الجامعة السعودية الالكترونية
Saudi Electronic University
2011 1432

College of Computing and Informatics

2020/2021 First Semester

Course Code	DS520
Course Name	big data processing and analysis
CRN	24541
Assignment type	Project
Module	All
Assignment Points	10

Student ID	G200002614	G200007615
Student Name	Enad S. Alotaibi	Abdulaziz M. Alqumayzi

- Task1:

1.1Literature Review:

Abstract:

Topic 1: Sentiment analysis is used in identifying the public opinion through text analytics. Big data tools can aid in the storage and processing of data for sentiment analysis. Through such analysis, companies can better plan their processes and sales accordingly.

Topic 2: Machine Learning algorithms are very important in the field of data science. With the increasing number of data, it is very important and advantageous to apply those algorithms on Big Data.

Sentimental Analysis Using Big Data:

With increase in unstructured data with increase in number of Social Media Applications, Sentimental Analysis has become one of the most important aspect of Big Data Analytics. Moreover, with increase in IOT more and more devices are now connected to Internet which entails rapid generation of Data which might be structured or unstructured in nature. According to a survey 90% of the overall data that is being generated is unstructured in its nature. These dataset analogous to Crude Oil if refined properly contains a huge value.

Big Data is a term which means dealing with massive scale datasets. Big Data is trending research area in Computer Science and Sentiment Analysis is one of the most important part of this research area. Big Data is considered as very large amount of data which can be found easily on web, Social media, remote sensing data and medical records etc. in form of structured, semi-structured or unstructured data and we can use these data for Sentiment Analysis.

Rapid increase in the volume of sentiment rich social media on the web has resulted in an increased interest among researchers regarding Sentimental Analysis and opinion mining.

However, with so much social media available on the web, sentiment analysis is now considered as a big data task. Hence the conventional sentiment analysis approaches fail to efficiently handle the vast amount of sentiment data available now a days. The main focus of the research was to find such a technique that can efficiently perform sentiment analysis on big data sets. A technique that can categorize the text as positive, negative and neutral in a fast and accurate manner. In the research, sentiment analysis was performed on a large data set of tweets using Hadoop and the performance of the technique was measured in form of speed and accuracy. The experimental results shows that the technique exhibits very good efficiency in handling big sentiment data sets.

One of the methods for Sentimental Analysis on Big Datasets is a dictionary-based technique i.e., a dictionary of sentiment bearing words was used to classify the text into positive, negative or neutral opinion. Machine learning techniques are not used because although they are more accurate than the dictionary-based approaches, they take far too much time performing Sentiment Analysis as they have to be trained first and hence are not efficient in handling big sentiment data.

Importance of Machine Learning Algorithms in Big Data:

As we know that a rapid rise in data has been seen in recent times and growing data has given a rise in requirement to analyse the data to getter better business insights from the data and utilize the true potential of Big Data. So, in this review we will try to understand the importance and role that Machine Learning plays in analysing the Big Datasets.

Big-data is an excellent source of knowledge and information from our systems and clients, but dealing with such amount of data requires automation, and this brings us to data mining and machine learning techniques. In the ICT sector, as in many other sectors of research and industry, platforms and tools are being served and developed in order to help professionals to treat their data and learn from it automatically; most of those platforms coming from big companies like Google or Microsoft, or from incubators at the Apache Foundation.

Dealing with big-data usually involves finding on it the relevant information, modelling the elements composing it, and transforming it into useful information and knowledge. For such goals most of professionals prefer to use Machine Learning techniques for modelling and prediction, data aggregation and clustering, and knowledge discovery. Machine Learning, as part of Data Mining, provides methods to treat and extract information from data automatically, where human operators and experts are not able to deal with because of the level of complexity or the volume to be treated per time unit.

There are majorly two types of Machine Learning Algorithms in Big Data: Supervised Learning and Unsupervised Learning. Unsupervised Learning contains algorithms like Clustering which are used to find data which are related with each other in the data. Supervised Learning algorithms have mainly two types of algorithms Regression and Classification. Regression is used to predict a dependent value based on multiple independent features and Classification is used to predict the depend column (which contains categorical data) based on the independent feature columns.

There are many tools to apply Machine Learning Algorithms in Big Data. Hadoop natively have Mahout which runs natively on Hadoop and have the potential to run Machine Learning Algorithms on Big Data. Moreover, Spark have Spark MLlib and Spark ML to run Machine Learning Algorithms on Spark RDDs and Spark Data frames. These tools have the capabilities to run most of the machine learning algorithms with Apache Spark even giving the potential to run Multilayer Neural Network and Artificial Neural Networks.

1.2 References:

[1] Kurian, D.D.M.K., Vishnupriya, S., Ramesh, R., Divya, G., Divya, D., Kurian, M.K., Vishnupriya, S., Ramesh, R., Divya, G. and Divya, D., 2015. Big data sentiment analysis using hadoop. *International Journal for Innovative Research in Science and Technology*, *1*(11), pp.92-96.

[2] Kaushik, C. and Mishra, A., 2014. A scalable, lexicon based technique for sentiment analysis. *arXiv preprint arXiv:1410.2265*.

[3] Yadav, K., Pandey, M. and Rautaray, S.S., 2016, November. Feedback analysis using big data tools. In *2016 International Conference on ICT in Business Industry & Government (ICTBIG)* (pp. 1-5). IEEE.

[4] Sagiroglu, S. and Sinanc, D., 2013, May. Big data: A review. In *2013 international conference on collaboration technologies and systems (CTS)* (pp. 42-47). IEEE.

[5] Berral-García, J.L., 2016, July. A quick view on current techniques and machine learning algorithms for big data analytics. In *2016 18th international conference on transparent optical networks (ICTON)* (pp. 1-4). IEEE.

[6] Rahul, K., Banyal, R.K., Goswami, P. and Kumar, V., 2021. Machine Learning Algorithms for Big Data Analytics. In *Computational Methods and Data Engineering* (pp. 359-367). Springer, Singapore.

