Discover necessity for Greenhouse Emission Test on Vehicle in New Zealand, based on datamining in last 15 years Fleets and Emission data

Tian Bai
tbai915
461962597
INFOSYS 722
23 Oct 2020

Table of Content

1 Business understanding

1.1 Business understanding

GitHub Link: https://github.com/tbai915/722BDAS

New Zealand as a member in the United Nations responding to the 17 sustainable development goals to provide future generations an eligible developing circumstance. The goal 13 mentioned about Climate Crisis which New Zealand arrange many different government agencies working form different ways to improve it. Nowadays, human is facing the climate crisis and the World meteorological organization confirmed “The year 2019 was the second warmest year on record after 2016, according to the World Meteorological Organization’s consolidated analysis of leading international datasets” (WMO confirms 2019 as second hottest year on record, para. 1).

The major department in this GOAL is Ministry for the Environment (MFE) in New Zealand. MFE mentioned the main pathway for improve climate crisis is to control the Greenhouse gas which made by Carbon Dioxide (CO2), Methane and so on. On one hand, we are able to reduce the existing CO2 to increase the area of plants in New Zealand. On the other hand, is to control the CO2 emission. Biofuels still place a key role in human transport, people are able to control the exhaust emission for reduce the CO2.

The Ministry of Transport apply stricter emission standard on the new imported fleets and also encourage people or commercial to select electric or hybrid fleets. However, there are still have a significant number of used vehicles in New Zealand. The question is, does the used vehicles can maintain the emission standard? There are no other tests will be introduced to the Greenhouse emission test comparing with other countries such as Germany and China. The current smoky exhaust test contained in warrant of fitness (Wof), based on the colour of smoke, and it will not test on Greenhouse Emission.

1.1.1 Study of Business

Hence, a study of this situation is about discovering the pattern emission distribution between different categories of fleets on the last 15 years data in New Zealand. The purpose is The Ministry of Transport currently could estimate the totally fleets emission annual, however, it will be hard to statistic emission information to each single vehicle and they cannot estimate the partitions for different age range of vehicles.

1.1.2 Business Success Goal

Therefore, the success goals of the study will be introduced with the following:

1. The models could accurate estimate the predict emission values against real values.

2. The models are able to predict partial emission value by input partial group statistics within one fleet group.

Suppose one fleet group able to category by different age ranges, input all the statistics from the whole group will get the totally emission predict value. If input each category statistics will return partial emission predict value. Under accurate models, blocking other fleets and remain a specific type of fleet inputs in one year, able to get a predict value of single type against totally predict value to estimate the emission partition of this type of fleet.

1.2 Access the Situation

For divide fleets emission data by age groups which require the data contains number of fleets, average age of fleets and corresponding CO2 emission. Since New Zealand have a fully vehicle registration system, these data are findable from The Ministry of Transport.

1.2.1 Resource inventory

The initial data collect from Ministry of Transport at page (https://www.transport.govt.nz/mot-resources/vehicle-fleet-statistics/).

iteration 2: SPSS

iteration 3: Python, PyCharm, Sci-kit learn, Pandas, xrnd, yellowbrick, matplotlib, numpy

iteration4: Python, Jupyter, import_ipynb, PySpark, Sci-kit learn, Pandas, xrnd, seaborn, matplotlib, numpy, Github, AWS EC2

1.2.2 Requirements

The requirements are the project participants needs to have some statistical base and in good knowledge of the above-mentioned software and language.

1.2.3 Assumptions

The assumption of this project is:

- As the Euro 5 Emission Standard published in 2009 and Euro 6 Emission published 5 years later, there will be over 50% fleets with age more than 10 years and this group will contribute over 70% CO2 emission of totally fleets.

1.2.4 Constraints

The constraints for this project are:

- These data should be corresponding and continues.
- For instance, the emission data should be corresponding with the data of fleets during a same time period.

1.2.5 Risks

Data risks:

- If some type of vehicles cannot split by age, in this case, if the majority group can split by age range then could dismiss the minor cannot groups.

ISAS risks:

- SPSS may not be accurate enough, and SPSS might not have many model options to choose from and more customize as the Sci-kit Learn library in Python does.

OSAS risks:

- Python visualization uses matplotlib, its common visualization is to generate the result as figures, therefore if the result needs to be demonstrated the whole program will need to be paused while demonstrating the output figure.

BSAS risks:

- As the use of git, this make the project go opensource, there are chances that it will occur a number of opensource merges, this require time and efforts to manage the merges or unwanted merges happens.

1.2.6 Contingencies

Since, in New Zealand the majority type of traveling resources are light fleets, the other less common fleets types, for example, heavy fleets might not able to find the age distribution as its minor amounts. We are using the majority sample of population to estimate the total population, this method is acceptable in Statistic and it is commonly used for population census

1.2.7 Benefit Analysis

Opensource enables other participants to join the project and this gives a better iteration of the program.

1.3 Datamining objectives

- The model built would able to estimate the CO2 emission from the number of flees and its average ages inputs. The estimated CO2 emission data is combined for multiple fleets type groups with over 80% accuracy.
- Assume the data would inputs many different types of fleets, the success model would still able to estimate parts of the CO2 emission with partial types of inputs with over 80% accuracy.
- In future, if Ministry of Transport considers CO2 emission is important to be considered and collecting more data in detail, there are more chance to obtaining a more accurate model.

1.3.1 Datamining Success criteria

- The first success criterion is when input values fleets to models reflect predict values of CO2 emission and it will have a lower residual with real values of CO2 emission thus to count the accuracy, $Real\ Value - Predict\ Value$.
- The second success criterion is the sum of the predict values within different groups should also not have a large residual compare with the predict values, $real\ value / \sum single\ group\ predict\ value$.

1.3.1.1 Methods for model assessment

- Running the model. Assume after input the features such as numbers of fleets and age of the fleets to the model, we will obtain an estimate CO2 emission. If the input features are all real data from the same year, the obtained estimate CO2 emission will be very close to the true CO2 emission from that year. If all the input features are 0, then then CO2 emission would be 0, as if there are no fleets in New Zealand, the total CO2 emission from fleets would be none.
- To evaluate the model's accuracy. Assume there are 3 features needs to be inputted to the model to estimates CO2 emission, by running the model 3 times with input each feature's value once while inputs the rest 2 features as 0.
  for example:
  - the first run: feature1's real value, 0, 0
  - the second run: 0, feature2's real value,0
  - the third run: 0,0, feature3's real value

  Each run will obtain an estimate CO2 emission, the total sum of these 3 estimated CO2 emission will be very close to the CO2 emission estimates obtained by input all 3 features' value to the modal. By this method, we are able to evaluate the accuracy of the model by comparing the total CO2 emission sum of each run with the estimates of CO2 emission gathered by input all 3 features.

1.3.1.2 Benchmark for evaluating success

- $small\ residuals$: $Real\ Value - Predict\ Value$
- $real\ value / \sum single\ group\ predict\ value$
- We are able to evaluate the feature's distribution on the CO2 mission with the same method mentioned above to evaluate the model's accuracy.
  $Co2\ Emission\ obtained\ each\ run / Co2\ Emission\ obtained\ by\ inputs\ all\ 3\ features$

1.3.1.3 Subjective measurements and arbiter of success

In this study, the measurement of CO2 emission is predicable to the successful model.

1.3.1.4 Successful deployment of model results as a part of datamining success

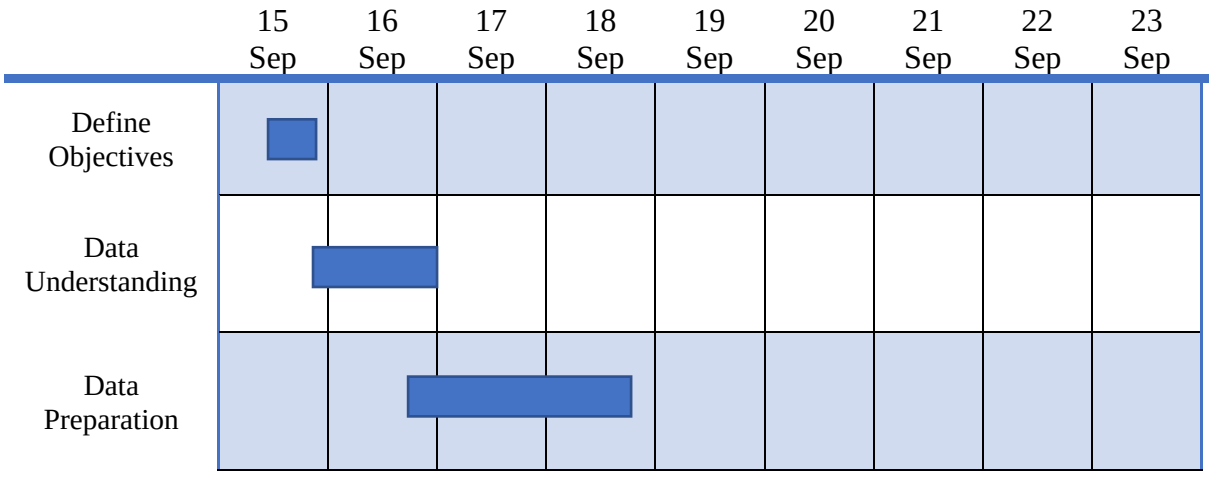
The reason is the input values might contain different fleets groups values, when input each group's values separately from total fleets group in the models, the predict value output of

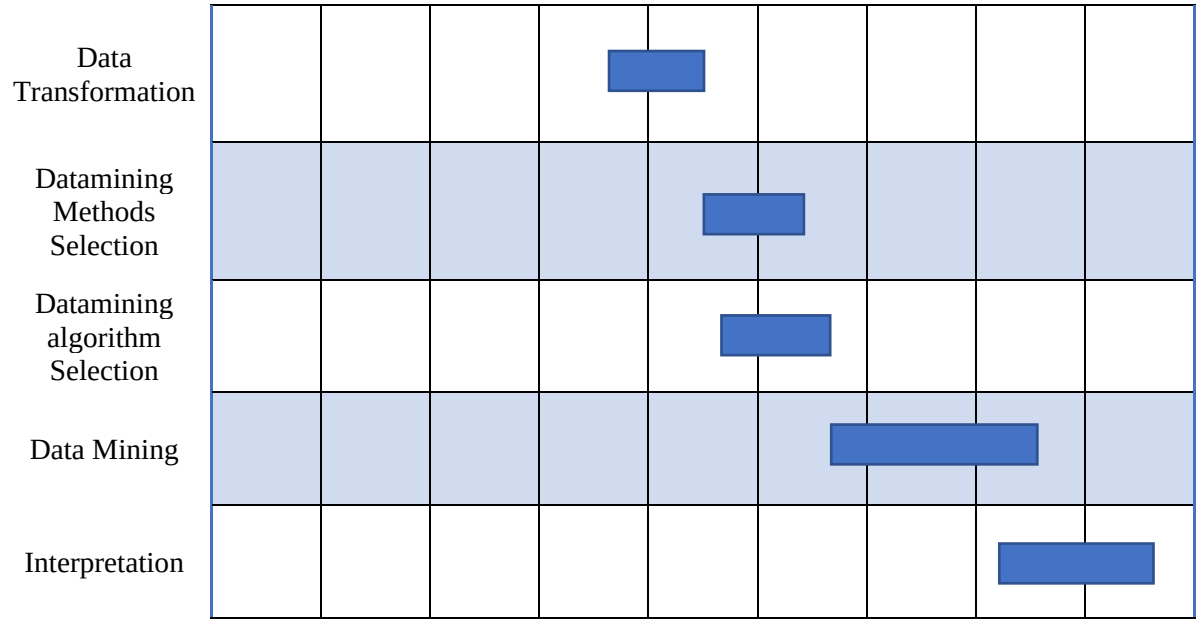
that one group against the total groups predict value will reflect the CO2 contribution for this group, $single\ group\ predict\ value / total\ groups\ predict\ value$.

1.4 Project Plan

Phase	Time	Risks
Iteration1 Proposal	24th July – 7th Aug (2 weeks)	• Unable to find suitable data
Iteration2 ISAS	7th Aug – 28th Aug (3 weeks)	• SPSS may not be accurate enough • SPSS might not have many model options to choose from and more customize as the Sci-kit Learn library in Python does.
Iteration3 OSAS	28th Aug – 2th Oct (6 weeks)	• When the result needs to be demonstrated the whole program will need to be paused while demonstrating the output figure
Iteration4 BDAS	9th Oct – 23th Oct (2 weeks)	• Require time and efforts to manage the merges or unwanted merges happens
Research paper	23th Oct – 30th Oct (1 week)	

1.4.1 Gantt Chart





2 Data Understanding

2.1 Data Collection

The initial data collect from Ministry of Transport at page (https://www.transport.govt.nz/mot-resources/vehicle-fleet-statistics/). The data of NZ-Vehicle-Fleet-Statistics-2018_web.xlsx is a combination of different data tables in an excel file**. It is issued by Ministry of Transport**, and it provides all the data the project needed, they are number of different fleets, age distribution between different fleets and CO2 emission for different fleets. There are some other data tables might use for mining in further insights, for example, average travel milage for different fleets.

A snapshot from this webpage: Download "2018 New Zealand Vehicle Fleet Annual Spreadsheet [XLSX, 1.2 MB] ". (Figure 1)

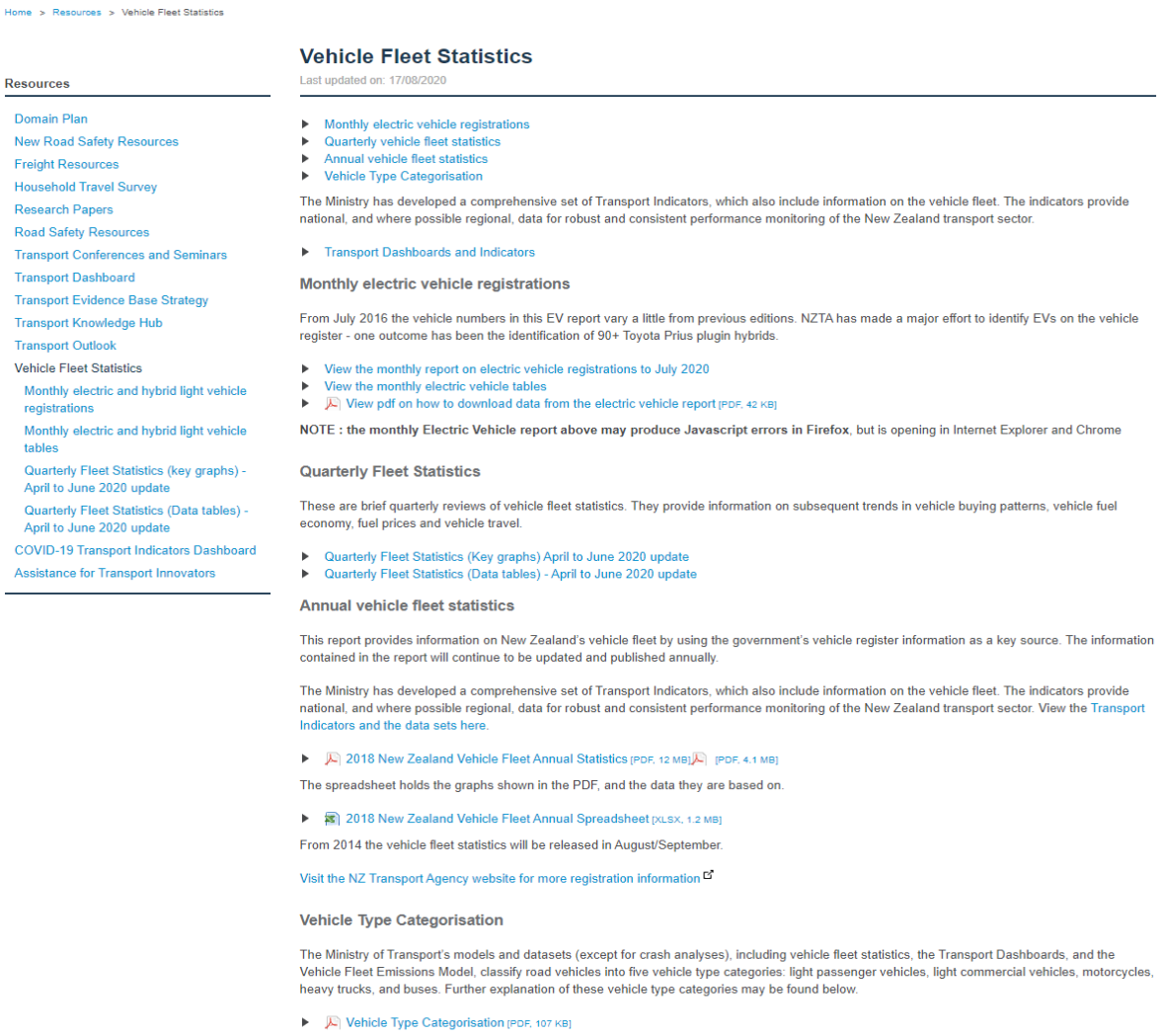


Figure 1: Screenshot for NZTA page.

2.2 Data Description

The NZ-Vehicle-Fleet-Statistics-2018_web.xlsx is a combination of different data tables in a excel file, illustrated in figure 2. All the feature values should be either integer or float, although there exist some features in percentage type that coding schema will convert it into float numbers, we will describe types within different data visualization in next step 2.3.

For the data that project need:

- Number of Fleets:
    - "Composition of Fleet" data at content in "figure 2.10 2.20 2.30 2.40" section of the Figure 2
    - This dataframe contains 34 columns and 19 rows as shown in red rectangle in figure 2.
    - All the data in numeric type, include integer, and float, more details in step 2.3
- Age Distribution of Fleets:

- Light Vehicles age distribution data at “figure 2.10”
- The age distribution of light fleets data table has 7 rows and 39 columns, shwon in figure 2
- All the data in numeric type, include integer, and float, more details in step 2.3

- CO2 Emission Data:
  - CO2 emissions data at “figure 1.10”
  - This file contains 17 rows and 5 columns, illustrated in figure 2.
  - All the data in float type, more details in step 2.3

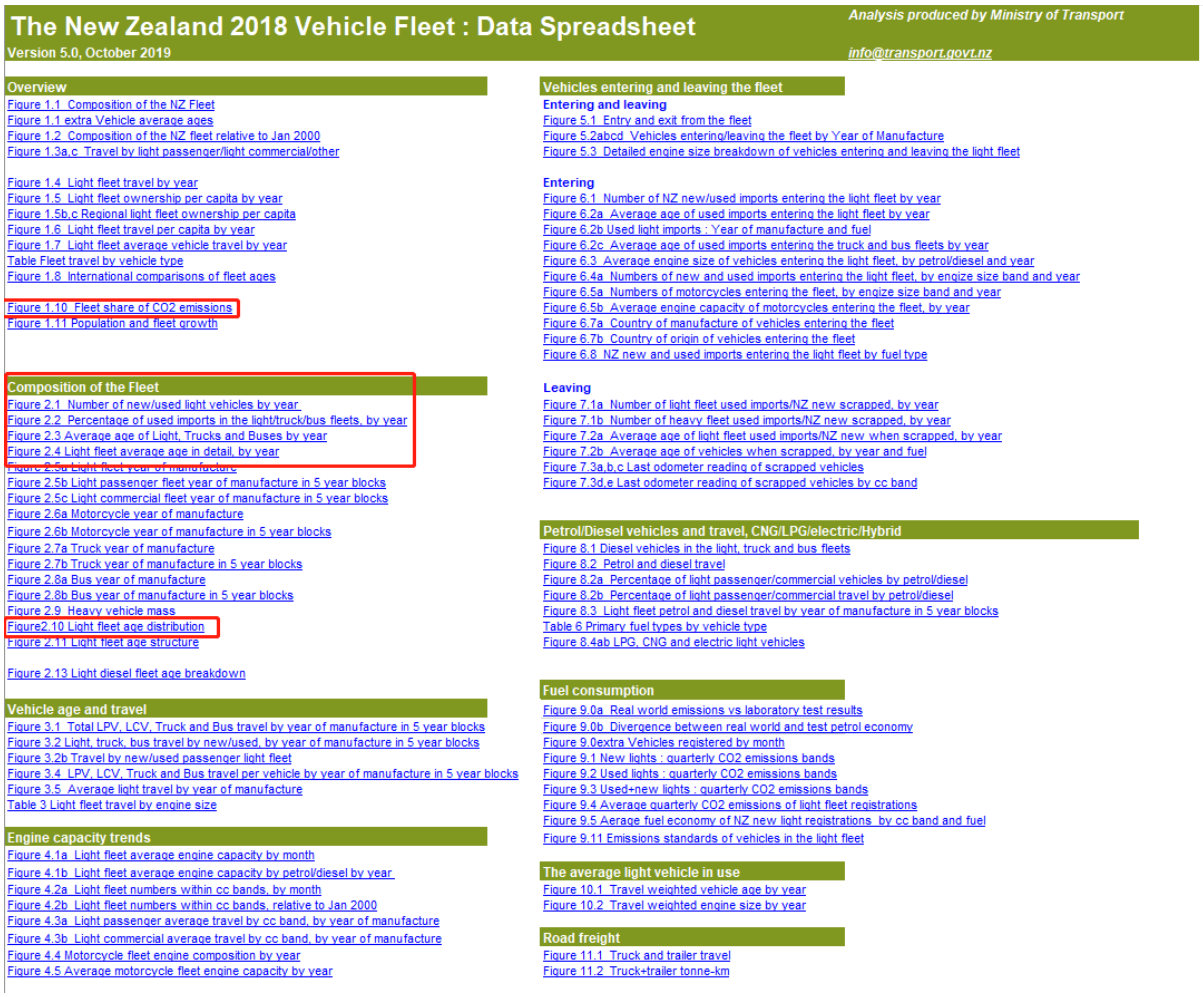


Figure 2: Screenshot for NZ-Vehicle-Fleet-Statistics-2018_web.xlsx contents page.

Risk appear:

After draft data description then we found that there are no data for the motorcycle age distribution and heavy age distribution, mentioned in risk we will further discuss details of data tables in section 2.3

2.3 Data Exploration

Important: In this project, we will combine using pandas dataframe with PySpark dataframe where pandas provided some help methods that PySpark's help method hardly to use or not have. PySpark doesn't have appropriate method for handling excel file which is our main file, and some other problems such as data visualization and data transformation in the next step 2.3 to 3. Expect these problems we will focus on PySpark, these two dataframes could convert to each other, example at figure 3.

```
In [ ]: # emxple of original pandas dfs
pd_number_fleets_df = load_average_vehicle(file_dir)
pd_light_fleet_age_df = load_light_fleet_age(file_dir)
pd_co2_emission_df = load_co2_emission(file_dir)

# example of convert to spark dfs
spark_number_fleets_df = spark.createDataFrame(number_fleets_df)
spark_light_fleet_age_df = spark.createDataFrame(number_fleets_df)
spark_co2_emission_df = spark.createDataFrame(number_fleets_df)
```

Figure 3:example of convert pandas to PySpark.

