IBM_Capstone_Project_ML

October 23, 2023

1 Capstone Project - Analyzing a Dataset on Automotive Engine Health for Predictive Maintenance

2 Background

Dr. Michael Treasure - Data scientist providing consulting services for Tiemac Techologies, Inc.(Tiemac)

Tiemac provides a Telematics and Fleet Management Solution for real time predictive analytics, data and business intelligence to measure, control and improve operational performance and profitability for carriers operating in the commercial over-the-road trucking sector.

Tiemac sees an opportunity: 98% of all trucking companies (carriers) are made up of 50 or less trucks in their fleets. These small carriers lack size, scale, ability to combine efforts, capacity and resources and generally operate within the confines of large enterprises fleets. From a helicopter viewpoint, Tiemac is interested in using data from the electronic logging device (ELD) install in trucks to analyze the performance of different types of truck manafacturer engines. The goal is to use the data to compare the performance of the manufacturers vehciles evaluating the general reliability of trucks engines from these manufacturers. This could help to inform the 98% of the market on which commercial truck manaufacture generally have trucks with more operating reliability for their individual use cases.

To achieve its goal, Tiemac has asked me to build a predictive maintenance model. In leiu of real data taken from Tiemac's CrewAccount sytems deployed in trucks across North America, I am going to use two other data souces for the datasets to be used in this project.

- Automotive Vehicles Engine Health Dataset. This dataset is from Kaggle and will be used under license of CCO 1.0 Universal Public Dedication.
- A Random dataset will be created to represent 5 different vehicle manaufacturers. Here the term vehicle manufacturers is to interpreted as commerical vehicle engine manufacturers. Although there are specialist commercial vechicle engine manufactures, the 5 vehicle manufacturers mentioned here do produce/sell their own engines. The assumption therefore is that engines being analysed here are the manufacturer's own engines.

In my analysis, the approach that will be taken is to use supervised machine learning algorithms, including regression.

Before building and training the algorithms, it will be necessary to perform ETL jobs and build ML pipelines. Therefore, in this project, using the referenced datasets, we preprocess the datasets. Such preprocessing will include, dropping any duplicate rows, and removing the rows with null/missing values, if necesssary, scaling features to be within a certain range and encoding categorial vairables such as the make of vehicles.

Once ETL is completed we will create different models, incuding regressions and ML pipe lines to create a preditive mainteance model that could be run against the similar dataset to generate alerts and or recommendations for maintenance and repairs based on vehicle engine manufacturers. As part of this process we will evaluate the model and persist the model.

2.1 Objectives

In this project we will:

- Part 1 Perform ETL activity
  - Import initial set of required libraries
  - Create a Spark Session
  - Load a csv dataset and create PySpark dataframe
  - Print top 5 rows of dataframe
  - Count totoal rows and display schema
  - Remove duplicates if any
  - Drop rows with null values if any
  - Rename columns to remove spaces in variable names and make them lower case
  - Create a 2nd dataset with truck manufacturers
  - Create a dataframe from the truck manufacturer dataset and randomly populatte rows without null values
  - join the two dataframes to create a new dataframe
  - Check point evaluation
  - Store the merged and cleaned data in parquet file format
- Part 2 Retrieve saved dataframe
  - Load datasete from previously saved parquet file
  - Count number of rows
  - Show the first 20 records
- Part 3 Visualize Variables and Distributions
  - Install required packakges
  - Import required libraries
  - Convert PySpark dataframe to Pandas Dataframe
  - Shape Pandas Dataframe to count rows and columns
  - Create Series of Scatter Plots
  - Create Box Plot
  - Create Histogram Plots and Correlation Matrix
- Part 4 Build Logistic Regression Models
  - Exploration with Manufacturers as target variable
  - Initial analysis on 1st of Logistic Regression exploration
  - Exploration with Engine Condition as target variable
    - Get Required Libraries
    - Plot Types & Count Engine Condition
    - Split Data Set into Training & Test sets
    - Build & Train Classifier
    - Calculate Metrics
- Part 5 Examining Evaluation Metrics

  - Summarize
  - Analysis
  - Final Remarks on initial ETL Proces
- Part 6 Create Baseline Machine Learning Pipeline Model
  - Get and Transform the PysSpark Dataframe
  - Import required functions & define Vector Assemble
  - Instantiate Classifier from SparkML
  - Import additional required ML functions, Build, Train and Evaluate
    * Build Pipeline
    * Train Model
    * Predict
    * Evaluate
- Part 7 Create Desired ML Pipeline Model - Project Essence
  - Import required ML functions
    * Create String Indexer
    * One Hot Encode
  - Create Vector Assembler
  - Build Classifier
  - Build Pipeline
  - Fit Model
  - Model Predict
  - Show Model
  - Evaluate Model
  - Interpret Model
- Part 8 Training the desired ML Model
  - Split Data Set
  - Fit desired ML Model to Training Data Set
  - Predict desired ML Model with Training Data Set
  - Show the desired ML Model with Training Data Set
  - Evaluate the desired trained ML Model on Trained Data Set
- Part 9 Predict and Evaluate the ML Model with Testing Data Set
  - Predict with ML Model on Testing Data Set
  - Calculate and Print MSE
  - Calculate and Print MAE
  - Calculate and Print R-Squared (R2)
  - Summarize ML Model with Testing Data Set
- Part 10 Persist the ML Model
  - Save the desired ML model for future production use
  - Load the stored ML model
  - Make predicitions on Test Data
  - Show predictions
- Part 11 Decode the One-Hot Encode ML Model Prediction
  - Method 1
  - Method 2
- Part 12 Project Conclusion