[7] Gupta, P., Sharma, A. and Jindal, R., 2016. Scalable machine-learning algorithms for big data analytics: a comprehensive review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *6*(6), pp.194-214.

[8] Suthaharan, S., 2016. Machine learning models and algorithms for big data classification. *Integr. Ser. Inf. Syst*, *36*, pp.1-12.

- **Task2:**

2.1 Introduction:

In this Report we will be analysing Big Dataset using multiple Big Data tools and try to understand the tools and technologies along with getting Insights from the data.

There are total eight tasks in the Report as given below:

2.2 Body section

Loading Dataset to Hadoop:

- We will be using Craigslist Cars Trucks dataset in this report.

- Firstly, the dataset is downloaded from https://www.kaggle.com/austinreese/craigslist-carstrucks-data which will download the data in our local system.
- The dataset size is 1.34 GB and contains 26 columns having details regarding vehicles like model, manufacturer, condition, price and so on.
- Vehicles dataset contains 475,057 rows in total.
- Now, we will load the dataset into Hadoop file system, but before that we will start the Hadoop services:

  Command:

  start-all.sh

  Output:

```
(base) hduser@spark-VirtualBox:~/work$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
21/03/29 13:51:18 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/softwares/hadoop/logs/hadoop-hduser-namenode-spark-VirtualBox.out
localhost: starting datanode, logging to /usr/local/softwares/hadoop/logs/hadoop-hduser-datanode-spark-VirtualBox.out
jStarting secondary namenodes [0.0.0.0]
p0.0.0.0: starting secondarynamenode, logging to /usr/local/softwares/hadoop/logs/hadoop-hduser-secondarynamenode-spark-VirtualBox.out
21/03/29 13:51:34 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
starting yarn daemons
starting resourcemanager, logging to /usr/local/softwares/hadoop/logs/yarn-hduser-resourcemanager-spark-VirtualBox.out
localhost: starting nodemanager, logging to /usr/local/softwares/hadoop/logs/yarn-hduser-nodemanager-spark-VirtualBox.out
(base) hduser@spark-VirtualBox:~/work$ jps
9329 SecondaryNameNode
9506 ResourceManager
3702
8888 NameNode
9690 NodeManager
9982 Jps
9087 DataNode
```

- So, now are Hadoop services are up, we will now copy our dataset from local file system to Hadoop File system

Command:

hdfs dfs –copyFromLocal vehicles.csv /mnt/data/source/.

Output:

```
(base) hduser@spark-VirtualBox:~/work/esaud_project$ hdfs dfs -copyFromLocal vehicles.csv /mnt/data/source/.
21/03/29 13:56:03 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
(base) hduser@spark-VirtualBox:~/work/esaud_project$ hdfs dfs -ls /mnt/data/source
21/03/29 13:56:27 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r--   1 hduser supergroup 1437679401 2021-03-29 13:56 /mnt/data/source/vehicles.csv
```

- Therefore, now our dataset is loaded in Hadoop File System (HDFS).

- Let us check the size of dataset in HDFS.

```
(base) hduser@spark-VirtualBox:~/work/esaud_project$ hdfs dfs -du -s -h /mnt/data/source/vehicles.csv
21/03/29 13:59:00 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
1.3 G  /mnt/data/source/vehicles.csv
```

- So, now we can move on to the next sections of our Report which will revolve around analysing the dataset using different Big Data Technologies.

- **<u>Task3 - Analysis using Map Reduce:</u>**
  - In this section, we will use Map Reduce to do analysis over the dataset.
  - We will find the count of each model sold by each manufacturer over all the years.
  - Map Reduce paradigm of analysis has 3 files namely Mapper, Reducer and Driver.
  - The Code for each of the files are given below:

<u>Mapper:</u>

```
public class VehicleMapper extends Mapper
      {
            public void Map(LongWritable key, Text value, Context
            context) throws IOException, InterruptedException
            {
                  String val = value.toString();
                  String [] input = val.split(",");

                  String manufacturer = input[8];
                  String model = input[7];
                  String key = manufacturer + model;

                  context.write(new Text(key), new IntWritable(1));
            }
      }
```

Reducer:

```
public class VehicleReducer extends Reducer
{
	public void Reduce(Text key, Iterable values, Context context)
	throws IOException, InterruptedException
	{
		int count = 0;
		while (values.hasNext())
		{
			count++;
		}
		context.write(key, new Text(String.valueOf(count)));
	}
}
```

Driver:

```
public class VehicleDriver {
	public static void main(String[] args) throws Exception {
		if(args.length != 3) {
			System.exit(-1);
		}

		Configuration c = new Configuration();
		Job job = Job.getInstance(c, "Job");

		job.setJarByClass(VehicleDriver.class);
		job.setJobName("Job");

		FileInputFormat.setInputPaths(job, new Path(args[0]));
		FileOutputFormat.setOutputPath(job, new Path(args[1]));


		job.setMapperClass(VehicleMapper.class);
		job.setReducerClass(VehicleReducer.class);

		job.setMapOutputKeyClass(Text.class);
		job.setMapOutputValueClass(Text.class);

		job.setOutputKeyClass(UserPairWritable.class);
		job.setOutputValueClass(Text.class);

		int a = job.waitForCompletion(true)?0:1;

		if(a != 0) {
			System.out.println("Job Failed ");
			System.exit(-1);
		}
	}
}
```

- <u>**Task4 - Loading Data to MongoDB:**</u>

    - In this section we will load CSV dataset to Mongo DB.

    - We will use below command to load vehicles.csv to MongoDB.