The reason that we did not using spark dataframe for step 2.3 is data visualization problem it will encounter indent problem as shown in figure 4, because we have too many columns to show. Pandas dataframe will handle this problem easily compare with figure 5.

```
In [5]: sparkDf = spark.createDataFrame(number_fleets_df)

        sparkDf.show()
```

```
+------+--------------+-----------------------+-------------+-------------+-------------+-------------+-----------+-----------+
|Period|Total light new|Total light used import|Total LPV new| Total LPV used|Total LCV new| Total LCV used|Total MC new|Total MC used|Total truck new|Total truck used|Total bus new|Total bus used|Light passenger NZ new|Light passenger used import|Light commercial NZ New|Light commercial used import|Motorcycle NZ New|Motorcycle Used Import|Truck NZ New|Truck Used Import|  Bus NZ New|Bus Used Import|       Light used %|       Truck used %|         Bus used %|Light fleet average age|Light passenger average age|Light commercial average age|Motorcycle average age|Truck fleet average age|Bus fleet average age|Light used average age|NZ new light average age|
+------+--------------+-----------------------+-------------+-------------+-------------+-------------+-----------+-----------+
| 2000|        1527641|                 966687|      1276498|       870680|       251143|        96007|      57728|       20310|          70872|           24593|         3461|          1120|          12.271011392|              10.897048284|           12.380349044|              12.465950399|      16.284931056|          15.417331364|15.065921662|     12.191741552|16.563854377|       14.4125| 0.3875540827028362|0.25761273765254283|  0.244488103034272|      11.809922351708526|         11.713869786193012|          12.402402756485915|      16.0591314488276|     14.325496255356414|   16.037873826412792|    11.052864577541555|      12.288986417504184|
+------+--------------+-----------------------+-------------+-------------+-------------+-------------+-----------+-----------+
```

Figure 4: example of PySpark visualization problem.

- Number of Fleets:

The first column is named period for indicate the year of the data of following columns, which show as red rectangle in Figure 5.

The Remain 33 columns are number of fleets, average age and percentage of number fleets which generate from number of fleets. There are mainly grouped by light, motorcycle, truck and bus 4 groups, for each group have 2 categories to imported and used imported. The light

fleets group has to two subgroups: passenger and commercial. Columns are either int64 or float 64 type, more details illustrate in figure 4.

19 rows are representing the data of different fleets with in different year from 2000 to 2018.

There are four different range or type of data in this data table:
The period data column is in range [2000, 2018] as integers in a red rectangle below.
The number of fleets columns are in large range integers, an example column is in a yellow rectangle.
The average ages of fleets columns are in a small range of floats, an example column is in a blue rectangle.
The percentage of fleets columns are in range [0,1] as floats, an example column is in a green rectangle.

```
In [14]: # ---- step 2.1 2.2 ----
         number_fleets_df = load_average_vehicle(file_dir)
         # ---- step 2.3 ----
         number_fleets_df
```

Out[14]:

	Period	Total light new	Total light used import	Total LPV new	Total LPV used	Total LCV new	Total LCV used	Total MC new	Total MC used	Total truck new	...	Truck used %	Bus used %	Light fleet average age	Light passeng average ag
0	2000	1527641	966687	1276498	870680	251143	96007	57728	20310	70872	...	0.257612737653	0.244488103034	11.809922351709	11.7138697861
1	2001	1510448	1052505	1256293	956915	254155	95590	58123	20440	70880	...	0.274290979830	0.269293589220	11.962536964427	11.8636262387
2	2002	1504464	1142837	1245805	1046144	258659	96693	59066	21015	71441	...	0.295766178718	0.313530734633	12.053391548336	11.9575762808
3	2003	1510494	1248263	1245233	1149489	265261	98774	61057	21829	72482	...	0.319947834082	0.339596381350	12.099701423831	12.0111378276
4	2004	1523241	1343089	1249158	1241364	274083	101725	64896	22867	74130	...	0.346509516295	0.354958145525	12.164872851407	12.0940873038
5	2005	1541884	1424588	1257339	1320880	284545	103708	72200	24471	75901	...	0.364653786915	0.369892146438	12.258855300159	12.2128254039
6	2006	1561044	1468085	1267476	1363917	293568	104168	80661	26640	76775	...	0.380547038890	0.387040534651	12.425075656933	12.4018728862
7	2007	1584733	1503359	1280762	1398318	303971	105041	89861	29146	77869	...	0.393302635782	0.413788432886	12.586453868459	12.5916794196
8	2008	1605137	1502981	1292558	1400142	312579	102839	100697	32149	79089	...	0.395024898456	0.426217038540	12.813787636121	12.8441783339
9	2009	1610696	1488717	1294924	1389668	315772	99049	104453	33238	78627	...	0.394552846781	0.416061925496	13.194887870688	13.2332130915
10	2010	1628515	1493451	1306995	1398139	321520	95312	105875	33639	77935	...	0.393464184541	0.409976247031	13.483027842213	13.5323331488
11	2011	1643564	1473590	1315826	1382331	327738	91259	106463	33563	77833	...	0.388384227318	0.399226622920	13.742635429573	13.8126819528
12	2012	1690422	1474965	1350457	1385912	339965	89053	108735	34070	78593	...	0.382446077083	0.394208583295	13.959251901830	14.0474095415
13	2013	1748586	1494485	1389608	1404935	358978	89550	111958	34912	80661	...	0.375790312720	0.387371249440	14.075851870341	14.1941625163
14	2014	1817743	1541012	1434731	1449199	383012	91813	115798	36262	83884	...	0.368114981319	0.379788278948	14.093225466185	14.2442578358
15	2015	1888590	1593749	1479943	1499088	408647	94661	120101	37761	86894	...	0.362077319512	0.370067014147	14.082412567985	14.2633450272
16	2016	1975136	1656445	1535175	1556538	439961	99907	124223	39092	89790	...	0.356496312700	0.348862729449	14.078110194875	14.3035126480
17	2017	2066267	1723994	1590094	1617627	476173	106367	128698	40858	93716	...	0.350587974416	0.332228744556	14.035353765863	14.3115412779
18	2018	2151898	1768376	1641232	1657190	510666	111186	133706	42961	97694	...	0.343582231957	0.310044819404	14.086867014703	14.4155344583

19 rows × 34 columns

Figure 5: Screenshot for number of fleets overview in pandas data frame.

```
In [20]: # ---- step 2.3 ----
         # number_fleets_df
         # light_fleet_age_df
         # co2_emission_df

         sparkdf = spark.createDataFrame(number_fleets_df)
         sparkdf.printSchema()
```

```
root
 |-- Period: long (nullable = true)
 |-- Total light new: long (nullable = true)
 |-- Total light used import: long (nullable = true)
 |-- Total LPV new: long (nullable = true)
 |--  Total LPV used: long (nullable = true)
 |-- Total LCV new: long (nullable = true)
 |--  Total LCV used: long (nullable = true)
 |-- Total MC new: long (nullable = true)
 |-- Total MC used: long (nullable = true)
 |-- Total truck new: long (nullable = true)
 |-- Total truck used: long (nullable = true)
 |-- Total bus new: long (nullable = true)
 |-- Total bus used: long (nullable = true)
 |-- Light passenger NZ new: double (nullable = true)
 |-- Light passenger used import: double (nullable = true)
 |-- Light commercial NZ New: double (nullable = true)
 |-- Light commercial used import: double (nullable = true)
 |-- Motorcycle NZ New: double (nullable = true)
 |-- Motorcycle Used Import: double (nullable = true)
 |-- Truck NZ New: double (nullable = true)
 |-- Truck Used Import: double (nullable = true)
 |-- Bus NZ New: double (nullable = true)
 |-- Bus Used Import: double (nullable = true)
 |-- Light used %: double (nullable = true)
 |-- Truck used %: double (nullable = true)
 |-- Bus used %: double (nullable = true)
 |-- Light fleet average age: double (nullable = true)
 |-- Light passenger average age: double (nullable = true)
 |-- Light commercial average age: double (nullable = true)
 |-- Motorcycle average age: double (nullable = true)
 |-- Truck fleet average age: double (nullable = true)
 |-- Bus fleet average age: double (nullable = true)
 |-- Light used average age: double (nullable = true)
 |-- NZ new light average age: double (nullable = true)
```

Figure 6: Temporary types of number of fleets overview in spark data frame.

- Age Distribution of Fleets:

It is order by different age groups in rows and the columns are the years. This data table storge the number of fleets in different age groups and the corresponding percentage of them from 2000 to 2018, thus there are 38 columns for then and first columns for the age groups information.

- The Age groups information are in string type as a red rectangle below for show the information for different groups.

- The Yellow rectangle is an example for the fleets of age groups in float type but they are integers.
- The Blue rectangle is an example for the percentages of fleets of corresponding age groups in float type between [0,1]

The percentage of fleets columns have a different start entry, 2000, …, 2018 comparing 2000.1, …, 2018.1. This is cause by when pandas load columns with same entry will rename it. There are some **missing values** in percentage of fleets for the total groups in 5th row, this is because the total percentage is 1 which is a trivial group in percentage columns. Instead, it using 15+ years group for the total recorded in 6th row, this is also explained why there also have some missing values in 6th row.

Under visualization we know that data are all float type expect nan values, we will impute for **missing values** in step 2.4, other problems will be mentioned in step 3.

```
In [17]: # ---- step 2.1 2.2 ----
         light_fleet_age_df = load_light_fleet_age(file_dir)

         # ---- step 2.3 ----
         light_fleet_age_df
```

Out[17]:

	Age	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2009.1	2010.1	201
0	0-4 years	375680.0	361278.0	371546.0	397164.0	422522.0	451885.0	475895.0	498470.0	503736.0	...	0.152160944232	0.143767925426	0.136656882(
1	5-9 years	720525.0	737685.0	730168.0	760055.0	761918.0	756408.0	661033.0	609594.0	592814.0	...	0.191736231004	0.202058513204	0.208288981(
2	10-14 years	785368.0	829982.0	887866.0	907729.0	926784.0	938797.0	1023210.0	1052695.0	1050524.0	...	0.321271571451	0.296735834709	0.259703575(
3	15-19 years	393886.0	404923.0	419316.0	446613.0	488853.0	536089.0	572331.0	616195.0	627768.0	...	0.213296470392	0.220129200804	0.246440177(
4	20+ years	218864.0	229080.0	238400.0	247192.0	266249.0	283289.0	296656.0	311134.0	333272.0	...	0.121534782922	0.137308525856	0.148910382(
5	Total	2494323.0	2562948.0	2647296.0	2758753.0	2866326.0	2966468.0	3029125.0	3088088.0	3108114.0	...	NaN	NaN	N
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.334831253313	0.357437726660	0.395350560(

7 rows × 39 columns

Figure 7: Screenshot for light fleet age overview in pandas data frame.

We cannot use PySpark get columns types, because currently the raw data using year as columns and the table need to be transpose which we will do in step3.

- CO2 Emission Data:

This file contains 17 rows and 5 columns, 4 columns for Light passenger, Light commercial, Motorcycle and Heavy fleet from 2001 to 2017, either integer type or float type.

Red rectangle shows the Year data which is similar with Period in the number of fleet data table with different column entry.

- Yellow rectangle is an example of the CO2 emission of 4 groups in float numbers, because the Million Tonnes CO2.

  There 2 rows of **missing data** founded in this file, it does not contain CO2 emission data in the year 2000 and 2018.

However, the differences with two files above are this file has no totally light fleets group and no bus group, these rows need to imputation in step 3.

```
In [21]: # ---- step 2.1 2.2 ----
co2_emission_df = load_co2_emission(file_dir)

# ---- step 2.3 ----
co2_emission_df
```

Out[21]:

	Year	Light passenger	Light commercial	Motorcycle	Heavy fleet
0	2001	6.76529055920	1.50467490711	0.02823485686	2.38874320690
1	2002	7.06637032283	1.55969757691	0.02796532254	2.57251006740
2	2003	7.37572755774	1.58502708961	0.02836421634	2.63913395708
3	2004	7.67104588050	1.58226037566	0.02939649819	2.65359933601
4	2005	7.54950688585	1.62864435086	0.03200675894	2.86099281184
5	2006	7.64481448926	1.66393447456	0.03743665722	2.91994167639
6	2007	7.79691413907	1.71895367454	0.04061596465	3.00990338266
7	2008	7.69252822020	1.76446421715	0.04533403409	3.07734898453
8	2009	7.62685643054	1.77460203601	0.04630074844	2.99064975410
9	2010	7.69716185765	1.83028339465	0.04643382175	3.10975125236
10	2011	7.57525756436	1.86172944903	0.04512402896	3.20049694840
11	2012	7.44477523066	1.87931042313	0.04446445691	3.21381765952
12	2013	7.41518700798	1.93747100946	0.04508948417	3.28880961212
13	2014	7.43075356143	1.99341682704	0.04530614932	3.34570990213
14	2015	7.66212375443	2.12184016233	0.04682491622	3.45417456437
15	2016	7.80025990339	2.23780183249	0.04741634669	3.50936613030
16	2017	8.07740664325	2.47222703164	0.04719994384	3.86107230524

Figure 8: Screenshot for CO2 emission overview in pandas data frame.

```
In [22]: # ---- step 2.3 ----
         # number_fleets_df
         # light_fleet_age_df
         # co2_emission_df

         sparkdf = spark.createDataFrame(co2_emission_df)
         sparkdf.printSchema()

         root
          |-- Year: double (nullable = true)
          |-- Light passenger: double (nullable = true)
          |-- Light commercial: double (nullable = true)
          |-- Motorcycle: double (nullable = true)
          |-- Heavy fleet: double (nullable = true)
```

Figure 9: Temporary types of CO2 emission overview in spark data frame.

2.3.1 Exploration of most important predictors (features)

Hypothesis among the used vehicles will contribute more CO2 emission compre with same age range of new vehicles.

There is a relationship between number, age and CO2 emssion in fleets.

Assumption: The age of fleets will be a significate feature, base on new vehicles will have apply harder emission standard.

The heavy vehicle and motorcycle data cannot split by age because The NZ Transport have not statistics the age distribution of them. However, they have taken 3.65% and 3.69% of totally vehicle, count by mean vehicle number/meanTotal numbe, over 18 years, so that dismiss these two types of fleets.

2.4 Data Quality

2.4.1 Missing Values Imputation

There are two missing rows can find in CO2 emission data, which will impute by mean imputation. Scikit-learn provide mean imputation.

- create two missing rows for year 2000 and year 2018 with np.nan.
- create scikit-learn mean imputation object fit for CO2 emission data and impute the missing rows.
- Adding these missing rows to the C
- CO2 emission data frame.

```
In [14]: def step_2_4_1_imputation_co2_emission_df(co2_emission_df):
             missing_values = [[2000] + [np.nan for i in range(4)], [2018] + [np.nan for i in range(4)]]

             imp = SimpleImputer(missing_values=np.nan, strategy='mean')
             imp.fit(co2_emission_df)
             X = imp.transform(missing_values)

             df = pd.DataFrame(X, columns=co2_emission_df.columns)
             co2_emission_df = pd.concat([co2_emission_df, df])
             co2_emission_df = co2_emission_df.sort_values(by=['Year'])
             co2_emission_df.index = [x for x in range(len(co2_emission_df.index))]

             return co2_emission_df
```

Figure 10: imputation information.

```
In [5]: # ---- step 2.4.1 ----
        co2_emission_df = step_2_4_1_imputation_co2_emission_df(co2_emission_df)
        print(co2_emission_df)
```

	Year	Light passenger	Light commercial	Motorcycle	Heavy fleet
0	2000.0	7.546587059314	1.830372872481	0.040206717949	3.064471855962
1	2001.0	6.765290559200	1.504674907110	0.028234856860	2.388743206900
2	2002.0	7.066370322830	1.559697576910	0.027965322540	2.572510067400
3	2003.0	7.375727557740	1.585027089610	0.028364216340	2.639133957080
4	2004.0	7.671045880500	1.582260375660	0.029396498190	2.653599336010
5	2005.0	7.549506885850	1.628644350860	0.032006758940	2.860992811840
6	2006.0	7.644814489260	1.663934474560	0.037436657220	2.919941676390
7	2007.0	7.796914139070	1.718953674540	0.040615964650	3.009903382660
8	2008.0	7.692528220200	1.764464217150	0.045334034090	3.077348984530
9	2009.0	7.626856430540	1.774602036010	0.046300748440	2.990649754100
10	2010.0	7.697161857650	1.830283394650	0.046433821750	3.109751252360
11	2011.0	7.575257564360	1.861729449030	0.045124028960	3.200496948400
12	2012.0	7.444775230660	1.879310423130	0.044464456910	3.213817659520
13	2013.0	7.415187007980	1.937471009460	0.045089484170	3.288809612120
14	2014.0	7.430753561430	1.993416827040	0.045306149320	3.345709902130
15	2015.0	7.662123754430	2.121840162330	0.046824916220	3.454174564370
16	2016.0	7.800259903390	2.237801832490	0.047416346690	3.509366130300
17	2017.0	8.077406643250	2.472227031640	0.047199943840	3.861072305240
18	2018.0	7.546587059314	1.830372872481	0.040206717949	3.064471855962

Figure 11: added information.

The reason not using spark imputation here is the imputation method both in PySpark and sklearn are similar, using spark we need to convert dataframe first and for the next step visualization we need to convert it back, thus is a waste of computation.

2.4.2 Check features normality

According to the CO2 emission data, check data normality within these files. If the fleet's groups follow the normal distribution or skew normal distribution then the average age of these groups must follow the same distribution, because they are from the same original fleets data.

Mean and standard deviation able to get from data, then build pdf for the mean and std dev to plot the figures for check the normality.

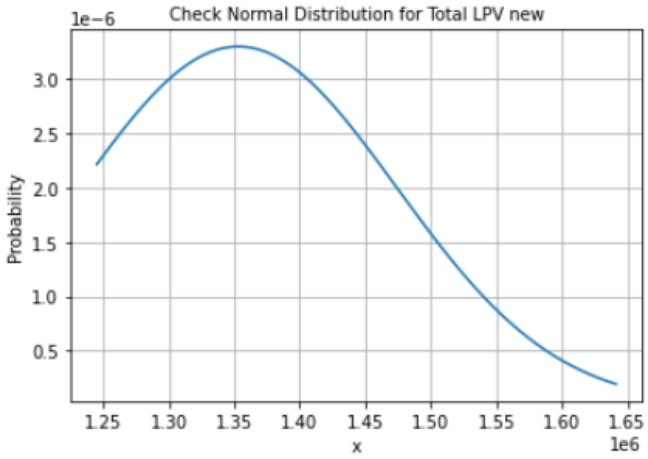
```
In [25]: def step_2_4_2_check_normality(number_fleets_df, light_fleet_age_df,co2_emission_df):
             plot_normality(number_fleets_df['Total LPV new'], 'Total LPV new')
             plot_normality(number_fleets_df[' Total LPV used'], 'Total LPV used')
             plot_normality(number_fleets_df['Total LCV new'], 'Total LCV new')
             plot_normality(number_fleets_df[' Total LCV used'], 'Total LCV used')

             plot_normality(np.array(light_fleet_age_df.iloc[0][1:20].astype(int)), '0-4 age group')
             plot_normality(np.array(light_fleet_age_df.iloc[1][1:20].astype(int)), '5-9 age group')
             plot_normality(np.array(light_fleet_age_df.iloc[2][1:20].astype(int)), '10-14 age group')
             plot_normality(np.array(light_fleet_age_df.iloc[3][1:20].astype(int)), '15-19 age group')
             plot_normality(np.array(light_fleet_age_df.iloc[4][1:20].astype(int)), '20+ age group')


             plot_normality(co2_emission_df['Light passenger'], 'Light passenger co2 emssion')
             plot_normality(co2_emission_df['Light commercial'], 'Light commercial co2 emission')
```

Figure 12: Plot normality for features.

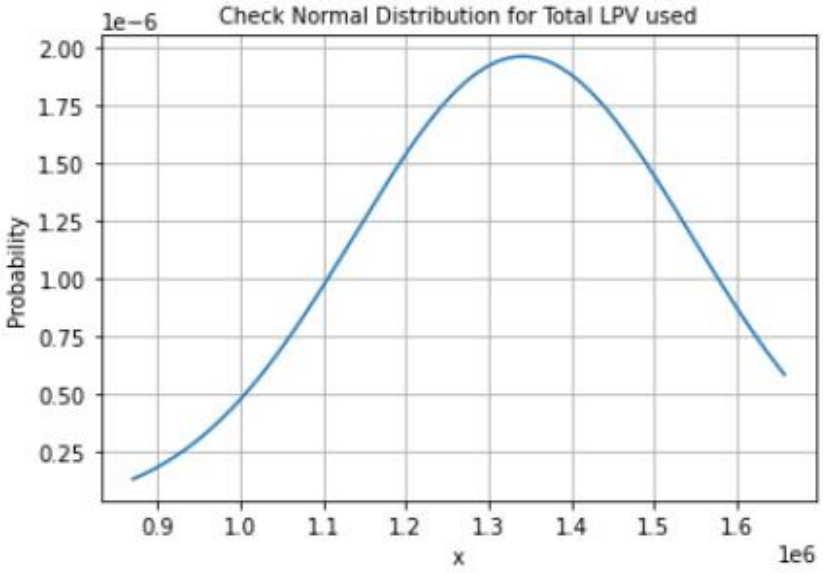
PySpark **does not** provided visualization and same for the pandas, we using matplotlib for this part.