2.2 Datasets

The dataset(s):

- Automotive Vehicle Engine Health Dataset - The original dataset may be found here https://www.kaggle.com/datasets/parvmodi/automotive-vehicles-engine-health-dataset
- Randomly created dataset of 5 commerical vehicle (engine) manufacturers that will be randomly matched to each row (record) of the Automotive Vehicle Enginer Health Dataset.

2.3 Setup

2.3.1 Installing Required Libraries

Spark Cluster environment with libraries like pyspark and findspark to connect to this cluster.

[138]:
```
!pip install pyspark==3.5.0 -q
!pip install findspark -q
```

2.3.2 Importing Required Libraries

We recommend you import all required libraries in one place (here):

[139]:
```
import findspark
findspark.init()
```

2.4 Part 1 - Perform ETL activity

2.4.1 Task 1 - Import required libraries

[140]:
```
#your code goes here
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.pipeline import PipelineModel
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import StandardScaler
```

2.4.2 Task 2 - Create a spark session

[141]:
```
#Create a SparkSession
spark = SparkSession.builder.appName("Capstone Project").getOrCreate()
```

2.4.3 Task 3 - Load the csv file into a dataframe

Download the data set from Kaggle.

NOTE : The data file downloaded from Kaggle is a zipped file. Therefore, it was downloaded to local environment and then will be loaded from the local drive into spark cluster.

Load the dataset into the spark dataframe

[142]:
```
# Load the dataset that you have downloaded in the previous task

df1 = spark.read.csv('///E:\\MLCapstone\\engine_data\\engine_data.csv',␣
↪header=True, inferSchema=True)
```

2.4.4 Task 4 - Print top 5 rows of the dataset

[143]:
```
#your code goes here
df1.show(5)
```

```
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+
|Engine rpm|Lub oil pressure|Fuel pressure|Coolant pressure|lub oil temp|Coolant
temp|Engine Condition|
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+
|       700|     2.493591821|  11.79092738|     3.178980794| 84.14416293|
81.6321865|               1|
|       876|     2.941605932|  16.19386556|     2.464503704| 77.64093415|
82.4457245|               0|
|       520|     2.961745579|  6.553146911|     1.064346764| 77.75226574|
79.64577667|               1|
|       473|     3.707834743|  19.51017166|     3.727455362| 74.12990715|
71.77462869|               1|
|       619|     5.672918584|  15.73887141|     2.052251454| 78.39698883|
87.00022538|               0|
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+
only showing top 5 rows
```

2.4.5 Task 6 - Print the total number of rows in the dataset and print the schema

[144]:
```
#your code goes here
rowcount1 = df1.count()
print(rowcount1)
```

```
19535
```

[145]:
```
df1.printSchema()
```

```
root
 |-- Engine rpm: integer (nullable = true)
 |-- Lub oil pressure: double (nullable = true)
 |-- Fuel pressure: double (nullable = true)
 |-- Coolant pressure: double (nullable = true)
 |-- lub oil temp: double (nullable = true)
 |-- Coolant temp: double (nullable = true)
```

```
 |-- Engine Condition: integer (nullable = true)
```

2.4.6 Task 7 - Drop all the duplicate rows from the dataset - if any

[146]:
```
df1 = df1.dropDuplicates()
```

2.4.7 Task 8 - Print the total number of rows in the dataset to check if any duplicate rows were present and henced dropped

[147]:
```
#count the records again to see if duplicates were present and dropped

rowcount2 = df1.count()
print(rowcount2)
```

```
19535
```

2.4.8 Task 9 - Drop all the rows that contain null values from the dataset - if any

[148]:
```
df1=df1.dropna()
```

2.4.9 Task 10 - Print the total number of rows in the dataset to see if any null value rows were present and dropped

[149]:
```
#count the remaining rows

rowcount3 = df1.count()
print(rowcount3)
```

```
19535
```

From the above count checks we can see that the original dataset contained no duplicates and no null values

2.4.10 Task 11 - Rename the columns in the dataframe to remove spaces in the names and use all lower cases

[150]:
```
from pyspark.sql.functions import col

new_columns = ['engine_rpm', 'lub_oil_pressure', 'fuel_pressure',␣
↪'coolant_pressure', 'lub_oil_temp', 'coolant_temp', 'engine_condition']

for i,n in zip(df1.columns,new_columns):
    df1 = df1.withColumnRenamed(i,n)
```