Code:

```
mongoimport --type csv -d project -c vehicles --headerline --drop vehicles.csv
```

Output:

```
(base) hduser@spark-VirtualBox:~/work/esaud_project$ mongoimport --type csv -d project -c vehicles --headerline --drop vehicles.csv
2021-03-29T22:12:49.592+0530    connected to: localhost
2021-03-29T22:12:49.593+0530    dropping: project.vehicles
2021-03-29T22:12:52.585+0530    [........................] project.vehicles     48.9MB/1.34GB (3.6%)
2021-03-29T22:12:55.586+0530    [#.......................] project.vehicles     106MB/1.34GB (7.7%)
2021-03-29T22:12:58.585+0530    [##......................] project.vehicles     151MB/1.34GB (11.0%)
2021-03-29T22:13:01.587+0530    [###.....................] project.vehicles     190MB/1.34GB (13.8%)
2021-03-29T22:13:04.585+0530    [####....................] project.vehicles     232MB/1.34GB (16.9%)
2021-03-29T22:13:07.584+0530    [####....................] project.vehicles     273MB/1.34GB (19.9%)
2021-03-29T22:13:10.585+0530    [#####...................] project.vehicles     316MB/1.34GB (23.0%)
2021-03-29T22:13:13.586+0530    [######..................] project.vehicles     362MB/1.34GB (26.4%)
2021-03-29T22:13:16.721+0530    [#######.................] project.vehicles     411MB/1.34GB (30.0%)
2021-03-29T22:13:19.585+0530    [########................] project.vehicles     465MB/1.34GB (33.9%)
2021-03-29T22:13:22.585+0530    [########................] project.vehicles     507MB/1.34GB (37.0%)
2021-03-29T22:13:25.587+0530    [#########...............] project.vehicles     554MB/1.34GB (40.4%)
2021-03-29T22:13:28.585+0530    [##########..............] project.vehicles     597MB/1.34GB (43.5%)
2021-03-29T22:13:31.585+0530    [###########.............] project.vehicles     639MB/1.34GB (46.6%)
2021-03-29T22:13:34.585+0530    [###########.............] project.vehicles     684MB/1.34GB (49.9%)
2021-03-29T22:13:37.585+0530    [############............] project.vehicles     728MB/1.34GB (53.1%)
2021-03-29T22:13:40.585+0530    [#############...........] project.vehicles     778MB/1.34GB (56.8%)
2021-03-29T22:13:43.743+0530    [##############..........] project.vehicles     808MB/1.34GB (58.9%)
2021-03-29T22:13:46.585+0530    [##############..........] project.vehicles     836MB/1.34GB (61.0%)
2021-03-29T22:13:49.585+0530    [###############.........] project.vehicles     873MB/1.34GB (63.6%)
2021-03-29T22:13:52.586+0530    [###############.........] project.vehicles     906MB/1.34GB (66.1%)
2021-03-29T22:13:55.617+0530    [################........] project.vehicles     923MB/1.34GB (67.4%)
2021-03-29T22:13:58.585+0530    [################........] project.vehicles     949MB/1.34GB (69.2%)
2021-03-29T22:14:01.585+0530    [#################.......] project.vehicles     1006MB/1.34GB (73.4%)
2021-03-29T22:14:04.585+0530    [##################......] project.vehicles     1.02GB/1.34GB (75.9%)
2021-03-29T22:14:07.585+0530    [##################......] project.vehicles     1.05GB/1.34GB (78.6%)
2021-03-29T22:14:11.081+0530    [###################.....] project.vehicles     1.09GB/1.34GB (81.6%)
2021-03-29T22:14:13.616+0530    [####################....] project.vehicles     1.12GB/1.34GB (83.9%)
2021-03-29T22:14:16.585+0530    [####################....] project.vehicles     1.15GB/1.34GB (86.0%)
2021-03-29T22:14:19.585+0530    [#####################...] project.vehicles     1.19GB/1.34GB (88.9%)
2021-03-29T22:14:22.585+0530    [#####################...] project.vehicles     1.22GB/1.34GB (91.5%)
2021-03-29T22:14:25.585+0530    [######################..] project.vehicles     1.26GB/1.34GB (93.9%)
2021-03-29T22:14:28.589+0530    [#######################.] project.vehicles     1.30GB/1.34GB (96.8%)
2021-03-29T22:14:31.585+0530    [#######################.] project.vehicles     1.33GB/1.34GB (99.1%)
2021-03-29T22:14:32.563+0530    [########################] project.vehicles     1.34GB/1.34GB (100.0%)
2021-03-29T22:14:32.563+0530    imported 458213 documents
```

- Now, let us check that the data is loaded correctly.

```
> show dbs;
admin    0.000GB
config   0.000GB
local    0.000GB
project  0.752GB
> use project;
switched to db project
> show collections;
vehicles
> db.vehicles.findOne()
{
        "_id" : ObjectId("60620389f9ea61dadc49f0c3"),
        "" : 0,
        "id" : NumberLong("7240372487"),
        "url" : "https://auburn.craigslist.org/ctd/d/auburn-university-2010-chevy-chevrolet/7240372487.html",
        "region" : "auburn",
        "region_url" : "https://auburn.craigslist.org",
        "price" : 35990,
        "year" : 2010,
        "manufacturer" : "chevrolet",
        "model" : "corvette grand sport",
        "condition" : "good",
        "cylinders" : "8 cylinders",
        "fuel" : "gas",
        "odometer" : 32742,
        "title_status" : "clean",
        "transmission" : "other",
        "VIN" : "1G1YU3DW1A5106980",
        "drive" : "rwd",
        "size" : "",
        "type" : "other",
        "paint_color" : "",
        "image_url" : "https://images.craigslist.org/00N0N_ipkbHVZYf4w_0gw0co_600x450.jpg",
        "description" : "Carvana is the safer way to buy a car During these uncertain times, Carvana is dedicated to ensuring safety for all of our customers. In addition to our 100% online shopping and s
elling experience that allows all customers to buy and trade their cars without ever leaving the safety of their house, we’re providing touchless delivery that make all aspects of our process even safer.
Now, you can get the car you want, and trade in your old one, while avoiding person-to-person contact with our friendly advocates. There are some things that can’t be put off. And if buying a car is one o
f them, know that we’re doing everything we can to keep you keep moving while continuing to put your health safety, and happiness first. Vehicle Stock# 2000721559📱 Want to instantly check this car’s avai
lability? Call us at  334-758-9176Just text that stock number to 855-976-4304 or head to http://www.carvanaauto.com/6143424-74502 and plug it into the search bar!Get PRE-QUALIFIED for your auto loan in 2
minutes - no hit to your credit:http://finance.carvanaauto.com/6143424-74502Looking for more cars like this one? We have 94 Chevrolet Corvette in stock for as low as $27990!Why buy with Carvana? We have o
ne standard: the highest. Take a look at just some of the qualifications all of our cars must meet before we list them.150-POINT INSPECTION: We put each vehicle through a 150-point inspection so that you
can be 100% confident in its quality and safety. See everything that goes into our inspections at:http://www.carvanaauto.com/6143424-74502NO REPORTED ACCIDENTS: We do not sell cars that have been in a rep
orted accident or have a frame or structural damage.7 DAY TEST OWN MONEY BACK GUARANTEE: Every Carvana car comes with a 7-day money-back guarantee. Why? It takes more than 15-minutes to make a decision on
 your next car. Learn more about test owning at http://about.carvanaauto.comFLEXIBLE FINANCING, TRADE INS WELCOME: We’re all about real-time financing without the middle man. Need financing? Pick a combin
```