Total LPV new:

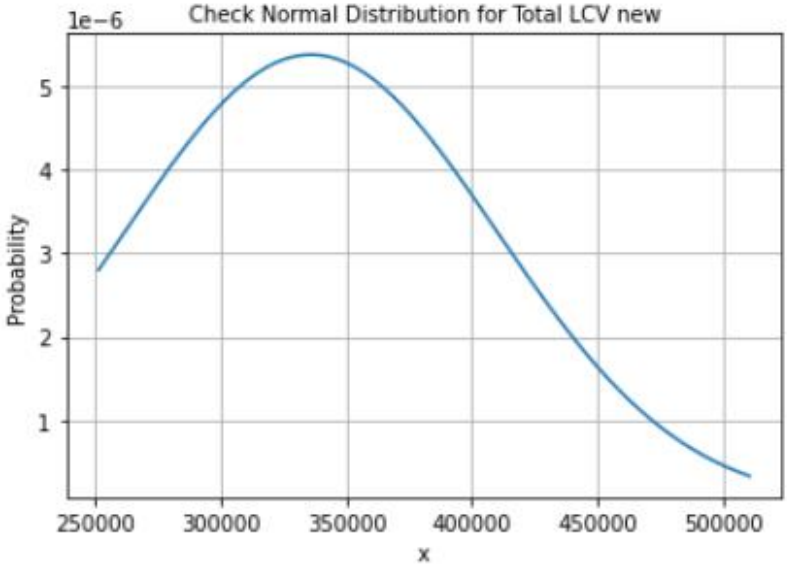
```
In [8]: # ---- step 2.4.2 ----
        step_2_4_2_check_normality(number_fleets_df, light_fleet_age_df, co2_emission_df)
```



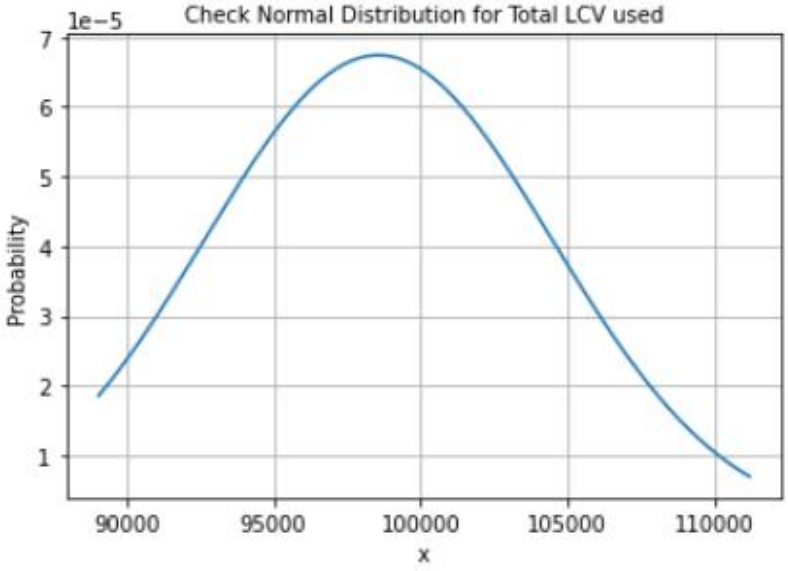
Total LPV used:



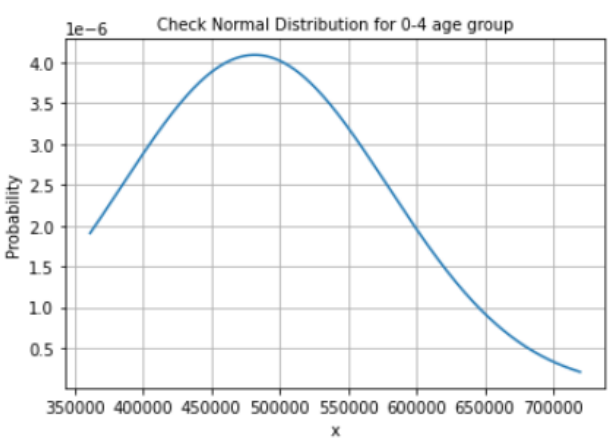
Total LCV new:



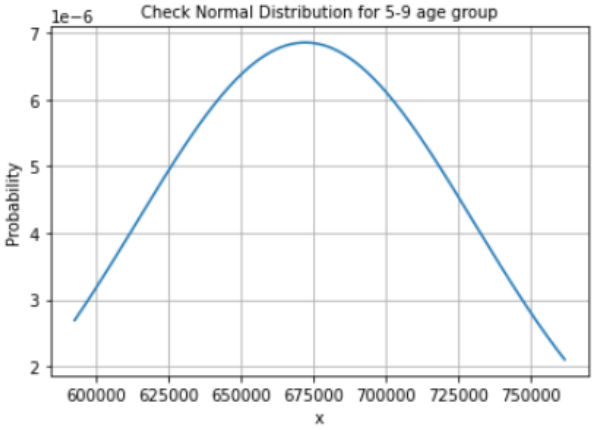
Total LCV used:



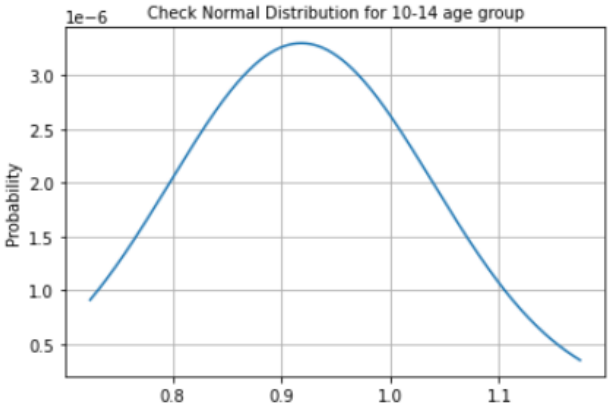
0-4 age group:



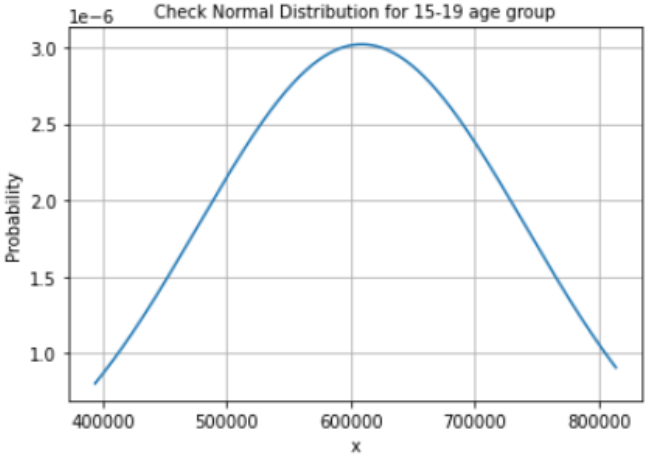
5-9 age group



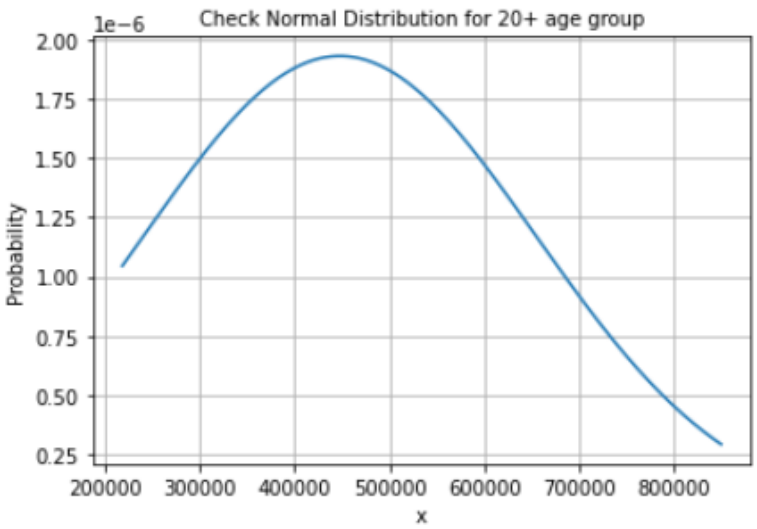
10-14 age group:



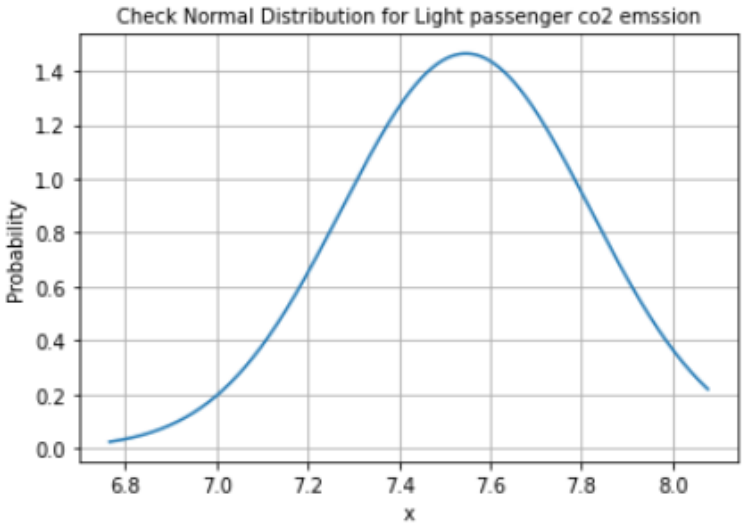
15-19 age group:



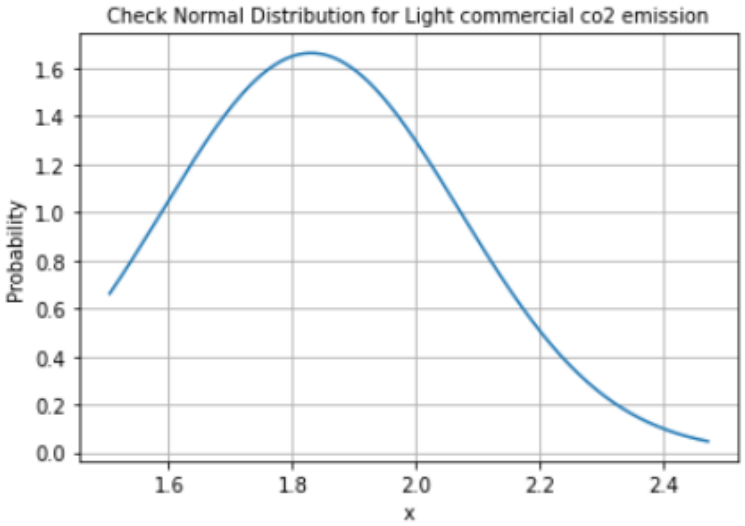
20+ age group:



Light passenger group:



Light commercial group:



All the groups are following the normal and skew normal distribution.

3 Data Preparation

We will continue combine using pandas dataframe rather than PySpark dataframe because of the data visualization indent problem same as step 2.

3.1 Data Selection

Considering the business and datamining objectives is around real emission values and model predict emission values. In the Data Selection part will start from CO2 emission data, and **there are no age distributions data for the motor cycle fleets and heavy fleets**, and we mentioned above in step 2.3.1 that we **discard these two groups data**. Hence, select all the columns related to Light Passenger group and Light commercial group.

- **Number of Fleets:**

There are 15 columns related to LPV and LCV in number of fleets data tables, marked with Figure 9 red rectangle below:

```
In [11]: # ---- step 3.1 ----
         number_fleets_df.columns

Out[11]: Index(['Period', 'Total light new', 'Total light used import', 'Total LPV new',
                ' Total LPV used', 'Total LCV new', ' Total LCV used', 'Total MC new',
                'Total MC used', 'Total truck new', 'Total truck used', 'Total bus new',
                'Total bus used', 'Light passenger NZ new',
                'Light passenger used import', 'Light commercial NZ New',
                'Light commercial used import', 'Motorcycle NZ New',
                'Motorcycle Used Import', 'Truck NZ New', 'Truck Used Import',
                'Bus NZ New', 'Bus Used Import', 'Light used %', 'Truck used %',
                'Bus used %', 'Light fleet average age', 'Light passenger average age',
                'Light commercial average age', 'Motorcycle average age',
                'Truck fleet average age', 'Bus fleet average age',
                'Light used average age', 'NZ new light average age'],
               dtype='object')
```

Figure 13: number of fleets select columns

We are able to select columns by create a columns list and applying this list on dataframe to construct a new dataframe with these columns.

```
In [9]: # ---- step 3.1 ----
num_fleets_select_cols = ['Period', 'Total light new', 'Total light used import', 'Total LPV new',
                          ' Total LPV used', 'Total LCV new', ' Total LCV used', 'Light passenger NZ new',
                          'Light passenger used import', 'Light commercial NZ New',
                          'Light commercial used import', 'Light fleet average age', 'Light passenger average age',
                          'Light commercial average age', 'Light used average age', 'NZ new light average age']

number_fleets_df[num_fleets_select_cols]
```

Figure 14: Example of construct new features of number of fleets by select columns

- **Light age distribution:**

All the light age distribution columns are needed; however, the age distribution data table need to transform for coincide with others.

- Construct data frames for numbers and percentages within different age groups
- Drop unnecessary columns which is using presentation better in Excel
- Reset index and columns appropriately.
- Needs to transpose first then divided into two groups of number of age groups and percentages of age groups.

```
In [ ]: def step_3_1_clean_light_age_distribution(light_fleet_age_df):
            light_fleet_age_df = light_fleet_age_df.T
            light_fleet_age_df.index = [x for x in range(len(light_fleet_age_df.index))]

            numbers = pd.DataFrame(light_fleet_age_df[1:20])
            numbers = numbers.drop(columns=[6])
            numbers.columns = ['0-4 years', '5-9 years', '10-14 years', '15-19 years', '20+ years', 'Total']
            numbers.index = [i for i in range(2000, 2019)]

            percentages = pd.DataFrame(light_fleet_age_df[20:])
            percentages = percentages.drop(columns=5)
            percentages.columns = ['0-4 years percentage', '5-9 years  percentage',
                                   '10-14 years percentage', '15-19 years percentage',
                                   '20+ years percentage', '15+ years percentage']
            percentages.index = [i for i in range(2000, 2019)]

            new_age_distribution = pd.concat([numbers, percentages], axis=1, join='inner')
            new_age_distribution.insert(0, 'Period', new_age_distribution.index)
            new_age_distribution.index = [i for i in range(len(new_age_distribution.index))]

            return new_age_distribution
```

Figure 15: Clean Nan values for light fleets age dataframe

New data frame has 13 columns 19 rows, 6 columns for numbers and 6 for columns from 2000 to 2018.

```
In [15]: new_age_distribution_df = step_3_1_clean_light_age_distribution(light_fleet_age_df)
         new_age_distribution_df
```

Out[15]:

	Period	0-4 years	5-9 years	10-14 years	15-19 years	20+ years	Total	0-4 years percentage	5-9 years percentage	10-14 years percentage	15-19 years percentage	20+ years percentage	15+ y percer
0	2000	375680	720525	785368	393886	218864	2494323	0.150614014304	0.288865956815	0.314862189059	0.157912988815	0.0877448510077	0.24565783
1	2001	361278	737685	829982	404923	229080	2562948	0.140961892321	0.28782675263	0.323838798134	0.157991110237	0.0893814466778	0.24737255
2	2002	371546	730168	887866	419316	238400	2647296	0.14034924693	0.275816531283	0.335385993859	0.158394074558	0.0900541533701	0.24844822
3	2003	397164	760055	907729	446613	247192	2758753	0.143965045077	0.275506723509	0.329035981112	0.161889447877	0.0896028024256	0.25149225
4	2004	422522	761918	926784	488853	266249	2866326	0.147408913013	0.26581693778	0.323335168435	0.170550384011	0.0928885967611	0.26343898
5	2005	451885	756408	938797	536089	283289	2966468	0.152330987558	0.254986064235	0.316469619763	0.180716259201	0.0954970692419	0.27621332
6	2006	475895	661033	1023210	572331	296656	3029125	0.157106425123	0.218225725251	0.337790616102	0.188942681467	0.0979345520571	0.28687723
7	2007	498470	609594	1052695	616195	311134	3088088	0.161417032157	0.197401757981	0.340888925445	0.1995393266	0.100752957817	0.30029228
8	2008	503736	592814	1050524	627768	333272	3108114	0.162071275378	0.19073109931	0.337994037542	0.201977147556	0.107226440214	0.30920358
9	2009	471609	594269	995752	661093	376686	3099409	0.152160944232	0.191736231004	0.321271571451	0.213296470392	0.121534782922	0.33483125
10	2010	448838	630819	926398	687235	428672	3121962	0.143767925426	0.202058513204	0.296735834709	0.220129200804	0.137308525856	0.3574377
11	2011	425980	649268	809535	768191	464176	3117150	0.136656882088	0.208288981923	0.259703575381	0.246440177726	0.148910382882	0.39535056
12	2012	423475	663850	752875	808744	516439	3165383	0.133783178844	0.209721856723	0.237846415426	0.255496412282	0.163152136724	0.41864854
13	2013	443043	704420	724437	813648	557519	3243067	0.136612348743	0.21720797011	0.223380213853	0.250888433696	0.171911033599	0.42279946
14	2014	502841	698847	765817	773998	617248	3358751	0.149710710916	0.208067522719	0.228006482171	0.230442209024	0.18377307517	0.41421528
15	2015	559050	665914	854452	728910	674011	3482337	0.160538741655	0.191226179431	0.245367407003	0.20931632981	0.193551342102	0.40286767
16	2016	620579	629335	961822	645677	774166	3631579	0.170884069987	0.173295142416	0.264849532393	0.177795113365	0.213176141838	0.39097125
17	2017	678208	607116	1081344	598489	825103	3790260	0.178934426662	0.160177929746	0.285295467857	0.157901832592	0.217690343143	0.37559217
18	2018	719677	601804	1174496	575220	849077	3920274	0.183578239684	0.153510698487	0.299595385425	0.146729539823	0.216586136581	0.36331567

Figure 16: New constructed light age distribution dataframe

- **CO2 Emission:**

CO2 emission data are using for target, mentioned in step2.2, thus only Light passenger and Light commercial columns are select for corresponding output:

```
In [16]: co2_select_cols = ['Light passenger', 'Light commercial']
         co2_emission_df[co2_select_cols]
```

Out[16]:

	Light passenger	Light commercial
0	7.546587059314	1.830372872481
1	6.765290559200	1.504674907110
2	7.066370322830	1.559697576910
3	7.375727557740	1.585027089610
4	7.671045880500	1.582260375660
5	7.549506885850	1.628644350860
6	7.644814489260	1.663934474560
7	7.796914139070	1.718953674540
8	7.692528220200	1.764464217150
9	7.626856430540	1.774602036010
10	7.697161857650	1.830283394650
11	7.575257564360	1.861729449030
12	7.444775230660	1.879310423130
13	7.415187007980	1.937471009460
14	7.430753561430	1.993416827040
15	7.662123754430	2.121840162330
16	7.800259903390	2.237801832490
17	8.077406643250	2.472227031640
18	7.546587059314	1.830372872481

Figure 17: new CO2 emission dataframe

Hence, one risk meet here that is we do not have enough age group data for heavy and motorcycle fleets. In step 1.2, the contingency method is using the majority groups data which are the LPV and LCV to measure the whole population.

3.2 Data Cleaning

The cleaning includes three aspects:

1. In step 2.4, we imputation the missing rows.
2. Cleaning for the null values, in step 3.1, we have cleaned the nan (not a number) values in light fleets age distribution table which we mentioned in step 2.3, because these nan values are just null values for make the excel looks more convenient.
3. Excluding the columns which are redundant columns. There are some fields are redundant for Light fleets groups. For example, the “Total Light New” field are including the number of new fleets both for Light passenger and Light commercial, thus we can clean some cross-hold information groups between Light Passenger and Light Commercial. The project needs columns direct have relationship with LPV and LCV, because they are the target output values columns.

- For number of fleets data frame, we select 5 columns and marked with yellow rectangle which distinct with red rectangle above.

```
In [17]: number_fleets_df.columns

Out[17]: Index(['Period', 'Total light new', 'Total light used import', 'Total LPV new',
                ' Total LPV used', 'Total LCV new', ' Total LCV used', 'Total MC new',
                'Total MC used', 'Total truck new', 'Total truck used', 'Total bus new',
                'Total bus used', 'Light passenger NZ new',
                'Light passenger used import', 'Light commercial NZ New',
                'Light commercial used import', 'Motorcycle NZ New',
                'Motorcycle Used Import', 'Truck NZ New', 'Truck Used Import',
                'Bus NZ New', 'Bus Used Import', 'Light used %', 'Truck used %',
                'Bus used %', 'Light fleet average age', 'Light passenger average age',
                'Light commercial average age', 'Motorcycle average age',
                'Truck fleet average age', 'Bus fleet average age',
                'Light used average age', 'NZ new light average age'],
               dtype='object')
```

Figure 18:

- For light fleets age distribution data frame, we clean the null values and next step is remaining all the percentage columns for construct new features that we will explained in 3.3.

```
In [18]: new_age_distribution_df.columns

Out[18]: Index(['Period', '0-4 years', '5-9 years', '10-14 years', '15-19 years',
                '20+ years', 'Total', '0-4 years percentage', '5-9 years  percentage',
                '10-14 years percentage', '15-19 years percentage',
                '20+ years percentage', '15+ years percentage'],
               dtype='object')
```

Figure 19:

- **For CO2 emission data frame will same as 3.1, because they are target values.**

Hence, after step 3.2, there are 12 columns for inputs and 2 columns for the target output. Storage these columns in lists then in the following steps could select these columns using these lists, by using df[columns].

```
In [ ]: # ---- step 3.2 ----
num_fleets_select_cols = ['Period', 'Total LPV new', ' Total LPV used', 'Total LCV new', ' Total LCV used',
                          'Light passenger average age', 'Light commercial average age']

percentage_columns = ['0-4 years percentage', '5-9 years  percentage','10-14 years percentage',
                      '15-19 years percentage', '20+ years percentage', '15+ years percentage']
```

Figure 209: Finally select rows

3.3 Data Construct

After we have cleaned the redundant fields, the next step for Light Passenger and Light Commercial is construct new features with “Light Fleet age distribution data”.

At first, the reason why remains the percentage of age groups in step 3.2 is, instead using percentage and the number of fleets for age groups are combine with Light Passenger and Light Commercial groups that is we cannot know how many vehicles exactly in an age range belongs to which groups (LPV or LCV). However, we can estimate the number of vehicles in different age ranges for different groups by construct new features with fields g to l in “Light Fleet age distribution data” which are the percentage for age ranges, if the Light Passenger and Light Commercial groups are following normal distribution.