[151]:
```
df1.show(5)
```

```
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+
```

```
|engine_rpm|lub_oil_pressure|fuel_pressure|coolant_pressure|lub_oil_temp|coolant
_temp|engine_condition|
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+
|       501|      2.99477498|   4.47831813|     1.637320865| 76.78483587|
63.97067607|               0|
|       780|     2.594671525|   5.25603517|     2.114896407| 74.99287021|
75.52746711|               0|
|      1086|     2.625588801|  3.765344712|     1.811358686| 77.04368528|
82.62726219|               0|
|      1384|     0.259479674|  12.14284087|     3.785984041| 77.78951386|
86.34632501|               1|
|       782|     1.961151236|   8.56439345|     2.511618166| 76.43830417|
77.37116979|               0|
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+
only showing top 5 rows
```

2.5 Task 12 - Create Second dataset

2.5.1 The Automotive Vehicle Engine Health Dataset does not contain information on vehicle engine manufacturer. Therefore, we are going to create another dataset with vehicle engine manufacturer information

Step 1 - is to define the number of rows as 19535, which is equal to the number of rows in dataframe df1 above and set columns = 1

Step 2 - is to create a list of vehicle engine manufacturers names. We will randomly select 5 commercial truck engine manufacturers' names, as follows: (1) Volvo (2) Mack (3) Freightliner (4) International (5) Kenworth

Step 2 - is to use random.choice function to randomly inject names from the manufacturers list into the rows

Step 4 - is to create a new dataframe with a single column containing the random selected names

[152]:
```python
import random
from pyspark.sql.functions import rand

# Define the number of rows and columns
rows = 19535
columns = 1

#Create the list of manufacturer's name
manufacturer = ['Volvo', 'Mack', 'Freightliner', 'International', 'Kenworth']

#Create a list of random manufacturers
random_manufacturer = [random.choice(manufacturer) for value in range(rows)]
```

```python
# Create a new dataframe with a single column containing the random
 ↪manufacturers
df2 = spark.createDataFrame([(value,) for value in random_manufacturer],
 ↪['manufacturers'])

#show the first 20 rows of this new dataframe
df2.show(20)
```

```
+-------------+
|manufacturers|
+-------------+
|International|
| Freightliner|
| Freightliner|
|         Mack|
|         Mack|
| Freightliner|
| Freightliner|
| Freightliner|
| Freightliner|
|         Mack|
| Freightliner|
|     Kenworth|
|        Volvo|
|        Volvo|
|     Kenworth|
|     Kenworth|
|International|
|     Kenworth|
|         Mack|
| Freightliner|
+-------------+
only showing top 20 rows
```

2.5.2 Task 13 - Print the total number of rows in this newly created dataset

[153]:
```python
#count the number of rows to check we have the same number of records as the in
 ↪the previous dataframe

rowcount4 = df2.count()
print(rowcount4)
```

```
19535
```

2.5.3 Task 14 - Create one new dataframe from the two dataframes above

Note - The method use below for joining the two dataframes is based on creating IDs. However, the ids are based on the number of partitions. So if the two DataFrames have a different number of partitions, this won't be guaranteed to work to preserve the same number of records in the new merged dataframe. However, a work around that will be used is to set the number of partions for each dataframe to be the same and to be large enough to capture most if not all the records. Once we have sufficient records remaining we can proceed.

[154]:
```
df1 = df1.repartition(36)
df2 = df2.repartition(36)
```

[155]:
```
# In creating the new dataframe, we are going to use the monotically function␣
↪to add and ID column to both dataframes and then use the ID column to join␣
↪creating a new dataframe
from pyspark.sql import functions as F
from pyspark.sql.functions import monotonically_increasing_id
df1 = df1.withColumn("id", monotonically_increasing_id())
df2 = df2.withColumn("id", monotonically_increasing_id())

df3 = df1.join(df2, "id", "inner").drop("id")
```

[156]:
```
#show the first 20 rows of this new dataframe to see if the join works as␣
↪intended
df3.show(20)
```

```
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+-------------+
|engine_rpm|lub_oil_pressure|fuel_pressure|coolant_pressure|lub_oil_temp|coolant
_temp|engine_condition|manufacturers|
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+-------------+
|       628|     1.998738288|  8.219071968|     1.457350001| 76.24542113|
92.02865188|               1|         Mack|
|       734|     4.148434572|   5.44578528|     1.430749525| 74.86696406|
84.89413913|               0|         Mack|
|       827|     4.313316622|  5.883723802|     3.266018879| 88.08943077|
75.50174012|               1|         Mack|
|       449|     2.541603901|  10.58610131|     3.576934088| 83.81795456|
79.16815651|               1|         Mack|
|      1096|     2.655848146|  4.646055253|     4.548715226| 75.92151314|
80.44826376|               1|        Volvo|
|       984|     2.492219616|  7.562085591|     2.026100067| 75.95299784|
74.48222711|               1|        Volvo|
|       649|     5.029143027|  5.083812853|     1.630588966| 76.75175212|
75.21988223|               1|        Volvo|
|       650|     3.169027313|  7.147442201|     3.078595125| 76.90938115|
87.17244583|               1|  Freightliner|
|      1425|     2.750815791|  2.677409041|     2.668441438| 83.38031364|
```

```
75.70386153|                 0| Freightliner|
|        673|     2.931258873|  6.155399147|     1.927421039|  76.9602504|
84.57597483|                 1| Freightliner|
|        704|     4.726887297|   8.06300596|     1.542422251| 74.11413484|
73.08409644|                 1|     Kenworth|
|        850|     3.671445967|  11.32976593|     2.592374639| 81.22461223|
76.29509474|                 1|     Kenworth|
|        764|     3.062610487|  4.620158063|     1.408026221| 77.37202096|
73.75508941|                 1|     Kenworth|
|        468|     2.989145428|  7.634780461|      1.35179479|  76.0566735|
70.86934932|                 1|     Kenworth|
|       1350|     2.551872217|  9.744406846|      5.33316718| 77.80713634|
81.91453674|                 0|International|
|        579|     3.870106059|   9.21702345|     1.767519447| 75.63777667|
74.12656436|                 0|International|
|        518|      4.86285687|   3.71893686|     2.630116962| 77.46924689|
83.72439214|                 1|International|
|        437|     2.968921935|  5.094935304|     2.301481623| 76.48690915|
83.4465197|                 1|         Mack|
|        726|     2.937762143|  7.208258611|     4.640791867| 84.71307087|
74.72029976|                 0|         Mack|
|        677|     3.277467593|  5.379075964|     1.243165123| 75.54849518|
78.75349725|                 1|         Mack|
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+-------------+
only showing top 20 rows
```

