked
[Service Life (1000 kms)]	real	Checked
[Max payload (tonnes)]	real	Checked
[Gross Weight (tonnes)]	real	Checked
[Unloaded Weight (tonnes)]	real	Checked
[Max_Speed (km hr)]	real	Checked
Engine	varchar(50)	Checked
[Fuel Tank (l)]	real	Checked
[Tire Size]	varchar(50)	Checked
[Tire Cost (1000AUD)]	real	Checked
[Tire Quantity]	smallint	Checked

DimMission		
Column Name	**Data Type**	**Allow Nulls**
Mission_Id	smallint	Unchecked
Mission_Name	varchar(50)	Checked
Misson_Category	varchar(50)	Checked
Mission_Schedule_Category	varchar(50)	Checked

DimLocation		
Column Name	**Data Type**	**Allow Nulls**
Location_Id	smallint	Unchecked
Location_Name	varchar(50)	Checked
Location_Code	varchar(50)	Checked
Lattitude	decimal(10,	Checked
Longitude	decimal(10,	Checked

DimDriver		
Column Name	**Data Type**	**Allow Nulls**
Driver_Id	smallint	Unchecked
Type	varchar(50)	Checked
Name	varchar(50)	Checked
Human_DOB	date	Checked
Age	int	Checked
Human_Gender	varchar(50)	Checked
Human_Relationship_Status	varchar(50)	Checked
Auto_Class	varchar(50)	Checked
Auto_Manufacturer	varchar(50)	Checked
Auto_Model	varchar(50)	Checked
Auto_Service_Entry_Date	date	Checked

Stockpile Management Data Cube

The stockpile management data cube was designed to enable management to view the performance of the ore flow control from the in-ground deposits to the port, as well as the current and historical stockpile levels, and to get meaningful data on affecting factors.

The stockpile management data cube was designed with the following parameters:

- Grain: the stockpile facts table has a grain of one day per stockpile, thus the fact table is a periodic snapshot fact table.
- Dimensions:
    - Date – holds date information
    - Stockpile – has data for each mine stockpile, port stockpile and in-ground deposit
    - Location – has data for the various mines
    - Type – has data for ranking of the in-ground resources
- Facts: the stockpile fact table has metrics on the day's inventory (loading) of each stockpile as well as data on the Ingress and Egress.

Some analyses that could be run on the stockpiles datacube are:

1. Timeseries analysis of the port stockpile levels and the meeting of demand. How does the system adjust to changing demand levels. This is important to ensuring that the system runs smoothly and supply disruptions or shortfalls do not impact the shipments as can be seen in the mid 2015 demand boom that cause the demand debt to skyrocket as the company could not increase supply enough to meet demand.
2. Timeseries analysis of the in-ground deposit levels by mine and changing of status. This can show where and how much resource remains in various company resource pools and how fast it is being depleted. This is important in relation to spurring exploration as well as to planning future extractions to meet demand.
3. Network analysis of the entire company ore movement system as a whole, good for understanding which nodes are producing how much ore and through which nodes and edges the ore is flowing.
4. Lead-time analysis of the current ore stockpiles, this is key for future contract negotiations as well as extraction planning.
5. Sankey diagram of the current flow magnitude between the stockpiles, again this is useful for understanding flow levels.
6. Geo breakdown of inventories by stockpile, helps to visualize where geographically the capacity is located.

The Schema and dependency diagram for the stockpiles management data cube is shown below.

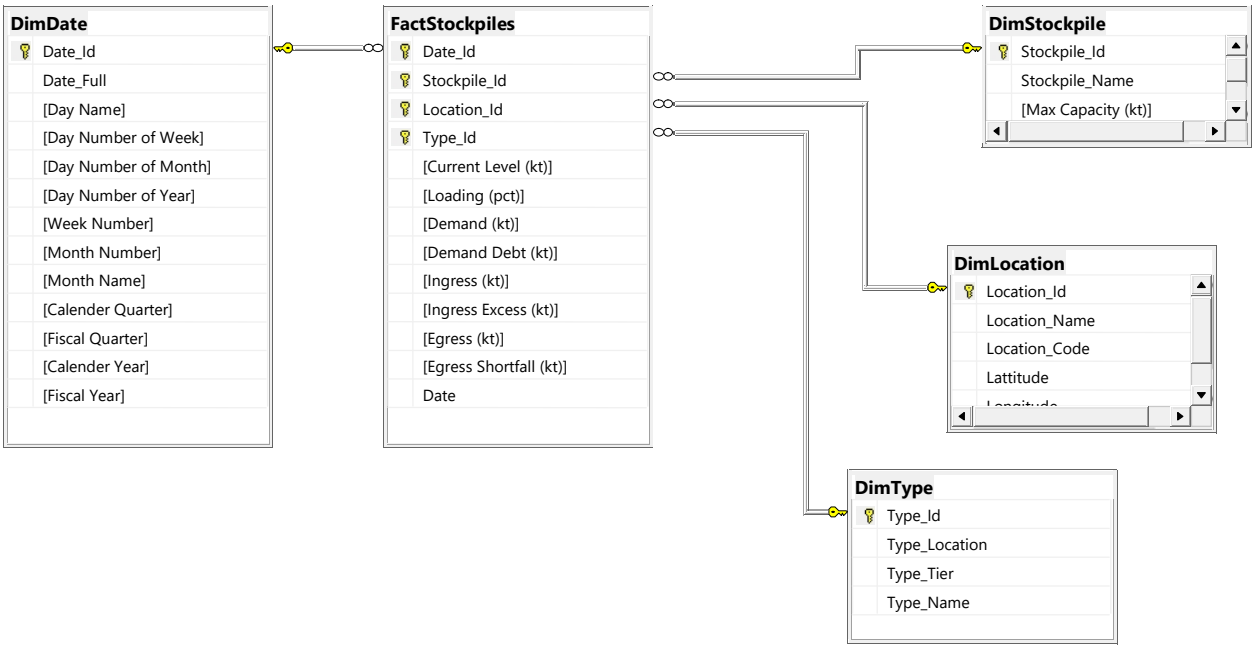


Fig.11 – Stockpile Data Cube Diagram and Dependencies

The Schema is shown below.

FactStockpiles

Column Name	Data Type	Allow Nulls
Date_Id	int	Unchecked
Stockpile_Id	smallint	Unchecked
Location_Id	smallint	Unchecked
Type_Id	smallint	Unchecked
[Current Level (kt)]	real	Checked
[Loading (pct)]	real	Checked
[Demand (kt)]	real	Checked
[Demand Debt (kt)]	real	Checked
[Ingress (kt)]	real	Checked
[Ingress Excess (kt)]	real	Checked
[Egress (kt)]	real	Checked
[Egress Shortfall (kt)]	real	Checked
Date	date	Checked

DimDate

Column Name	Data Type	Allow Nulls
Date_Id	int	Unchecked
Date_Full	date	Checked
[Day Name]	varchar(50)	Checked
[Day Number of Week	smallint	Checked
[Day Number of Mont	smallint	Checked
[Day Number of Year]	smallint	Checked
[Week Number]	smallint	Checked
[Month Number]	smallint	Checked
[Month Name]	varchar(50)	Checked
[Calender Quarter]	smallint	Checked
[Fiscal Quarter]	smallint	Checked
[Calender Year]	smallint	Checked
[Fiscal Year]	smallint	Checked

DimStockpile

Column Name	Data Type	Allow Nulls
Stockpile_Id	smallint	Unchecked
Stockpile_Name	varchar(50)	Checked
[Max Capacity (kt)]	real	Checked

DimType

Column Name	Data Type	Allow Nulls
Type_Id	smallint	Unchecked
Type_Location	varchar(50)	Checked
Type_Tier	varchar(50)	Checked
Type_Name	varchar(50)	Checked

Order Management Data Cube

The order management data cube was designed to enable management to quickly see the breadth and depth of the order pipeline and to quantify the performance of the various supply contracts across customers and countries.

The order management data cube was designed with the following parameters:

- Grain: the order facts table has a grain of one shipment from the company's port loading facilities.
- Dimensions:
    - Date – this is a role playing dimension for date as it appears several times in the Fact Table
    - Customer – Holds information of the company's customers
    - Port – holds information on the various destination ports
    - Transporter – holds information on the various shipping service providers
    - Country – holds information on the various countries
    - Currency – holds information on currencies and exchange rates
    - Deal – holds information on the particular deal framework applied to an order
- Facts: the order fact table has metrics on shipment dates, quantities, pricing levels and status.

Some analyses that can be performed with the orders datacube are:

1. Looking at the breakdown of customer or country in relation to order volume and price/tone for last x years and looking forward x years. This can help to identify the most important regions/countries and customers over periods of time.
2. Looking at geo-heatmaps or pie charts or flow diagrams representing customer volume or other importance metrics. Again this helps to identify important customers and regions over certain periods of time.
3. Looking at the order volume breakdown over time and customer/country. Would help to see how the volumes are changing and would potentially help to predict future volumes and regions.
4. Scatter plots for late shipment penalties vs late payment penalties, helps to identify problem customers or relationships, could be due to geographic distance, transporter issues, customer culture etc.

The data cube dependency diagram is show below.

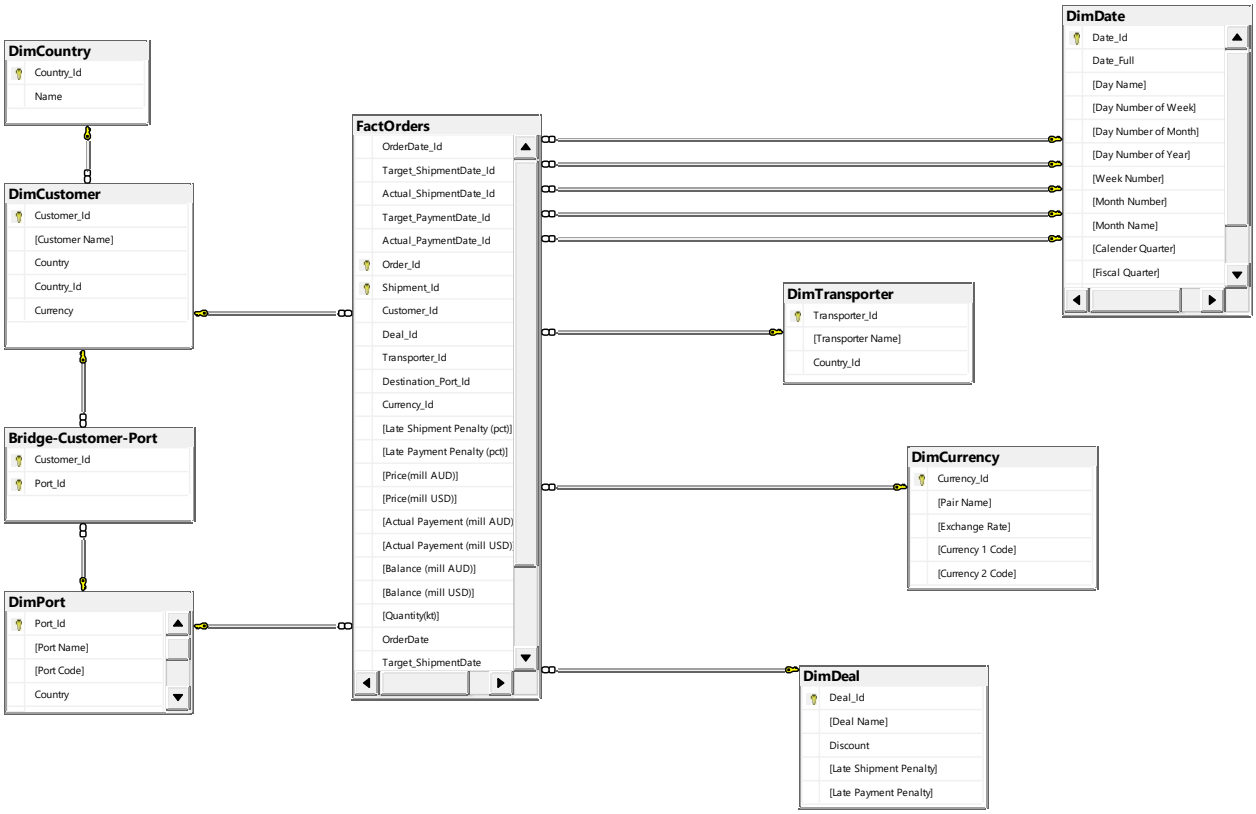


Fig.12 – Orders Data Cube Diagram and Dependencies

The following are the schema for this data cube.

FactOrders

Column Name	Data Type	Allow Nulls
OrderDate_Id	int	Checked
Target_ShipmentDate_Id	int	Checked
Actual_ShipmentDate_Id	int	Checked
Target_PaymentDate_Id	int	Checked
Actual_PaymentDate_Id	int	Checked
Order_Id	smallint	Unchecked
Shipment_Id	smallint	Unchecked
Customer_Id	smallint	Checked
Deal_Id	smallint	Checked
Transporter_Id	smallint	Checked
Destination_Port_Id	smallint	Checked
Currency_Id	smallint	Checked
[Late Shipment Penalty (pct)]	real	Checked
[Late Payment Penalty (pct)]	real	Checked
[Price(mill AUD)]	real	Checked
[Price(mill USD)]	real	Checked
[Actual Payement (mill AUD)]	real	Checked
[Actual Payement (mill USD)]	real	Checked
[Balance (mill AUD)]	real	Checked
[Balance (mill USD)]	real	Checked
[Quantity(kt)]	real	Checked
OrderDate	date	Checked
Target_ShipmentDate	date	Checked
Actual_ShipmentDate	date	Checked
Target_PaymentDate	date	Checked
Actual_PaymentDate	date	Checked

DimDate

Column Name	Data Type	Allow Nulls
Date_Id	int	Unchecked
Date_Full	date	Checked
[Day Name]	varchar(50	Checked
[Day Number of Week]	smallint	Checked
[Day Number of Month]	smallint	Checked
[Day Number of Year]	smallint	Checked
[Week Number]	smallint	Checked
[Month Number]	smallint	Checked
[Month Name]	varchar(50	Checked
[Calender Quarter]	smallint	Checked
[Fiscal Quarter]	smallint	Checked
[Calender Year]	smallint	Checked
[Fiscal Year]	smallint	Checked

DimTransporter

Column Name	Data Type	Allow Nulls
Transporter_Id	smallint	Unchecked
[Transporter Name]	varchar(50	Checked
Country_Id	smallint	Checked

DimPort

Column Name	Data Type	Allow Nulls
Port_Id	smallint	Unchecked
[Port Name]	varchar(50	Checked
[Port Code]	varchar(50	Checked
Country	varchar(50	Checked
Country_Id	smallint	Checked
Lat	real	Checked
Lon	real	Checked

DimDeal

Column Name	Data Type	Allow Nulls
Deal_Id	smallint	Unchecked
[Deal Name]	varchar(50	Checked
Discount	real	Checked
[Late Shipment Penalty]	real	Checked
[Late Payment Penalty]	real	Checked

DimCustomer

Column Name	Data Type	Allow Nulls
Customer_Id	smallint	Unchecked
[Customer Name]	varchar(50	Checked
Country	varchar(50	Checked
Country_Id	smallint	Checked
Currency	varchar(50	Checked

DimCurrency

Column Name	Data Type	Allow Nulls
Currency_Id	smallint	Unchecked
[Pair Name]	varchar(50	Checked
[Exchange Rate]	real	Checked
[Currency 1 Code]	varchar(50	Checked
[Currency 2 Code]	varchar(50	Checked

DimCountry

Column Name	Data Type	Allow Nulls
Country_Id	smallint	Unchecked
Name	varchar(50	Checked

BridgeCustomerPort

Column Name	Data Type	Allow Nulls
Customer_Id	smallint	Unchecked
Port_Id	smallint	Unchecked

Task 4 – Analytical Visualisation with Power BI

Note: that as these are screenshots, they have no user interaction, which is somewhat critical for some of them. This is a power bi issue as the platform prohibits exporting of interactive plots.