Next step is applying the percentage of age groups to the number of Light Passenger and Light Commercial to get the number of age groups in LPV new, LPV used, LCV new or LCV used. There are 24 columns which generated by 6 age groups * 4 number fleets group and 1 column for the Period. i.e. Elements in yellow rectangle shown below times blue rectangle above.

```
In [ ]: # ---- step 3.3 ----
nums_columns = ['Total LPV new', ' Total LPV used', 'Total LCV new', ' Total LCV used']
new_age_distribution_df = step_3_3_construct_new_distribution_df(nums_columns, percentage_columns,
                                                                 number_fleets_df, new_age_distribution_df)
```

Figure 21: split columns

```
In [ ]: def step_3_3_construct_new_distribution_df(nums_columns, percentage_columns, number_fleets_df, new_age_distribution_df):

    # new_table = {'Period':[i for i in range(2000,2019)]} for internal output
    new_table = {}
    for num_column in nums_columns:
        for percenate_column in percentage_columns:
            new_column = new_age_distribution_df[percenate_column] * number_fleets_df[num_column]
            new_column_name = '%s of %s' % (percenate_column[:-11].strip(), num_column[6:].strip())
            new_table[new_column_name] = new_column

    new_age_distribution_df = pd.DataFrame(new_table)

    return new_age_distribution_df
```

Figure 22: method of construct new dataframe

```
In [9]: # ---- step 3.3 ----
nums_columns = ['Total LPV new', ' Total LPV used', 'Total LCV new', ' Total LCV used']
new_age_distribution_df = step_3_3_construct_new_distribution_df(nums_columns, percentage_columns,
                                                                 number_fleets_df, new_age_distribution_df)
new_age_distribution_df
```

Out[9]:

	0-4 years of LPV new	5-9 years of LPV new	10-14 years of LPV new	15-19 years of LPV new	20+ years of LPV new	15+ years of LPV new	0-4 years of LPV used	5-9 years of LPV used	10-14 years of LPV used	15-19 years of LPV used
0	192258.488031	368736.816142	401920.954609	201575.614396	112006.126822	313581.741218	131136.609974	251509.811279	274144.21077	137491.681101
1	177089.438589	361594.734542	406836.415224	198483.125853	112289.285791	310772.411644	134888.54919	275425.736993	309886.203516	151184.063253
2	174847.793571	343613.613755	417825.54808	197328.130054	112189.914539	309518.044593	146825.52258	288543.809303	350862.04516	165703.010734
3	179270.024976	343070.063835	409726.461868	201590.082849	111576.366473	313166.449321	165486.2357	316691.948099	378223.240892	186090.139551
4	184137.022961	332047.354364	403896.712332	213044.37659	116032.533753	329076.910343	182988.117893	329975.577151	401376.638029	211715.106897
5	191531.691566	320603.92302	397909.595244	227221.600628	120072.189544	347293.790171	201210.954846	336805.992527	418018.391353	238704.492454
6	199128.623289	276595.869338	428141.498935	239480.314136	124129.694303	363610.008439	214280.124034	297641.776507	460718.363742	257702.135279
7	206736.80094	252824.670355	436597.581931	255562.387014	129040.55976	384602.946774	225712.341572	276030.431416	476671.120451	279019.432092
8	209486.52356	246531.008262	436876.897177	261067.177891	138596.39311	399663.571002	226922.79965	267050.62285	473239.647712	282796.687334
9	197036.858548	248283.847197	416022.26839	276202.718625	157378.30724	433581.025865	211453.195048	266449.704667	446460.822155	296411.279416
10	187903.959693	264089.466465	387832.252286	287707.764805	179461.556752	467169.321556	201007.543488	282505.887593	414877.943204	307771.220683
11	179816.67853	274072.057927	341724.71678	324272.393297	195940.153466	520212.546763	188905.044473	287924.31667	358996.30306	340661.897317
12	180668.430353	283220.349465	321201.356637	345036.918442	220329.945104	565366.863546	185411.712959	290656.037895	329634.201296	354095.543739
13	189837.612712	301833.932928	310410.932212	348636.574571	238888.947577	587525.522148	191931.470181	305163.079486	313834.68075	352481.941594
14	214794.597983	298520.924938	327127.968172	330622.580995	263664.927912	594287.508906	216960.612549	301531.245857	330426.765956	333956.618875
15	237588.186942	283003.845665	363129.776422	309776.237087	286444.953884	596221.190971	240661.701151	286664.87087	367827.335429	313783.598222
16	262336.952143	266038.370259	406590.380892	272946.613161	327262.683546	600209.296707	265987.54853	269740.474386	412248.361453	276744.850167
17	284522.558229	254697.965022	453646.611667	251078.756593	346148.108489	597226.865083	289449.159798	259108.143962	461501.651783	255426.26775
18	301294.481473	251946.670699	491705.533611	240817.216103	355468.098114	596285.314216	304224.023022	254396.394425	496486.476772	243158.726099

19 rows × 24 columns

Figure 23: new constructed age distribution dataframe

3.4 Data Integration

There are current 3 data frames:

1. Number of fleets which has 6 columns after data cleaning
2. New constructed age distribution has 24 columns which we constructed in step 3.3
3. CO2 emission data has 2 columns for LPV and LCV.

All data frames are using for the models then we merge these two data frames into one. Hence, we got one data frame for input values with 30 columns and 1 period column, last 2columns for CO2 emission target values, explained in red rectangle below.

Examples:

- Period: Yellow Rectangle
- 30 Columns for features: Blue Rectangle
- 2 Columns for targets: Green Rectangle

No problems find when integrate data frames, since pandas provided concat dataframe object method and able to select columns with their column names.

In [10]:
```
# ---- step 3.4 ----
cleaned_data_df = pd.concat([number_fleets_df[num_fleets_select_cols], new_age_distribution_df, co2_emission_df[co2_select_cols]], axis=1)
cleaned_data_df
```

Out[10]:

	Period	Total LPV new	Total LPV used	Total LCV new	Total LCV used	Light passenger average age	Light commercial average age	0-4 years of LPV new	5-9 years of LPV new	10-14 years of LPV new	...	20+ years of LCV new	15+ year LCV
0	2000	1276498	870680	251143	96007	11.713869786193	12.404022756486	192258.488031	368736.816142	401920.954609	...	22036.5051166	61695.2468
1	2001	1256293	956915	254155	95590	11.863626238794	12.588450156197	177089.438589	361594.734542	406836.415224	...	22716.7415804	62870.9722
2	2002	1245805	1046144	258659	96693	11.957576280878	12.671380771511	174847.793571	343613.613755	417825.54808	...	23293.3172566	64263.3701
3	2003	1245233	1149489	265261	98774	12.011137827668	12.682297031759	179270.024976	343070.063835	409726.461868	...	23768.1289742	66711.0858
4	2004	1249158	1241364	274083	101725	12.094087303847	12.633976658351	184137.022961	332047.354364	403896.712332	...	25459.1852661	72204.1461
5	2005	1257339	1320880	284545	103708	12.212825403901	12.564519784656	191531.691566	320603.92302	397909.595244	...	27173.2135674	78595.1215
6	2006	1267476	1363917	293568	104168	12.401872886236	12.578583532490	199128.623289	276595.869338	428141.498935	...	28750.4505783	84217.9756
7	2007	1280762	1398318	303971	105041	12.591679419682	12.552225851255	206736.80094	252824.670355	436597.581931	...	30625.9773407	91280.1459
8	2008	1292558	1400142	312579	102839	12.844178333979	12.616798020544	209486.52356	246531.008262	436876.897177	...	33516.7334557	96650.5482
9	2009	1294924	1389668	315772	99049	13.233213091535	12.946859006949	197036.858548	248283.847197	416022.26839	...	38377.2814727	105730.334
10	2010	1306995	1398139	321520	95312	13.532333148849	13.163048902595	187903.959693	264089.466465	387832.252286	...	44147.4372334	114923.377
11	2011	1315826	1382331	327738	91259	13.812681952800	13.291566527000	179816.67853	274072.057927	341724.71678	...	48803.5910649	129571.402
12	2012	1350457	1385912	339965	89053	14.047409541544	13.396963530653	180668.430353	283220.349465	321201.356637	...	55466.0161614	142325.853
13	2013	1389608	1404935	358978	89550	14.194162516350	13.338720213832	189837.612712	301833.932928	310410.932212	...	61712.2790192	151775.70
14	2014	1434731	1449199	383012	91813	14.244257835865	13.175904807221	214794.597983	298520.924938	327127.968172	...	70387.293067	158649.424
15	2015	1479943	1499088	408647	94661	14.263345027207	13.011490975382	237588.186942	283003.845665	363129.776422	...	79094.1752958	164630.665
16	2016	1535175	1556538	439961	99907	14.303512648064	12.787276704545	262336.952143	266038.370259	406590.380892	...	93789.1885392	172012.104
17	2017	1590094	1617627	476173	106367	14.311541277982	12.514544065992	284522.558229	254697.965022	453646.611667	...	103658.263765	178846.853
18	2018	1641232	1657190	510666	111186	14.415534458399	12.343552002496	301294.481473	251946.670699	491705.533611	...	110603.176023	185532.963

19 rows × 33 columns

Figure 24: integrated, cleaned total data

In [10]:
```
# ---- step 3.4 ----
cleaned_data_df = pd.concat([number_fleets_df[num_fleets_select_cols], new_age_distribution_df, co2_emission_df[co2_select_cols]], axis=1)
cleaned_data_df
```

Out[10]:

...	20+ years of LCV new	15+ years of LCV new	0-4 years of LCV used	5-9 years of LCV used	10-14 years of LCV used	15-19 years of LCV used	20+ years of LCV used	15+ years of LCV used	Light passenger	Light commercial
...	22036.5051166	61695.2468666	14459.9996713	27733.1539159	30228.974185	15160.7523172	8424.11991069	23584.8722279	7.546587059314	1.830372872481
...	22716.7415804	62870.9722027	13474.547287	27513.3592839	30955.7507136	15102.3702276	8543.97248793	23646.3427155	6.765290559200	1.504674907110
...	23293.3172566	64263.3701875	13570.7897334	26669.5278594	32429.4779042	15315.5982512	8707.60625181	24023.204503	7.066370322830	1.559697576910
...	23768.1289742	66711.0858076	14220.0033624	27212.9011079	32500.1999983	15990.4683246	8850.42720679	24840.8955314	7.375727557740	1.585027089610
...	25459.1852661	72204.1461669	14995.1716762	27040.2279957	32891.2700091	17349.2378135	9449.09250553	26798.330319	7.671045880500	1.582260375660
...	27173.2135674	78595.1215418	15797.9420577	26444.0947497	32820.4313264	18741.7218092	9903.81005694	28645.5318662	7.549506885850	1.628644350860
...	28750.4505783	84217.9756913	16365.4620922	22732.1373479	35186.9728981	19681.7812431	10201.6464187	29883.4276618	7.644814489260	1.663934474560
...	30625.9773407	91280.1459864	16955.4064748	20735.2780601	35807.3136177	20959.8104053	10583.1914421	31543.0018474	7.796914139070	1.718953674540
...	33516.7334557	96650.5482617	16667.2478886	19614.5955219	34758.9688268	20771.1278775	11027.0598852	31798.1877627	7.692528220200	1.764464217150
...	38377.2814727	105730.334521	15071.3893652	18991.2819447	31821.6278807	21126.8020958	12037.8987136	33164.7008094	7.626856430540	1.774602036010
...	44147.4372334	114923.377876	13702.8085082	19258.6010105	28282.4858778	20980.954387	13087.1502164	34068.1046035	7.697161857650	1.830283394650
...	48803.5910649	129571.402033	12471.1704025	19008.2442013	23700.2885857	22489.8841791	13589.4126314	36079.2968105	7.575257564360	1.861729449030
...	55466.0161614	142325.853963	11913.7934256	18676.3605068	21180.9368329	22752.722003	14529.1872317	37281.9092347	7.444775230660	1.879310423130
...	61712.2790192	151775.70717	12233.6358299	19450.9737233	20003.6981505	22467.0592374	15394.6330588	37861.6922962	7.415187007980	1.937471009460
...	70387.293067	158649.424429	13745.3895013	19103.3034634	20933.9591476	21157.5905371	16872.7573506	38030.3478877	7.430753561430	1.993416827040
...	79094.1752958	164630.665523	15196.7578238	18101.6613711	23226.7241143	19814.0930961	18321.7635947	38135.8566908	7.662123754430	2.121840162330
...	93789.1885392	172012.104411	17072.5147802	17313.3977934	26460.3222328	17762.976391	21297.7888026	39060.7651936	7.800259903390	2.237801832490
...	103658.263765	178846.853096	19032.7181608	17037.6458533	30346.0230296	16795.5442273	23155.0687291	39950.6129564	8.077406643250	2.472227031640
...	110603.176023	185532.963207	20411.3301575	17068.240522	33310.8125238	16314.2706148	24081.3461819	40395.6167967	7.546587059314	1.830372872481

Figure 25: integrated, cleaned total data

3.5 Data Formatting

Since the new construct data is object type shown as figure below which actually is float, in this step we need to convert to integer, because there are not exist that parts of vehicles could run on the road.

Convert object type shown in rectangle below to int64 type in yellow below:
age and CO2 emission columns already in float64 type, others should be in int64 type, only new constructed columns are in object type thus it will only change type for these columns.

```
In [ ]: def step_3_5_convert_object_to_int(data_df):
            for column in data_df.columns:
                if data_df.dtypes[column] != np.float64:
                    data_df[column] = data_df[column].astype(np.int64)

            return data_df
```

Figure 26: convert object types to int64

```
In [11]: cleaned_data_df.dtypes

Out[11]: Period                              int64
         Total LPV new                       int64
          Total LPV used                     int64
         Total LCV new                       int64
          Total LCV used                     int64
         Light passenger average age       float64
         Light commercial average age      float64
         0-4 years of LPV new               object
         5-9 years of LPV new               object
         10-14 years of LPV new             object
         15-19 years of LPV new             object
         20+ years of LPV new               object
         15+ years of LPV new               object
         0-4 years of LPV used              object
         5-9 years of LPV used              object
         10-14 years of LPV used            object
         15-19 years of LPV used            object
         20+ years of LPV used              object
         15+ years of LPV used              object
         0-4 years of LCV new               object
         5-9 years of LCV new               object
         10-14 years of LCV new             object
         15-19 years of LCV new             object
         20+ years of LCV new               object
         15+ years of LCV new               object
         0-4 years of LCV used              object
         5-9 years of LCV used              object
         10-14 years of LCV used            object
         15-19 years of LCV used            object
         20+ years of LCV used              object
         15+ years of LCV used              object
         Light passenger                   float64
         Light commercial                  float64
         dtype: object
```

Figure 27: converted types of intergration dataframe

```
In [25]: # ---- step 3.5 ----
cleaned_data_df = step_3_5_convert_object_to_int(cleaned_data_df)

sparkdf = spark.createDataFrame(cleaned_data_df)
sparkdf.printSchema()
```

```
root
 |-- Period: long (nullable = true)
 |-- Total LPV new: long (nullable = true)
 |--  Total LPV used: long (nullable = true)
 |-- Total LCV new: long (nullable = true)
 |--  Total LCV used: long (nullable = true)
 |-- Light passenger average age: double (nullable = true)
 |-- Light commercial average age: double (nullable = true)
 |-- 0-4 years of LPV new: long (nullable = true)
 |-- 5-9 years of LPV new: long (nullable = true)
 |-- 10-14 years of LPV new: long (nullable = true)
 |-- 15-19 years of LPV new: long (nullable = true)
 |-- 20+ years of LPV new: long (nullable = true)
 |-- 15+ years of LPV new: long (nullable = true)
 |-- 0-4 years of LPV used: long (nullable = true)
 |-- 5-9 years of LPV used: long (nullable = true)
 |-- 10-14 years of LPV used: long (nullable = true)
 |-- 15-19 years of LPV used: long (nullable = true)
 |-- 20+ years of LPV used: long (nullable = true)
 |-- 15+ years of LPV used: long (nullable = true)
 |-- 0-4 years of LCV new: long (nullable = true)
 |-- 5-9 years of LCV new: long (nullable = true)
 |-- 10-14 years of LCV new: long (nullable = true)
 |-- 15-19 years of LCV new: long (nullable = true)
 |-- 20+ years of LCV new: long (nullable = true)
 |-- 15+ years of LCV new: long (nullable = true)
 |-- 0-4 years of LCV used: long (nullable = true)
 |-- 5-9 years of LCV used: long (nullable = true)
 |-- 10-14 years of LCV used: long (nullable = true)
 |-- 15-19 years of LCV used: long (nullable = true)
 |-- 20+ years of LCV used: long (nullable = true)
 |-- 15+ years of LCV used: long (nullable = true)
 |-- Light passenger: double (nullable = true)
 |-- Light commercial: double (nullable = true)
```

Figure 28: show result by PySpark

4 Data Transformation

4.1 Data Reduction

Although, most of features are needed for our project success criteria, we go through on Principle Component Analysis (PCA) for find are there any features could be reduced.

we set all features (n) go through PCA, thus there will be return n floats values (ratios) which sum equal to 1, it will discuss how many variances that the n features under this PCA component can explained. In this case, we are aim for the components can explained over 90% variances.

The PCA components have n floats for n features, and how the bigger of float is then how the feature important is. We run PCA separately on LPV_input and LCV_input data frames, and rank importance of all the features among the components which explained over 90% variances.

We use PySpark for running PCA algorithms:

For LPV_input:

- Select LPV columns from integrated dataframe generated from step 3
- Convert Pandas Dataframe into spark dataframe
- assemble spark dataframe columns into one column name features for run PCA algorithm

```
In [16]:
from pyspark.ml.feature import PCA
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

default_LPV_cols = ['Period', 'Total LPV new',' Total LPV used', 'Light passenger average age','0-4 years of LPV new',
                    '5-9 years of LPV new', '10-14 years of LPV new', '15-19 years of LPV new', '20+ years of LPV new',
                    '15+ years of LPV new', '0-4 years of LPV used', '5-9 years of LPV used', '10-14 years of LPV used',
                    '15-19 years of LPV used', '20+ years of LPV used', '15+ years of LPV used', ]

LPV = cleaned_data_df[default_LPV_cols]
sparkdf = spark.createDataFrame(LPV)

assembler = VectorAssembler(
    inputCols= sparkdf.columns,
    outputCol="features")
output = assembler.transform(sparkdf)
```

Figure 29: prepare for run PCA with spark

- Run PCA on all 16 LPV features to check the explained variance ration.

```
In [18]:
pca = PCA(k=16, inputCol="features", outputCol="pcaFeatures")
model = pca.fit(output)
print(model.explainedVariance)
```

```
[0.8153903239063524, 0.09419784479044002, 0.08171192437520455, 0.007500595594773108, 0.0008887751685419086, 0.0002986294755813989
3, 7.933526453875018e-06, 2.1451451515221327e-06, 1.6878328922024958e-06, 1.4018325629744476e-07, 6.510190066429272e-13, 4.360991
5641636746e-13, 1.3640688972886595e-13, 1.0623806710864865e-13, 2.277654778367207e-14, 2.162438752600682e-16]
```

Figure 30: LPV PCA explained variance ration

- The first 2 components explained 0.9095881686967918 variance; thus, we will rank the features importance for first 2 components.

```
In [19]:
result = model.transform(output).select("pcaFeatures")
result.show(truncate=False)
```

```
+-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------+
|pcaFeatures
|
+-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------+
|[-1264814.7339085278,330939.0612045998,557593.0671360802,316761.2507637557,-619659.1341903186,-763837.6681354614,-102166.96049414932,-
3420.8128091675208,-31750.660636702545,14141.603817835858,650.1402938608371,-698.6420832456715,-1073.7961956343133,-773.6143319782423,-
977.3899880792524,514.0761860972468] |
|[-1326682.608241482,384563.07795103145,504665.46029208624,345052.7680601755,-598378.9938770655,-771472.0561358365,-100712.34203331357,
-3536.914961317372,-33938.37153158732,14103.3437136308,650.1347643322806,-698.9343069242786,-1073.740791881908,-773.6790729743952,-977.
3704471717501,514.0711926297492]      |
|[-1398261.2627433748,447100.21316404635,459687.09794044716,357616.6774785518,-587615.7491873506,-773416.0330319153,-99145.77074983387,
-4846.256810321473,-32323.0475520848,14166.678540154917,650.3330330427561,-698.3472897704196,-1073.8466662035498,-773.6134743436851,-97
7.4222812066364,514.0788488411132]    |
```

Figure 31: LPV PCA first 2 components

- Rank the importance for components in ascending:

```
In [45]: result = model.transform(output).select("pcaFeatures")
         # result.show(truncate=False)
         LPV_PCA_list = result.head(2)

         print(LPV_PCA_list[0][0])
         get_sort_array_index(LPV_PCA_list[0][0])
         print(LPV_PCA_list[1][0])
         get_sort_array_index(LPV_PCA_list[1][0])
```

```
[-1264814.7339085278,330939.0612045998,557593.0671360802,316761.2507637557,-619659.1341903186,-763837.6681354614,-102166.96049414932,-3420.8128091675208,-31750.660636702545,14141.603817835858,650.1402938608371,-698.6420832456715,-1073.7961956343133,-773.6143319782423,-977.3899880792524,514.0761860972468]
[0, 5, 4, 6, 8, 7, 12, 14, 13, 11, 15, 10, 9, 3, 1, 2]
[-1326682.608241482,384563.07795103145,504665.46029208624,345052.7680601755,-598378.9938770655,-771472.0561358365,-100712.34203331357,-3536.914961317372,-33938.37153158732,14103.3437136308,650.1347643322806,-698.9343069242786,-1073.740791881908,-773.6790729743952,-977.3704471717501,514.0711926297492]
[0, 5, 4, 6, 8, 7, 12, 14, 13, 11, 15, 10, 9, 3, 1, 2]
```

Figure 32: Rank LPV PCA first 2 components

- All the components 1 or 2 indicate 0, 5, 4 and 6 features are the least variance features, Mark the top 4 unnecessary columns with rectangles:

```
default_LPV_cols = ['Period', 'Total LPV new', ' Total LPV used', 'Light passenger average age', '0-4 years of LPV new',
                    '5-9 years of LPV new', '10-14 years of LPV new', '15-19 years of LPV new', '20+ years of LPV new',
                    '15+ years of LPV new', '0-4 years of LPV used', '5-9 years of LPV used', '10-14 years of LPV used',
                    '15-19 years of LPV used', '20+ years of LPV used', '15+ years of LPV used', ]
```

Figure 33: Mark LPV least relevant components

Explanation of ranks: the period is the least relevant feature in LPV set combining with first and second components, because it has lowest variance.

0-4 years, 5-9 years, and 10-14-year group, these groups do not have large variance than others in this component analysis, we cannot satisfy project requirement if we ignore these.

However, we can reduce 15+ years groups, because it combines with 15-19 years and 20+ years groups which is an overlap of the project objectives that will influence the project accuracy. Alert when we predict one-year CO2 emission it will count 15-20+ years groups twice so as to reduce the importance of other groups.