[157]:

```python
#count the number of rows to check if we have sufficient records to proceed.

rowcount5 = df3.count()
print(rowcount5)
```

```
19515
```

2.5.4 Task 15 - Check Point Evaluation

The code cell below is to just create a check point. It provides a summary of the data wrangling done. If the code throws up any errors, we will go back and review the code written.

[158]:

```python
print("Part 1 - Evaluation")

print("Total rows = ", rowcount1)
print("Total rows after dropping duplicate rows = ", rowcount2)
print("Total rows after dropping duplicate rows and rows with null values = ",␣
 ↪rowcount3)
print("Total rows after creating dataframe randomly populated with␣
 ↪manufacturers = ", rowcount4)
```

```
print("Total rows after merging dataframes = ", rowcount5)

import os

print("engine_man_data.parquet exists :", os.path.isdir("engine_man_data.
↪parquet"))
```

```
Part 1 - Evaluation
Total rows =  19535
Total rows after dropping duplicate rows =  19535
Total rows after dropping duplicate rows and rows with null values =  19535
Total rows after creating dataframe randomly populated with manufacturers =
19535
Total rows after merging dataframes =  19515
engine_man_data.parquet exists : True
```

2.5.5 Task 16 - Save the merged dataframe in parquet format, name the file as "engine__man__data.parquet"

[159]:
```
# We are saving the dataframe in parquet format to take advantage of the␣
↪benefits the parquet file format offers such as efficient storage, faster␣
↪query performance and schema preservation for evolution
df3.write.mode("overwrite").parquet("engine_man_data.parquet")
```

2.6 Part - 2 Retrieve the saved Dataframe

2.6.1 Task 1 - Load data from "engine__man__data.parquet" into a dataframe

[160]:
```
#load the merged dataset

df4 = spark.read.parquet("engine_man_data.parquet")
```

2.6.2 Task 2 - Print the total number of rows in the dataset

[161]:
```
#show the total number of rows in the loaded dataset

rowcount6 = df4.count()
print(rowcount6)
```

```
19515
```

2.6.3 Task 3 - Show the first 20 records in the dataset

[162]:
```
#show top 20 rows
df4.show(20)
```

```
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+-------------+
|engine_rpm|lub_oil_pressure|fuel_pressure|coolant_pressure|lub_oil_temp|coolant
```

```
_temp|engine_condition|manufacturers|
+----------+---------------+------------+---------------+-----------+-------
-----+---------------+-------------+
|       811|     5.452593264| 5.079589925|     2.680202261| 76.54589478|
90.08560545|               0|        Mack|
|       558|     4.443161459| 5.610184593|     3.112912175| 76.66414132|
67.08213633|               1|        Mack|
|      1234|     2.981909274| 12.63864452|     2.102405305| 77.37659228|
74.82047801|               1|        Mack|
|       469|     4.328405379| 5.216832349|     6.625636587| 76.09339431|
78.95197818|               1|       Volvo|
|       770|     2.961174741| 2.240656493|      2.31300994| 74.53125761|
80.90553092|               0|       Volvo|
|       619|     3.934368926| 7.698462273|     2.278726856| 86.06308412|
88.42011639|               1|       Volvo|
|      1606|     4.398524811| 5.503495054|     1.675167803| 76.97447105|
87.88562744|               1| Freightliner|
|       728|     2.483021887| 6.562360645|      1.72643451| 74.17618041|
76.73653723|               1| Freightliner|
|       524|     2.271233227| 5.599198784|     3.374839294| 75.19588881|
80.25312165|               0| Freightliner|
|       916|     2.331835634| 6.329489495|     2.220629546| 75.21374294|
84.35683965|               0| Freightliner|
|       581|     2.532502622| 5.211893945|     1.020389138| 84.03211401|
74.97293084|               0|    Kenworth|
|       696|     3.229960878| 13.50813113|     2.711880305| 74.91004261|
76.23301589|               1|    Kenworth|
|       523|     2.768377533| 10.41567564|     1.603632275| 76.31189436|
74.70228332|               1|    Kenworth|
|       848|     4.143512581|  2.86629221|     2.521902353| 86.77647934|
77.87564948|               1|International|
|       823|     2.080203637| 7.236541038|     2.573919401| 77.34109804|
81.94345215|               0|International|
|      1031|     4.899692719| 4.732332599|     1.347354638| 78.53583698|
77.58817477|               0|International|
|       723|     2.359133956| 10.50397534|     2.719321943| 76.05518118|
74.4970153|               0|International|
|       814|     3.159342477| 5.551390389|     3.180498063| 77.95522031|
85.68445037|               0|        Mack|
|       819|     2.383461976| 6.738287649|     3.231114826| 77.41499188|
78.06126422|               1|        Mack|
|       680|     0.808593578| 3.780912195|     0.947805614| 77.76832404|
71.91843029|               0|        Mack|
+----------+---------------+------------+---------------+-----------+-------
-----+---------------+-------------+
only showing top 20 rows
```