Trucks

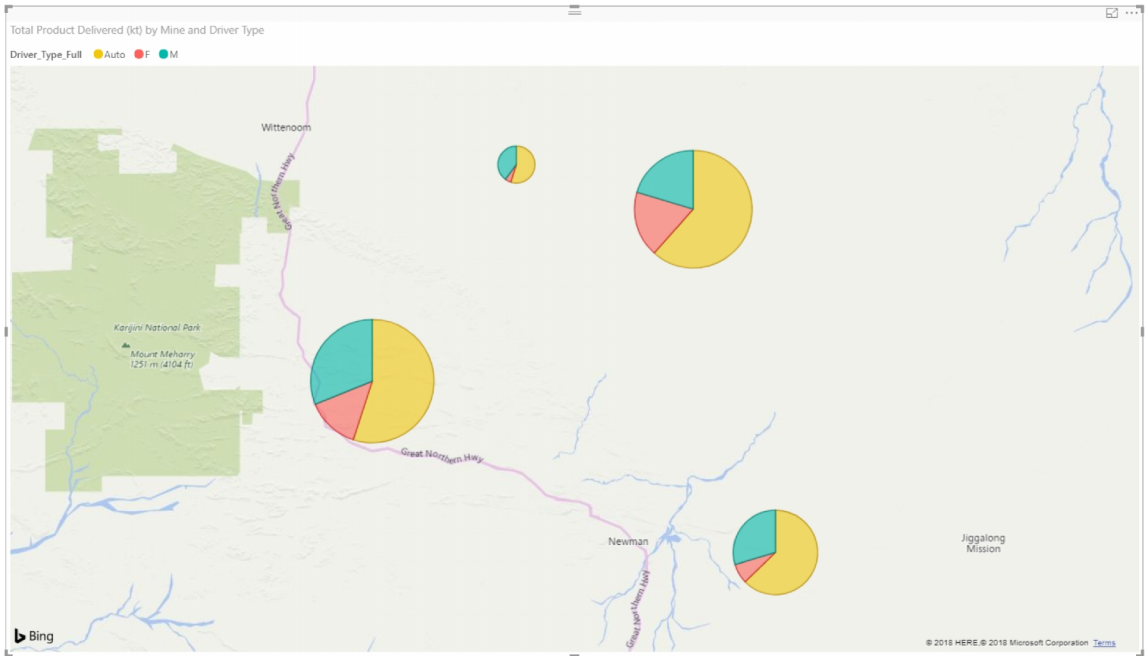


Fig.13 – Trucks – Last Year Product Delivered by Mine and Driver Type

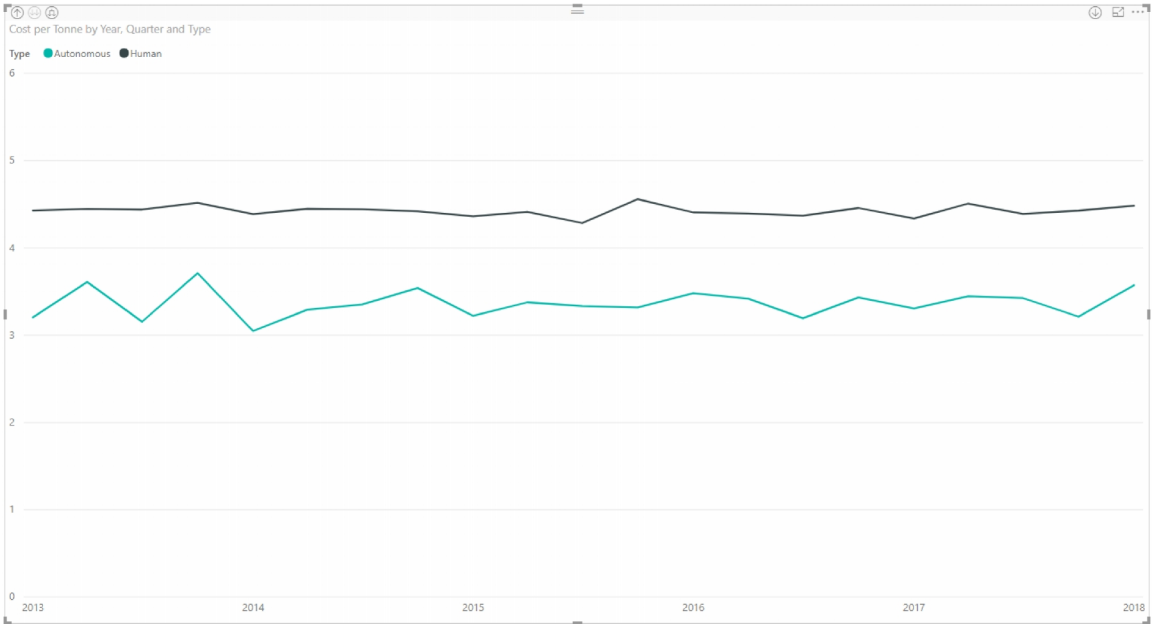


Fig.14 – Trucks – Human vs Autonomous Cost/Tonne Timeseries

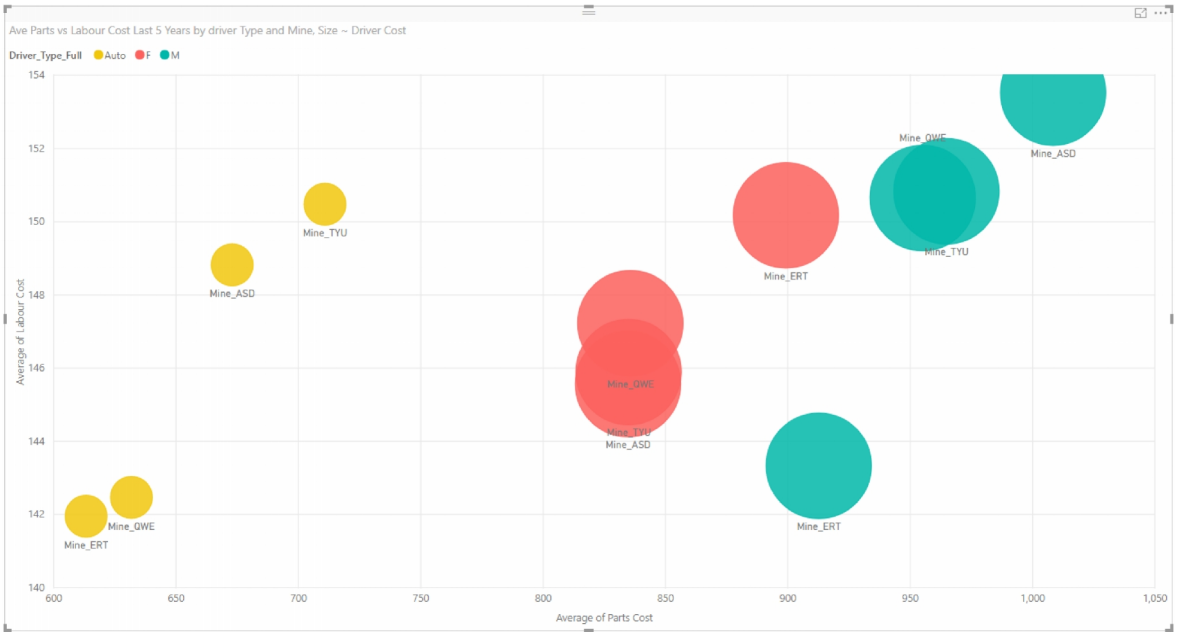


Fig.15 – Trucks – Parts Cost(x axis) vs Labour Cost(y axis) vs Driver Cost(size) by Driver Type(colour) and Mine

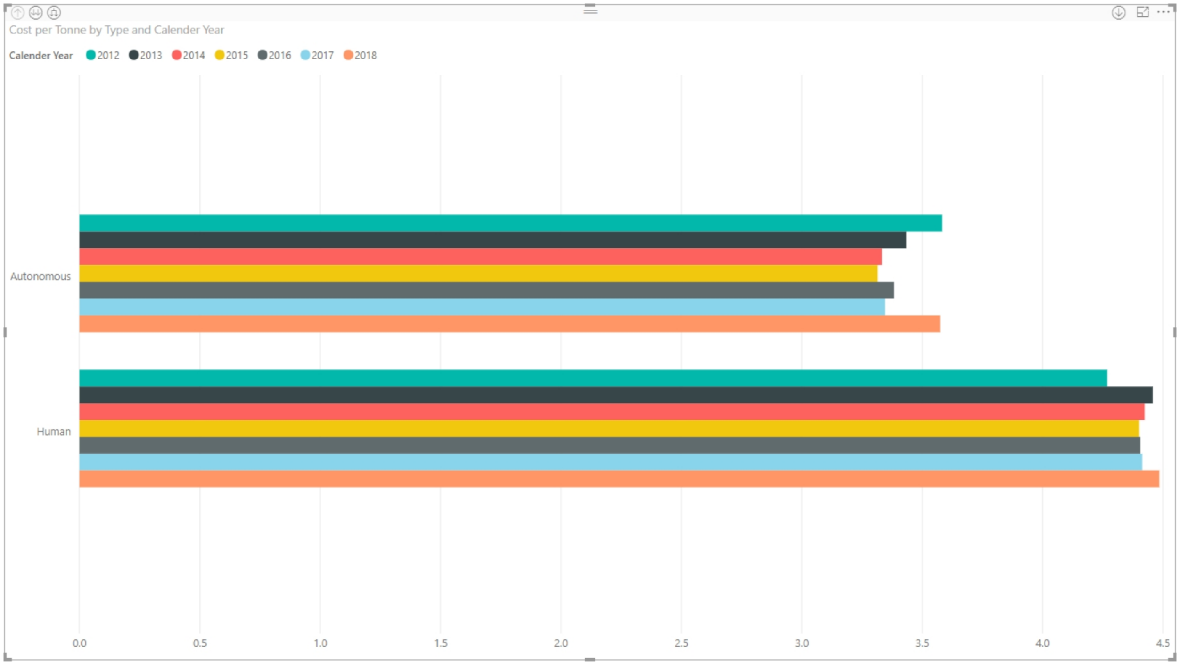


Fig.16 – Trucks – Mean Ore Production Cost per Tonne by Driver Type by Year

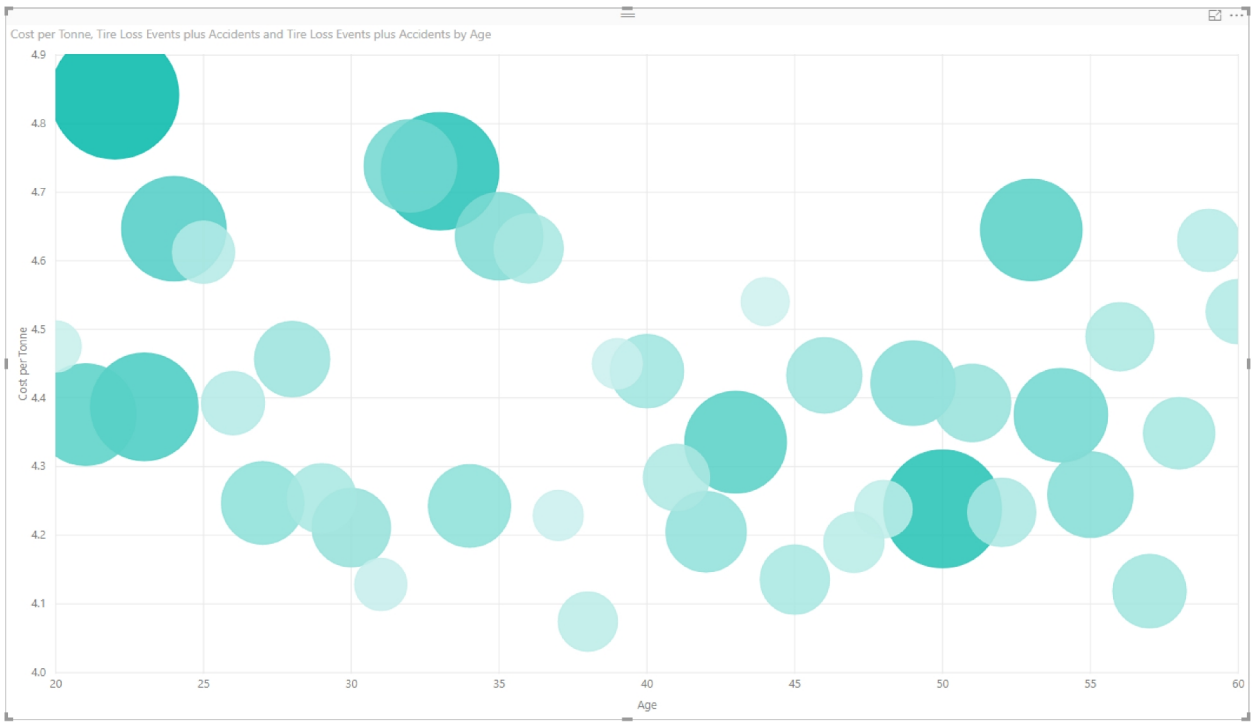


Fig.17 – Trucks – Ore Production Cost per Tonne by Driver Age by Major Negative Events(size)

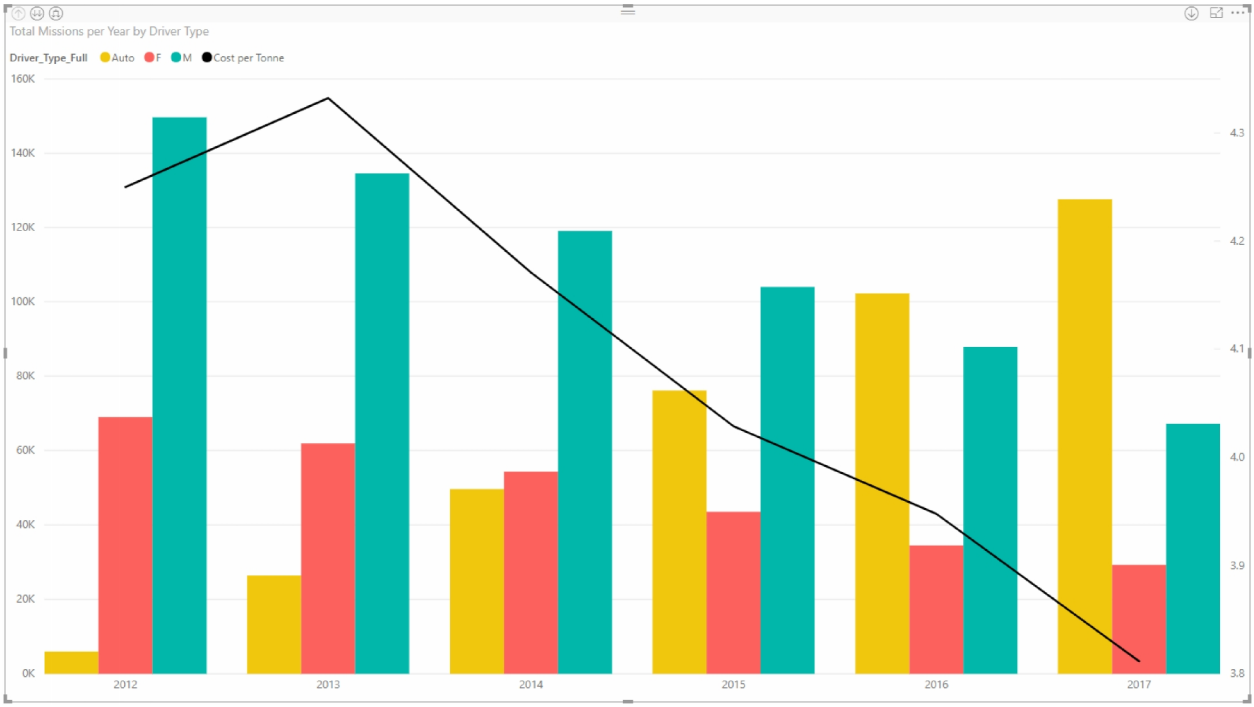


Fig.18 – Trucks – Number of Missions Driven per Year by Driver Type + Mean Production Cost/Tonne(line)