- So, the data is now loaded successfully to MongoDB.

- **Task5 - Analysis in MongoDB:**
    - In this section, we will analyse the dataset using MongoDB.
    - MongoDB is a NoSQL database.
    - Dataset is already loaded to MongoDB collections which we will use in this section.
    - First Analysis: How many vehicles ‘Chevrolet’ manufacturer has sold:

Code:

```
db.vehicles.find({"manufacturer":"chevrolet"}).count()
```

Output:

```
> db.vehicles.find({"manufacturer":"chevrolet"}).count()
64977
```

- Second Analysis: Find top 10 most expensive vehicles from the dataset.

Code:

```
db.vehicles.find({},
        {"manufacturer":1,"model":1,"price":1}).sort({"price":-1}).limit(10)
```

Output:

```
> db.vehicles.find({},{"manufacturer":1,"model":1,"price":1}).sort({"price":-1}).limit(10)
{ "_id" : ObjectId("606203dcf9ea61dadc4fd3e5"), "price" : NumberLong("3615215112"), "manufacturer" : "chevrolet", "model" : "silverado 2500hd" }
{ "_id" : ObjectId("606203e8f9ea61dadc506f5d"), "price" : NumberLong("2857993261"), "manufacturer" : "jeep", "model" : "" }
{ "_id" : ObjectId("60620390f9ea61dadc4a86d0"), "price" : NumberLong("2808348671"), "manufacturer" : "gmc", "model" : "" }
{ "_id" : ObjectId("60620389f9ea61dadc49f721"), "price" : 1234567890, "manufacturer" : "chevrolet", "model" : "" }
{ "_id" : ObjectId("606203c8f9ea61dadc4e9e2c"), "price" : 123456789, "manufacturer" : "ram", "model" : "3500 crewcab tradesma" }
{ "_id" : ObjectId("606203d0f9ea61dadc4eec31"), "price" : 123456789, "manufacturer" : "chrysler", "model" : "royal" }
{ "_id" : ObjectId("606203d2f9ea61dadc4f1f1a"), "price" : 123456789, "manufacturer" : "chevrolet", "model" : "camaro" }
{ "_id" : ObjectId("606203d2f9ea61dadc4f1f29"), "price" : 123456789, "manufacturer" : "jeep", "model" : "wrangler unlimited" }
{ "_id" : ObjectId("606203e4f9ea61dadc5045f5"), "price" : 123456789, "manufacturer" : "ford", "model" : "explorer" }
{ "_id" : ObjectId("606203e4f9ea61dadc5045f7"), "price" : 123456789, "manufacturer" : "", "model" : "Navagater" }
```

- Third Analysis: Find the count of each Cylinder type from the dataset.

Code:

```
db.vehicles.aggregate([{$group : {_id: "$cylinders", totalCount: {$sum:1}}}])
```

Output:

```
> db.vehicles.aggregate([{$group : {_id: "$cylinders", totalCount: {$sum:1}}}])
{ "_id" : "3 cylinders", "totalCount" : 550 }
{ "_id" : "5 cylinders", "totalCount" : 2058 }
{ "_id" : "other", "totalCount" : 1112 }
{ "_id" : "10 cylinders", "totalCount" : 1543 }
{ "_id" : "8 cylinders", "totalCount" : 81179 }
{ "_id" : "6 cylinders", "totalCount" : 105677 }
{ "_id" : "4 cylinders", "totalCount" : 94767 }
{ "_id" : "12 cylinders", "totalCount" : 187 }
{ "_id" : "", "totalCount" : 171140 }
```

- **Task6 - Analysis in Hive:**
  - We will analyse the dataset using HIVE in this section.
  - Firstly, we will create a table in Hive to contain the dataset.

```
    > CREATE TABLE IF NOT EXISTS VEHICLES
    > ( `_C0` STRING,
    > `ID` STRING,
    > `URL` STRING,
    > `REGION` STRING,
    > `REGION_URL` STRING,
    > `PRICE` STRING,
    > `YEAR` STRING,
    > `MANUFACTURER` STRING,
    > `MODEL` STRING,
    > `CONDITION` STRING,
    > `CYLINDERS` STRING,
    > `FUEL` STRING,
    > `ODOMETER` STRING,
    > `TITLE_STATUS` STRING,
    > `TRANSMISSION` STRING,
    > `VIN` STRING,
    > `DRIVE` STRING,
    > `SIZE` STRING,
    > `TYPE` STRING,
    > `PAINT_COLOR` STRING,
    > `IMAGE_URL` STRING,
    > `DESCRIPTION` STRING,
    > `STATE` STRING,
    > `LAT` STRING,
    > `LONG` STRING,
    > `POSTING_DATE` STRING)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > LINES TERMINATED BY '\n'
    > STORED AS TEXTFILE;
OK
Time taken: 1.022 seconds
```