3 Part 3 - Visualize relationships and variables distributions

To more easily visaulize through graphing we are going to go through a couple steps - Step 1 - install required packages - Step 2 - import required libraries - Step 3 - convert the PySpark dataframe to a Pandas Dataframe. - Step 4 - create a series of scatter plots to look at possible correlation relationships

3.0.1 Step 1 - install required packages

[163]:
```
# install required package for visulization
!pip install pandas
!pip install scikit-learn
!pip install numpy
!pip install matplotlib
```

```
Requirement already satisfied: pandas in e:\anaconda\anaconda3\lib\site-packages
(1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in
e:\anaconda\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in e:\anaconda\anaconda3\lib\site-
packages (from pandas) (2022.7)
Requirement already satisfied: numpy>=1.21.0 in e:\anaconda\anaconda3\lib\site-
packages (from pandas) (1.24.3)
Requirement already satisfied: six>=1.5 in e:\anaconda\anaconda3\lib\site-
packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
Requirement already satisfied: scikit-learn in e:\anaconda\anaconda3\lib\site-
packages (1.3.0)
Requirement already satisfied: numpy>=1.17.3 in e:\anaconda\anaconda3\lib\site-
packages (from scikit-learn) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in e:\anaconda\anaconda3\lib\site-
packages (from scikit-learn) (1.10.1)
Requirement already satisfied: joblib>=1.1.1 in e:\anaconda\anaconda3\lib\site-
packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
e:\anaconda\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: numpy in e:\anaconda\anaconda3\lib\site-packages
(1.24.3)
Requirement already satisfied: matplotlib in e:\anaconda\anaconda3\lib\site-
packages (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in
e:\anaconda\anaconda3\lib\site-packages (from matplotlib) (1.0.5)
Requirement already satisfied: cycler>=0.10 in e:\anaconda\anaconda3\lib\site-
packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
e:\anaconda\anaconda3\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
e:\anaconda\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.20 in e:\anaconda\anaconda3\lib\site-
```

```
packages (from matplotlib) (1.24.3)
Requirement already satisfied: packaging>=20.0 in
e:\anaconda\anaconda3\lib\site-packages (from matplotlib) (23.0)
Requirement already satisfied: pillow>=6.2.0 in e:\anaconda\anaconda3\lib\site-
packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
e:\anaconda\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
e:\anaconda\anaconda3\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in e:\anaconda\anaconda3\lib\site-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

3.0.2 Step 2 - import required libraries

[164]:
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
```

3.0.3 Step 3 - convert the PySpark dataframe to a Pandas Dataframe.

[165]:
```
# Convert the PySpark DataFrame to a Pandas DataFrame
pdf1 = df4.toPandas()
```

[166]:
```
# show 5 random rows from the pandas dataset
pdf1.sample(5)
```

[166]:
```
       engine_rpm  lub_oil_pressure  fuel_pressure  coolant_pressure  \
18488         559          4.417817      10.065641          2.355551
5078         1356          3.869504       2.358005          4.906734
800           641          4.582983       4.772423          2.413080
7309          913          2.871362       6.143667          3.194376
977           685          2.415466       6.113998          2.637209

       lub_oil_temp  coolant_temp  engine_condition manufacturers
18488     81.962337     72.678071                 0          Mack
5078      80.799674     75.002857                 0      Kenworth
800       76.017130     74.647635                 0          Mack
7309      77.073719     74.248095                 0   Freightliner
977       77.006586     79.737073                 1      Kenworth
```

[167]:
```
# Let's find out the number of rows and columns in the pandas dataset:
pdf1.shape
```