Fig.19 – Trucks – Mean Daily Product Delivered vs Fuel Burn by Driver Type

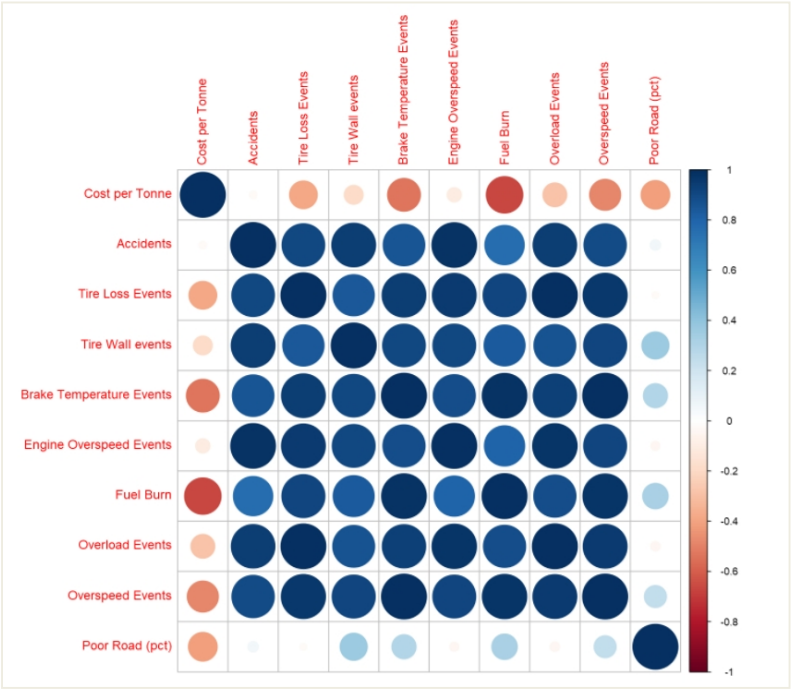


Fig.20 – Trucks – Correlation Matrix Between Major Fact Measurements

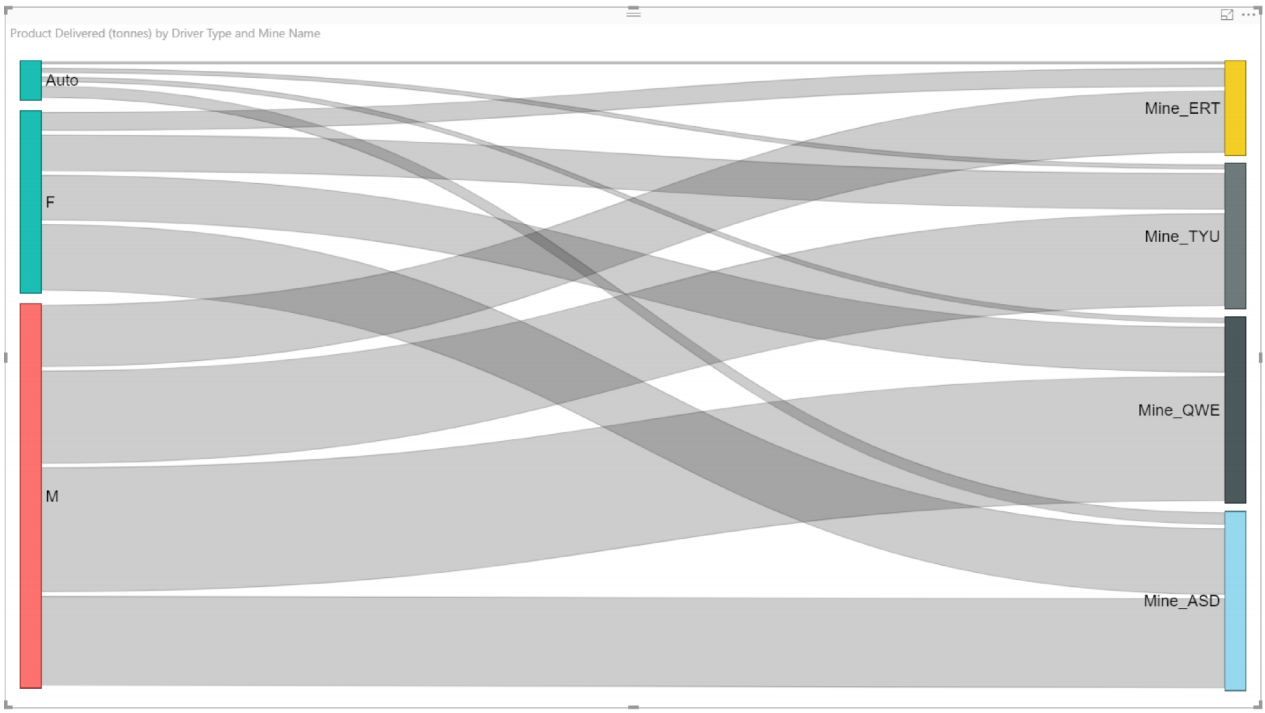


Fig.21 – Trucks – Total 2012 Product Delivered by Driver Type and Mine

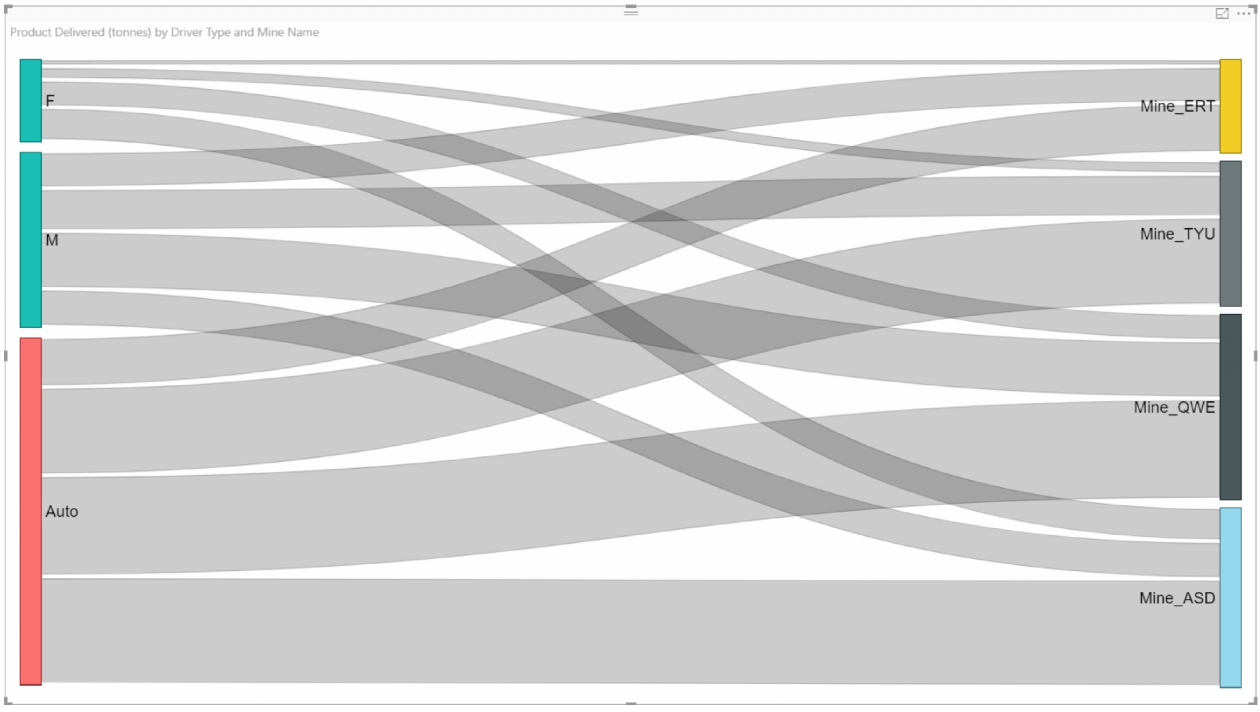


Fig.22 – Trucks – Total 2017 Product Delivered by Driver Type and Mine

Stockpiles

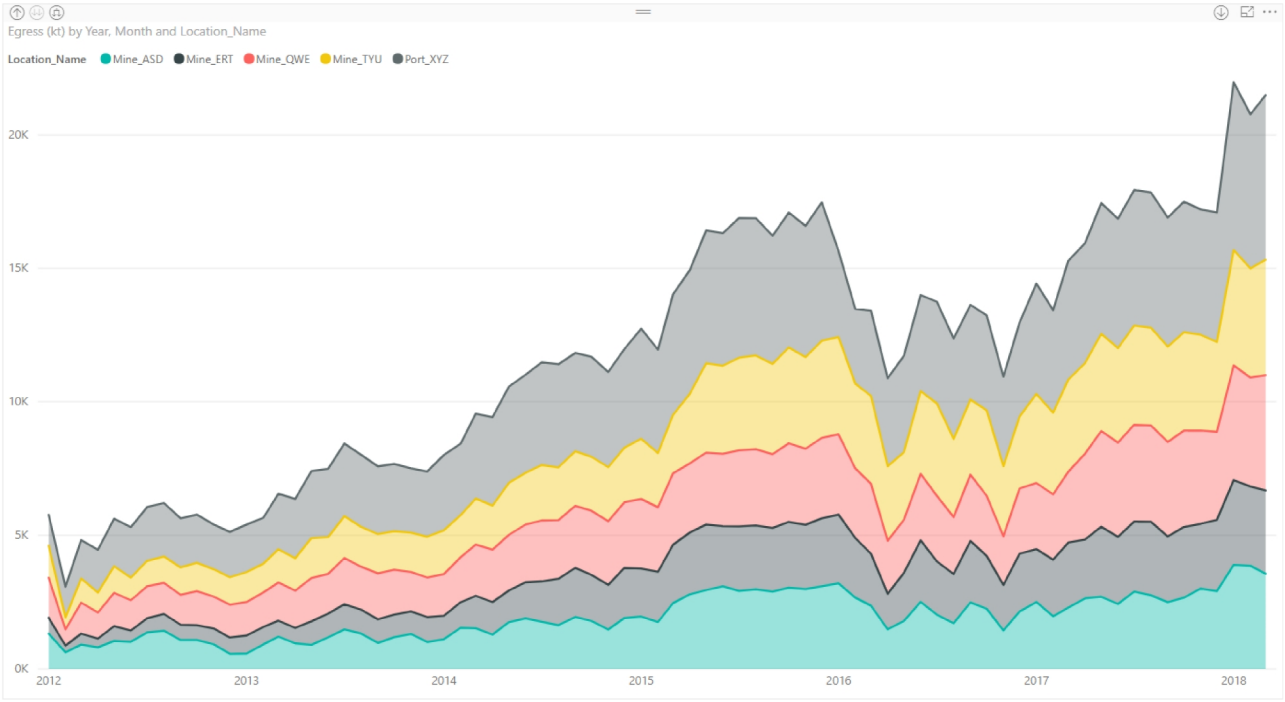


Fig.23 – Stockpiles – Mine Output by Year - Timeseries

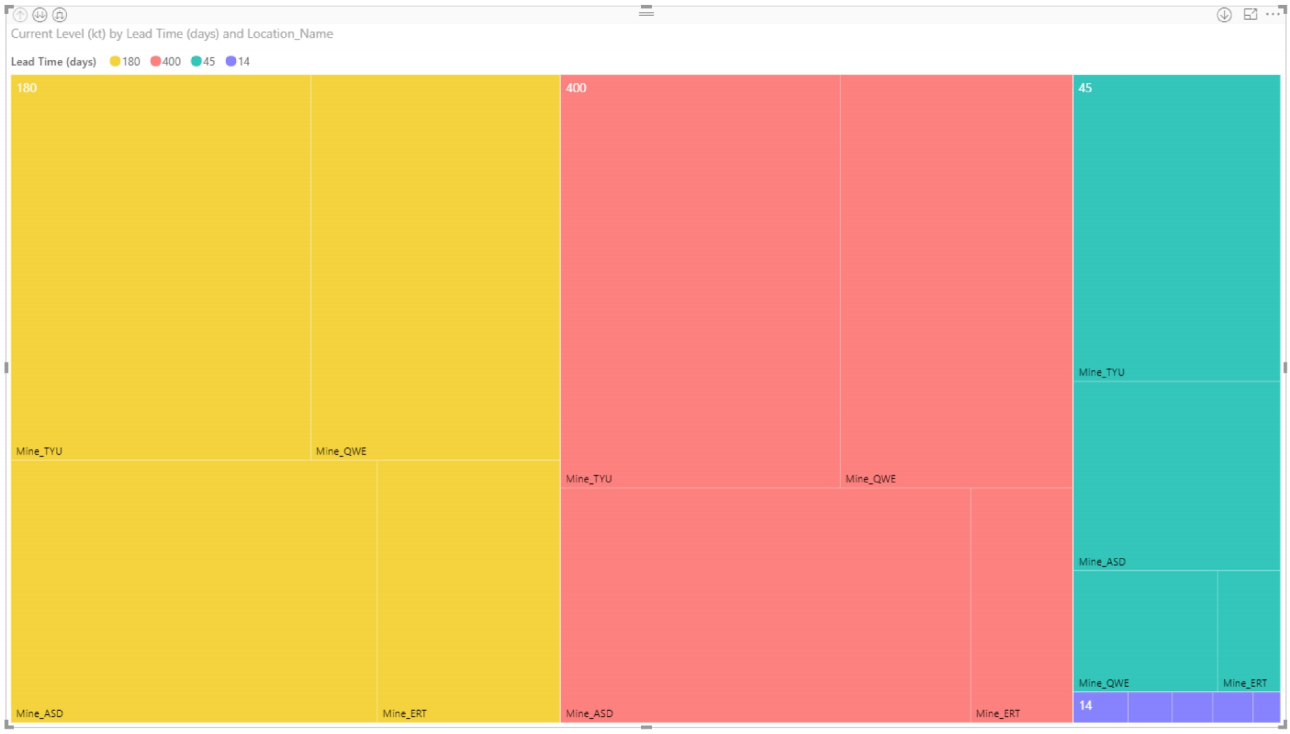


Fig.24 – Stockpiles – Current Leadtime(colour) by Stockpile and Quantity(size)