For LCV_input:

- Ignore 15+ year group first for coincide with LPV result, 14 remaining features.
- Similar steps with LPV to get the component explained variance ration:

```
In [46]: default_LCV_cols = ['Period', 'Total LCV new', ' Total LCV used', 'Light commercial average age', '0-4 years of LCV new',
                           '5-9 years of LCV new', '10-14 years of LCV new', '15-19 years of LCV new', '20+ years of LCV new',
                           '0-4 years of LCV used', '5-9 years of LCV used', '10-14 years of LCV used','15-19 years of LCV used',
                           '20+ years of LCV used', ]

         LCV = cleaned_data_df[default_LCV_cols]
         LCV_sparkdf = spark.createDataFrame(LCV)

         assembler = VectorAssembler(
             inputCols= LCV_sparkdf.columns,
             outputCol="features")
         LCV_output = assembler.transform(LCV_sparkdf)
```

```
In [47]: pca = PCA(k=14, inputCol="features", outputCol="pcaFeatures")
         model = pca.fit(LCV_output)
         print(model.explainedVariance)
```

```
[0.9447786697563921,0.043323538528097456,0.010452615866478996,0.0010278587466225784,0.0003027918049081290 3,0.00010032857531640 58,1.2260389980107051e-05,1.6347450561570994e-06,1.6380342374135463e-07,1.3777179056607615e-07,7.172124054411663e-12,4.107951709765241e-12,6.3982118130028 7e-13,1.3642235131927117e-14]
```

Figure 34: LCV PCA explained variance ration

- The first component explained 0.94477867 variance; thus, we will rank the features importance for first components.

```
In [50]: result = model.transform(LCV_output).select("pcaFeatures")
         result.show(truncate=False)
         LCV_PCA_list = result.head(2)

+--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-+
|pcaFeatures
|
+--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-+
|[-261496.55746783238,-58249.78886673159,-6432.804006297585,-124469.95262869389,40715.52428986373,22309.625605630932,1180.560203768419
6,-366.31804284231293,4774.375266859259,657.9417811165006,-511.50959261962726,1812.3140054679284,-519.7993149442373,148.24423767496629]
|
```

Figure 35: LCV PCA first components

- Rank the importance for components in ascending:

```
In [51]: print(LCV_PCA_list[0][0])
         get_sort_array_index(LCV_PCA_list[0][0])

[-261496.55746783238,-58249.78886673159,-6432.804006297585,-124469.95262869389,40715.52428986373,22309.625605630932,1180.5602037684196,
-366.31804284231293,4774.375266859259,657.9417811165006,-511.50959261962726,1812.3140054679284,-519.7993149442373,148.24423767496629]
[0, 3, 1, 2, 12, 10, 7, 13, 9, 6, 11, 8, 5, 4]
```

Figure 36: Rank LCV PCA first components

- Mark the top 4 unnecessary columns with red rectangles:

```
default_LCV_cols = ['Period', 'Total LCV new', ' Total LCV used', 'Light commercial average age', '0-4 years of LCV new',
                   '5-9 years of LCV new', '10-14 years of LCV new', '15-19 years of LCV new', '20+ years of LCV new',
                   '0-4 years of LCV used', '5-9 years of LCV used', '10-14 years of LCV used','15-19 years of LCV used',
                   '20+ years of LCV used', ]
```

Figure 37: Mark LCV least relevant components

Explanation of ranks: As shown on the figure we know that Period is not a relevant feature and it will not attend model predict, thus we can ignore it.

Total LCV new, Total LCV used, and Light commercial average age has a lower rank because it has a low variance. it is important component of LCV used we cannot ignore it for this project success.

Step 4.1 sum up, we reduce features related to 15+ year groups both for LPV and LCV inputs, since they have potential influence on the model accuracy, their high variance are made from other groups. Period we will use for indicate the result and it won't attend the datamining model.

```
In [ ]: # ---- step 4.1 ----
        reduced_LPV_cols = ['Total LPV new', ' Total LPV used', 'Light passenger average age','0-4 years of LPV new', '5-9 years of LPV new',
                            '10-14 years of LPV new', '15-19 years of LPV new', '20+ years of LPV new', '0-4 years of LPV used',
                            '5-9 years of LPV used', '10-14 years of LPV used', '15-19 years of LPV used', '20+ years of LPV used',
                            'Light passenger']

        reduced_LCV_cols = ['Total LCV new', ' Total LCV used', 'Light commercial average age', '0-4 years of LCV new', '5-9 years of LCV new',
                            '10-14 years of LCV new', '15-19 years of LCV new', '20+ years of LCV new', '0-4 years of LCV used',
                            '5-9 years of LCV used', '10-14 years of LCV used', '15-19 years of LCV used', '20+ years of LCV used',
                            'Light commercial']
```

Figure 38: Reduced LPV and LCV columns for next steps

4.2 Data Transformation/ Normalization

In Data Normalization, because our data set columns have different ranges. for example, for the Total LPV used column range [870690,1657190] and CO2 emission range [6.76, 7.54]. As shown in the different colour rectangles below. If not apply log transformation, this might influence parameters like (β_i) in linear regression model $y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i \;\; i = 1, \ldots, n.$

For avoid overfitting in next steps of machine learning models this step we will do normalization under L2 normalization.

$$L' = L + \frac{\lambda}{2n}\sum_{w} w^2 \quad w \leftarrow \left(1 - \frac{\eta\lambda}{n}\right) w - \eta\frac{\partial L}{\partial w}$$

Figure 39: L2 normalization mathmatic model

The above equations are explained the L2 normalization will consider all the feature ranges, it will prevent weights from going to large, and transform the values for all data in range (0,1), but not equal to 0 or 1.

Important: spark machine learning models provide **more powerful** L2 normalization on the models, it could change the parameter lambda and omega. If the model input values already done l2 normalization then the model will force do l2 normalization again which will influence the accuracy. Hence here we represent we are **able to apply** l2 normalization on data before the training models and applying powerful normalization method on spark models from step 6.

LPV before l2 normalization:

```
In [16]: # ---- step 4.2 ----
         cleaned_data_df[reduced_LPV_cols]
```

Out[16]:

	Total LPV new	Total LPV used	Light passenger average age	0-4 years of LPV new	5-9 years of LPV new	10-14 years of LPV new	15-19 years of LPV new	20+ years of LPV new	0-4 years of LPV used	5-9 years of LPV used	10-14 years of LPV used	15-19 years of LPV used	20+ years of LPV used	Light passenger
0	1276498	870680	11.713869786193	192258	368736	401920	201575	112006	131136	251509	274144	137491	76397	7.546587059314
1	1256293	956915	11.863626238794	177089	361594	406836	198483	112289	134888	275425	309886	151184	85530	6.765290559200
2	1245805	1046144	11.957576280878	174847	343613	417825	197328	112189	146825	288543	350862	165703	94209	7.066370322830
3	1245233	1149489	12.011137827668	179270	343070	409726	201590	111576	165486	316691	378223	186090	102997	7.375727557740
4	1249158	1241364	12.094087303847	184137	332047	403896	213044	116032	182988	329975	401376	211715	115308	7.671045880500
5	1257339	1320880	12.212825403901	191531	320603	397909	227221	120072	201210	336805	418018	238704	126140	7.549506885850
6	1267476	1363917	12.401872886236	199128	276595	428141	239480	124129	214280	297641	460718	257702	133574	7.644814489260
7	1280762	1398318	12.591679419682	206736	252824	436597	255562	129040	225712	276030	476671	279019	140884	7.796914139070
8	1292558	1400142	12.844178333979	209486	246531	436876	261067	138596	226922	267050	473239	282796	150132	7.692528220200
9	1294924	1389668	13.233213091535	197036	248283	416022	276202	157378	211453	266449	446460	296411	168892	7.626856430540
10	1306995	1398139	13.532333148849	187903	264089	387832	287707	179461	201007	282505	414877	307771	191976	7.697161857650
11	1315826	1382331	13.812681952800	179816	274072	341724	324272	195940	188905	287924	358996	340661	205843	7.575257564360
12	1350457	1385912	14.047409541544	180668	283220	321201	345036	220329	185411	290656	329634	354095	226114	7.444775230660
13	1389608	1404935	14.194162516350	189837	301833	310410	348636	238888	191931	305163	313834	352481	241523	7.415187007980
14	1434731	1449199	14.244257835865	214794	298520	327127	330622	263664	216960	301531	330426	333956	266323	7.430753561430
15	1479943	1499088	14.263345027207	237588	283003	363129	309776	286444	240661	286664	367827	313783	290150	7.662123754430
16	1535175	1556538	14.303512648064	262336	266038	406590	272946	327262	265987	269740	412248	276744	331816	7.800259903390
17	1590094	1617627	14.311541277982	284522	254697	453646	251078	346148	289449	259108	461501	255426	352141	8.077406643250
18	1641232	1657190	14.415534458399	301294	251946	491705	240817	355468	304224	254396	496486	243158	358924	7.546587059314

Figure 40: LPV before L2 normalization

```
In [30]: # ---- step 4.2 ----
LPV_df, LCV_df = step_4_2_normalization(cleaned_data_df,reduced_LPV_cols,reduced_LCV_cols)
LPV_df
```

Out[30]:

	Total LPV new	Total LPV used	Light passenger average age	0-4 years of LPV new	5-9 years of LPV new	10-14 years of LPV new	15-19 years of LPV new	20+ years of LPV new	0-4 years of LPV used	5-
0	0.742510709923	0.506455337114	0.000006813699	0.111832234808	0.214485591935	0.233787992251	0.117251728050	0.065151417844	0.076278916580	0.1462
1	0.714235993429	0.544031635655	0.000006744787	0.100679807848	0.205575808994	0.231297089630	0.112842866022	0.063839283882	0.076687416615	0.1565
2	0.687310716569	0.577157727152	0.000006596996	0.096463103664	0.189571319149	0.230514085391	0.108865873134	0.061894680131	0.081003364058	0.1591
3	0.660092946235	0.609339441433	0.000006367055	0.095030297520	0.181860010990	0.217194085355	0.106862038696	0.059145983578	0.087723455209	0.1678
4	0.638363673527	0.634380665396	0.000006180504	0.094100483488	0.169687695795	0.206405061876	0.108872977208	0.059296433091	0.093513304075	0.1686
5	0.621636080669	0.653051139139	0.000006038095	0.094694096156	0.158508081250	0.196728639788	0.112339450129	0.059364330127	0.099479452869	0.1665
6	0.613374294298	0.660045339996	0.000006001684	0.096364741009	0.133853629522	0.207191839321	0.115892431887	0.060070200759	0.103697303761	0.1440
7	0.608718792218	0.664590645332	0.000005984556	0.098257200189	0.120161841094	0.207505218398	0.121463153948	0.061329952753	0.107276087227	0.1311
8	0.611819784687	0.662743626956	0.000006079667	0.099158164984	0.116693056202	0.206791014604	0.123573530727	0.065603071490	0.107411326363	0.1264
9	0.616349577400	0.661445215724	0.000006298659	0.093783925028	0.118176141708	0.198015469548	0.131464847340	0.074907765855	0.100646035744	0.1268
10	0.619061376415	0.662231954797	0.000006409623	0.089000715238	0.125086400358	0.183697574771	0.136273123787	0.085002141303	0.095207456868	0.1338
11	0.626165715044	0.657813631166	0.000006573079	0.085569531394	0.130423391735	0.162617133882	0.154312202942	0.093242503344	0.089894738666	0.1370
12	0.634350720120	0.651005011802	0.000006598495	0.084865253690	0.133037046683	0.150877877380	0.162073901699	0.103495231476	0.087093185024	0.1365
13	0.639754966500	0.646811290565	0.000006534782	0.087398146510	0.138959448135	0.142908172054	0.160506856970	0.109980501290	0.088362193133	0.1404
14	0.641032021600	0.647496265621	0.000006364277	0.095969092497	0.133377531459	0.146159023629	0.147720575526	0.117804011305	0.096936852557	0.1347
15	0.640317289155	0.648600631487	0.000006171229	0.102795650978	0.122445062940	0.157112656970	0.134028762300	0.123933857976	0.104125225853	0.1240
16	0.639310558085	0.648206997547	0.000005956576	0.109247593640	0.110789260020	0.169320943744	0.113666037805	0.136285473552	0.110768021505	0.1123
17	0.636917702684	0.647946141951	0.000005732538	0.113966280360	0.102019772492	0.181709488968	0.100570169408	0.138650789795	0.115939807410	0.1037
18	0.638267727303	0.644473721576	0.000005606136	0.117171878583	0.097980663808	0.191221858247	0.093652645870	0.138239903005	0.118311342376	0.0985

Figure 41: LPV after L2 normalization

```
In [30]: # ---- step 4.2 ----
LPV_df, LCV_df = step_4_2_normalization(cleaned_data_df,reduced_LPV_cols,reduced_LCV_cols)
LPV_df
```

Out[30]:

5-9 years of LPV new	10-14 years of LPV new	15-19 years of LPV new	20+ years of LPV new	0-4 years of LPV used	5-9 years of LPV used	10-14 years of LPV used	15-19 years of LPV used	20+ years of LPV used	Light passenger
4485591935	0.233787992251	0.117251728050	0.065151417844	0.076278916580	0.146297233636	0.159463513504	0.079975479803	0.044438448557	0.000004389683
5575808994	0.231297089630	0.112842866022	0.063839283882	0.076687416615	0.156586440019	0.176178435333	0.085952126161	0.048626080475	0.000003846248
9571319149	0.230514085391	0.108865873134	0.061894680131	0.081003364058	0.159189195814	0.193570593020	0.091418358144	0.051975112716	0.000003898517
1860010990	0.217194085355	0.106862038696	0.059145983578	0.087723455209	0.167876610430	0.200494473246	0.098645551768	0.054598290588	0.000003909843
9687695795	0.206405061876	0.108872977208	0.059296433091	0.093513304075	0.168628830918	0.205117253242	0.108193811464	0.058926443626	0.000003920174
8508081250	0.196728639788	0.112339450129	0.059364330127	0.099479452869	0.166518449002	0.206670652202	0.118016715460	0.062364386387	0.000003732522
3853629522	0.207191839321	0.115892431887	0.060070200759	0.103697303761	0.144038497241	0.222956946025	0.124710670963	0.064640954138	0.000003699583
0161841094	0.207505218398	0.121463153948	0.061329952753	0.107276087227	0.131191156683	0.226551533701	0.132611764470	0.066959152694	0.000003705707
6693056202	0.206791014604	0.123573530727	0.065603071490	0.107411326363	0.126405525710	0.224003087742	0.133858741991	0.071063525130	0.000003641184
8176141708	0.198015469548	0.131464847340	0.074907765855	0.100646035744	0.126822677276	0.212503152560	0.141083796877	0.080388125346	0.000003630182
5086400358	0.183697574771	0.136273123787	0.085002141303	0.095207456868	0.133809183772	0.196507505127	0.145776486429	0.090929901644	0.000003645780
0423391735	0.162617133882	0.154312202942	0.093242503344	0.089894738666	0.137015180836	0.170836407730	0.162111281167	0.097955071021	0.000003604859
3037046683	0.150877877380	0.162073901699	0.103495231476	0.087093185024	0.136529962011	0.154839113927	0.166329189482	0.106212621897	0.000003497037
8959448135	0.142908172054	0.160506856970	0.109980501290	0.088362193133	0.140492530873	0.144484531002	0.162277038090	0.111193616310	0.000003413842
3377531459	0.146159023629	0.147720575526	0.117804011305	0.096936852557	0.134722834110	0.147633003517	0.149210193273	0.118992041775	0.000003320031
2445062940	0.157112656970	0.134028762300	0.123933857976	0.104125225853	0.124029043942	0.159145310000	0.135762444866	0.125537308834	0.000003315121
0789260020	0.169320943744	0.113666037805	0.136285473552	0.110768021505	0.112330926401	0.171677169671	0.115247682568	0.138181948078	0.000003248352
2019772492	0.181709488968	0.100570169408	0.138650789795	0.115939807410	0.103786613940	0.184855836640	0.102311775987	0.141051306867	0.000003235433
7980663808	0.191221858247	0.093652645870	0.138239903005	0.118311342376	0.098933457765	0.193081167597	0.094563050218	0.139583925828	0.000002934834

Figure 42: LPV after L2 normalization

Step summary, the above figures are explained the data under L2 normalization will transform to range 0 to 1 which make the input smoother for the model.

5 Datamining Method Selection

5.1 Understanding Datamining Objectives

There are two datamining objectives mentioned in step 1.3:

- The first objective is for predict CO2 emission values for fleets.
- The second objective is basing on the models created for the first object to divide the CO2 emission for age groups among fleets, for second objective, needs to running the models several times and base on the benchmarks mentioned in step 1.3.1.2.

Explanation: hence, we know the model need to predict values rather than labels by Objectives and could run multiple times with stable result.

5.1.1 Method of Machine Learning in Datamining

There are three main categories of Machine Learning:

- Supervised Learning: indicate to problem that the model input data **have** labels, two problems are derived, they are:
    - Classification: predict a **label** for given features vector, accepted numeric, String, and Boolean type of data.
    - Regression: predict a **value** for given features vector, accepted numeric type of data, but the data should have some variance, better not equal to 0 or 1 which is same as Boolean values. If some values are equal to 0 it might make some features useless, because whatever 0 relation to other features will be useless. If some values are equal to 2 then it might cause this feature become useless, because everything relation to1 will be themselves.
- Unsupervised Learning: indicate to problem that the model input data **don't have** labels; one problem belongs to this is:
    - Clustering: Partitioning data into **n groups**.
- Reinforcement Learning: build software agents in an environment to optimized the notion of cumulative reward.

5.1.2 Matching Datamining Objectives and Data Type accessible

Explanation: After match Machine Learning Algorithms and Objectives, this project is a supervised learning problem its data have labels, otherwise it will be an un-supervised learning problem. It will be either a classification problem or regression problem. The Objective is for predict values, which is means the target predict value have relation with all feature values thus it is a regression problem.

Data Type: In step 3.5 we know that there are all numeric features. For the model training aspect, good data will cause a more reliable result. Data transformation able to reduce the

chance of overfitting, our data have a large range for different columns and we have done data transformation at step 4.2.

5.1.3 Modelling Requirement Assumptions and Criteria

The general modelling requirement for Regression Problem.

- Splitting the dataset into training and test set, if you do not have unseen data for the model then it will easily cause overfitting.
- A loss function, using for training and tell the model how close to the real predict value.
- For the spark requirement, needs to normalized the data, which will be force normalized during training and we proved that we are able to do in step 4.2
- Learning rate, for control the learning speed and able to avoid over fitting.
- Stop criteria, a method for avoid over fitting, might set maximum iteration times or wait until model convergence.

Noisy data, for avoid over fitting and it will be vectors with all 0 for this problem to indicate no vehicles on road means no emission

5.2 Datamining Method Selection

In step 5.1 conclude matching objectives belongs to **Supervised Learning, Regression Problem**, hence there are some methods for Regression Problem.

5.2.1 Linear Model

- The most traditional way is model working on **Linear Model**, here is its equation denote equation 1:

$$equation\ 1: \ y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i \ \ i = 1, \ldots, n.$$

where y_i is our target values, where $(\beta_0\, \beta_1 \ldots \beta_p)$ are the model coefficients, $(x_{i1} \ldots x_{ip})$ are the features. Objective for the linear model is for find best coefficients based on our data that could predict accuracy target values. We will discuss how to choose the best method of optimization in step 5.2.

5.2.2 Neural Network

- Second model is **Neural Network**, equation 2:

Illustrated in Figure 43, Similar with equation 1, x indicates input features, then follow some arrows to the next node in hidden layer, the arrows indicate some activation function with

coefficients like equation 1. After feed-forward propagation from input nodes through hidden nodes to the output nodes. The objective of Neural Network is similar with method above which is for find the best coefficients to have an accuracy predict output.

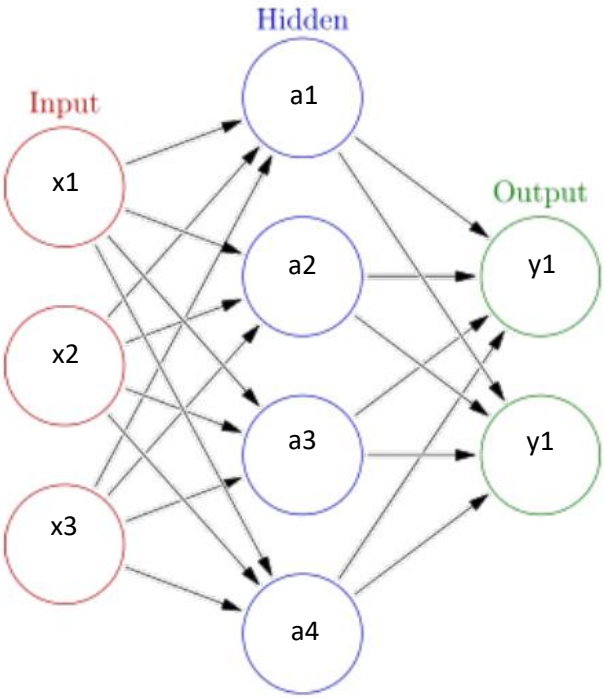


Figure 43: Neural Network Model

5.2.3 Empirical Based Algorithm

- **Empirical Based Algorithm**, such as Decision Tree (C5.0), SVM and KNN.

There are more rely on the training data their objective is for finding similar one to the training data that will not fulfil our datamining objective 2, because we only have aggregate data of whole age groups of fleets and we want to divide separate groups from them, thus no similar data. These algorithms are actually applying Classification to find answers as predict value.

Decision trees are using tree structure to find nearest instance. SVM is trying to find a hyper-plane could divide N-dimensional data instances into two groups then find the nearest instance, and KNN is using for base on one instance find nearest K vectors.

Step sum up, we select multiple algorithms offered by PySpark and aiming for:

- Verify the Empirical Based Algorithm, such as Decision Tree (C5.0), SVM and KNN, are not suit for this problem.
- Select the best regression algorithm provided by PySpark.

The objective for select one right algorithm for conduct this iteration is which one able to reflect the most accuracy predict. The procedures:

1. Run multiple algorithms, where PySpark offered. For prove assumption above that SVM, Decision Tree and KNN might cause over fitting or under fitting problem, refer to step 6.1.
2. Comparing models to select one with accuracy more than 80%, refer to step 6.2.
3. Each selected algorithm run serial times with different setting for find the best setting to get the most accuracy models, refer to step 6.3

6 Datamining Algorithm Selection

All the model constructed by **PySpark**, this project will only select regression models where **PySpark** offered.

6.1 Conduct Datamining Exploratory analysis and discuss

For compare regression models we need set a criterion to measure its performance.

Definition: Residuals means the difference values between real values with predict values.

$$Residual(y, \hat{y}) = (y_i - \hat{y}) \text{ where } y \text{ is real value, } \hat{y} \text{ is the predict value}$$

There are four main methods for use residuals to measure model performance.