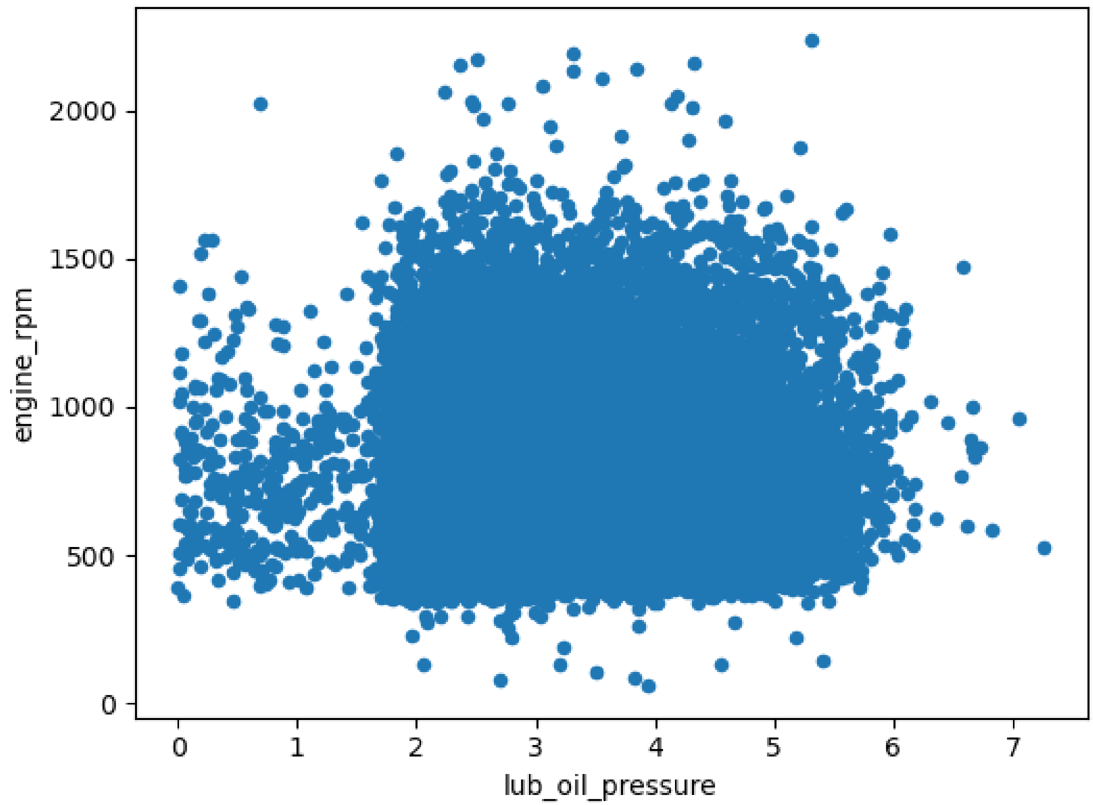
[167]:
```
(19515, 8)
```

3.0.4 Step 4 - create a series of scatter plots to look at possible correlation relationships

[168]:
```
# Let's create a scatter plot of lube oil pressure engine rpm . This will help␣
↪us visualize the relationship between them.
```

[169]:
```
pdf1.plot.scatter(x = "lub_oil_pressure", y = "engine_rpm")
```

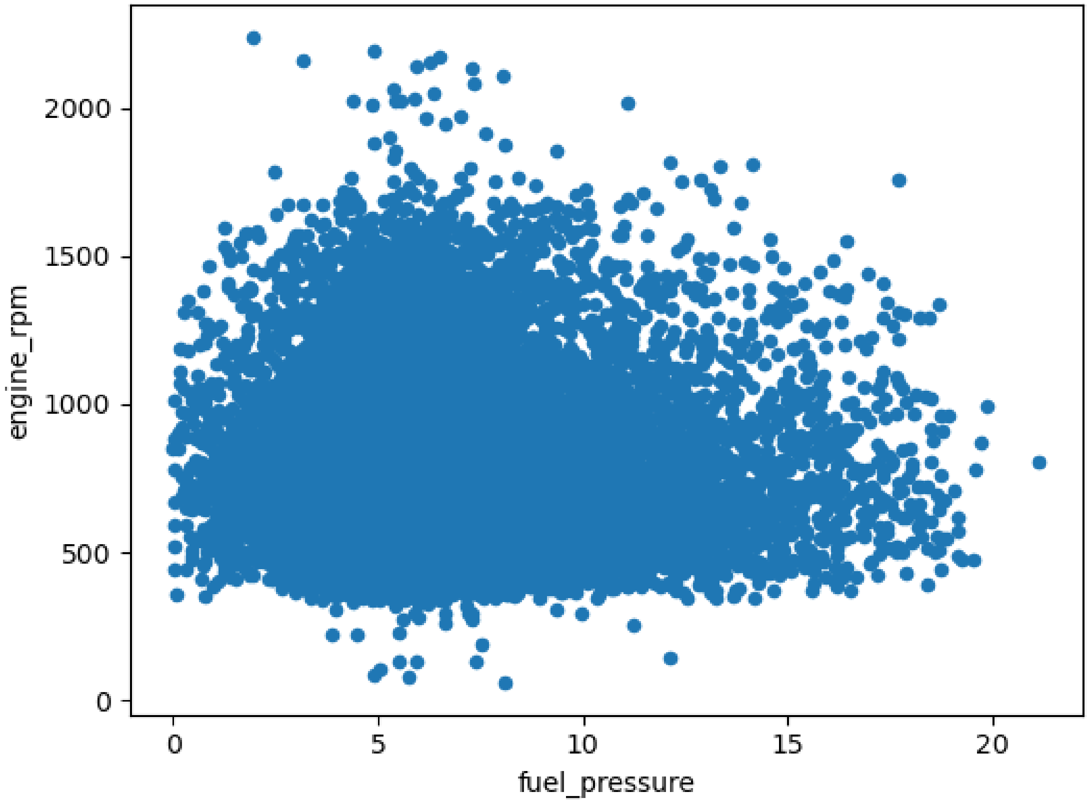
[169]: <Axes: xlabel='lub_oil_pressure', ylabel='engine_rpm'>