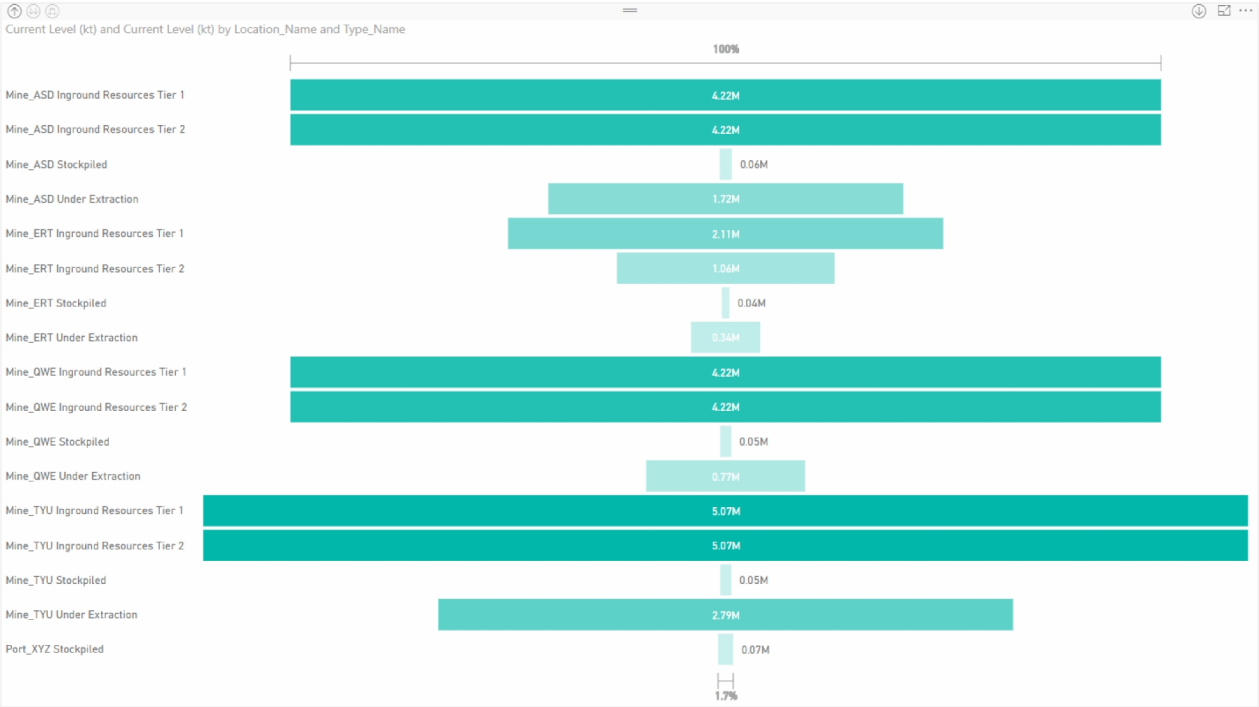


Fig.25 – Stockpiles – Current Stockpile Levels

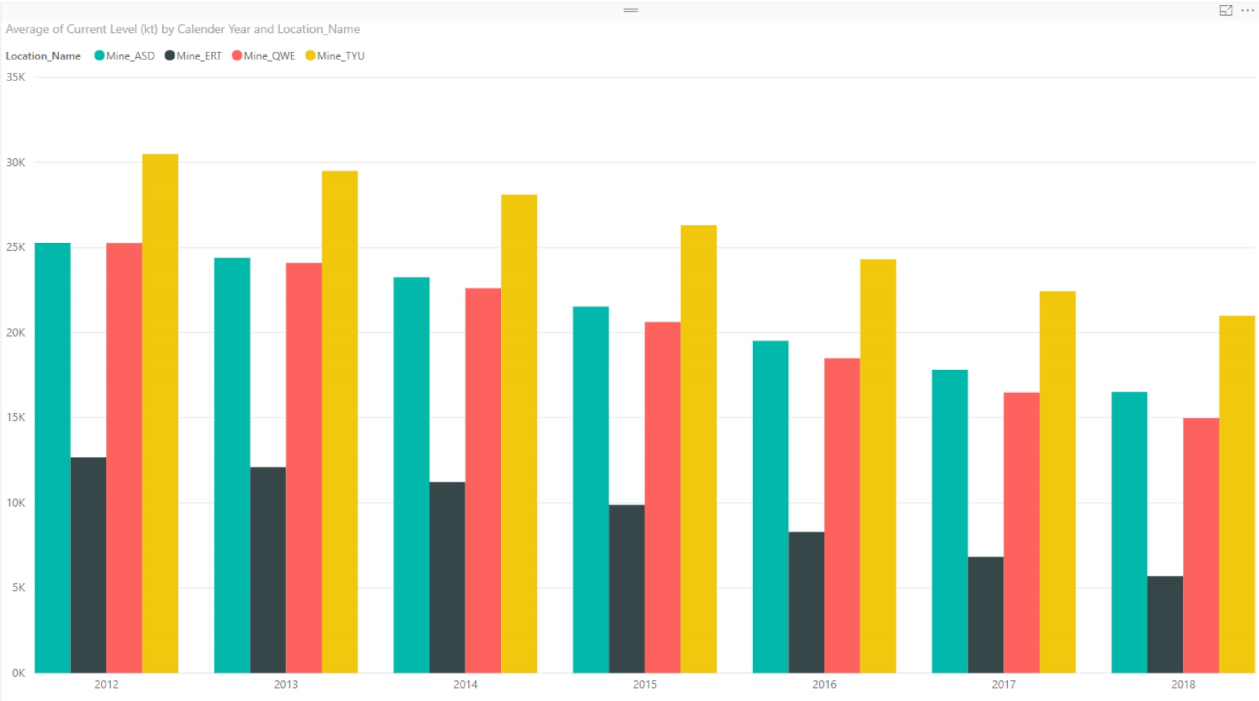


Fig.26 – Stockpiles – Mean Levels by Location - Timeseries

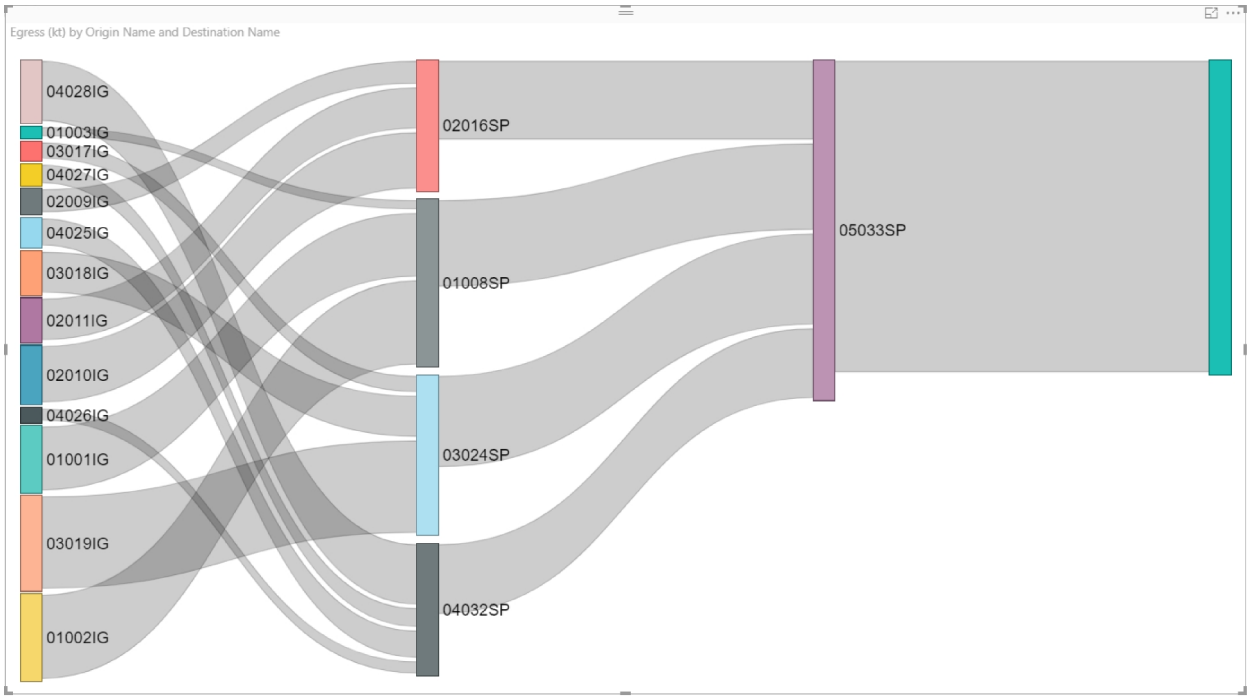


Fig.27 – Stockpiles – 2017 Stockpile Flow Chain – shows volumes going through stockpile nodes

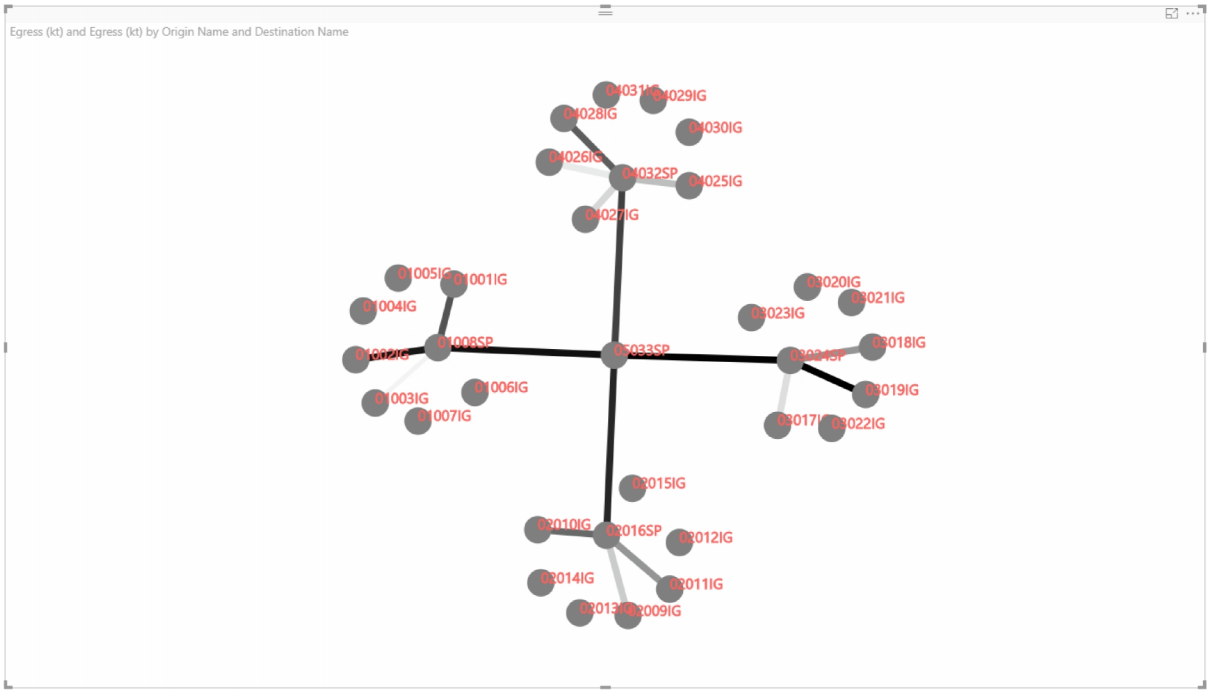


Fig.28 – Stockpiles – 2017 Stockpile Network Traffic Diagram – Whole Network

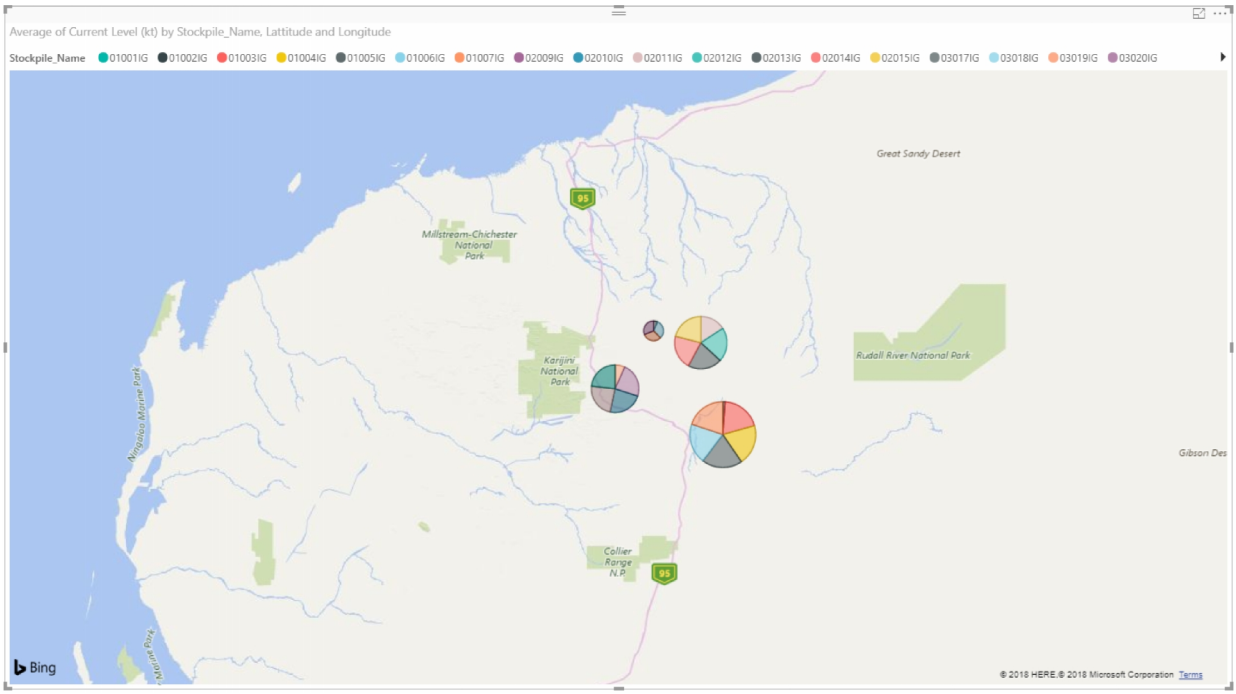


Fig.29 – Stockpiles – Current Stockpile Levels by Stockpile and Location - Geo

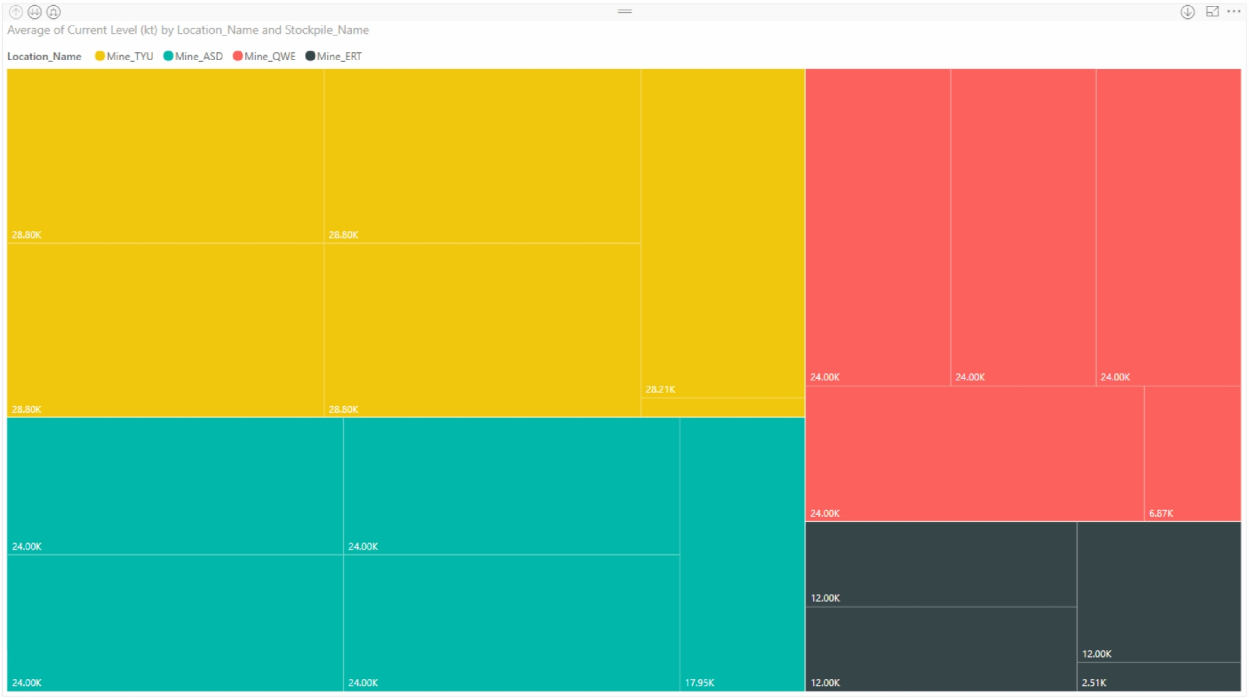


Fig.30 – Stockpiles – Current Stockpile Levels(size) by Stockpile(box) and Location(colour)

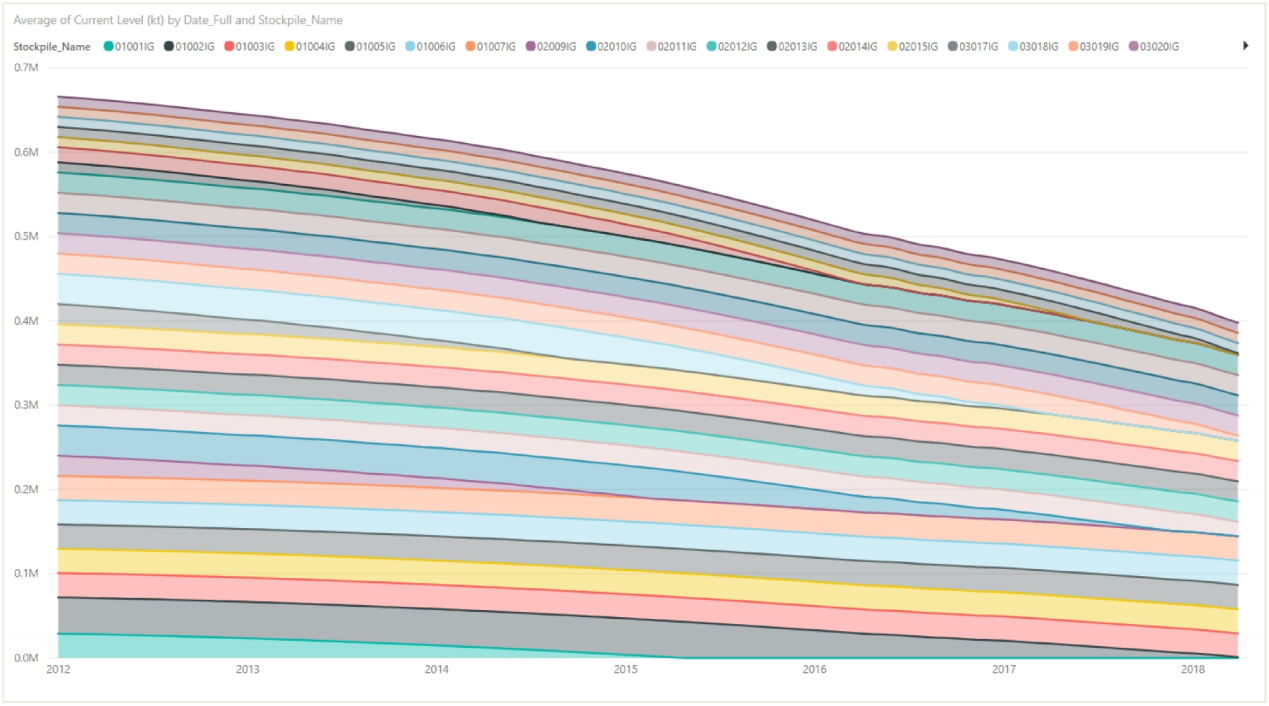


Fig.31 – Stockpiles – In Ground Deposits Decrease Over Last 5 Years - timeseries

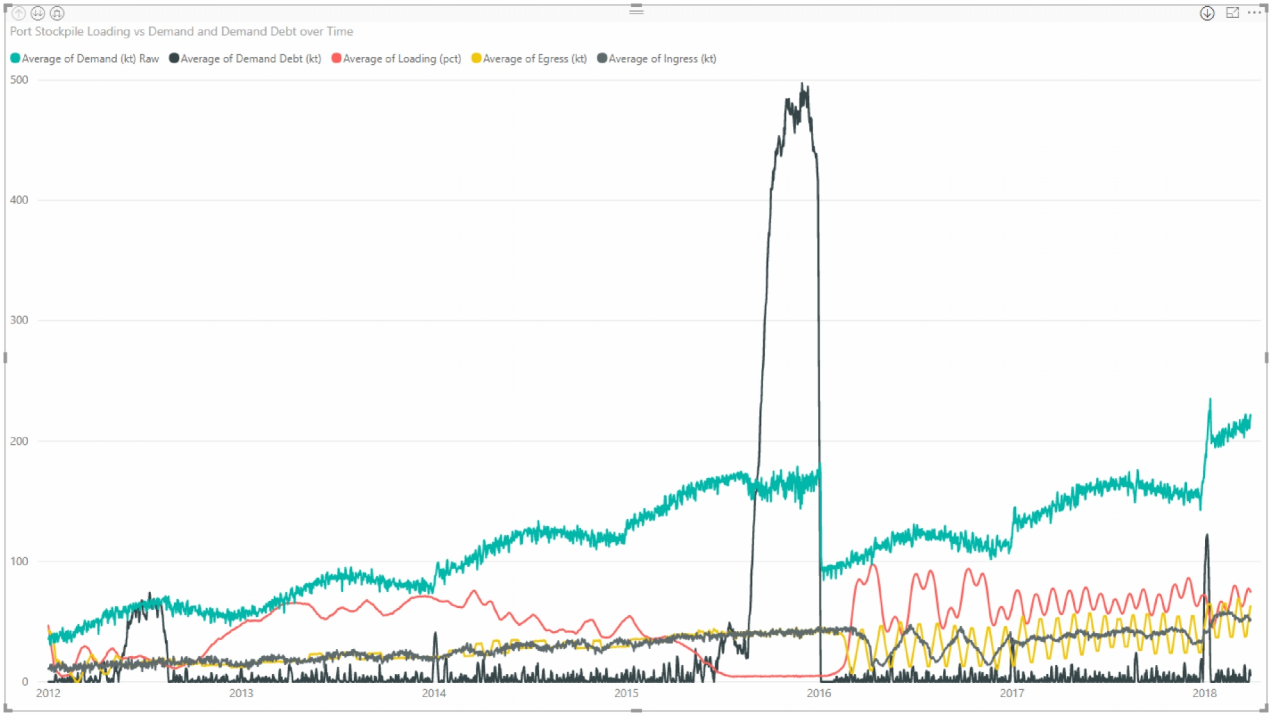


Fig.32 – Stockpiles – Port Stockpile Level, Demand, Demand Debt, Port Ingress/Egress - Timeseries