Definition: MSE is for sum of the square residuals with predict values and real values.

$$\text{MSE}(y, \hat{y}) = \sum_i (y_i - \hat{y}_i)_2$$

Definition: RMSE is root of MSE.

$$\text{RMSE}(y, \hat{y}) = \sqrt{\sum_i (y_i - \hat{y}_i)_2}$$

Definition: MAE sum of absolute value of residuals.

$$\text{MAE}(y, \hat{y}) = \sum_i |y_i - \hat{y}_i|_2$$

Definition: R2 is combine use sum of squares and sum of residuals.

$$\text{R}_2(y, \hat{y}) = 1 - \sum_{ni=1} (y_i - \hat{y})_2 \sum_{ni=1} (y_i - \overline{y})_2$$

R2 will return a value from 0 – 1, good models will more and more close to 1. In this iteration using R2 measurement for comparing models in a Regression problem.

Select model Criteria One: The success models will have R2 with greater than 0.8. For Matching Datamining Objective One.

Select model Criteria Two: The success models will have lower residuals, which means avoid model overfitting in the case of optimized with R2 to 1 but have residuals away from 0. For Matching Datamining Objective Two.

6.1.1 Linear Regression

The Linear Regression is a common statistical technique to predict values. It is optimizing linear model which we mention as equation 1 in step 5.2.1.

Advantages: simple to use, able to train data fast, predicting values not rely on other data.
Disadvantages: easily overfitting or underfitting.

Loss function: during the training procedures to indicate model how bad or good the performance is then tuning the parameters.

There are several types of Linear Regression:
Ordinary least squares Regression: using original data for training.

Ridge Regression: Applying L2 regularization on data before training.

Lasso Regression: Applying L1 regularization on data before training.

Elastic Net Regression: Combine Applying L1 + L2 regularization on data before training.

6.1.2 Artificial Neural Network Regression

Artificial Neural Network is inspired from animal's brain to simulated biological neural networks. It is a power technique cross computer science, statistics, and optimization. It could solve classification or regression problems with the model mentioned in step 5.2.2.

Advantages: Accuracy, after training model, feedforward run the model able to get the predicts, predicting values not rely on other data.
Disadvantages: Hardly to train, need much time to train the model, gradient vanish and gradient exploding problems.

The Artificial Neural Network models are always different when they have different architectures, optimization algorithms, loss functions, or activity functions.

6.1.3 Tree-Based Regression

Tree Based Regression is a computer science technique to predict values. Setting different condition as tree nodes to divided data into different groups, for unseen data to find the most similar groups.

Advantages: Easily to build model with little data preparation, accept numerical or categorical data.
Disadvantages: Rely on the training data.

There are different tree-based regression models such as Decision Tree, Random forest, and Gradient-boost tree regression. The difference is using different method to generate or update tree nodes.

6.1.4 KNN Regression

KNN is a lazy learning, similar with Tree-Based Regression. They are both applying classification methods to divided unseen data to a group, rather than using tree hierarchy, this using vector distance measurement.

Advantages: Easily to build model with little data preparation, very fast.
Disadvantages: Rely on the training data.

The different K regression is using different distance measurement, such as Euclidean distance and Mahalanobis distance.

6.1.5 SVM Regression

SVM is an associated learning algorithm, similar with KNN and Tree-based method, they are all classify unseen data to predict its value. SVM is basing on the kernel function which will build a hyper plane to divide data into two groups.

Advantages: Able to handle high dimensional vector space.
Disadvantages: Rely on the training data, data vector might couple in some dimensional that not easily to divide by hyper plain.

There are some regression models not offered by spark. they are Neural Network Regression, SVM Regression, KNN regression. Hence, we only consider the spark provided models in next step, Discuss the R2 result with predict results for select best algorithm(s) further in step 6.2.

6.2 Select data-mining algorithm based on discussion

Since there are no Neural Network Regression provided by PySpark, referring to the regression methods, the most fitting algorithm for this research is **Linear Model Regression.** The project data are all numeric and the predict values should not rely on training data, all of these reasons implies Linear Model is best fitting for this problem. Hence, our selection bases on two procedures:

First Step: prove Tree-Based Regression not suit for this research problem.

1. Assemble features into one vector.

2. Simply splitting 70% data set into training set, reminder 30% to test set, under random seed 722.

3. Add noisy data into training set and test set respectively.

```
In [ ]: def assemble_dataset(df, seed):
            spark_df = spark.createDataFrame(df)
            assembler = VectorAssembler(inputCols= spark_df.columns[:-1],outputCol="features")
            spark_assembler_df = assembler.transform(spark_df)

            selected_data = spark_assembler_df.select('features',spark_df.columns[-1])
            train_data,test_data = selected_data.randomSplit([0.7,0.3],seed=seed)

            noisy_df = spark.createDataFrame([[Vectors.dense(np.zeros(13)),0.0]])
            train_data,test_data = train_data.union(noisy_df),test_data.union(noisy_df)

            return train_data,test_data
```

Figure 44: assemble features, splitting for training and testing, add noisy data.

2. Check LPV and LCV training and test sets.

```
In [23]: LPV_train_data,LPV_test_data = assemble_dataset(LPV_df,SEED)
         LCV_train_data,LCV_test_data = assemble_dataset(LCV_df,SEED)

         print('LPV training set:')
         LPV_train_data.show()
         print('LPV testing set:')
         LPV_test_data.show()
         print('LCV training set:')
         LCV_train_data.show()
         print('LCV testing set:')
         LCV_test_data.show()
```

Figure 45: training and testing

LPV training set:

features	Light passenger
[1276498.0,870680...	7.546587059314116
[1245805.0,104614...	7.066370322830001
[1256293.0,956915...	6.7652905591999986
[1249158.0,124136...	7.6710458805
[1257339.0,132088...	7.54950688585
[1267476.0,136391...	7.64481448926
[1280762.0,139831...	7.796914139070001
[1292558.0,140014...	7.6925282202
[1294924.0,138966...	7.626856430539999
[1306995.0,139813...	7.697161857649999
[1350457.0,138591...	7.4447752306599995
[1479943.0,149908...	7.6621237544300005
[1535175.0,155653...	7.800259903390001
[1590094.0,161762...	8.07740664325
[1641232.0,165719...	7.546587059314116
[0.0,0.0,0.0,0.0,...	0.0

LCV training set:

features	Light commercial
[251143.0,96007.0...	1.8303728724811765
[254155.0,95590.0...	1.50467490711
[258659.0,96693.0...	1.55969757691
[274083.0,101725....	1.58226037566
[284545.0,103708....	1.6286443508600001
[293568.0,104168....	1.66393447456
[303971.0,105041....	1.71895367454
[312579.0,102839....	1.76446421715
[315772.0,99049.0...	1.7746020360100003
[321520.0,95312.0...	1.83028339465
[339965.0,89053.0...	1.87931042313
[408647.0,94661.0...	2.1218401623300003
[439961.0,99907.0...	2.2378018324899998
[476173.0,106367....	2.4722270316399997
[510666.0,111186....	1.8303728724811765
[0.0,0.0,0.0,0.0,...	0.0

LPV testing set:

features	Light passenger
[1245233.0,114948...	7.375727557739999
[1315826.0,138233...	7.575257564359999
[1389608.0,140493...	7.41518700798
[1434731.0,144919...	7.43075356143
[0.0,0.0,0.0,0.0,...	0.0

LCV testing set:

features	Light commercial
[265261.0,98774.0...	1.58502708961
[327738.0,91259.0...	1.86172944903
[358978.0,89550.0...	1.93747100946
[383012.0,91813.0...	1.99341682704
[0.0,0.0,0.0,0.0,...	0.0

Figure 45: training and testing sets for LPV and LCV

3. Build a proper Decision Tree Regression Model, set features vector and label vector.

```
In [17]: from pyspark.ml.regression import DecisionTreeRegressor
         from pyspark.ml.evaluation import RegressionEvaluator


         dt = DecisionTreeRegressor(featuresCol='features', labelCol='Light passenger')
         dt_model =dt.fit(train_data)
```

Figure 46: Build proper Decision Tree Model

4. Using Decision Tree Regression model on test data, check r2 value and prediction values.

LPV Decision Tree Regression Model Result:

```
predictions =dt_model.transform(LPV_test_data)
predictions.show()
evaluator = RegressionEvaluator(
    labelCol="Light passenger", predictionCol="prediction", metricName="r2")
r2 = evaluator.evaluate(predictions)
print("R2 on test data = %g" % r2)
```

```
+--------------------+-----------------+------------------+
|            features|  Light passenger|        prediction|
+--------------------+-----------------+------------------+
|[1245233.0,114948...|7.375727557739999|       7.6710458805|
|[1315826.0,138233...|7.575257564359999| 7.4447752306599995|
|[1389608.0,140493...|    7.41518700798| 7.7391149136349995|
|[1434731.0,144919...|    7.43075356143| 7.7391149136349995|
|[0.0,0.0,0.0,0.0,...|              0.0|               0.0|
+--------------------+-----------------+------------------+

R2 on test data = 0.99315
```

Figure 47: LPV Decision Tree Regressor Result

LCV Decision Tree Regression Model Result:

```
predictions =dt_model.transform(LCV_test_data)
predictions.show()
evaluator = RegressionEvaluator(
    labelCol="Light commercial", predictionCol="prediction", metricName="r2")
r2 = evaluator.evaluate(predictions)
print("R2 on test data = %g" % r2)
```

```
+--------------------+----------------+------------------+
|            features|Light commercial|        prediction|
+--------------------+----------------+------------------+
|[265261.0,98774.0...|   1.58502708961| 1.6249464003600005|
|[327738.0,91259.0...|   1.86172944903|      1.87931042313|
|[358978.0,89550.0...|   1.93747100946|      1.87931042313|
|[383012.0,91813.0...|   1.99341682704| 2.1218401623300007|
|[0.0,0.0,0.0,0.0,...|             0.0|               0.0|
+--------------------+----------------+------------------+

R2 on test data = 0.992277
```

Figure 48: LCV Decision Tree Regressor Result

The Decision Tree Regressor has R2 score 0.99315 and 0.992277 for LPV and LCV data set. However, the reason why **not using Tree Based algorithm** on this problem is the predictions where marked in figure 47 and 48 are same appearance in the training sets in figure 45 which means DT is finding the closest vectors in the training data then it is reasonable to have a higher R2 score.

Considering these data have no abilities to split into single group for the data mining success criteria 2, mentioned in step 1.3 and 5, **Hence we not using Tree model for this iteration, they are Decision Tree Regressor, Random forest Regressor and Gradient-boosted tree Regressor.**

Second Step: verify Linear Model fitting abilities.

1. Using same data sets in first step.

2. Build a proper Linear Model.

```
In [20]: from pyspark.ml.regression import LinearRegression

         lr = LinearRegression(labelCol='Light passenger', regParam=0.1, elasticNetParam=0, loss='squaredError')
         lrModel = lr.fit(LPV_train_data)
```

Figure 49: Build proper Linear Regression Model

3. Using Linear Regression model on test data, check r2 value and prediction values.

LPV Linear Regression Model Result:

```
predictions = lrModel.transform(LPV_test_data)
predictions.show()

test_results = lrModel.evaluate(LPV_test_data)
test_results.residuals.show()
print("R2: %.4f"%(float(test_results.r2)))
```

```
+--------------------+-----------------+--------------------+
|            features|  Light passenger|          prediction|
+--------------------+-----------------+--------------------+
|[1245233.0,114948...|7.375727557739999|  7.390609331995027|
|[1315826.0,138233...|7.575257564359999|  7.548119472356868|
|[1389608.0,140493...|    7.41518700798|  7.685914980139722|
|[1434731.0,144919...|    7.43075356143|  7.730085571701273|
|[0.0,0.0,0.0,0.0,...|              0.0|0.07990859376154384|
+--------------------+-----------------+--------------------+

+--------------------+
|           residuals|
+--------------------+
|-0.01488177425502...|
| 0.02713809200313122|
|-0.27072797215972155|
|-0.29933201027127243|
|-0.07990859376154384|
+--------------------+

R2: 0.9962
```

Figure 50: LPV Linear Regressor Result

LCV Linear Regression Model Result:

```
predictions = lrModel.transform(LCV_test_data)
predictions.show()

test_results = lrModel.evaluate(LCV_test_data)
test_results.residuals.show()
print("R2: %.4f"%(float(test_results.r2)))
```

```
+--------------------+---------------+--------------------+
|            features|Light commercial|          prediction|
+--------------------+---------------+--------------------+
|[265261.0,98774.0...|   1.58502708961|   1.621669267865094|
|[327738.0,91259.0...|   1.86172944903|   1.874454188756759|
|[358978.0,89550.0...|   1.93747100946|  2.0046767755355823|
|[383012.0,91813.0...|   1.99341682704|  2.0489793121000397|
|[0.0,0.0,0.0,0.0,...|             0.0| 0.060567040479487716|
+--------------------+---------------+--------------------+

+--------------------+
|           residuals|
+--------------------+
|-0.03664217825509386|
|-0.01272473972675...|
|-0.06720576607558226|
| -0.0555624850600398|
|-0.06056704047948...|
+--------------------+

R2: 0.9955
```

Figure 51: LCV Linear Regressor Result

The Linear Regressor has R2 score 0.9962 and 0.9955 for LPV and LCV data set as good as Decision Tree Regressor. We set elasticNetParam with 0 means the Linear Regressor is under l2 normalization which improve the model accuracy. All the predict results are different under a linear model which is count for a new value rather than fining a similar value. The reason about **choose Linear Regressor** is it sufficient for the datamining objectives (one, two), hence it will be select for the data mining.

6.3 Build Appropriate Models

After we decide using Linear Model for solver regression problem in this problem, the next step is building an appropriate model, the environments are same as step 6.2.

As we mentioned in Step 6.1, there are many types of linear regressions, Ordinary least squares Regression, Ridge Regression, Lasso Regression, and Elastic Net Regression. We mentioned in step 4.2, spark provide more powerful normalization which allowed you tune the lambda and omega for combining L2 and L1 in the equation to control the learning. Hence, the main focus on this step is tuning the parameters elasticNetParam, and regParam to control the lambda and omega in the normalization.

Ordinary least squares Regression:

The naïve Linear Regression which the input data without any normalization, thus it is the model does not set elasticNetParam, and regParam.

```
In [27]: from pyspark.ml.regression import LinearRegression

         lr = LinearRegression(labelCol='Light passenger',  loss='squaredError')
         lrModel = lr.fit(LPV_train_data)
```

Figure 52: Ordinary least squares Regression

LPV under Ordinary least squares Regression:

```
+--------------------+-----------------+--------------------+
|            features|  Light passenger|          prediction|
+--------------------+-----------------+--------------------+
|[1245233.0,114948...|7.375727557739999|   7.359201270924787|
|[1315826.0,138233...|7.575257564359999|   7.935712225078278|
|[1389608.0,140493...|    7.41518700798|   6.912512126551265|
|[1434731.0,144919...|    7.43075356143|    7.66513774981806|
|[0.0,0.0,0.0,0.0,...|              0.0|0.001807022968678...|
+--------------------+-----------------+--------------------+

+--------------------+
|           residuals|
+--------------------+
|0.016526286815212288|
|-0.36045466071827903|
|  0.5026748814287352|
| -0.2343841883880593|
|-0.00180702296867...|
+--------------------+

R2: 0.9901
```

Figure 53: LPV under Ordinary least squares Regression:

LCV under Ordinary least squares Regression:

```
+--------------------+---------------+--------------------+
|            features|Light commercial|          prediction|
+--------------------+---------------+--------------------+
|[265261.0,98774.0...|   1.58502708961|  1.1365920009369401|
|[327738.0,91259.0...|   1.86172944903|  2.1521935642266325|
|[358978.0,89550.0...|   1.93747100946|   1.915433810704013|
|[383012.0,91813.0...|   1.99341682704|   1.802621492752547|
|[0.0,0.0,0.0,0.0,...|             0.0|-4.77145464575325...|
+--------------------+---------------+--------------------+

+--------------------+
|           residuals|
+--------------------+
|  0.4484350886730599|
| -0.2904641151966325|
|0.022037198755987086|
| 0.19079533428745288|
|4.771454645753255...|
+--------------------+

R2: 0.8857
```

Figure 54: LCV under Ordinary least squares Regression:

Ridge Regression:

The naïve Linear Regression under :2 normalization which set elasticNetParam equal 0, and tuning regParam. The Step 6.2 set regParam equal 0.1, thus this time set it to 1.

```
In [ ]: from pyspark.ml.regression import LinearRegression

        lr = LinearRegression(labelCol='Light passenger', regParam=1, elasticNetParam=0, loss='squaredError')
        lrModel = lr.fit(LPV_train_data)
```

Figure 55: Ridge Regression with lambda equal 1.

LPV under Ridge Regression:

```
+--------------------+-----------------+------------------+
|            features|  Light passenger|        prediction|
+--------------------+-----------------+------------------+
|[1245233.0,114948...|7.375727557739999| 7.331374615479455|
|[1315826.0,138233...|7.575257564359999| 7.526846681079377|
|[1389608.0,140493...|    7.41518700798| 7.685400237983518|
|[1434731.0,144919...|    7.43075356143| 7.750477327339305|
|[0.0,0.0,0.0,0.0,...|              0.0|0.5663426091638106|
+--------------------+-----------------+------------------+

+--------------------+
|           residuals|
+--------------------+
|0.044352942260544204|
|0.048410883280622485|
| -0.2702132300035176|
| -0.3197237659093046|
| -0.5663426091638106|
+--------------------+

R2: 0.9887
```

Figure 56: LPV under Ridge Regression:

LCV under Ridge Regression:

```
+--------------------+----------------+--------------------+
|            features|Light commercial|        prediction|
+--------------------+----------------+--------------------+
|[265261.0,98774.0...|   1.58502708961| 1.6399321847896506|
|[327738.0,91259.0...|   1.86172944903| 1.7852033332049837|
|[358978.0,89550.0...|   1.93747100946|  1.857373946775586|
|[383012.0,91813.0...|   1.99341682704| 1.9080335596069655|
|[0.0,0.0,0.0,0.0,...|             0.0|0.32674998320264415|
+--------------------+----------------+--------------------+

+--------------------+
|           residuals|
+--------------------+
|-0.05490509517965059|
| 0.07652611582501634|
| 0.08009706268441397|
| 0.08538326743303437|
|-0.32674998320264415|
+--------------------+

R2: 0.9541
```

Figure 57: LCV under Ridge Regression:

Lasso Regression:

The naïve Linear Regression under L1 normalization which set elasticNetParam equal 1, and tuning regParam equal 0.1 to compare with Ridge Regressor in step 6.2.

```
In [ ]: from pyspark.ml.regression import LinearRegression

        lr = LinearRegression(labelCol='Light passenger', regParam=0.1, elasticNetParam=1, loss='squaredError')
        lrModel = lr.fit(LPV_train_data)
```

Figure 58: Lasso Regression with lambda equal 0.1.

LPV under Lasso Regression:

```
+--------------------+-----------------+------------------+
|            features|  Light passenger|        prediction|
+--------------------+-----------------+------------------+
|[1245233.0,114948...|7.375727557739999|  7.43831873644325|
|[1315826.0,138233...|7.575257564359999| 7.527188669861334|
|[1389608.0,140493...|    7.41518700798| 7.605208420052604|
|[1434731.0,144919...|    7.43075356143|  7.62939548651731|
|[0.0,0.0,0.0,0.0,...|              0.0|0.4470105066650957|
+--------------------+-----------------+------------------+

+--------------------+
|           residuals|
+--------------------+
| -0.0625911787032507|
|  0.04806889449866514|
|-0.19002141207260337|
|-0.19864192508730927|
| -0.4470105066650957|
+--------------------+

R2: 0.9937
```

Figure 56: LPV under Lasso Regression:

LCV under Lasso Regression:

```
+--------------------+---------------+--------------------+
|            features|Light commercial|          prediction|
+--------------------+---------------+--------------------+
|[265261.0,98774.0...|   1.58502708961| 1.6216692678650924|
|[327738.0,91259.0...|   1.86172944903| 1.8744541887567605|
|[358978.0,89550.0...|   1.93747100946| 2.0046767755355828|
|[383012.0,91813.0...|   1.99341682704| 2.0489793121000397|
|[0.0,0.0,0.0,0.0,...|             0.0|0.06056704047948878|
+--------------------+---------------+--------------------+

+--------------------+
|           residuals|
+--------------------+
|-0.03664217825509...|
|-0.01272473972676047|
| -0.0672057660755827|
| -0.0555624850600398|
|-0.06056704047948878|
+--------------------+

R2: 0.9955
```

Figure 59: LCV under Lasso Regression.

Elastic Net Regression:

The naïve Linear Regression combine use L2 and L1 normalization which set elasticNetParam equal 0.5, and tuning regParam equal 0.1 to compare with Ridge Regressor in step 6.2.

```
In [ ]: from pyspark.ml.regression import LinearRegression

        lr = LinearRegression(labelCol='Light passenger', regParam=0.1, elasticNetParam=0.5, loss='squaredError')
        lrModel = lr.fit(LPV_train_data)
```

Figure 60: ElasticNet Regression with lambda equal 0.1, L2 and L1 equal weight.

LPV under Elastic Net Regression:

```
+-------------------+----------------+------------------+
|           features|  Light passenger|        prediction|
+-------------------+----------------+------------------+
|[1245233.0,114948...|7.375727557739999|7.4267645253394825|
|[1315826.0,138233...|7.575257564359999| 7.555215091157053|
|[1389608.0,140493...|    7.41518700798|7.6663608730497375|
|[1434731.0,144919...|    7.43075356143| 7.672373277255685|
|[0.0,0.0,0.0,0.0,...|             0.0| 0.280230233535028|
+-------------------+----------------+------------------+

+--------------------+
|           residuals|
+--------------------+
|-0.05103696759948306|
|0.020042473202946276|
|-0.25117386506973727|
|-0.24161971582568498|
|  -0.280230233535028|
+--------------------+

R2: 0.9954
```

Figure 61: LPV under ElasticNet Regression:

LCV under Lasso Regression:

```
+-------------------+---------------+------------------+
|           features|Light commercial|        prediction|
+-------------------+---------------+------------------+
|[265261.0,98774.0...|   1.58502708961| 1.6388289885660017|
|[327738.0,91259.0...|   1.86172944903|   1.85731194465398|
|[358978.0,89550.0...|   1.93747100946| 1.9765938364617683|
|[383012.0,91813.0...|   1.99341682704|  2.016404858833665|
|[0.0,0.0,0.0,0.0,...|             0.0|0.22973527666239904|
+-------------------+---------------+------------------+

+--------------------+
|           residuals|
+--------------------+
|-0.05380189895600...|
|0.004417504376019954|
|-0.03912282700176828|
|-0.02298803179366...|
|-0.22973527666239904|
+--------------------+

R2: 0.9795
```

Figure 62: LPV under ElasticNet Regression:

The result shows that under same environment all the regressor performance similar, and **Ridge Regressor** performance best according to these four regressors. After increase lambda to 1 then the r2 score decreased. The next procedures will tune the lambda (regParam).