[170]:
```
# Let's create a scatter plot of fuel pressure vs engine rpm . This will help␣
↪us visualize the relationship between them.
```

[171]:
```
pdf1.plot.scatter(x = "fuel_pressure", y = "engine_rpm")
```

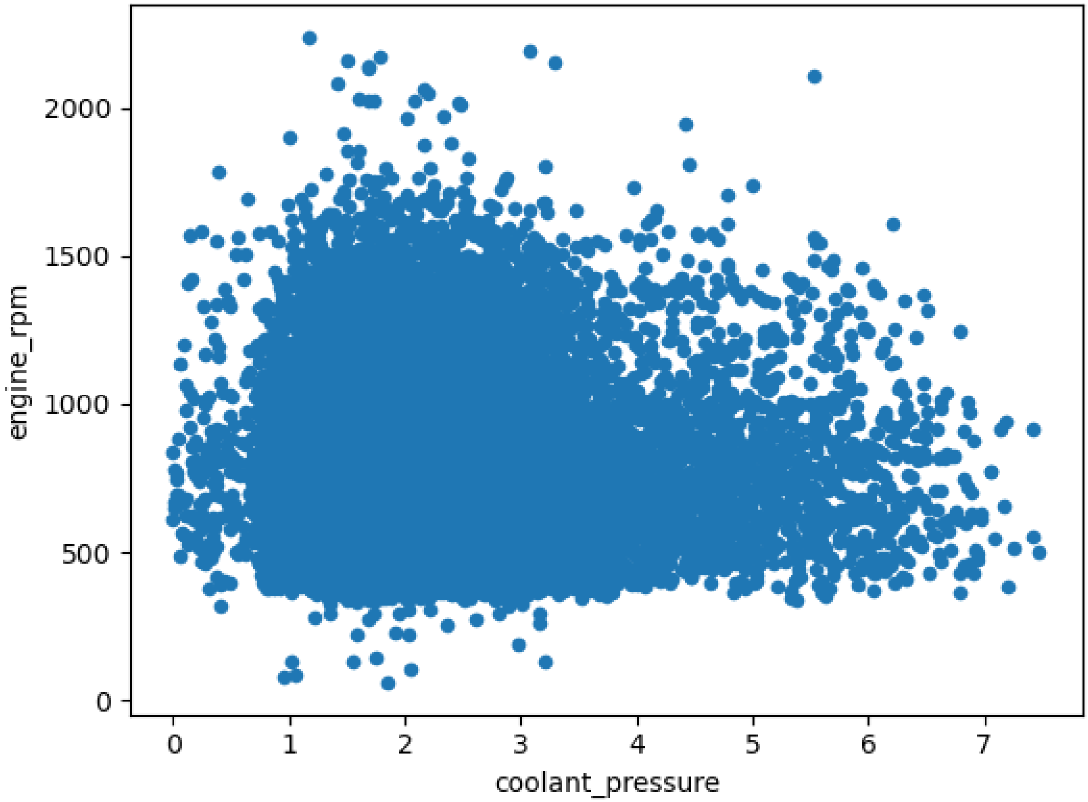
[171]: <Axes: xlabel='fuel_pressure', ylabel='engine_rpm'>



[172]:
```
# Let's create a scatter plot of coolant pressure vs engine rpm . This will␣
↪help us visualize the relationship between them.
```