Orders

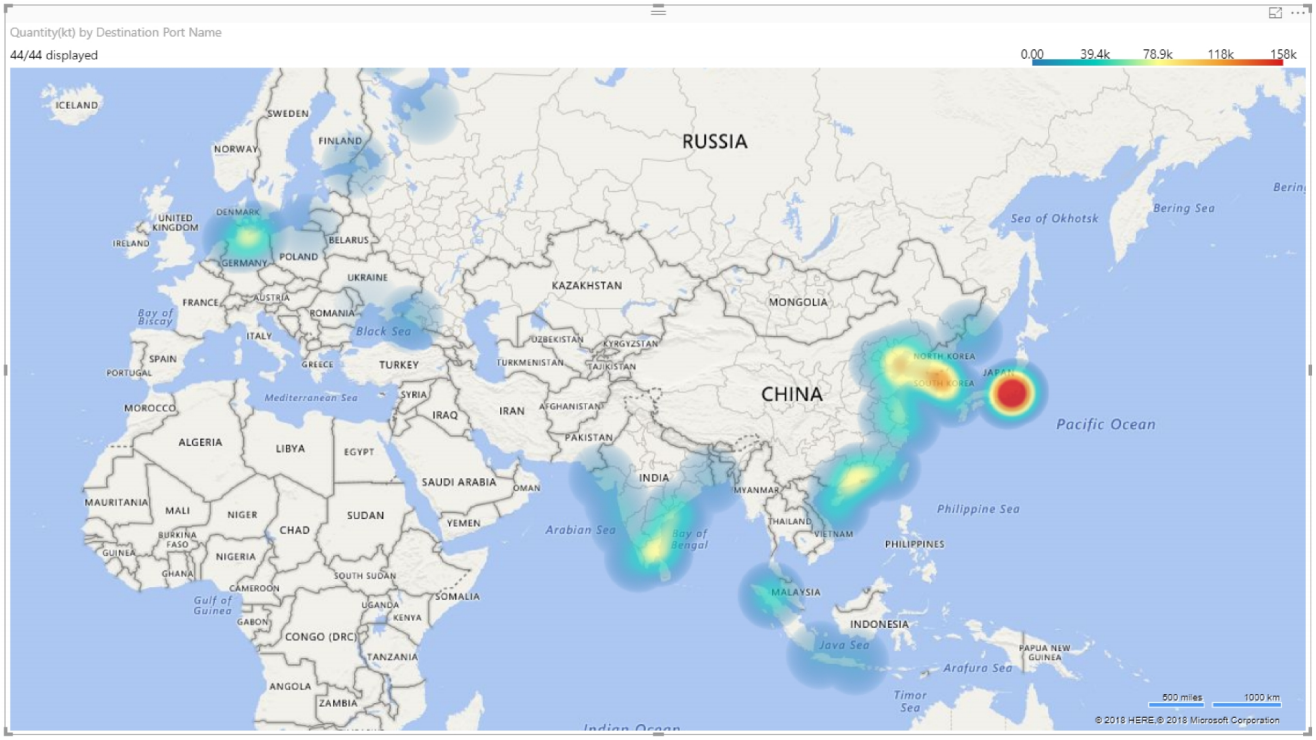


Fig.33 – Orders – 2017 Customer Order Volume Heatmap - Geo

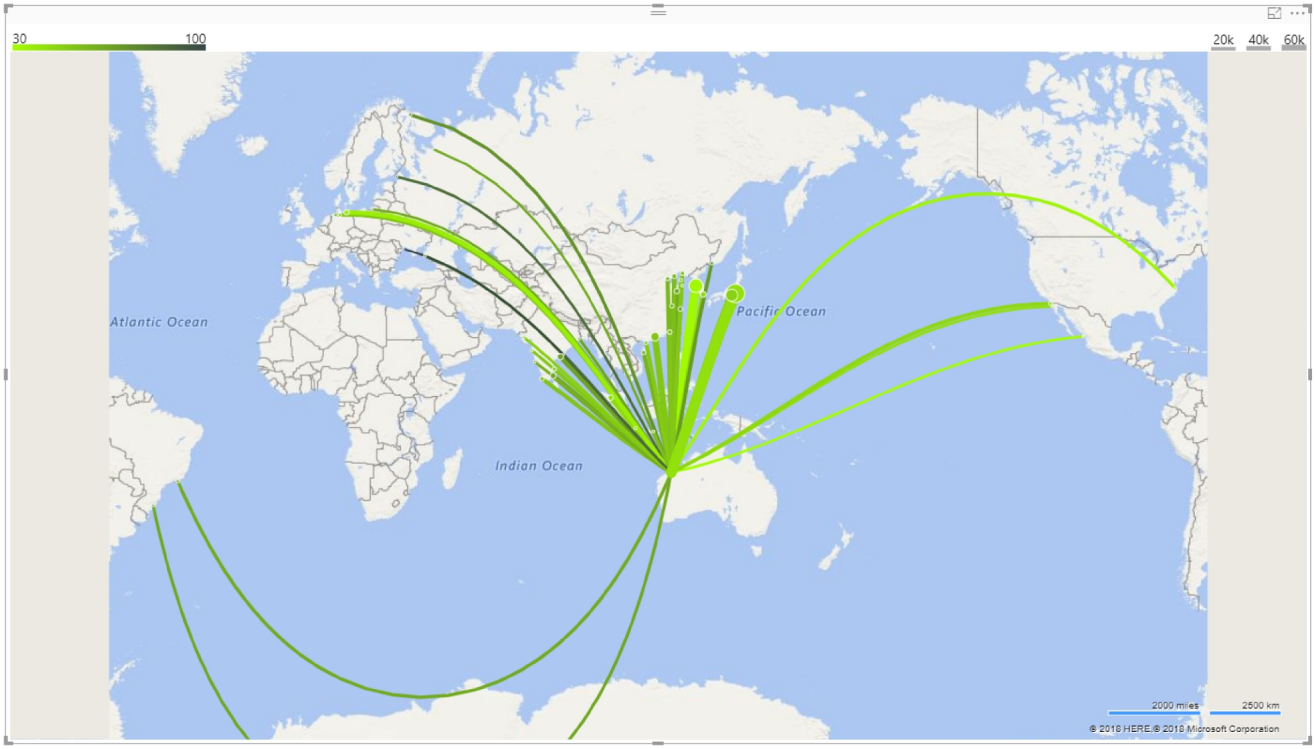


Fig.34 – Orders – 2017 Port to Port Flows - Geo

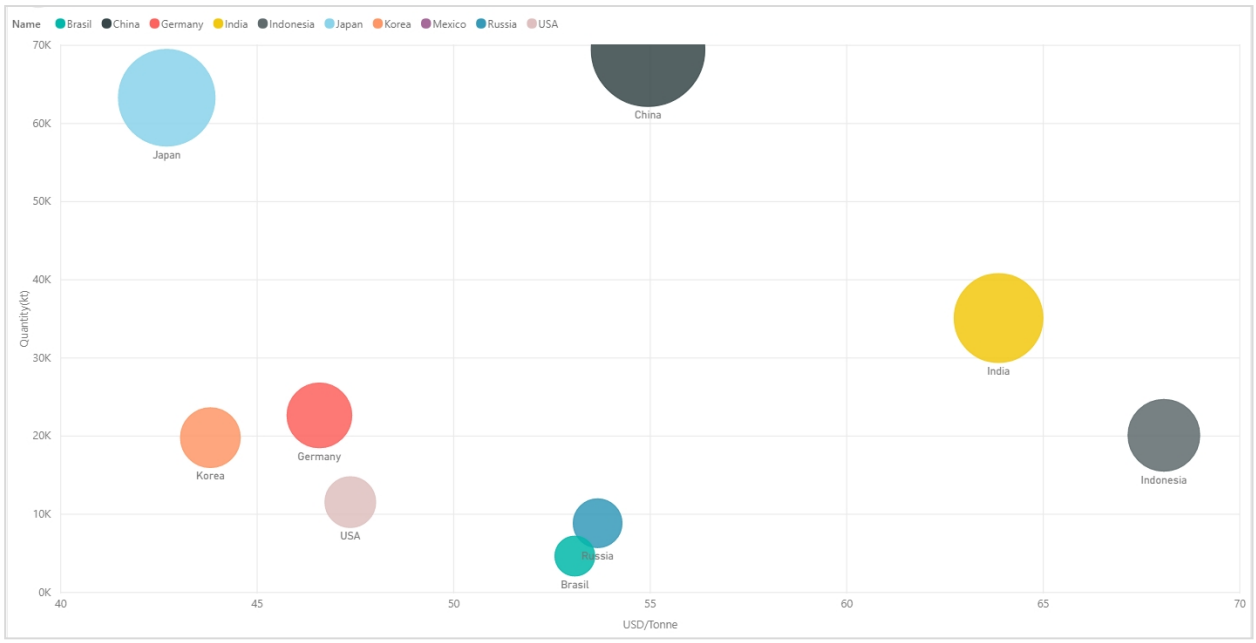


Fig.35 – Orders – Last 3 Year Volume vs USD/Tonne by Country

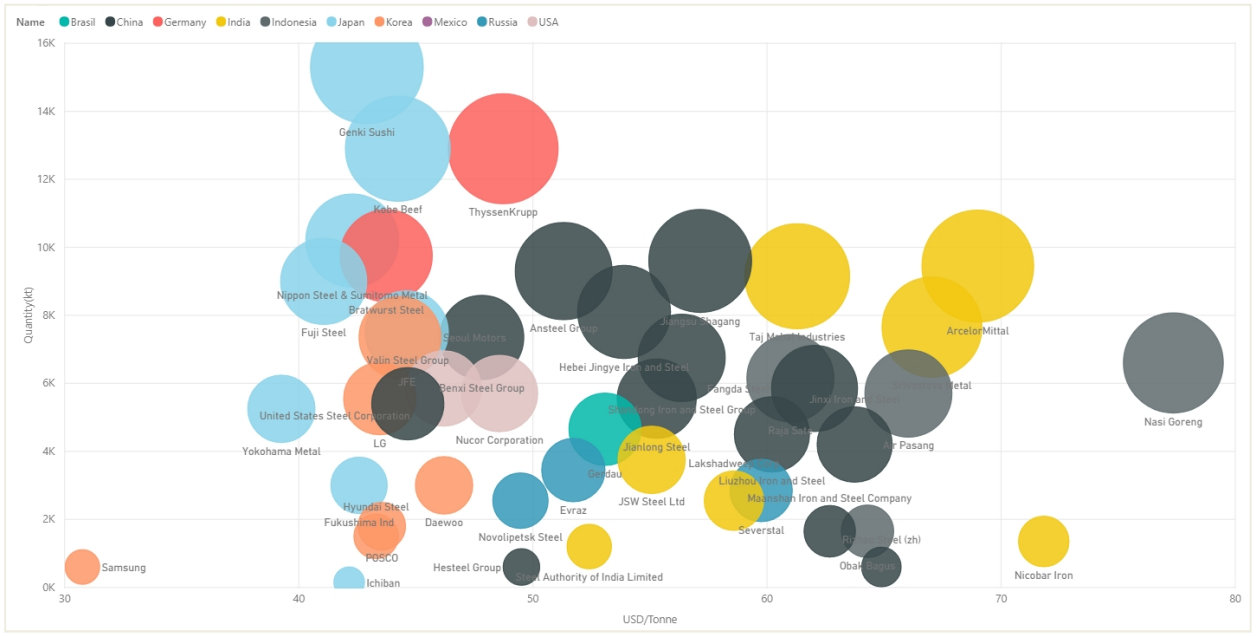


Fig.36 – Orders – Last 3 Year Volume vs USD/Tonne by Customer/Country

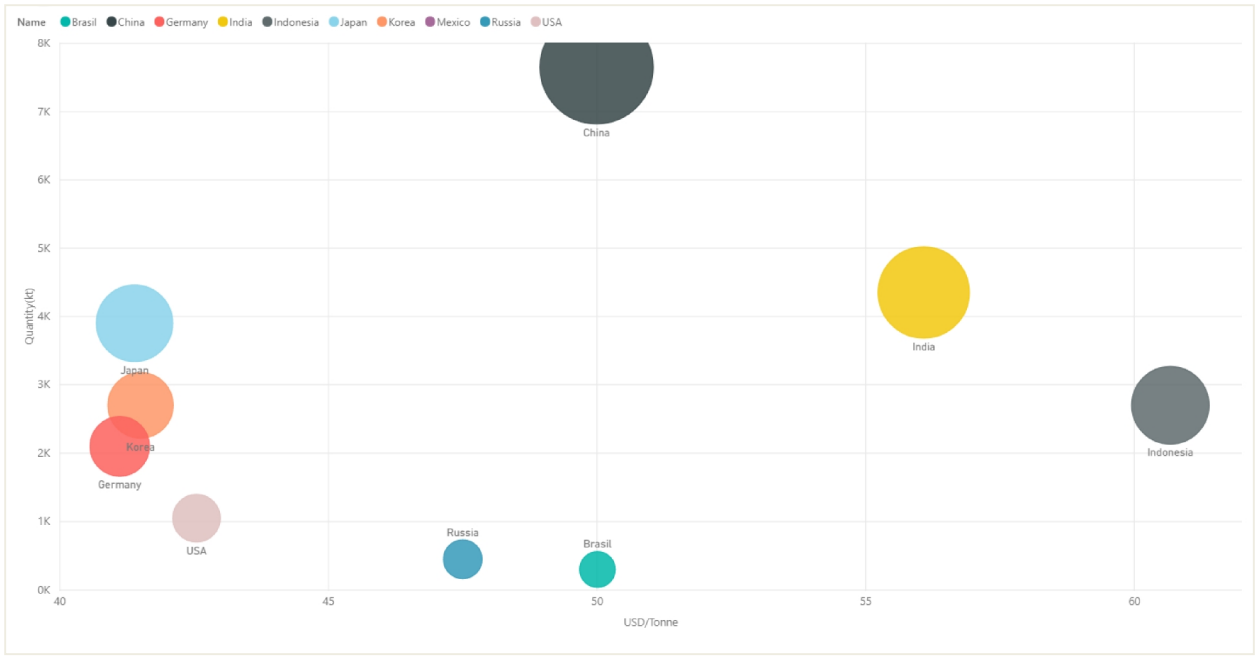


Fig.37 – Orders – Next 3 Year Volume vs USD/Tonne by Country

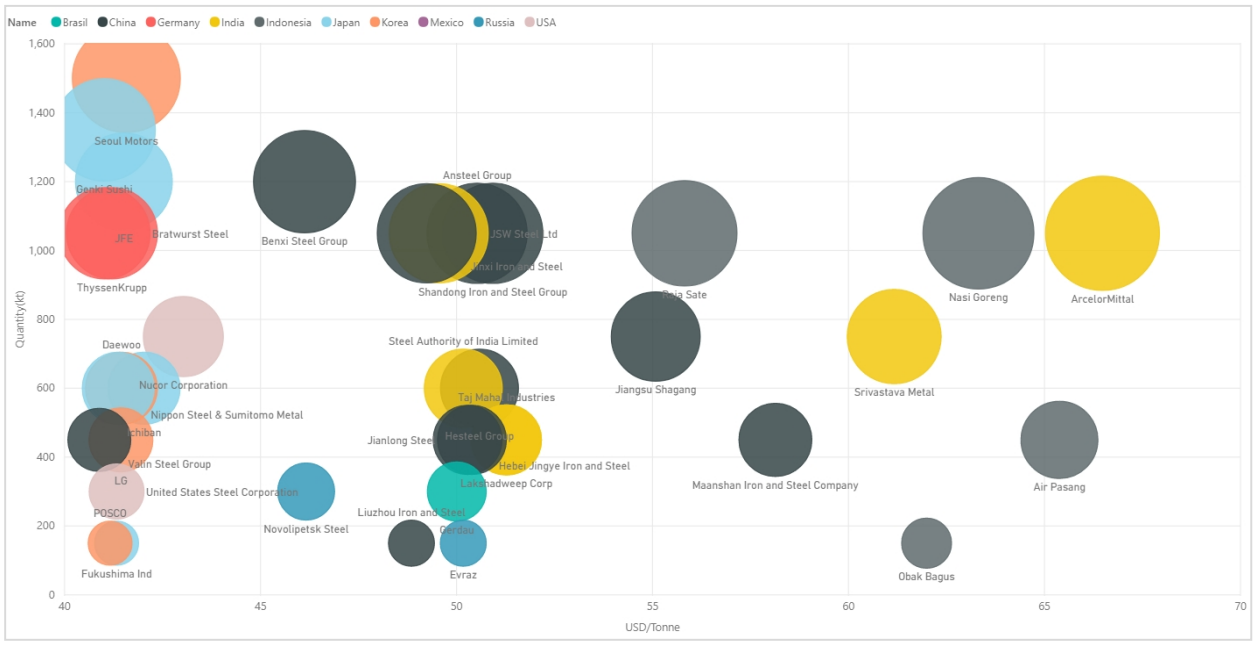


Fig.38 – Orders – Next 3 Year Volume vs USD/Tonne by Customer/Country

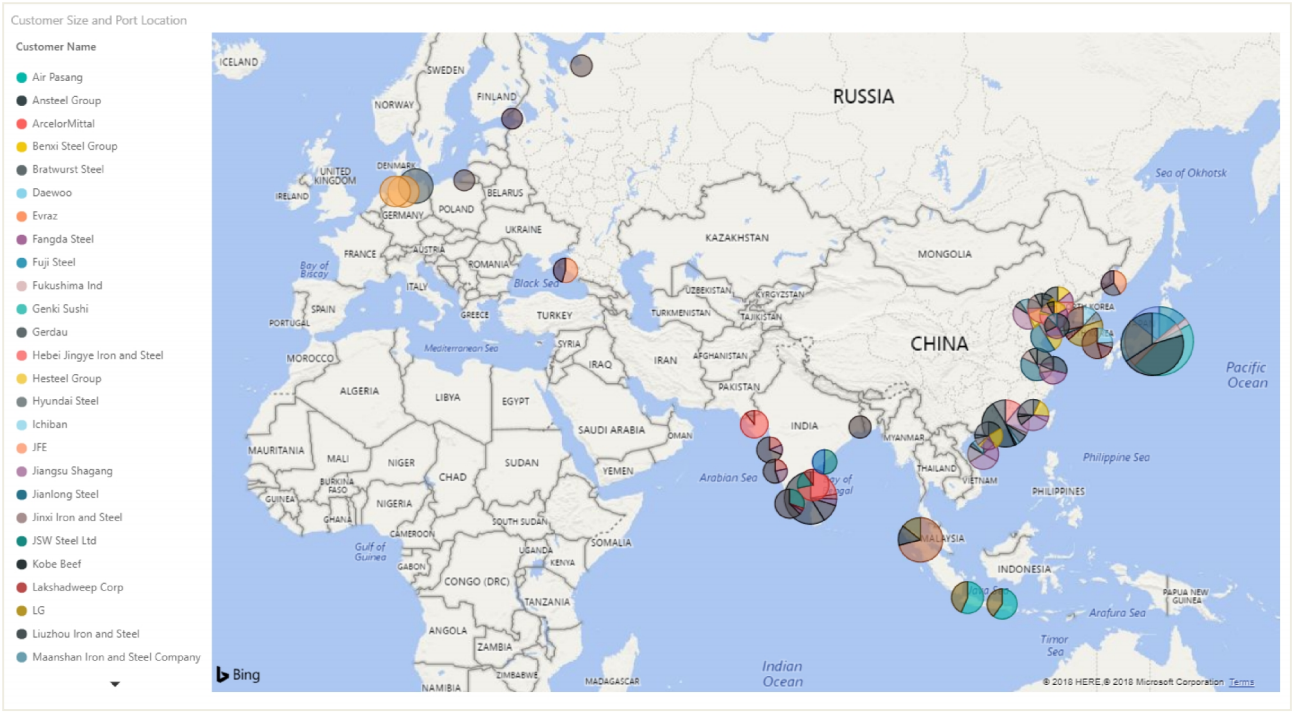


Fig.39 – Orders – Last 3 Years Customer Size by Port - Geo

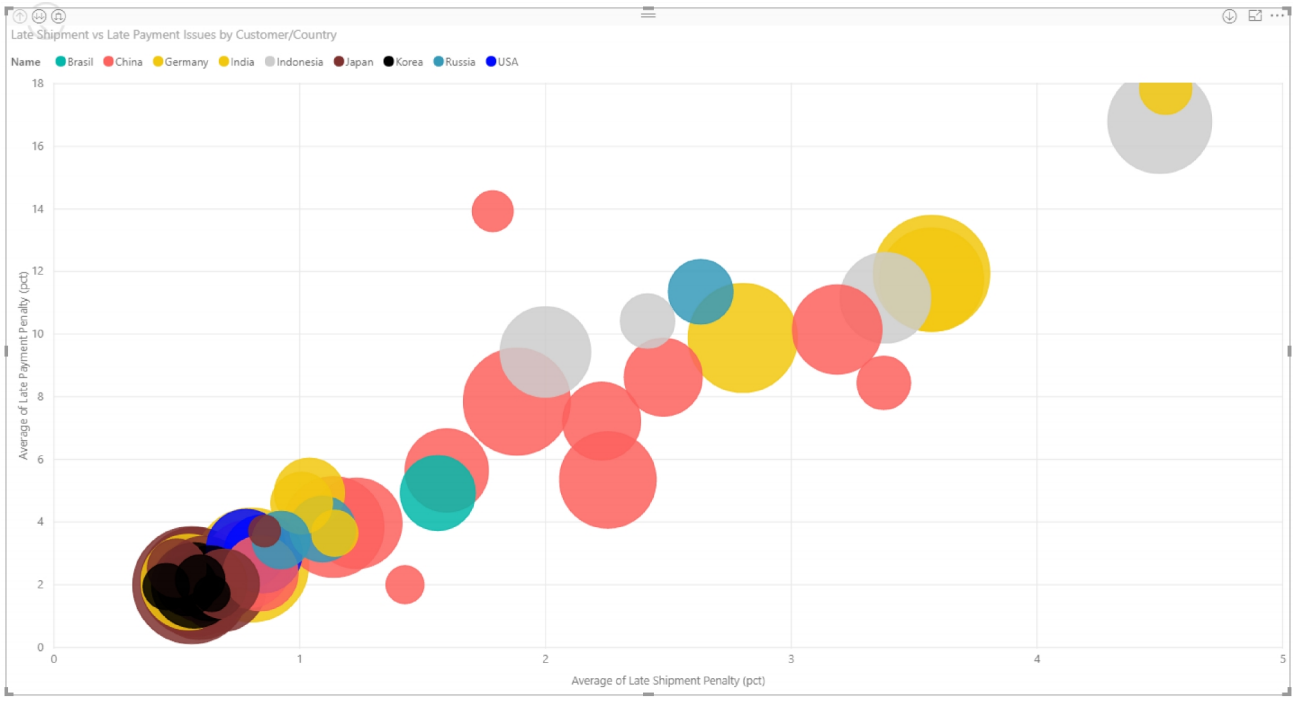


Fig.40 – Orders – Last 3 Year Late Payment Issues vs Late Shipment Issues by Volume(size) Customer/Country(colour)

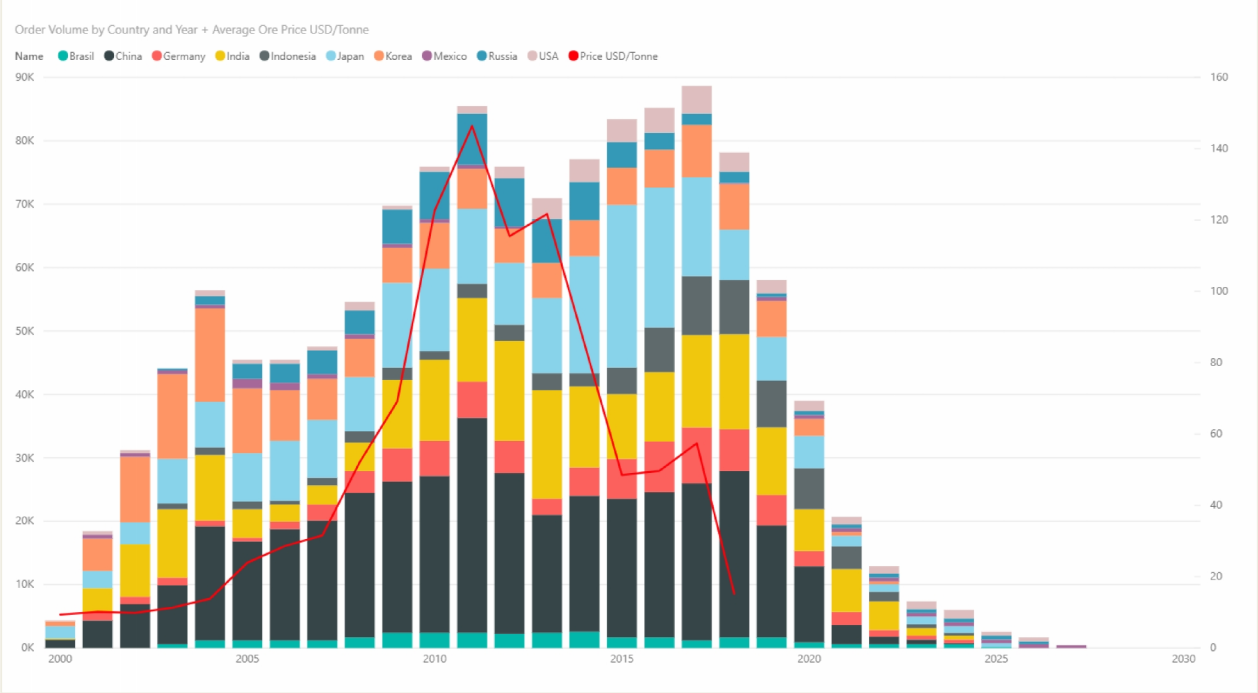


Fig.41 – Orders – Historical Volume by Country + USD/Tonne(line) - Timeseries

Appendix 1 – Data Generation Simulator Code

```
#Program for CITS5504 project
#Date: 12mar18
#Author: Nathan Scott

import Modules.utility_functions as utils
import Modules.trucks as tk
import Modules.Stockpiles as sp
import Modules.Orders as od
import os

def main():
    tk.run_truck_simulation()
    sp.run_stockpile_simulation()
    od.run_orders_simulation()

main()
############################################################
############################################################

#Trucks Simulator
import sys
import time
import math
import re
import os
import tkinter as tk
from tkinter import filedialog
import csv
import json
from datetime import datetime, timedelta
from random import randint
import pandas as pd
import numpy as np
import requests as rq
from bs4 import BeautifulSoup as bsup
import lxml   #even though not used, it is used in the reading of beautiful soup content, so you need to pip install it and import it



#UTILITY FUNCTIONS
#################################################################
def error_exit_procedure(e_msg):
    #prints any runtime errors to the console, gives the user time to read and then quits
    print(e_msg)
    sys.exit()

def load_csv_file_to_db(file_path):
    #function to load in the given file to a dict data structure
    output_data = {}
    import_data, e_msg = read_csv(file_path)
    if e_msg != '':
        error_exit_procedure(e_msg)
    attr_keys = [x for x in import_data[0][1:]]
    for row in import_data[1:]:
        mainkey = row[0]