- Now using LOAD DATA LOCAL INPATH command, we will load the dataset to HIVE table we created.

```
    > LOAD DATA LOCAL INPATH '/home/hduser/work/esaud_project/vehicles.csv' OVERWRITE INTO TABLE VEHICLES;
Loading data to table default.vehicles
OK
Time taken: 12.565 seconds
```

- Let us check the dataset in HIVE table to see if the data is loaded properly.

```
    > SELECT * FROM VEHICLES LIMIT 5;
OK
vehicles._c0    vehicles.id     vehicles.url    vehicles.region vehicles.region_url     vehicles.price  vehicles.year   vehicles.manufacturer   vehicles.model  vehicles.condition      vehicles.cylinders v
ehicles.fuel    vehicles.odometer       vehicles.title_status   vehicles.transmission   vehicles.vin    vehicles.drive  vehicles.size   vehicles.type   vehicles.paint_color    vehicles.image_url      vehi
cles.description        vehicles.state  vehicles.lat    vehicles.long   vehicles.posting_date
        id      url     region  region_url      price   year    manufacturer    model   condition       cylinders       fuel    odometer        title_status    transmission    VIN     drive   size    type
paint_color     image_url       description     state   lat     long    posting_date
0       7240372487      https://auburn.craigslist.org/ctd/d/auburn-university-2010-chevy-chevrolet/7240372487.html      auburn  https://auburn.craigslist.org   35990   2010.0  chevrolet       corvette gra
nd sport        good    8 cylinders     gas     32742.0 clean   other   1G1YU3DW1A5106980       rwd             other           https://images.craigslist.org/00N0N_ipkbHVZYf4w_0gw0co_600x450.jpg      "Car
vana is the safer way to buy a car During these uncertain times  Carvana is dedicated to ensuring safety for all of our customers. In addition to our 100% online shopping and selling experience that allow
s all customers to buy and trade their cars without ever leaving the safety of their house      we're providing touchless delivery that make all aspects of our process even safer. Now        you can get
 the car you want       and trade in your old one
1       7240309422      https://auburn.craigslist.org/cto/d/auburn-2014-hyundai-sonata-20t/7240309422.html      auburn  https://auburn.craigslist.org   7500    2014.0  hyundai sonata  excellent       4 cy
linders gas     93600.0 clean   automatic       5NPEC4AB0EH813529       fwd             sedan           https://images.craigslist.org/00s0s_gBHYmJ5o7yM_0ne0hq_600x450.jpg      "I'll move to another city a
nd try to sell my car. The car is in very good condition        everything works and fully cleaned. It equipped with a heated seat      power seat      backup camera   Bluetooth
2       7240224296      https://auburn.craigslist.org/cto/d/auburn-2006-bmw-x3/7240224296.html  auburn  https://auburn.craigslist.org   4900    2006.0  bmw     x3 3.0i good    6 cylinders     gas     8704
6.0     clean   automatic                       SUV     blue    https://images.craigslist.org/00B0B_5zgEGWPOrt0_07L0ak_600x450.jpg      "Clean 2006 BMW X3 3.0I.  Beautiful and rare Blue Water Meta
llic exterior and tan interior color combination. 5-speed automatic transmission        AWD     CD/AM/FM radio  cold A/C (just serviced along w/oil change)     alloy wheels
3       7240103965      https://auburn.craigslist.org/cto/d/lanett-truck/7240103965.html        auburn  https://auburn.craigslist.org   2000    1974.0  chevrolet       c-10    good    4 cylinders     gas1
90000.0 clean   automatic               rwd     full-size       pickup  blue    https://images.craigslist.org/00M0M_6o7KcDpArwl_0CI0t2_600x450.jpg      1974 chev. truck (LONG BED) NEW starter front and ba
ck breaks       al      32.8616 -85.2161        2020-12-01T15:54:45-0600
Time taken: 1.383 seconds, Fetched: 5 row(s)
```

- Let us check the count of data in table.

```
    > SELECT COUNT(1) FROM VEHICLES;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine
Query ID = hduser_20210329173240_1961f1c5-8e16-4c01-baff-d88efe7ee174
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1617019272949_0001, Tracking URL = http://spark-VirtualBox:8088/proxy/application_1617019272949_0001/
Kill Command = /usr/local/softwares/hadoop/bin/hadoop job  -kill job_1617019272949_0001
Hadoop job information for Stage-1: number of mappers: 6; number of reducers: 1
2021-03-29 17:32:51,941 Stage-1 map = 0%,  reduce = 0%
2021-03-29 17:33:06,568 Stage-1 map = 17%,  reduce = 0%, Cumulative CPU 5.02 sec
2021-03-29 17:33:07,616 Stage-1 map = 33%,  reduce = 0%, Cumulative CPU 11.6 sec
2021-03-29 17:33:08,641 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 36.78 sec
2021-03-29 17:33:11,747 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 39.02 sec
MapReduce Total cumulative CPU time: 39 seconds 20 msec
Ended Job = job_1617019272949_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 6  Reduce: 1   Cumulative CPU: 39.02 sec   HDFS Read: 1437787344 HDFS Write: 106 SUCCESS
Total MapReduce CPU Time Spent: 39 seconds 20 msec
OK
480154
Time taken: 33.137 seconds, Fetched: 1 row(s)
```

- In the analysis, we will try to find average selling price of each manufacturer from the year 1995 to 2020.