Here, we set 5 different lambda values, the lower bound value is zero which means no L2 normalization here, and upper bound is one means the fastest normalization. Since we have got a good result at lambda equal 0.1, and we know the upper bound is one. Next is test values between this range and less than 0.1. The 3 different untested values are 0.01, 0.35, and 0.5.

```
In [20]: regParam_list = [0.01, 0.1, 0.35, 0.5, 1]

         for i in regParam_list:
             _,_,LPV_r2 = ridge_regressor(LPV_train_data, LPV_test_data,'Light passenger',i)
             _,_,LCV_r2 = ridge_regressor(LCV_train_data, LCV_test_data,'Light commercial',i)

             print('regparam = %f, LPV r2 = %.4f, LCV r2 = %.4f'%(i, LPV_r2, LCV_r2) )

regparam = 0.010000, LPV r2 = 0.9956, LCV r2 = 0.9916
regparam = 0.100000, LPV r2 = 0.9962, LCV r2 = 0.9955
regparam = 0.350000, LPV r2 = 0.9949, LCV r2 = 0.9910
regparam = 0.500000, LPV r2 = 0.9938, LCV r2 = 0.9839
regparam = 1.000000, LPV r2 = 0.9887, LCV r2 = 0.9541
```

Figure 62: Test on Lambda.

As result shown the regParam equal to 0.1 still be the best result for both LPV and LCV dataset. Decreasing the regParam cannot guarantee a better result than 0.1, and the training will become really slow. Increasing the regParam will fast the training and reduce the r2 score. To sum up of step 6, choosing **ridge regressor** for the data mining model, which means the **linear regressor with regParam = 0.1 and elasticNet = 0**, it satisfies all the data mining goals.

7 Datamining

7.1 Create and Justify Test Design

For Datamining Objective One:

Splitting data set to 75% as a training set and 25% as a testing set to avoid overfitting and underfitting the model. More the training data will give a more accuracy result, we use 70% data to select algorithms then need more data on the datamining. After splitting the training set and testing set, adding noisy data into these files respectively, for avoid overfitting and also make the model learn that no vehicles on road indicate to no CO2 emissions. In figure 63, yellow highlight shows the splitting method, and red highlight show the adding noisy.

```
In [ ]: def step_7_1_assemble_dataset(df, seed):
            spark_df = spark.createDataFrame(df)
            assembler = VectorAssembler(inputCols= spark_df.columns[:-1],outputCol="features")
            spark_assembler_df = assembler.transform(spark_df)

            selected_data = spark_assembler_df.select('features',spark_df.columns[-1])
            train_data,test_data = selected_data.randomSplit([0.75,0.25],seed=seed)

            noisy_df = spark.createDataFrame([[Vectors.dense(np.zeros(13)),0.0]])
            train_data,test_data = train_data.union(noisy_df),test_data.union(noisy_df)

            return train_data,test_data
```

Figure 63: assemble dataset for datamining.

LPV training set:

features	Light passenger
[1276498.0,870680...	7.546587059314116
[1245805.0,104614...	7.066370322830001
[1256293.0,956915...	6.7652905591999986
[1249158.0,124136...	7.6710458805
[1257339.0,132088...	7.54950688585
[1267476.0,136391...	7.64481448926
[1280762.0,139831...	7.796914139070001
[1292558.0,140014...	7.6925282202
[1294924.0,138966...	7.626856430539999
[1306995.0,139813...	7.697161857649999
[1350457.0,138591...	7.4447752306599995
[1389608.0,140493...	7.41518700798
[1434731.0,144919...	7.43075356143
[1479943.0,149908...	7.6621237544300005
[1535175.0,155653...	7.800259903390001
[1590094.0,161762...	8.07740664325
[1641232.0,165719...	7.546587059314116
[0.0,0.0,0.0,0.0,...	0.0

LCV training set:

features	Light commercial
[251143.0,96007.0...	1.8303728724811765
[254155.0,95590.0...	1.50467490711
[258659.0,96693.0...	1.55969757691
[274083.0,101725....	1.58226037566
[284545.0,103708....	1.6286443508600001
[293568.0,104168....	1.66393447456
[303971.0,105041....	1.71895367454
[312579.0,102839....	1.76446421715
[315772.0,99049.0...	1.7746020360100003
[321520.0,95312.0...	1.83028339465
[339965.0,89053.0...	1.87931042313
[358978.0,89550.0...	1.93747100946
[383012.0,91813.0...	1.99341682704
[408647.0,94661.0...	2.1218401623300003
[439961.0,99907.0...	2.2378018324899998
[476173.0,106367....	2.4722270316399997
[510666.0,111186....	1.8303728724811765
[0.0,0.0,0.0,0.0,...	0.0

LPV testing set:

features	Light passenger
[1245233.0,114948...	7.375727557739999
[1315826.0,138233...	7.575257564359999
[0.0,0.0,0.0,0.0,...	0.0

LCV testing set:

features	Light commercial
[265261.0,98774.0...	1.58502708961
[327738.0,91259.0...	1.86172944903
[0.0,0.0,0.0,0.0,...	0.0

Figure 64: assembled training and testing dataset for datamining.

For Datamining Objective Two:

Prepare splitting age groups set base on the data in 2017, One reason is the real emission value for 2018 is missing which we imputed in Data Pre-processing. Comparing using mean values, data in 2017 are more fit current situation. We create files with 10 instances of 2017 and trim it only contains one group data.

```
In [20]: split_2017_LPV_rows = [[1.59009400e+06, 1.61762700e+06, 1.43115413e+01, 2.84522000e+05, 2.54697000e+05,
                                 4.53646000e+05, 2.51078000e+05, 3.46148000e+05, 2.89449000e+05, 2.59108000e+05,
                                 4.61501000e+05, 2.55426000e+05, 3.52141000e+05],
                                [284522.0, 0.0, 2.975101564, 284522.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                                [254697.0, 0.0, 7.975101564, 0.0, 254697.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                                [453646.0, 0.0, 12.97510156, 0.0, 0.0, 453646.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                                [251078.0, 0.0, 17.97510156, 0.0, 0.0, 0.0, 251078.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                                [346148.0, 0.0, 22.97510156, 0.0, 0.0, 0.0, 0.0, 346148.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                                [0.0, 289449.0, 9.130000000, 0.0, 0.0, 0.0, 0.0, 0.0, 289449.0, 0.0, 0.0, 0.0, 0.0],
                                [0.0, 259108.0, 14.13000000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 259108.0, 0.0, 0.0, 0.0],
                                [0.0, 461501.0, 19.13000000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 461501.0, 0.0, 0.0],
                                [0.0, 255426.0, 24.13000000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 255426.0, 0.0],
                                [0.0, 352141.0, 29.13000000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 352141.0], ]

         split_2017_LCV_rows = [[4.76173000e+05, 1.06367000e+05, 1.25145441e+01, 8.52030000e+04, 7.62720000e+04,
                                 1.35849000e+05, 7.51880000e+04, 1.03658000e+05, 1.90320000e+04, 1.70370000e+04,
                                 3.03460000e+04, 1.67950000e+04, 2.31550000e+04],
                                [085203.0, 0.0, 2.868356783, 085203.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                                [076272.0, 0.0, 7.868356783, 0.0, 076272.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                                [135849.0, 0.0, 12.86835678, 0.0, 0.0, 135849.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                                [075188.0, 0.0, 17.86835678, 0.0, 0.0, 0.0, 075188.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                                [103658.0, 0.0, 22.86835678, 0.0, 0.0, 0.0, 0.0, 103658.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                                [0.0, 019032.0, 07.98000000, 0.0, 0.0, 0.0, 0.0, 0.0, 019032.0, 0.0, 0.0, 0.0, 0.0],
                                [0.0, 017037.0, 12.98000000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 017037.0, 0.0, 0.0, 0.0],
                                [0.0, 030346.0, 17.98000000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 030346.0, 0.0, 0.0],
                                [0.0, 016795.0, 22.98000000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 016795.0, 0.0],
                                [0.0, 023155.0, 27.98000000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 023155.0], ]
```

```
In [21]: LPV_2017_df = spark.createDataFrame([ [Vectors.dense(i)] for i in split_2017_LPV_rows ],["features"])
         LCV_2017_df = spark.createDataFrame([ [Vectors.dense(i)] for i in split_2017_LCV_rows ],["features"])

         print('LPV 2017 obj 2 splitted data')
         LPV_2017_df.show()
         print('LCV 2017 obj 2 splitted data')
         LCV_2017_df.show()
```

Figure 65: prepare dataset for datamining objective two.

The first rows will be the 2017 data; thus, we can compare with each group predict values against the whole groups, and we can also compare with the whole groups predict values with real values, these are mentions in step 1.3.1.2 benchmark.

Preparing similar 2017 data set form LPV and LCV and convert it to **Spark Data Frame** and assemble to one vector for testing the Objective 2.

```
LPV 2017 obj 2 splitted data
+--------------------+
|            features|
+--------------------+
|[1590094.0,161762...|
|[284522.0,0.0,2.9...|
|[254697.0,0.0,7.9...|
|[453646.0,0.0,12....|
|[251078.0,0.0,17....|
|[346148.0,0.0,22....|
|[0.0,289449.0,9.1...|
|[0.0,259108.0,14....|
|[0.0,461501.0,19....|
|[0.0,255426.0,24....|
|[0.0,352141.0,29....|
+--------------------+
```

```
LCV 2017 obj 2 splitted data
+--------------------+
|            features|
+--------------------+
|[476173.0,106367....|
|[85203.0,0.0,2.86...|
|[76272.0,0.0,7.86...|
|[135849.0,0.0,12....|
|[75188.0,0.0,17.8...|
|[103658.0,0.0,22....|
|[0.0,19032.0,7.98...|
|[0.0,17037.0,12.9...|
|[0.0,30346.0,17.9...|
|[0.0,16795.0,22.9...|
|[0.0,23155.0,27.9...|
+--------------------+
```

Figure 66: assembled dataset for datamining objective two.

7.2 Conduct Datamining

Run the pipeline for LPV and LCV dataset, setting Random Seed with 722, split the training set and test set with 75% and 25% respectively, which visualized in figure 64 at step 7.1

7.2.1 Conduct Datamining for Objective One:

Dataset select and split from the step 1-5 above, convert from original data to spark dataframe. Spark Dataframe need to assemble all features columns into one column and remain the label column. In figure 68 show the vector and the label column.

```
In [37]: # ---- step 7.2 objective 1 ----
         LPV_train_data.printSchema()
         LPV_test_data.printSchema()
         LCV_train_data.printSchema()
         LCV_test_data.printSchema()

         root
          |-- features: vector (nullable = true)
          |-- Light passenger: double (nullable = true)

         root
          |-- features: vector (nullable = true)
          |-- Light passenger: double (nullable = true)

         root
          |-- features: vector (nullable = true)
          |-- Light commercial: double (nullable = true)

         root
          |-- features: vector (nullable = true)
          |-- Light commercial: double (nullable = true)
```

Figure 67: assemble data for conduct datamining.

Construct ridge regressor with squared error loss function, set features column is the combined vector and the label column is the specific column for LPV and LCV. Reuse the function from step 6 by set the regParam equal to 0.1 for our selected algorithm. Verification of fulfil of Data Mining Objective One with the test set, which is the predict values do not have a large gap comparing real value (Step1.3), and one measurement of this situation introduced in Step 6 is R2 score.

```
In [ ]: def ridge_regressor(train_data, test_data, labelcol, reg_param):
            lr = LinearRegression(labelCol=labelcol, regParam=reg_param, elasticNetParam=0, loss='squaredError')
            lrModel = lr.fit(train_data)

            predictions = lrModel.transform(test_data)
            test_results = lrModel.evaluate(test_data)

            return predictions, test_results.residuals, test_results.r2
```

```
In [22]: # ---- step 7.2 objective 1 ----
         LPV_predictions, LPV_residuals, LPV_r2 = ridge_regressor(LPV_train_data, LPV_test_data,'Light passenger', 0.1)
         LCV_predictions, LCV_residuals, LCV_r2 = ridge_regressor(LCV_train_data, LCV_test_data,'Light commercial', 0.1)
```

```
In [32]: print('Data Mining Objective One')
         print('LPV R2 score = %f, LCV R2 score = %f'%(LPV_r2, LCV_r2))

Data Mining Objective One
LPV R2 score = 0.999629, LCV R2 score = 0.998748
```

Figure 68: conduct datamining for datamining objective one.

7.2.2 Conduct Datamining for Objective Two:

For training the model, the setting and the training set same with step 7.2.1 from the objective one. For test set which replace with datasets designed and built from step 7.1 for datamining objective two, assemble the features into one vector as same as test set in step 7.2.1. Using regression models for predict single age group values is test the fitness of Data Mining Objective Two.

```
In [38]: #  ---- step 7.2 objective 2 ----
         LPV_2017_df.printSchema()
         LCV_2017_df.printSchema()

root
 |-- features: vector (nullable = true)

root
 |-- features: vector (nullable = true)
```

Figure 69: assemble data for conduct datamining.

The details are the sum of the single group values should not have a large residual with real values.

```
In [39]: print('LPV 2017 real emission = %f, sum of single group prediction in 2017 = %f'%(LPV_2017_real, LPV_2017_predict) )
         print('LCV 2017 real emission = %f, sum of single group prediction in 2017 = %f'%(LCV_2017_real, LCV_2017_predict) )

LPV 2017 real emission = 7.786092, sum of single group prediction in 2017 = 18.562246
LCV 2017 real emission = 2.161407, sum of single group prediction in 2017 = 5.432101
```

Figure 70: conduct datamining for datamining objective two.

Step sum up, the model we selected, and constructed with parameters tuned in step 6 is **fulfil for our datamining objective One**, results shown on figure 68, all the datasets have over 0.99 r2 score in range [0,1] that indicates our predict values have a tiny gap to real values.

The model **needs more discover** for the **Objective Two results** shown on figure 70, because as our project assumption is the sum of single groups value has a slight gap between LPV real emission value in 2017 and a small gap with LCV 2017 emission value. However, it has a large gap, we need to discover two things in next steps.

1. Is that the project assumption is right? We need to discover with the visualization and search for more patterns, in next step 7.3 to step 8.4.

2. Is the PySpark model problem? We have proved in iteration 2 and 3, both SPSS and Scikit-Learn able to satisfy this assumption. we need to re-run the iteration, for verify is this cause by PySpark or model parameters selection, in step 8.5.

7.3 Search for patterns

For discovering patterns, we need to get all the predict values for the Objective One output and Two.

For the Objective One, we output the predict result of test set and compare with real values count as residuals.

```
In [43]: print('LPV objective one results')
         LPV_predictions.show()
         LPV_residuals.show()
         print('LCV objective one results')
         LCV_predictions.show()
         LCV_residuals.show()
```

Figure 71: Code for patterns find from objective one test results.

```
Data Mining Objective One
LPV R2 score = 0.998174, LCV R2 score = 0.998793
```

Figure 72: Model accuracy.

LPV objective one results

features	Light passenger	prediction
[1245233.0,114948...	7.375727557739999	7.38886035344829
[1315826.0,138233...	7.575257564359999	7.5447334786043045
[1389608.0,140493...	7.41518700798	7.688787765854149
[1434731.0,144919...	7.43075356143	7.736590929673956
[0.0,0.0,0.0,0.0,...	0.0	0.03749330012153006
[0.0,0.0,0.0,0.0,...	0.0	0.03749330012153006
[0.0,0.0,0.0,0.0,...	0.0	0.03749330012153006

residuals
-0.01313279570829...
0.030524085755694763
-0.2736007578741484
-0.3058373682439557
-0.03749330012153006
-0.03749330012153006
-0.03749330012153006

Figure 73: Model LPV residuals.

LCV objective one results

features	Light commercial	prediction
[265261.0,98774.0...	1.58502708961	1.6178263118146199
[327738.0,91259.0...	1.86172944903	1.8594483389471486
[358978.0,89550.0...	1.93747100946	1.9836393098022254
[383012.0,91813.0...	1.99341682704	2.030001513394828
[0.0,0.0,0.0,0.0,...	0.0	0.029488317968545348
[0.0,0.0,0.0,0.0,...	0.0	0.029488317968545348
[0.0,0.0,0.0,0.0,...	0.0	0.029488317968545348

residuals
-0.03279922220461984
0.002281110082851...
-0.0461683003422253
-0.03658468635482803
-0.02948831796854...
-0.02948831796854...
-0.02948831796854...

Figure 74: Model LCV residuals.

For the Objective Two, we output the predict result of the 2017 test set we designed in Step 7.1.

```
print('LPV 2017 single group predictions')
for idx,i in enumerate(LPV_2017_predvalues[1:]):
    print('%s : %f ,contributed:  %f %%' %(columns[idx], i, i/LPV_2017_predict *100) )
```

```
LPV 2017 single group predictions
0-4 years of LPV new : 1.473137 ,contributed:  8.053784 %
5-9 years of LPV new : 1.863923 ,contributed:  10.190249 %
10-14 years of LPV new : 2.321547 ,contributed:  12.692122 %
15-19 years of LPV new : 2.300353 ,contributed:  12.576253 %
20+ years of LPV new : 1.429951 ,contributed:  7.817680 %
0-4 years of LPV used : 1.206645 ,contributed:  6.596845 %
5-9 years of LPV used : 1.720232 ,contributed:  9.404677 %
10-14 years of LPV used : 2.106328 ,contributed:  11.515502 %
15-19 years of LPV used : 2.273164 ,contributed:  12.427608 %
20+ years of LPV used : 1.595962 ,contributed:  8.725280 %
```

Figure 75: LPV single grouples pattern.

```
print('LCV 2017 single group predictions')
for idx,i in enumerate(LCV_2017_predvalues[1:]):
    print('%s : %f ,contributed:  %f %%' %(columns[idx], i,i/LCV_2017_predict * 100) )
```

```
LCV 2017 single group predictions
0-4 years of LCV new : 0.249914 ,contributed:  4.612629 %
5-9 years of LCV new : 0.477053 ,contributed:  8.804919 %
10-14 years of LCV new : 0.357675 ,contributed:  6.601561 %
15-19 years of LCV new : 0.701987 ,contributed:  12.956495 %
20+ years of LCV new : 0.733979 ,contributed:  13.546962 %
0-4 years of LCV used : 0.365656 ,contributed:  6.748880 %
5-9 years of LCV used : 0.456571 ,contributed:  8.426886 %
10-14 years of LCV used : 0.457761 ,contributed:  8.448852 %
15-19 years of LCV used : 0.719705 ,contributed:  13.283511 %
20+ years of LCV used : 0.897730 ,contributed:  16.569303 %
```

Figure 76: LCV single grouples pattern.

The relaitionships that we discover for datamining are:

1. Figures 72 to 74 show the model are accuracy enough to predict CO2 emission by the fleet's information, proved by all residuals are smaller.
2. Figure 75 compare with 76, new import fleets have more CO2 emissions than Used import fleets
3. Figure 75 in LPV categories, 10-19 age groups contribute the most CO2 emissions among new import groups or used groups.
4. Figure 76 in LCV categories, 15 – 20+ age groups contribute the most CO2 emissions among new import groups or used groups.

8 Result Analysis

8.1 Study and discuss the mined patterns

Re-call the LPV and LCV 2017 input vectors to analysis the mined patterns:

```
for idx,i in enumerate(reduced_LPV_cols[:-1]):
    print('%s : %f' % (i, split_2017_LPV_rows[0][idx]) )
```

```
Total LPV new : 1590094.000000
 Total LPV used : 1617627.000000
Light passenger average age : 14.311541
0-4 years of LPV new : 284522.000000
5-9 years of LPV new : 254697.000000
10-14 years of LPV new : 453646.000000
15-19 years of LPV new : 251078.000000
20+ years of LPV new : 346148.000000
0-4 years of LPV used : 289449.000000
5-9 years of LPV used : 259108.000000
10-14 years of LPV used : 461501.000000
15-19 years of LPV used : 255426.000000
20+ years of LPV used : 352141.000000
```

Figure 77: LPV 2017 input vector.

```
for idx,i in enumerate(reduced_LCV_cols[:-1]):
    print('%s : %f' % (i, split_2017_LCV_rows[0][idx]) )
```

```
Total LCV new : 476173.000000
 Total LCV used : 106367.000000
Light commercial average age : 12.514544
0-4 years of LCV new : 85203.000000
5-9 years of LCV new : 76272.000000
10-14 years of LCV new : 135849.000000
15-19 years of LCV new : 75188.000000
20+ years of LCV new : 103658.000000
0-4 years of LCV used : 19032.000000
5-9 years of LCV used : 17037.000000
10-14 years of LCV used : 30346.000000
15-19 years of LCV used : 16795.000000
20+ years of LCV used : 23155.000000
```

Figure 78: LCV 2017 input vector.

The Datamining goal is a regression problem, the first needed is the model should be accuracy and the mined **pattern 1** proved that the residuals are quite small in range [0.02 to

0.3] compare with target values range [6.90 to 8.0]. Under an accuracy model then we can discuss the next project needed which is finding CO2 distribution of age groups.

The Pattern 2, 3 and 4 results caused by the fleets numbers and age trend, **it means the models explained age trend in correct way.** Becase in the pattern 4, 20+ year groups have samiliar number of fleets, but the used fleets contribute more CO2 emission, it is same as common sense that is 20+ year is counting since they come into New Zealand. They are import as used vehicles that they must have higher used age than new import group. The pattern 2 caused by the new import fleets group size are bigger than used import fleets. The pattern 3 also proved the result follow the normal distribution, the median position groups have the biggest proportion. For example, in figure77 and figure 75, there are 453646 vehicles and 461601 vehicles for the median position group (10-14 age group) and they contribute the most CO2 emission (12.69, 11.51) .

Although, the patterns illustrated the model mined what the project needed. it does not satisfy one assumption which is the sum of the single groups predicts should be similar with the total group predict, where we mentioned in step 7.2. Hence, the data mining objective two is **not satisfiable**.

However, we are not able to conclude this assumption is right, because the percentage of single age groups seems reasonable that we do not know is this benchmark necessary for this project or this is caused by PySpark problem. For Linear model will have an offset item, it will simultaneously increase or decrease on all the predict. **If all the single group value decreases a constant variable then the percentage of single groups occupation won't be change and this benchmark will be fit, thus we need to rerun the iteration to discuss the benchmark necessary.** We will rerun this iteration to find is that caused by spark problem or assumption problem in step 8.5.

8.2 Visualize the data, results, models and patterns

For the Datamining Objective One, reflect to the pattern 1, figure73 and 74.

To visualized the regression models, predict values against real values. Y-axis for the predict values and X-axis for the real values. A diagonal line corss the origin, Scatters aways from the line means the residuals, if all the points on the lines means no residuals within the Y and X.