        attr_vals = row[1:]
        output_data[mainkey] = dict(zip(attr_keys, attr_vals))
    return output_data

def load_csv_file_to_bridge_db(file_path):
    #function to load in the given file to a list of lists
    output_data = []
    import_data, e_msg = read_csv(file_path)
    if e_msg != '':
        error_exit_procedure(e_msg)
    for row in import_data:
        output_data.append((row[0],row[1]))
    return output_data

def get_file_dialog():
    #prompts the user to select a .txt file containing the text to be analysed
    #and returns the chosen complete file path
    error = ''
    root = tk.Tk()
    root.withdraw()
```

```
    file_path = filedialog.askopenfilename(title='Please select a single file with your given text to analyse.', filetypes = (('text', '*.txt'),('All files','*.*')))

    #check the file exists, if not spit back to user
    if not os.path.exists(file_path):
        error = "file or directory doesn't exist"

    return file_path, error

def load_file(file_path):
    #this loads the file into a string
    #the error string is filled if there are any problems
    #like: file already open error, or file doesn't exist error
    error = ''
    data = ''

    #try to read the file, catch various errors and spit back to user
    try:
        file = open(file_path, 'r')
        data = file.read()
        file.close()
    except Exception as e:
        error = str(e)

    return data, error

def read_csv(file_path):
    #loads a csv into a list of lists
    error = ''
    data = []

    #try to read the file, catch various errors and spit back to user
    try:
        csv_reader = csv.reader(open(file_path, newline=''), dialect='excel', delimiter=',', quotechar='"')
        for r in csv_reader:
            data.append(tuple(r))
    except Exception as e:
        error = str(e)

    return data, error


def write_csv(data, file_path, append_flag):
    # writes the list of lists to a csv file
    error = ''

    #try to read the file, catch various errors and spit back to user
    try:
        csv_writer = csv.writer(open(file_path, 'a' if append_flag else 'w', newline=''), delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
        for r in data:
            csv_writer.writerow(r)
    except Exception as e:
        error = str(e)

    return error


#####################################################################
#####################################################################
#####################################################################
```