Code:

```
WITH CTE AS (
      SELECT
      MANUFACTURER,
      CAST(YEAR AS INTEGER) AS YEAR,
      CAST(PRICE AS INTEGER) AS PRICE
      FROM VEHICLES
      WHERE CAST(YEAR AS INTEGER) BETWEEN 1995 AND 2020
      )
      SELECT MANUFACTURER, YEAR, AVG(PRICE) AS AVG_PRICE
      FROM CTE
      GROUP BY MANUFACTURER, YEAR
      ORDER BY MANUFACTURER, YEAR
      LIMIT 50;
```

College of Computing and Informatics

Output:

```
    > WITH CTE AS (
    > SELECT MANUFACTURER,
    > CAST(YEAR AS INTEGER) AS YEAR,
    > CAST(PRICE AS INTEGER) AS PRICE
    > FROM VEHICLES_2
    > WHERE CAST(YEAR AS INTEGER) BETWEEN 1995 AND 2020
    > )
    > SELECT
    > MANUFACTURER,
    > YEAR,
    > AVG(PRICE) AS AVG_PRICE FROM CTE
    > GROUP BY MANUFACTURER, YEAR
    > ORDER BY MANUFACTURER, YEAR
    > LIMIT 50;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different e
Query ID = hduser_20210329205835_d398bab1-a1a4-4ea1-8f21-e5521b9f559b
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1617031582049_0002, Tracking URL = http://spark-VirtualBox:8088/proxy/application_1617031582049_0002/
Kill Command = /usr/local/softwares/hadoop/bin/hadoop job  -kill job_1617031582049_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2021-03-29 20:58:42,349 Stage-1 map = 0%,  reduce = 0%
2021-03-29 20:58:48,493 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.9 sec
2021-03-29 20:58:53,656 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.9 sec
MapReduce Total cumulative CPU time: 6 seconds 900 msec
Ended Job = job_1617031582049_0002
```

```
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1617031582049_0003, Tracking URL = http://spark-VirtualBox:8088/proxy/application_1617031582049_0003/
Kill Command = /usr/local/softwares/hadoop/bin/hadoop job  -kill job_1617031582049_0003
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2021-03-29 20:59:06,243 Stage-2 map = 0%,  reduce = 0%
2021-03-29 20:59:10,458 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.99 sec
2021-03-29 20:59:16,599 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.94 sec
MapReduce Total cumulative CPU time: 2 seconds 940 msec
Ended Job = job_1617031582049_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.9 sec   HDFS Read: 52051611 HDFS Write: 1630 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.94 sec   HDFS Read: 7945 HDFS Write: 1843 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 840 msec
```

```
2002.0  2000    NULL
2003.0  1995    NULL
2005.0  2000    NULL
2006    2005    2004.0
2006.0  2000    NULL
2007.0  2000    NULL
acura   1995    3400.0
acura   1997    1200.0
acura   1998    4998.333333333333
acura   1999    2333.3333333333335
acura   2000    2400.0
acura   2001    3884.714285714286
acura   2002    2838.0
acura   2003    3675.0
acura   2004    13142.153846153846
acura   2005    4658.0625
acura   2006    3935.214285714286
acura   2007    6691.214285714285
acura   2008    6398.458333333333
acura   2009    9506.375
acura   2010    10487.095238095239
acura   2011    12611.931034482759
acura   2012    14299.414634146342
acura   2013    14343.411764705883
```

- **<u>Task7 - Analysis using Spark SQL:</u>**

  - The aim in this section is to analyse the data using Spark SQL.
  - Firstly, spark session is initialized and the data is loaded in spark dataframe.

```
In [12]: from pyspark.sql.session import SparkSession
         from pyspark.sql import functions as fn

         import matplotlib.pyplot as plt
```

```
In [19]: spark = SparkSession.builder.appName("Analysis").getOrCreate()
```

```
In [1]: # Reading dataset

        vehicles_df = spark.read.csv("hdfs://localhost:9000/mnt/data/source/vehicles.csv", inferSchema=True, header=True)
```

```
In [4]: # Dropping unused columns

        vehicles_df = vehicles_df.drop("_c0")
```

  - Temp view is created in Spark SQL so we can query on that table.

```
In [6]: vehicles_df.createOrReplaceTempView("vehicles")
```

  - Let us checkout some data from the dataset.

```
spark.sql("SELECT * FROM VEHICLES LIMIT 10").toPandas().head()
```

	id	url	region	region_url	price	year	manufacturer	model	condition	cylinders	...	drive	size	type
0	7240372487	https://auburn.craigslist.org /ctd/d/auburn-uni...	auburn	https://auburn.craigslist.org	35990	2010.0	chevrolet	corvette grand sport	good	8 cylinders	...	rwd	None	other
1	7240309422	https://auburn.craigslist.org /cto/d/auburn-201...	auburn	https://auburn.craigslist.org	7500	2014.0	hyundai	sonata	excellent	4 cylinders	...	fwd	None	sedan
2	7240224296	https://auburn.craigslist.org /cto/d/auburn-200...	auburn	https://auburn.craigslist.org	4900	2006.0	bmw	x3 3.0i	good	6 cylinders	...	None	None	SUV
3	7240103965	https://auburn.craigslist.org /cto/d/lanett-tru...	auburn	https://auburn.craigslist.org	2000	1974.0	chevrolet	c-10	good	4 cylinders	...	rwd	full-size	pickup
4	7239983776	https://auburn.craigslist.org /cto/d/auburn-200...	auburn	https://auburn.craigslist.org	19500	2005.0	ford	f350 lariat	excellent	8 cylinders	...	4wd	full-size	pickup

5 rows × 25 columns

  - In the first analysis, we will find the top 10 manufacturers who sold maximum cars. The distribution will be based on percentage cars sold with respect to total number of cars.