Regression Line Charts for LPV (predict values against real values):

```
In [71]: sns.regplot(x="real_value", y="pred_value", data=LPV_test_real_pred_df)
Out[71]: <AxesSubplot:xlabel='real_value', ylabel='pred_value'>
```

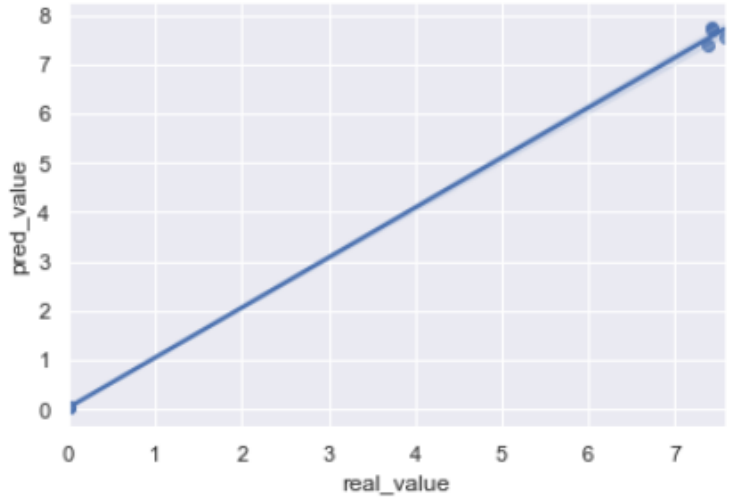


Figure 79: LPV regression model visualization.

Regression Line Charts for LCV (predict values against real values):

```
In [72]: sns.regplot(x="real_value", y="pred_value", data=LCV_test_real_pred_df)
Out[72]: <AxesSubplot:xlabel='real_value', ylabel='pred_value'>
```

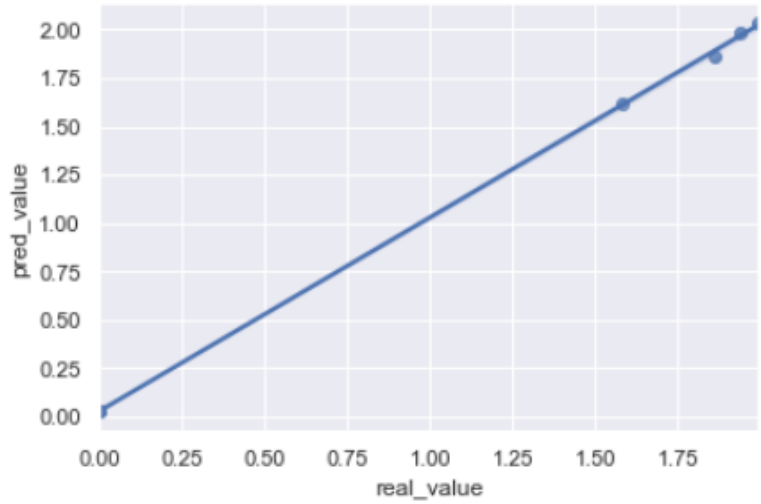


Figure 80: LCV regression model visualization.

Residuals Distribution Charts for LPV:

```
In [68]: sns.residplot(x="real_value", y="pred_value", data=LPV_test_real_pred_df, scatter_kws={"s": 80})
Out[68]: <AxesSubplot:xlabel='real_value', ylabel='pred_value'>
```

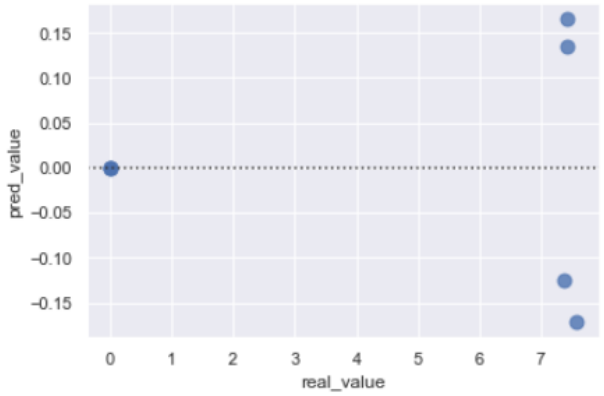


Figure 81: LPV residuals visualization.

Residuals Distribution Charts for LCV:

```
In [73]: sns.residplot(x="real_value", y="pred_value", data=LCV_test_real_pred_df, scatter_kws={"s": 80})
Out[73]: <AxesSubplot:xlabel='real_value', ylabel='pred_value'>
```

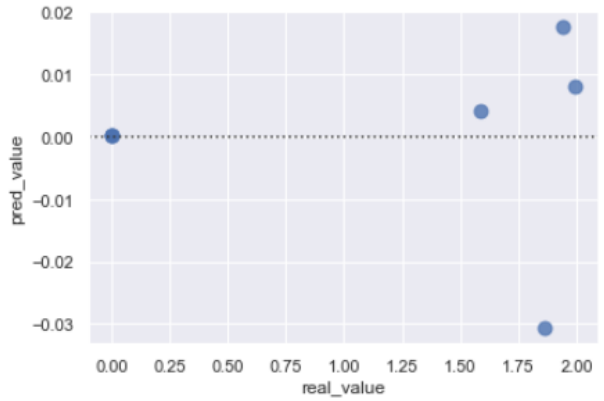


Figure 82: LCV residuals visualization.

For the Datamining Objective Two, reflect to the pattern 2 to 4, figure75 and 76.

To visualized the Percentage Occupation among the age groups. We only know the results in the numeric aspect, however we do not have the sense about proportion be occupied, Pie Chat would help us on this aspect.

Pie Chart for LPV Percentage:

```
fig = plt.figure(figsize=(9, 5.0625))
ax1 = fig.add_subplot(121)

ratios = LPV_percentage_list
labels = LPV_columns
# rotate so that first wedge is split by the x-axis
angle = -180 * ratios[0]
ax1.pie(ratios, autopct='%1.1f%%', startangle=angle, labels=labels, )
plt.show()
```

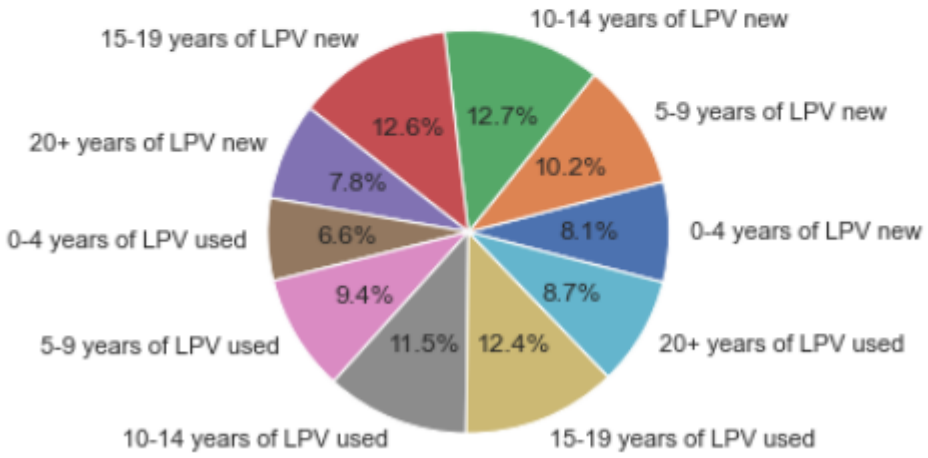


Figure 83: LPV percentage visualization.

Pie Chart for LCV Percentage:

```
fig = plt.figure(figsize=(9, 5.0625))
ax1 = fig.add_subplot(121)

ratios = LCV_percentage_list
labels = LCV_columns
# rotate so that first wedge is split by the x-axis
angle = -180 * ratios[0]
ax1.pie(ratios, autopct='%1.1f%%', startangle=angle, labels=labels, )
plt.show()
```

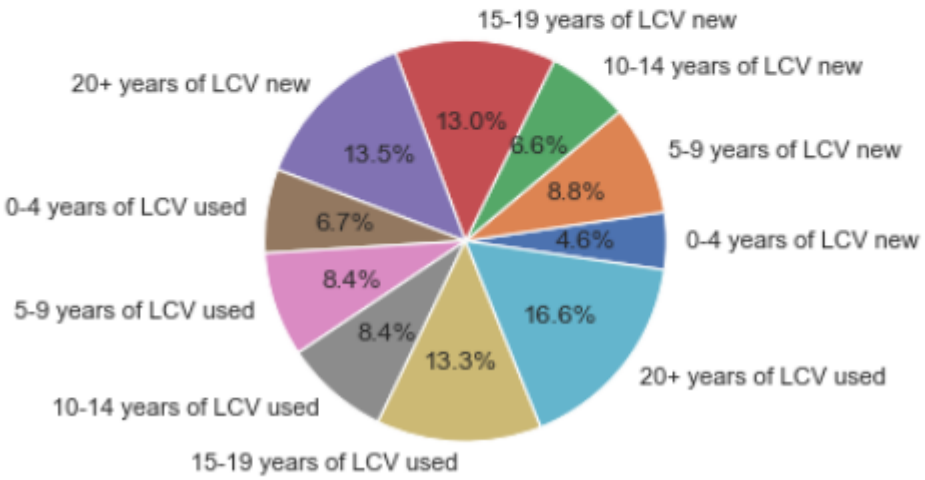


Figure 84: LCV percentage visualization.

8.3 Interpret the results, models and patterns

In this step we interpret the results models and pattern in business aspect.

8.3.1 Interpretation of regression model, indicate Pattern One:

Visualization interpretation: The regression figures 79 and 80, the solid diagonal lines are for the fitting regression lines. it shows LCV model have better result than LPV model because LCV scatters are closer the line and LPV scatters seem a small away from line.

The residual figures 80 and 81 could find test data distributed on both side of distribution graph, fit for the normal distribution performance. It shows that there are no over fitting and under fitting problem because LPV have r2 score 0.9982 test sets and 0.9987 for LCV All the accuracy is exceeded 80%.

- According to the models, these models could apply for the subjective where we mentioned in step 1.3 for predict CO2 emissions in the future, because it could predict emission values in a reasonable range. For the Decision maker of New Zealand, they are able to control the number of fleets registrate or work off from road then they can base on these data to estimate trend of CO2 emission.

8.3.2 Interpretation of single groups predictions, indicate Pattern Two to Four:

Visualization interpretation: It has shown that the percentage of ages could be divided from whole groups, from Figure 83, and 84. There percentage are giving the common sense for the real business decision making, we will assess these percentage in real business aspect in step8.4 for decision making for the project business objectives.

- The business objective is for find the different age group of fleets take the occupation of the CO2 emission, and notice that the assumption of this project in Step 1.2 mentioned if the fleets with age greater or equal to 10 take 70% of CO2 Emission then the NZ Transport needs to consider apply extra Greenhouse Exhaust Test. The fleets with over 10 years are 5 - 20+ groups in used import vehicles and 10 – 20+ groups in new import vehicles.
- Counting the occupation:

Compute the weight average of occupation among LPV and LCV:

$$\left(\frac{\#\,LPV}{\#\text{LPV}+\#\text{LCV}} * 65.785\% + \frac{\#\,LCV}{\#\text{LPV}+\#\text{LCV}} * 71.406\%\right) * 100\% = \left(\frac{3207721}{3207721+582540} * 65.785\% + \frac{582540}{3207721+582540} * 71.406\%\right) * 100\% = 66.64893419\%$$

In conclusion, rely on the project assumption and the weight average of the occupation of CO2 emissions reflect that it is **NOT** necessary to suggest the NZ Transport apply extra Greenhouse Exhaust Test.

8.4 Assess and evaluate results, models and patterns

8.4.1 Assessment of the Prediction:

After above interpretation, all of our prediction is basing on vehicle's **age**, and **number of vehicles**. Compare with same size of groups, Higher age groups will contribute more CO2 emission than Lower age groups. People always have common sense of this situation. We compare with 0-4 average groups with 5-9 average groups where 0-4 groups contribute significantly smaller than 5-9 groups. This could prove **Harder Emission Standard** reduce the CO2 emission in other side. This model able to help New Zealand Transport to estimate future emission combine with Emission Standard to make decision about New Zealand Fleets control.

8.4.2 Assessment of the Percentage:

Follow the results percentage, we can find in the pie char graph that used vehicle used take near over 50% CO2 Emission Proportion in Commercial vehicles, However the used vehicles

are just nearly 30% of the new vehicles. Connect to the real-world situation, government are more encouraging reducing CO2 emission and the taxies you can see on the road are always hybrid engine system which give some evidence that hybrid or electrical vehicles will significantly improve the CO2 emission. New Zealand Transport could use this to enlarge investment on replacing old commercial vehicles to more eco selections.

8.5 Multiple Iteration

The objective of multiple iteration is for verify is that problem in step 7.2.2 caused by PySpark or the assumption is not suitable.

Hence, we first test the pipeline robust:

We test the robust through change the random seed, while the random seed has changed then the training and test data set will be different, because they splitting with different random seed.

After a different iteration of different seed then the model should reflect a stable result.

Second, we test on a different linear model to get a better result:

We replace the liner model with l2 normalization to the **Isotonic regression** model. Others provided by PySpark like generalized linear regression is very similar with the current linear model.

By the last iterations, we know that **SPSS** and **Scikit-Learn** able to **fulfil** the assumption, and to verify the spark problem. We will focus on the R2 score for test satisfy the Datamining Objective One, and the single groups against total groups real value for the Datamining Objective Two.

If after the two changes above able to satisfy the Datamining Objective Two, then we continue focus on the new results, otherwise remain the current results.

8.5.1 Test the Pipeline Robust:

Set random seed to 123 for run the pipeline again, where we just need to change the hyper parameter at project pipeline class front.

```
In [1]: import os
        import random
        import import_ipynb

        from BDAS_help import *

        # pd.set_option('display.precision', 12)

        # SEED = 722
        SEED = 123
        random.seed(SEED)
        np.random.seed(SEED)

        importing Jupyter notebook from BDAS_help.ipynb
```

Figure 85: Set different random seed.

For Data Mining Objective One:

R2 score:

```
In [26]: print('Data Mining Objective One')
         print('LPV R2 score = %f, LCV R2 score = %f'%(LPV_r2, LCV_r2))

         Data Mining Objective One
         LPV R2 score = 0.994009, LCV R2 score = 0.912100
```

Figure 86: random seed 123 r2 score.

The all training set and test set of LPV or LCV are different under seed 123 comparing seed 722. The R2 scores for all set of LPV or LCV are over 0.9.

For Data Mining Objective Two:

```
In [29]: #  ---- step 7.2 objective 2 ----
         LPV_2017_predictions, LPV_2017_predvalues, LPV_2017_real, LPV_2017_predict = step_7_2_objective2_model(LPV_model,  LPV_2017_df)
         LCV_2017_predictions, LCV_2017_predvalues, LCV_2017_real, LCV_2017_predict = step_7_2_objective2_model(LCV_model, LCV_2017_df)

In [30]: print('LPV 2017 real emission = %f, sum of single group prediction in 2017 = %f'%(LPV_2017_real, LPV_2017_predict) )
         print('LCV 2017 real emission = %f, sum of single group prediction in 2017 = %f'%(LCV_2017_real, LCV_2017_predict) )

         LPV 2017 real emission = 8.053384, sum of single group prediction in 2017 = 18.626556
         LCV 2017 real emission = 2.402113, sum of single group prediction in 2017 = 4.550043
```

Figure 87: random seed 123 for sum of single groups.

The results show that the sum of single groups against the whole groups real values have small residuals that indicate for **NOT** satisfy data mining objective two. It could prove the pipeline has a good robustly and stability for predict whole populations.

8.5.2 Get a better result:

Set random seed back to 722 and set IsotonicRegression:

```
In [ ]: def ridge_regressor(train_data, test_data, labelcol, reg_param):
#     lr = LinearRegression(labelCol=labelcol, regParam=reg_param, elasticNetParam=0, loss='squaredError')
        lr = IsotonicRegression(labelCol=labelcol)

        lrModel = lr.fit(train_data)

        predictions = lrModel.transform(test_data)
#     test_results = lrModel.evaluate(test_data)

        evaluator = RegressionEvaluator(labelCol=labelcol, predictionCol="prediction", metricName="r2")
        r2 = evaluator.evaluate(predictions)

#     return predictions, test_results.residuals, test_results.r2, lrModel
        return predictions, predictions.select('prediction'), r2,lrModel
```

Figure 88: random seed 722, isotonic regression.

For Data Mining Objective One:

R2 score:

```
In [26]: print('Data Mining Objective One')
         print('LPV R2 score = %f, LCV R2 score = %f'%(LPV_r2, LCV_r2))

         Data Mining Objective One
         LPV R2 score = 0.997763, LCV R2 score = 0.999509
```

Figure 89: random seed 722, isotonic regression r2 score.

We can find that using saga solver will still satisfy the data mining objective. The R2 score increase a little to 0.99763, 0.999509.

For Data Mining Objective Two:

```
In [30]: print('LPV 2017 real emission = %f, sum of single group prediction in 2017 = %f'%(LPV_2017_real, LPV_2017_predict) )
         print('LCV 2017 real emission = %f, sum of single group prediction in 2017 = %f'%(LCV_2017_real, LCV_2017_predict) )

         LPV 2017 real emission = 7.811997, sum of single group prediction in 2017 = 9.019206
         LCV 2017 real emission = 2.180134, sum of single group prediction in 2017 = 3.070119
```

Figure 90: random seed 722, for sum of single groups.

In here, we find using isotonic regressor will reduce the gap between sum of single groups with total groups. However, we keep going on its single group predictions, it won't predict values for the used groups and remaining groups values are similar with Ridge Regressor.

Finally, we could confirm PySpark is **NOT proper** for this problem, since using different data and models, it still cannot satisfy the project datamining assumption two. **we need to applying other models at final research paper.**

```
LPV 2017 single group predictions
0-4 years of LPV new : 1.613846 ,contributed:  17.893441 %
5-9 years of LPV new : 1.444675 ,contributed:  16.017763 %
10-14 years of LPV new : 2.573140 ,contributed:  28.529562 %
15-19 years of LPV new : 1.424148 ,contributed:  15.790165 %
20+ years of LPV new : 1.963397 ,contributed:  21.769069 %
0-4 years of LPV used : 0.000000 ,contributed:  0.000000 %
5-9 years of LPV used : 0.000000 ,contributed:  0.000000 %
10-14 years of LPV used : 0.000000 ,contributed:  0.000000 %
15-19 years of LPV used : 0.000000 ,contributed:  0.000000 %
20+ years of LPV used : 0.000000 ,contributed:  0.000000 %
```

Figure 91: random seed 722, LPV single groups.

```
LCV 2017 single group predictions
0-4 years of LCV new : 0.549349 ,contributed:  17.893399 %
5-9 years of LCV new : 0.491766 ,contributed:  16.017809 %
10-14 years of LCV new : 0.875890 ,contributed:  28.529517 %
15-19 years of LCV new : 0.484777 ,contributed:  15.790159 %
20+ years of LCV new : 0.668338 ,contributed:  21.769116 %
0-4 years of LCV used : 0.000000 ,contributed:  0.000000 %
5-9 years of LCV used : 0.000000 ,contributed:  0.000000 %
10-14 years of LCV used : 0.000000 ,contributed:  0.000000 %
15-19 years of LCV used : 0.000000 ,contributed:  0.000000 %
20+ years of LCV used : 0.000000 ,contributed:  0.000000 %
```

Figure 92: random seed 722, LCV single groups.

9 Version Control - GitHub

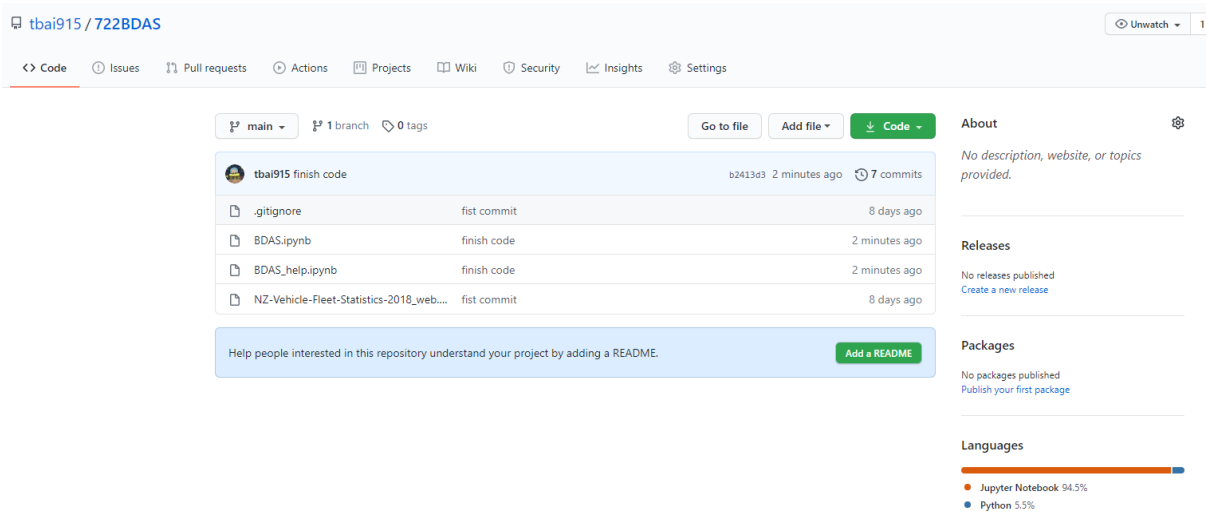


Figure 93: Github.

Figure 94: AWS EC2.

Figure 95: AWS EC2.

Reference:

1. World Meteorological Organization. (15 January 2020). “WMO confirms 2019 as second hottest year on record”. Retrieved from https://public.wmo.int/en/media/press-release/wmo-confirms-2019-second-hottest-year-record

2. Ministry of Transport. (05 August 2020).” Vehicle Fleet Statistics”. Retrieved from https://www.transport.govt.nz/mot-resources/vehicle-fleet-statistics/

Disclaimer

I acknowledge that the submitted work is my own original work in accordance with the University of Auckland guidelines and policies on academic integrity and copyright. (See: https://www.auckland.ac.nz/en/students/forms-policies-and-guidelines/student-policies-and-guidelines/academic-integrity-copyright.htmlLinks to an external site.).

I also acknowledge that I have appropriate permission to use the data that I have utilised in this project. (For example, if the data belongs to an organisation and the data has not been published in the public domain then the data must be approved by the rights holder.) This includes permission to upload the data file to Canvas. The University of Auckland bears no responsibility for the student's misuse of data.