Truck simulator

```
import Modules.utility_functions as utils


def run_truck_simulation():
    e_msg=''

    # local functions
    # ------------------------------------------------------
    def get_driver_id(p_date, truck):
        d_id = ''
        if truck_DB[truck]['Auto_Since'] == 'Not Yet':
            d_id = truck_DB[truck]['Driver_ID_Human']
        else:
            auto_date = utils.datetime.strptime(truck_DB[truck]['Auto_Since'], '%d-%b-%y')
            if auto_date < p_date:
                d_id = truck_DB[truck]['Driver_ID_Auto']
            else:
                d_id = truck_DB[truck]['Driver_ID_Human']
        return d_id

    def get_tire_limit(truck, poor_road_pct):
        tire_limit = float(truck_DB[truck]['Tire_Life_Poor_Surface (1000kms)']) *1000 * poor_road_pct/100\
                     + float(truck_DB[truck]['Tire_Life_Standard_Surface (1000kms)']) * 1000 * (1 -
poor_road_pct / 100)
        return round(utils.np.random.normal(tire_limit, tire_limit * 0.15, 1)[0], 0)

    def get_service_limit(service_code):
        return round(utils.np.random.normal(float(mission_DB[str(service_code)]['Service period mean
(km)']),float(mission_DB[str(service_code)]['Service period SD (km)']), 1)[0], 0)

    def get_trip_time(mission_id):
        #i am doing some weird rayleigh conversion to get trip duration here, I like the distribution
        mean = float(mission_DB[str(mission_id)]['Duration Mean (hrs)'])
        sd = 0.65*float(mission_DB[str(mission_id)]['Duration SD (hrs)'])
        duration = mean * (1 -  sd / mean) + utils.np.random.rayleigh(sd)
        return duration

    def get_trip_dist(mission_id):
        mean = float(mission_DB[str(mission_id)]['Distance Mean (km)'])
        sd = 0.8*float(mission_DB[str(mission_id)]['Distance SD (km)'])
        if mean == 0:
            return 0
        distance = mean*(1-sd/mean) + utils.np.random.rayleigh(sd)
        return distance

    def get_rest_time(mean, sd):
        rest_time = mean + utils.np.random.rayleigh(sd)
        return rest_time

    def get_trip_production(mission_id, truck, trip_duration):
        production = 0
        if mission_id == 6:
            # assume the truck max payload can be carried on a mission type 6 job 1x/hr
            production = float(truck_DB[truck]['Max payload (tonnes)']) * trip_duration
        return production

    def get_fuel_burn(truck, trip_time, d_id, poor_road_pct, mission_id):
        if float(mission_DB[str(mission_id)]['Use Fuel']) == 0:
            return 0
        idle_time = float(driver_DB[driver_id]['Time Waste Hrs'])
        fuel_burn_idle = float(truck_DB[truck]['Fuel_Burn_Idle (l/hr)']) * idle_time
        non_idle_loaded_time = (trip_time - idle_time) * 0.6
        non_idle_unloaded_time = (trip_time - idle_time) * 0.4
        fuel_burn_loaded = float(truck_DB[truck]['Fuel_Burn_Loaded_Poor_Surface (l/hr)']) * poor_road_pct / 100
+ \
                           float(truck_DB[truck]['Fuel_Burn_Loaded_Standard_Surface (l/hr)']) * (1 -
poor_road_pct / 100)
        fuel_burn_loaded *= non_idle_loaded_time * (1 + float(driver_DB[d_id]['Fuel Burn mult']))
        fuel_burn_unloaded = float(truck_DB[truck]['Fuel_Burn_Unloaded_Poor_Surface (l/hr)']) * poor_road_pct /
100 + \
                             float(truck_DB[truck]['Fuel_Burn_Unloaded_Standard_Surface (l/hr)']) * (
                                 1 - poor_road_pct / 100)
        fuel_burn_unloaded *= non_idle_unloaded_time * (1 + float(driver_DB[d_id]['Fuel Burn mult']))
        fuel_burn_total = fuel_burn_loaded + fuel_burn_unloaded + fuel_burn_idle
        return round(utils.np.random.normal(fuel_burn_total, fuel_burn_total * 0.1, 1)[0], 0)

    def get_parts_cost(mission_id, tire_change, accident_cost, truck):
        cost = 0
        if mission_id == 1:
```

```
            cost = float(truck_DB[truck]['Service_Type_1_Cost_Parts'])
        elif mission_id == 2:
            cost = float(truck_DB[truck]['Service_Type_2_Cost_Parts'])
        elif mission_id == 3:
            cost = float(truck_DB[truck]['Service_Type_3_Cost_Parts'])
        elif mission_id == 4:
            if tire_change > 0:
                cost = float(truck_DB[truck]['Tire Cost (1000AUD)']) * 1000 * tire_change
            else:
                cost = float(truck_DB[truck]['Service_Type_1_Cost_Parts'])
        elif mission_id == 5:
            if tire_change > 0:
                cost = float(truck_DB[truck]['Tire Cost (1000AUD)']) * 1000 * tire_change
            elif accident_cost > 0:
                cost = accident_cost
            else:
                cost = float(truck_DB[truck]['Service_Type_1_Cost_Parts'])
        return cost

    def get_labour_cost(mission_id, tire_change, accident_cost, truck):
        cost = 0
        if mission_id == 1:
            cost = float(truck_DB[truck]['Service_Type_1_Cost_Labour'])
        elif mission_id == 2:
            cost = float(truck_DB[truck]['Service_Type_2_Cost_Labour'])
        elif mission_id == 3:
            cost = float(truck_DB[truck]['Service_Type_3_Cost_Labour'])
        elif mission_id == 4:
            if tire_change > 0:
                cost = 500 * tire_change
            else:
                cost = float(truck_DB[truck]['Service_Type_1_Cost_Labour'])
        elif mission_id == 5:
            if tire_change > 0:
                cost = 500 * tire_change
            elif accident_cost > 0:
                cost = accident_cost / 7
            else:
                cost = float(truck_DB[truck]['Service_Type_1_Cost_Labour'])
        return cost

    def calc_events(d_id, duration):
        events = {}
        #this will scale the lambda (poisson mean by the duration of the trip, if more, there is more time, so
more chance for an event etc...
        time_adjust = duration/4.2
        events['Brake Temperature Events'] = utils.np.random.poisson(time_adjust*float(driver_DB[d_id]['Brake
Temperature Events']))
        events['Overspeed Events'] = utils.np.random.poisson(time_adjust*float(driver_DB[d_id]['Overspeed
Events']))
        events['Engine Overspeed Events'] = utils.np.random.poisson(time_adjust*float(driver_DB[d_id]['Engine
Overspeed Events']))
        events['Overload Events'] = utils.np.random.poisson(time_adjust*float(driver_DB[d_id]['Overload
Events']))
        events['Tire Wall Events'] = utils.np.random.poisson(time_adjust*float(driver_DB[d_id]['Tire Wall
events']))
        events['Tire Loss Events'] = utils.np.random.poisson(time_adjust*float(driver_DB[d_id]['Tire Loss
Events']))
        events['Accidents'] = utils.np.random.poisson(time_adjust*float(driver_DB[d_id]['Accidents']))
        return events

    def update_ufs_uws_tire_limits(events, uws, ufs, tire):
        tot_events = sum(list(events.values())[:5])
        uws += -12*tot_events
        ufs += -16*tot_events
        tire += -19*tot_events
        return uws, ufs, tire


    # ------------------------------------------------------------
    # ------------------------------------------------------------
    # load the csvs
    root_dir = 'D:/A.Nathan/1a.UWA 2018/CITS5504 - Data Warehousing/Assignment/Data/Truck Data/'
    truck_DB = utils.load_csv_file_to_db(root_dir + 'Trucks.csv')
    driver_DB = utils.load_csv_file_to_db(root_dir + 'Drivers.csv')
    location_DB = utils.load_csv_file_to_db(root_dir + 'Locations.csv')
    mission_DB = utils.load_csv_file_to_db(root_dir + 'Missions.csv')

    output_DB = [[
        'Date',
        'Time Out',
```

```
    'Time In',
    'Truck_Id',
    'Driver_Id',
    'Location_Id',
    'Mission_Id',
    'Duration (hrs)',
    'Distance (kms)',
    'Product Delivered (tonnes)',
    'Ave Speed',
    'Parts Cost',
    'Labour Cost',
    'Fuel Burn',
    'Brake Temperature Events',
    'Overspeed Events',
    'Engine Overspeed Events',
    'Overload Events',
    'Tire Wall events',
    'Tire Loss Events',
    'Accidents']]

# write headers to csv
utils.write_csv(output_DB, root_dir + 'Output.csv', False)

#truck_DB = {k: v for k, v in truck_DB.items() if float(k) <= 1}

i = 0
tot = len(truck_DB)
for truck in truck_DB:
    i += 1
    print('Doing truck ' + str(i) + ' of ' + str(tot))

    output_DB = []
    start_date = utils.datetime.strptime('01-01-2012', '%d-%m-%Y')
    end_date = utils.datetime.strptime('29-03-2018', '%d-%m-%Y')
    current_date = start_date

    # init counter vars
    poor_road_pct = float(location_DB[truck_DB[truck]['Location_Id']]['Poor Road pct'])
    driver_id = get_driver_id(current_date, truck)
    location_id = truck_DB[truck]['Location_Id']
    tire_limit = get_tire_limit(truck, poor_road_pct)
    s1_limit = get_service_limit(1)
    s2_limit = get_service_limit(2)
    s3_limit = get_service_limit(3)
    uws_limit = get_service_limit(4)
    ufs_limit = get_service_limit(5)
    s1_dist = 0
    s2_dist = 0
    s3_dist = 0
    tire_dist = 0
    uws_dist = 0  # unsched_workshop service
    ufs_dist = 0  # unsched_field service
    trip_distance = 0

    break_flag = False
    while break_flag is False:
        # increment counters
        s1_dist += trip_distance
        s2_dist += trip_distance
        s3_dist += trip_distance
        tire_dist += trip_distance
        uws_dist += trip_distance  # unsched_workshop service
        ufs_dist += trip_distance  # unsched_field service

        # reset vars
        tire_change = 0
        accident_cost = 0
        trip distance = 0
        trip_duration = 0

        # start - around a 15 mins break with rayleigh dist
        rest_time = get_rest_time(10, 5)
        time_out = current_date + utils.timedelta(minutes=rest_time)
        driver_id = get_driver_id(current_date, truck)

        #init events_DB
        event_DB = calc_events(driver_id, 0)

        # test for mission type
        if s1_dist > s1_limit:
            s1_limit = get_service_limit(1)
```

```
                s1_dist = 0
                mission_id = 1
                s1_limit = get_service_limit(1)
            elif s2_dist > s2_limit:
                s2_limit = get_service_limit(2)
                s2_dist = 0
                mission_id = 2
            elif s3_dist > s3_limit:
                s3_limit = get_service_limit(3)
                s3_dist = 0
                mission_id = 3
            elif uws_dist > uws_limit:
                uws_limit = get_service_limit(4)
                uws_dist = 0
                mission_id = 4
            elif ufs_dist > ufs_limit:
                ufs_limit = get_service_limit(5)
                ufs_dist = 0
                mission_id = 5
            elif tire_dist > tire_limit:
                tire_limit = get_tire_limit(truck, poor_road_pct)
                tire_dist = 0
                mission_id = 4
                tire_change = 6
            else:
                # toss a coin for 6 or 7
                mission_id = 6
                if utils.np.random.choice(a=[0, 1], size=1, p=[0.5, 0.5])[0] == 0: mission_id = 7
                #calc trip dist and duration early for this type of mission
                trip_distance = get_trip_dist(mission_id)
                trip_duration = get_trip_time(mission_id)
                #only calc events for field missions (id = 6 or 7)
                event_DB = calc_events(driver_id, trip_duration)
                #update the ufs-uws-tire limits due to events, basically shorten a bit depending on the number
of events, so worse drivers will be more expensive
                uws_limit, ufs_limit, tire_limit = update_ufs_uws_tire_limits(event_DB, uws_limit, ufs_limit,
tire_limit)
                # test for mission ending event (tire failure, breakdown)
                if event_DB['Tire Loss Events'] > 0:
                    ufs_limit = get_service_limit(5)
                    ufs_dist = 0
                    mission_id = 5
                    tire_change = 1
                elif event_DB['Accidents'] > 0:
                    ufs_limit = get_service_limit(5)
                    ufs_dist = 0
                    mission_id = 5
                    accident_cost = utils.np.random.normal(20000, 5000, 1)[0]

            # get mission dist and duration if not calculated already for mission 6 or 7
            if mission_id not in [6,7]:
                trip_distance = get_trip_dist(mission_id)
                trip_duration = get_trip_time(mission_id)

            # now calc output for given mission
            trip_production = get_trip_production(mission_id, truck, trip_duration)
            if trip_duration == 0:
                ave_speed = 0
            else:
                ave_speed = trip_distance / trip_duration
            parts_cost = get_parts_cost(mission_id, tire_change, accident_cost, truck)
            labour_cost = get_labour_cost(mission_id, tire_change, accident_cost, truck)
            fuel_burn = get_fuel_burn(truck, trip_duration, driver_id, poor_road_pct, mission_id)

            # fill out output data
            current_date += utils.timedelta(hours=trip_duration)
            time_in = current_date
            new data = [utils.datetime.date(time out), time out, time in, truck, driver id, location id,
mission_id,
                        trip_duration, trip_distance, trip_production, ave_speed, parts_cost, labour_cost,
fuel_burn]
            new_data += list(event_DB.values())
            output_DB.append(new_data)

            if current_date > end_date:
                break_flag = True

        # when finished with each truck, append to csv
        utils.write_csv(output_DB, root_dir + 'Output.csv', True)
    # ------------------------------------------------------------------------------
```