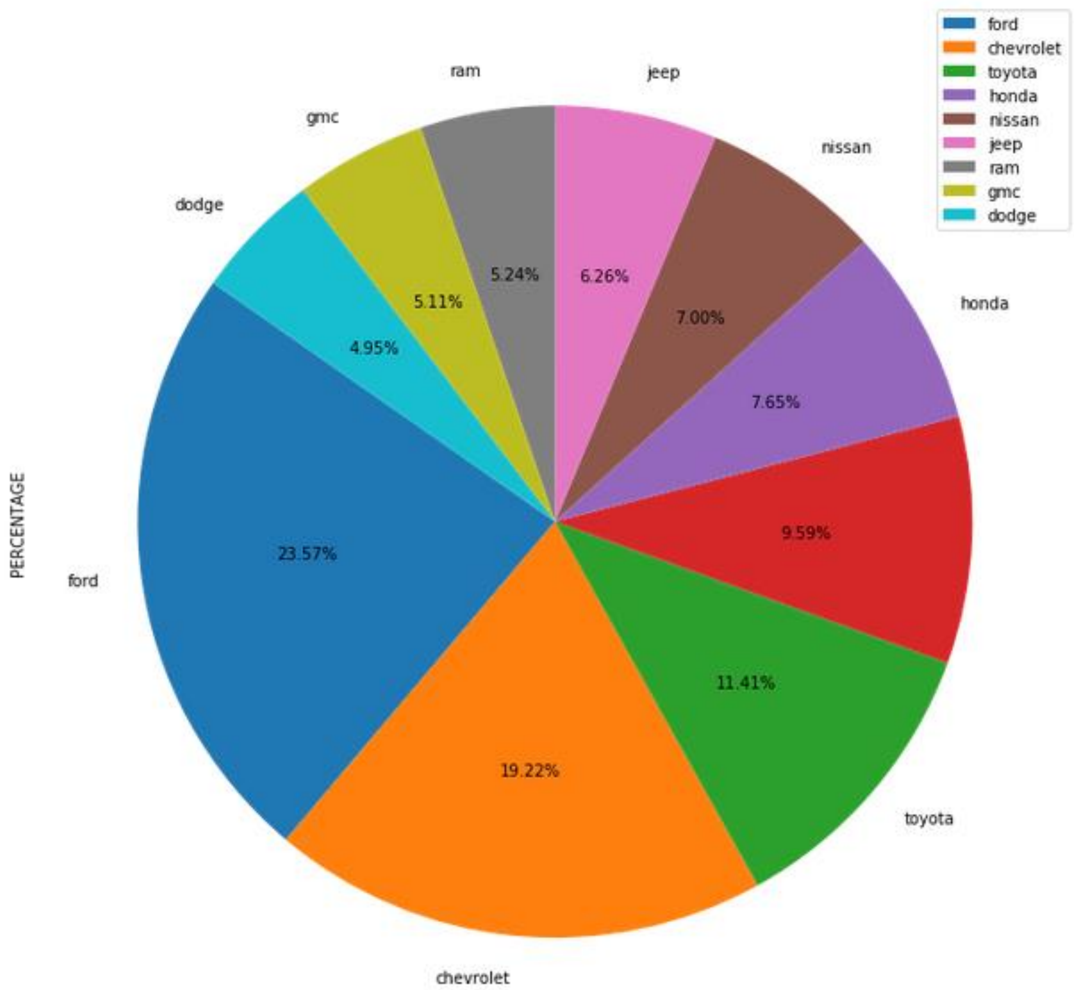
College of Computing and Informatics

Code:

```
pd_df = spark.sql("SELECT MANUFACTURER, 100*(COUNT(1)/(SELECT COUNT(1) FROM VEHICLES)) AS PERCENTAGE \
                    FROM VEHICLES \
                    GROUP BY MANUFACTURER \
                    ORDER BY PERCENTAGE DESC \
                    LIMIT 10").toPandas()

pd_df.plot.pie(y="PERCENTAGE", labels=pd_df["MANUFACTURER"], figsize=(12,12), startangle=145, autopct='%.2f%%')
```

Output:



- In second analysis, we will find the trend of number of cars sold in each year from 1995 to 2020 and visualize the data susing a line chart.

Code:

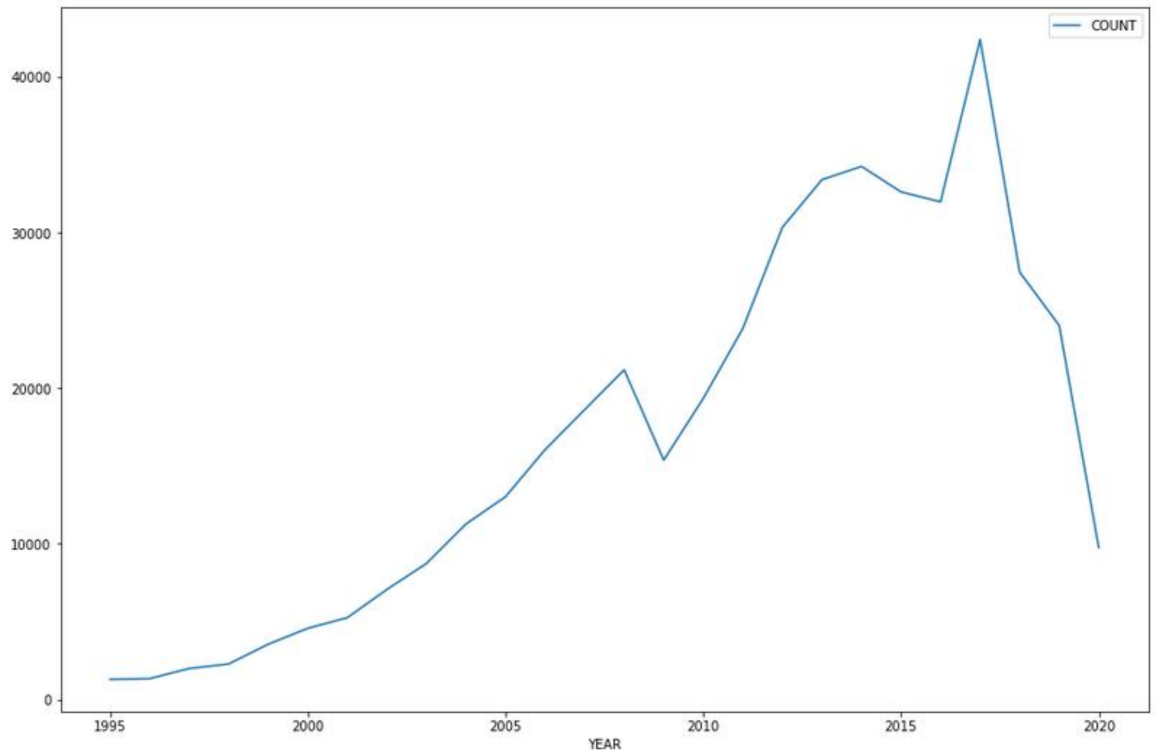
```
spark.sql("DROP TABLE IF EXISTS VEHICLES_STG")

          spark.sql("CREATE TABLE VEHICLES_STG AS \
                      SELECT CAST(YEAR AS INTEGER) YEAR \
                      FROM VEHICLES")

          pd_df = spark.sql("SELECT YEAR, COUNT(1) AS COUNT \
                              FROM VEHICLES_STG \
                              WHERE YEAR>=1995 AND YEAR<=2020 \
                              GROUP BY YEAR \
                              ORDER BY YEAR").toPandas()

          pd_df.plot(x="YEAR",    y="COUNT",    kind="line",
          figsize=(15, 10))
```

Output:



- **Task8 - Analysis using Spark ML:**
  - In this section, we will use Spark Machine Learning technologies to create a model on the dataset.
  - Firstly, as the dataset contains categorical values, we will encode the categorical column data to vector format using StringIndexer and OneHotEncoder. We will typecast the numerical columns.

In [46]:
```
from pyspark.ml.feature import VectorAssembler, StringIndexer, OneHotEncoderEstimator, StandardScaler
from pyspark.ml import Pipeline

from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
```

In [94]:
```
cat_features = ["region", "manufacturer", "model", "condition", "cylinders", "fuel", "transmission",
                "size", "type", "paint_color", "state"]

cat_features_ix = ["region_ix", "manufacturer_ix", "model_ix", "condition_ix", "cylinders_ix",
                   "fuel_ix", "transmission_ix", "size_ix", "type_ix", "paint_color_ix", "state_ix"]

cat_features_vec = ["region_vec", "manufacturer_vec", "model_vec", "condition_vec", "cylinders_vec",
                    "fuel_vec", "transmission_vec", "size_vec", "type_vec", "paint_color_vec", "state_vec"]

num_features = ["year", "odometer"]

for col in num_features:
    vehicles_df = vehicles_df.withColumn(col, fn.col(col).cast("Double"))

vehicles_df = vehicles_df.withColumn("price", fn.col("price").cast("Double"))

vehicles_df = vehicles_df.dropna()

for col in cat_features:
    indexer = StringIndexer(inputCol=col, outputCol=col+"_ix", handleInvalid="skip")
    vehicles_df = indexer.fit(vehicles_df).transform(vehicles_df)
```

In [95]:
```
oneHotEncoder = OneHotEncoderEstimator(inputCols=cat_features_ix, outputCols=cat_features_vec,
                                       handleInvalid="keep")

vehicles_df = oneHotEncoder.fit(vehicles_df).transform(vehicles_df)
```

  - Now, we will use Vector Assembler to assemble all the feature columns to one feature vector column.

In [96]:
```
assembler = VectorAssembler(inputCols = cat_features_vec+num_features,
                            outputCol = "features", handleInvalid="skip")

features_df = assembler.transform(vehicles_df)
```

  - We will scale the values using Standard Scaler, as scaled values will create the model faster and efficiently.

In [97]:
```
scaler = StandardScaler(inputCol="features", outputCol="sc_features")

scaled_df = scaler.fit(features_df).transform(features_df)
```

- Now, we will split the dataset in 80%-20% ratio for training and testing the model. Then, we will train Random Forest Regression Model on top of training dataset.

```
In [98]: train, test = scaled_df.randomSplit([0.8, 0.2])

randomForest = RandomForestRegressor() \
                        .setFeaturesCol("sc_features") \
                        .setLabelCol("price")

model = randomForest.fit(train)
```

- Finally, using Regression Evaluator we will evaluate the model. Metric to calculate accuracy will be R2.

```
In [100]: predictions = model.transform(test)

evaluator = RegressionEvaluator() \
                .setLabelCol("price") \
                .setMetricName("r2")

print("R2 ERROR RATE: {}".format(evaluator.evaluate(predictions)))

R2 ERROR RATE: 0.9305430818729231
```

- **<u>Conclusion:</u>**

  - So, in this report we did analysis of vehicles dataset using several Big Data technologies like Hive, Map Reduce, Spark and MongoDB. In the conclusion part, we will discuss some of the key insights gained from the project.
    - Chevrolet, Jeep and BMC sells some of the most expensive cars and Chevrolet have sold almost 650K cars until now.
    - Most of the vehicles sold had cylinder types as 4 Cylinders, 6 Cylinders or 8 Cylinders.
    - There has been stagnant increase in average selling price of vehicles each year.
    - Ford, Chevrolet and Toyota have sold most number of vehicles until now. These three manufacturers have sold more than 50% of vehicles out of the total vehicles sold.

  - There has been stable increase in the number of cars sold from 1995 until recently where there has been a sharp drop in the number of cars sold in couple of recent years.
- We also understood the importance of Big Data technologies in analyzing Big Datasets and how the parallel processing can help us transform the datasets in lesser time span.
- As future steps, we can focus more on predictive analysis using the given dataset and understand how the feature columns are dependent on the dependent columns like prices.