Stockpiles Simulator

```
import Modules.utility_functions as utils



# trains from mines to port can carry aroudn 25K tonnes => normally distribute or something but cap it
# trucks, we know, get stats from your truck analysis on ave daily Tput
# deposit size ave and distribution
# stockpile capacities
# ship Tput



#Stockpile_Id,Stockpile_Name,Location_Id,Max Capacity (kt),Type,Tier,Lead Time (mths)
#Location_Id,Location_Name,Location_Code,Poor Road pct,Ave Train Capacity (kt/d),Max Train Capacity (kt/d),Ave Shipping Capacity (kt/d),Max Shipping Capacity (kt/d)

def run_stockpile_simulation():
    e_msg = ''


    def convert_type(type):
        if type == 'IG_0':
            type = 2
        elif type == 'IG_1':
            type = 3
        elif type == 'IG_2':
            type = 4
        elif type == 'IG_3':
            type = 5
        elif type == 'SP_0':
            type = 1
        return type

    def calc_new_level(incoming, outgoing, pile_id):
        current_level = stockpile_DB[pile_id]['Quantity (kt)']
        max_level = float(stockpile_DB[pile_id]['Max Capacity (kt)'])
        new_level_0 = current_level + incoming - outgoing
        new_level = min(max_level,max(0,new_level_0))
        stockpile_DB[pile_id]['Quantity (kt)'] = new_level
        #workout excess and shortfalls
        excess, shortfall = 0,0
        if new_level < new_level_0:
            excess = new_level_0 - new_level
        elif new_level > new_level_0:
            shortfall = new_level - new_level_0
        #calc real ingress and egress
        stockpile_DB[pile_id]['Ingress (kt)'] = incoming - excess
        stockpile_DB[pile_id]['Egress (kt)'] = outgoing - shortfall
        stockpile_DB[pile_id]['Ingress Excess (kt)'] = excess
        stockpile_DB[pile_id]['Egress Shortfall (kt)'] = shortfall
        return excess

    def move_ig_tier(loc_id):
        #this moves one tier 3 to tier 2, one tier 2 to tier 1 and one tier 1 to tier 0
        t1_flag, t2_flag, t3_flag = False, False, False
        for pile_id in [x for x in stockpile_DB if stockpile_DB[x]['Location_Id'] == loc_id and float(stockpile_DB[x]['Tier']) > 0]:
            if t1_flag is False and stockpile_DB[pile_id]['Tier'] == '1':
                stockpile_DB[pile_id]['Tier'] = '0'
                stockpile_DB[pile_id]['Lead Time (mths)'] = '2'
                t1_flag = True
            if t2_flag is False and stockpile_DB[pile_id]['Tier'] == '2':
                stockpile_DB[pile_id]['Tier'] = '1'
                stockpile_DB[pile_id]['Lead Time (mths)'] = '6'
                t2_flag = True
            if t3_flag is False and stockpile_DB[pile_id]['Tier'] == '3':
                stockpile_DB[pile_id]['Tier'] = '2'
                stockpile_DB[pile_id]['Lead Time (mths)'] = '12'
                t3_flag = True

    def calc_new_params(days, demand_debt):
        demand_year_rand = [1,1,1.1,1.2,.7,.8,1,1,1,1]
        train_year_rand = [1,1,1,1,1,1,1,1,1,1]
        truck_year_rand = [1,1,1,1,1,.8,1,1,1,1]
        #calc new demand and max caps
```

```
    #do demand first
    demand = demand_year_rand[int(days/365)]*(demand_int + demand_slope*(days**1.2))
    demand += demand_noise_const*utils.math.cos(2*utils.math.pi*(days%365)/365+utils.math.pi)
    demand = max(0,demand - demand_noise_const*abs(utils.np.random.normal(0,0.5,1)[0]))
    demand += demand_debt

    #do ships
    ship_cap = float(location_DB['5']['Orig Shipping Capacity (kt/d)']) + ship_slope*(days)
    ship_cap = max(0,ship_cap - ship_noise_const*abs(utils.np.random.normal(0,0.5,1)[0]))
    location_DB['5']['Cur Shipping Capacity (kt/d)'] = ship_cap

    #do mines trains and trucks
    for item in [x for x in location_DB if x != '5']:
        train_cap = train_year_rand[int(days/365)]*(float(location_DB[item]['Orig Train Capacity (kt/d)']) + train_slope * (days))
        train_cap = max(0,train_cap - train_noise_const * abs(utils.np.random.normal(0, max(0.25,0.5-0.25*days/2300), 1)[0]))
        location_DB[item]['Cur Train Capacity (kt/d)'] = train_cap
        truck_cap = truck_year_rand[int(days/365)]*(float(location_DB[item]['Orig Mine Production Capacity (kt/d)']) + truck_slope * (days))
        truck_cap = max(0,truck_cap - truck_noise_const * abs(utils.np.random.normal(0, max(0.25,0.5-0.25*days/2300), 1)[0]))
        location_DB[item]['Cur Mine Production Capacity (kt/d)'] = truck_cap

    return demand
#--------------------------------------------------------------------------------
#--------------------------------------------------------------------------------



# load the csvs
root_dir = 'D:/A.Nathan/1a.UWA 2018/CITS5504 - Data Warehousing/Assignment/Data/'
stockpile_DB = utils.load_csv_file_to_db(root_dir + 'Stockpile Data/Stockpiles.csv')
location_DB = utils.load_csv_file_to_db(root_dir + 'Truck Data/Locations.csv')

output_DB = [[
        'Date',
        'Stockpile_Id',
        'Location_Id',
        'Type_Id',
        'Current Level (kt)',
        'Loading (pct)',
        'Demand (kt)',
        'Demand Debt (kt)',
        'Ingress (kt)',
        'Ingress Excess (kt)',
        'Egress (kt)',
        'Egress Shortfall (kt)'
        ]]

# write headers to csv
utils.write_csv(output_DB, root_dir + 'Stockpile Data/Output.csv', False)

output_DB = []
start_date = utils.datetime.strptime('01-01-2012', '%d-%m-%Y')
end_date = utils.datetime.strptime('29-03-2018', '%d-%m-%Y')
current_date = start_date

#initialise stockpiles
#add the current quantity, ingress and egress holders per stockpile Id
for pile in stockpile_DB:
    stockpile_DB[pile]['Ingress (kt)'] = 0
    stockpile_DB[pile]['Egress (kt)'] = 0
    stockpile_DB[pile]['Ingress Excess (kt)'] = 0
    stockpile_DB[pile]['Egress Shortfall (kt)'] = 0
    if stockpile_DB[pile]['Type'] == 'IG':
        stockpile_DB[pile]['Quantity (kt)'] = float(stockpile_DB[pile]['Max Capacity (kt)'])
    else:
        stockpile_DB[pile]['Quantity (kt)'] = float(stockpile_DB[pile]['Max Capacity (kt)'])/2

#start time
cap_low_limit_port = 30
cap_limit_port = 80
cap_limit_mine = 80
#low limit, we schedule tier move at this level of tier 0
cap_limit_low_ig0 = 20

demand = 50     #original demand kt/d
demand_debt = 0
ship_reduction = 0
ship_reduction_step = 2
train_reduction = 0
```

```
    train_reduction_next = 0
    train_reduction_step = 2
    truck_reduction = {'1':0,'2':0,'3':0,'4':0}
    truck_reduction_next = {'1':0,'2':0,'3':0,'4':0}
    truck_reduction_step = 2
    extraction_project = []
    extraction_project_lead_time = 60

    #parameter models
    demand_int = demand
    demand_slope = 0.016
    demand_noise_fraction = 0.05
    demand_noise_const = 10
    ship_slope = 0.095
    ship_noise_const = 10
    train_slope = 0.05
    train_noise_const = 5
    truck_slope = 0.02
    truck_noise_const = 15

    break_flag = False
    while break_flag is False:
        #this calcs new values daily for demand and max capacities of all nodes => basically from different growth curves + noise
        demand = calc_new_params((current_date - start_date).days, demand_debt)
        print(str(demand_debt))

        #check for expiring extraction projects
        #it will move one tier 1 to 0, 1 tier 2 to 1 and one tier 3 to 2
        for project in extraction_project:
            project[1] += -1
            if project[1] == 0:
                move_ig_tier(project[0])

        #get rid of processed extraction projects
        extraction_project = [x for x in extraction_project if x[1] > 0]

        #do the processing
        # port first
        pile_id = [x for x in stockpile_DB if stockpile_DB[x]['Location_Id'] == '5'][0]
        egress=0
        egress = min(demand,min((1-ship_reduction/100)*float(location_DB['5']['Cur Shipping Capacity (kt/d)']),float(stockpile_DB[pile_id]['Quantity
(kt)'])))
        demand_debt = demand - egress
        ingress = (1-train_reduction/100)*sum([min(stockpile_DB[x]['Quantity (kt)'],float(location_DB[stockpile_DB[x]['Location_Id']]['Cur Train Capacity
(kt/d)'])) for x in stockpile_DB if stockpile_DB[x]['Location_Id'] != '5' and stockpile_DB[x]['Type'] == 'SP'])
        excess = calc_new_level(ingress, egress, pile_id)
        #if excess > 0:
        #    train_reduction = 100*(1-(ingress-excess)*(1-train_reduction/100)/ingress)
        # check port overload levels
        pct_load = 100*stockpile_DB[pile_id]['Quantity (kt)']/float(stockpile_DB[pile_id]['Max Capacity (kt)'])
        if pct_load > cap_limit_port:
            train_reduction_next = min(100, train_reduction + train_reduction_step)
        elif pct_load < cap_limit_port - 20:
            train_reduction_next = max(0, train_reduction - train_reduction_step)
        #control shipping capcity to meet demand
        if demand_debt > 0:
            ship_reduction = max(0, ship_reduction - ship_reduction_step)
        else:
            ship_reduction = min(100, ship_reduction + ship_reduction_step)

        #now work through each mine
        for mine in [x for x in location_DB if x != '5']:
            # main mine stockpile next
            ig0piles2process = [x for x in stockpile_DB if stockpile_DB[x]['Location_Id'] == mine and stockpile_DB[x]['Type'] == 'IG' and stockpile_DB[x]['Tier']
== '0' and stockpile_DB[x]['Quantity (kt)'] > 0 and mine != '5']
            total_ig0_quantity = sum([stockpile_DB[x]['Quantity (kt)'] for x in ig0piles2process])
            pile_id = [x for x in stockpile_DB if stockpile_DB[x]['Location_Id'] == mine and stockpile_DB[x]['Type'] == 'SP' and mine != '5'][0]
            egress = (1-train_reduction/100)*min(stockpile_DB[pile_id]['Quantity (kt)'], float(location_DB[mine]['Cur Train Capacity (kt/d)']))
            ingress = (1-truck_reduction[mine]/100)*min(total_ig0_quantity, float(location_DB[mine]['Cur Mine Production Capacity (kt/d)']))
            excess = calc_new_level(ingress, egress, pile_id)
        #    if excess > 0:
        #        truck_reduction[mine] = 100 * (1 - (ingress - excess) * (1 - truck_reduction[mine] / 100) / ingress)
            #check mine overload levels
            pct_load = 100 * stockpile_DB[pile_id]['Quantity (kt)'] / float(stockpile_DB[pile_id]['Max Capacity (kt)'])

            #if mine == '1': print(pct_load)

            if pct_load > cap_limit_mine:
```

```
            truck_reduction_next[mine] = min(100, truck_reduction[mine] + truck_reduction_step)
        elif pct_load < cap_limit_mine - 20:
            truck_reduction_next[mine] = max(0, truck_reduction[mine] - truck_reduction_step)

        # ig0 - under extraction ore next
        for pile_id in ig0piles2process:
            if total_ig0_quantity == 0:
                egress = 0
            elif len(ig0piles2process) == 1:
                egress = (1 - truck_reduction[mine] / 100) * float(location_DB[mine]['Cur Mine Production Capacity (kt/d)'])
            else:    # len(ig0piles2process) > 1:
                #give more capacity to the emptier piles to finish them sooner
                egress = (1-truck_reduction[mine]/100)*float(location_DB[mine]['Cur Mine Production Capacity (kt/d)'])*(1-stockpile_DB[pile_id]['Quantity (kt)']/total_ig0_quantity)/(len(ig0piles2process)-1)
            ingress = 0     #we just use it up till it's gone
            excess = calc_new_level(ingress, egress, pile_id)
        #check for low ig0 levels
        if len(ig0piles2process) == 0:
            # trigger new extraction project at this mine
            extraction_project.append([mine, extraction_project_lead_time])
        elif 100*sum([stockpile_DB[x]['Quantity (kt)'] for x in ig0piles2process])/sum([float(stockpile_DB[x]['Max Capacity (kt)']) for x in ig0piles2process]) < cap_limit_low_ig0 and len([x for x in extraction_project if x[0] == mine]) == 0:
            #trigger new extraction project at this mine
            extraction_project.append([mine,extraction_project_lead_time])

    #now output the data
    for pile_id in stockpile_DB:
        output_DB.append([
            current_date,
            pile_id,
            stockpile_DB[pile_id]['Location_Id'],
            convert_type(stockpile_DB[pile_id]['Type'] + '_' + stockpile_DB[pile_id]['Tier']),
            round(stockpile_DB[pile_id]['Quantity (kt)'],2),
            round(100*float(stockpile_DB[pile_id]['Quantity (kt)'])/float(stockpile_DB[pile_id]['Max Capacity (kt)']),1),
            round(demand,2),
            round(demand_debt,2),
            round(stockpile_DB[pile_id]['Ingress (kt)'],2),
            round(stockpile_DB[pile_id]['Ingress Excess (kt)'],2),
            round(stockpile_DB[pile_id]['Egress (kt)'],2),
            round(stockpile_DB[pile_id]['Egress Shortfall (kt)'],2)])

    #increment date and test for exit
    current_date += utils.timedelta(days=1)
    if current_date > end_date:
        break_flag = True

    train_reduction = train_reduction_next
    truck_reduction = truck_reduction_next

# append data to csv
utils.write_csv(output_DB, root_dir + 'Stockpile Data/Output.csv', True)

return e_msg


################################################
################################################
################################################
```

Orders Simulator

```
import Modules.utility_functions as utils

# trains from mines to port can carry aroudn 25K tonnes => normally distribute or something but cap it
# trucks, we know, get stats from your truck analysis on ave daily Tput
# deposit size ave and distribution
# stockpile capacities
# ship Tput

#Stockpile_Id,Stockpile_Name,Location_Id,Max Capacity (kt),Type,Tier,Lead Time (mths)
#Location_Id,Location_Name,Location_Code,Poor Road pct,Ave Train Capacity (kt/d),Max Train Capacity (kt/d),Ave Shipping Capacity (kt/d),Max Shipping Capacity (kt/d)

def run_orders_simulation():
    def d2s(date):
        return utils.datetime.strftime(date, '%Y-%m-%d')

    def s2d(date_s):
        return utils.datetime.strptime(date_s, '%Y-%m-%d')

    def get_nearest_date(dates, target):
        #dates = [s2d(x) for x in dates]
        return min(dates, key=lambda x: abs(s2d(x) - target))

    #--------------------------------------------------------------------------
    #--------------------------------------------------------------------------



    # load the csvs
    root_dir = 'D:/A.Nathan/1a.UWA 2018/CITS5504 - Data Warehousing/Assignment/Data/'
    order_seed = utils.load_csv_file_to_db(root_dir + 'Order Data/Order Start.csv')
    customers_db = utils.load_csv_file_to_db(root_dir + 'Order Data/DimCustomer.csv')
    countries_db = utils.load_csv_file_to_db(root_dir + 'Order Data/DimCountry.csv')
    port_db = utils.load_csv_file_to_db(root_dir + 'Order Data/DimPort.csv')
    transporter_db = utils.load_csv_file_to_db(root_dir + 'Order Data/DimTransporter.csv')
    deal_db = utils.load_csv_file_to_db(root_dir + 'Order Data/DimDeal.csv')
    ore_price_db = utils.load_csv_file_to_db(root_dir + 'Order Data/IronOre Price.csv')
    audusd_db = utils.load_csv_file_to_db(root_dir + 'Order Data/AUDUSD.csv')
    bridge_db = utils.load_csv_file_to_bridge_db(root_dir + 'Order Data/Bridge-Customer-Port.csv')

    # write headers to csv
    output_db = [[
            'OrderDate_Id',
            'Target_ShipmentDate_Id',
            'Actual_ShipmentDate_Id',
            'Target_PaymentDate_Id',
            'Actual_PaymentDate_Id',
            'Order_Id',
            'Shipment_Id',
            'Customer_Id',
            'Deal_Id',
            'Transporter_Id',
            'Destination_Port_Id',
            'Currency_Id',
            'Late Shipment Penalty (pct)',
            'Late Payment Penalty (pct)',
            'Price(mill AUD)',
            'Price(mill USD)',
            'Actual Payement (mill AUD)',
            'Actual Payement (mill USD)',
            'Balance (mill AUD)',
            'Balance (mill USD)',
            'Quantity(kt)'
            ]]
    utils.write_csv(output_db, root_dir + 'Order Data/Output.csv', False)



    output_db = []
    #start_date = utils.datetime.strptime('01-01-2012', '%d-%m-%Y')
    #end_date = utils.datetime.strptime('6-04-2018', '%d-%m-%Y')
    current_date = utils.datetime.now()

    #go through each order
    temp=1
    for k, v in order_seed.items():
```

```
        #if temp > 10:
        #    break
        #print('{0}: {1}'.format(k,v))
        #go through each shipment - the FT grain
        num_shipments = int(v['shipment number'])
        shipment_size = float(v['shipment quantity (kt)'])
        shipment_freq = 30 if v['shipment freq'] == 'M' else 90
        order_date_d = s2d(v['OrderDate_Id'])
        customer_id = v['Customer_Id']
        deal_id = v['Deal_Id']
        transporter_id = v['Transporter_Id']
        order_id = k
        leadtime_order_days = round(utils.np.random.uniform(90,365),0)
        for shipment_id in range(0,num_shipments-1):
            #shipping
            target_ship_date_d = order_date_d + utils.timedelta(days=leadtime_order_days) + utils.timedelta(days=shipment_freq*shipment_id)
            actual_ship_date_lag_mean = 9 - utils.math.log(float(utils.datetime.strftime(target_ship_date_d,'%Y'))-1999,1.5)
            actual_ship_date_d = target_ship_date_d + utils.timedelta(days=round(utils.np.random.normal(actual_ship_date_lag_mean,
actual_ship_date_lag_mean/2),0))
            late_ship_penalty = actual_ship_date_d-target_ship_date_d
            late_ship_penalty = min(100,max(0,int(late_ship_penalty.days)/7)*int(deal_db[deal_id]['Late Shipment Penalty']))
            if current_date < actual_ship_date_d:
                actual_ship_date_d = None
                late_ship_penalty = 0

            #get the port
            port_id = utils.np.random.choice([x[1] for x in bridge_db if x[0] == customer_id],size=None,replace=False)

            #payment
            target_pay_date_d = target_ship_date_d + utils.timedelta(days=14)
            actual_pay_date_lag_mean = 11 - utils.math.log(float(utils.datetime.strftime(target_pay_date_d,'%Y'))-1999,2)
            actual_pay_date_d = target_pay_date_d +
utils.timedelta(days=round(utils.np.random.normal(actual_pay_date_lag_mean,actual_pay_date_lag_mean/2),0))
            late_pay_penalty = actual_pay_date_d-target_pay_date_d
            late_pay_penalty = min(100,max(0,int(late_pay_penalty.days)/7)*int(deal_db[deal_id]['Late Payment Penalty']))
            if current_date < actual_pay_date_d:
                actual_pay_date_d = None
                late_pay_penalty = 0

            #determine pricing
            if actual_ship_date_d is not None:
                exch_rate = float(audusd_db[get_nearest_date(audusd_db.keys(), actual_ship_date_d)]['AUDUSD'])
                price_usd = float(ore_price_db[get_nearest_date(ore_price_db.keys(),actual_ship_date_d)]['Value'])*shipment_size*1000/1e6*(1-
float(deal_db[deal_id]['Discount'])/100-late_ship_penalty/100+(late_pay_penalty if late_pay_penalty is not None else 0)/100)
                price_aud = price_usd/exch_rate
            else:
                price_usd = 'TBD'
                price_aud = 'TBD'

            actual_payment_usd = 'TBD'
            actual_payment_aud = 'TBD'
            balance_usd = 'TBD'
            balance_aud = 'TBD'
            if actual_pay_date_d is not None:
                actual_payment_usd = price_usd -
round(abs(utils.np.random.normal(0,float(countries_db[customers_db[customer_id]['Country_Id']]['Dodgyness'])**2*price_usd/100)),2)
                actual_payment_aud = actual_payment_usd/exch_rate
                balance_usd = actual_payment_usd - price_usd
                balance_aud = balance_usd/exch_rate

            output_db = [[
                d2s(order_date_d),
                d2s(target_ship_date_d),
                'NULL' if actual_ship_date_d is None else d2s(actual_ship_date_d),
                d2s(target_pay_date_d),
                'NULL' if actual_pay_date_d is None else d2s(actual_pay_date_d),
                order_id,
                shipment_id+1,
                customer_id,
                deal_id,
                transporter_id,
                port_id,
                1,
                late_ship_penalty,
                late_pay_penalty,
                price_aud,
                price_usd,
                actual_payment_aud,
```

```
            actual_payment_usd,
            balance_aud,
            balance_usd,
            shipment_size
            ]]
        utils.write_csv(output_db, root_dir + 'Order Data/Output.csv', True)
    temp += 1
#-------------------------------------------------------------------------
```