[173]:
```
pdf1.plot.scatter(x = "coolant_pressure", y = "engine_rpm")
```

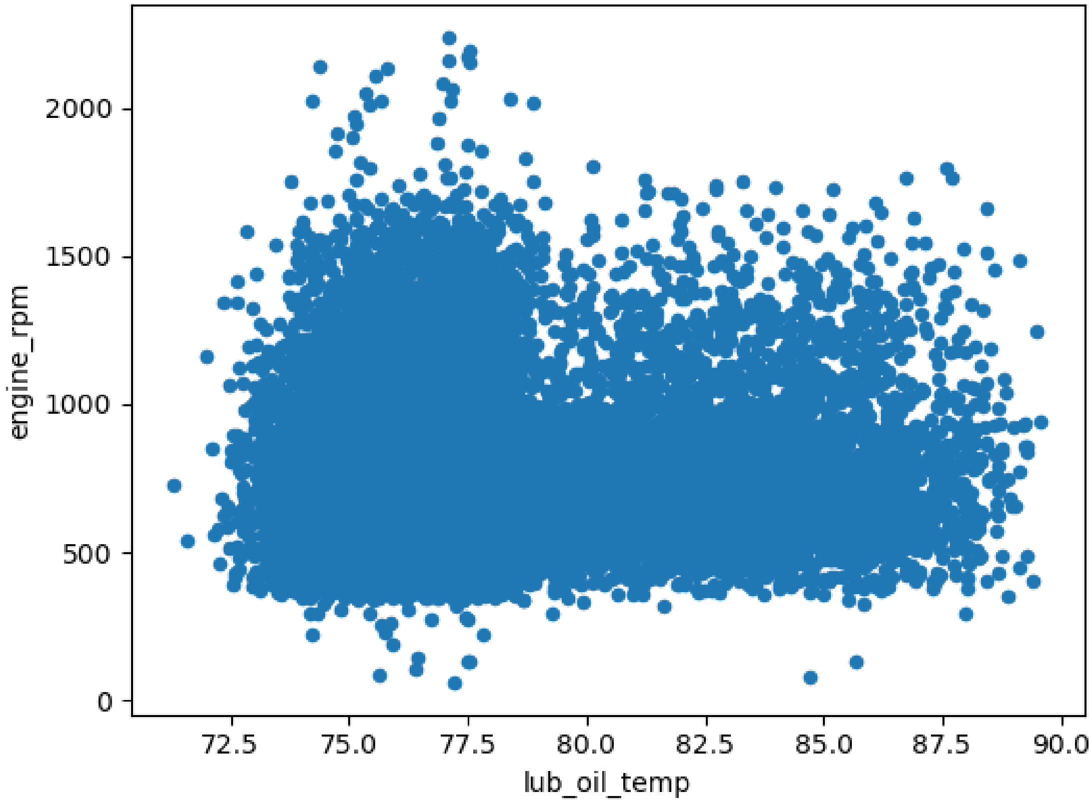
[173]: `<Axes: xlabel='coolant_pressure', ylabel='engine_rpm'>`



[174]:
```
# Let's create a scatter plot of lube oil temperature vs engine rpm . This will
↪help us visualize the relationship between them.
```

[175]:
```
pdf1.plot.scatter(x = "lub_oil_temp", y = "engine_rpm")
```

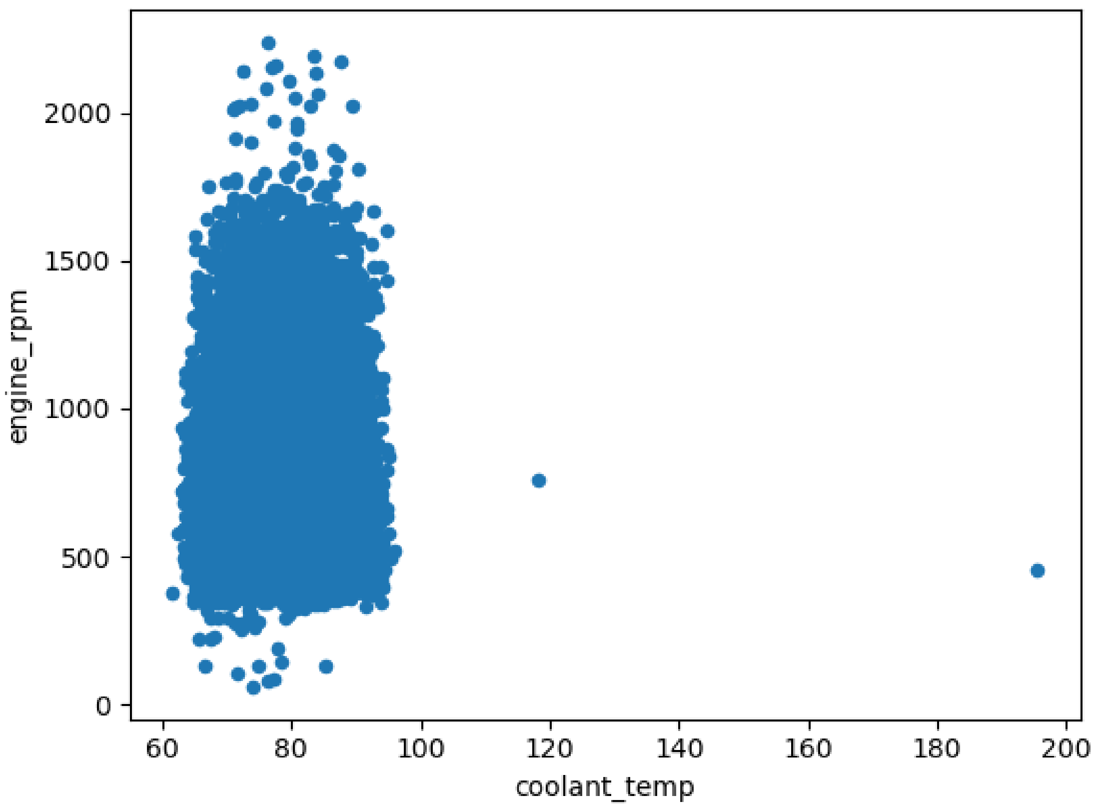
[175]: <Axes: xlabel='lub_oil_temp', ylabel='engine_rpm'>



[176]:
```
# Let's create a scatter plot of coolant temperature vs engine rpm . This will
↪help us visualize the relationship between them.
```

[177]:
```
pdf1.plot.scatter(x = "coolant_temp", y = "engine_rpm")
```

[177]: <Axes: xlabel='coolant_temp', ylabel='engine_rpm'>

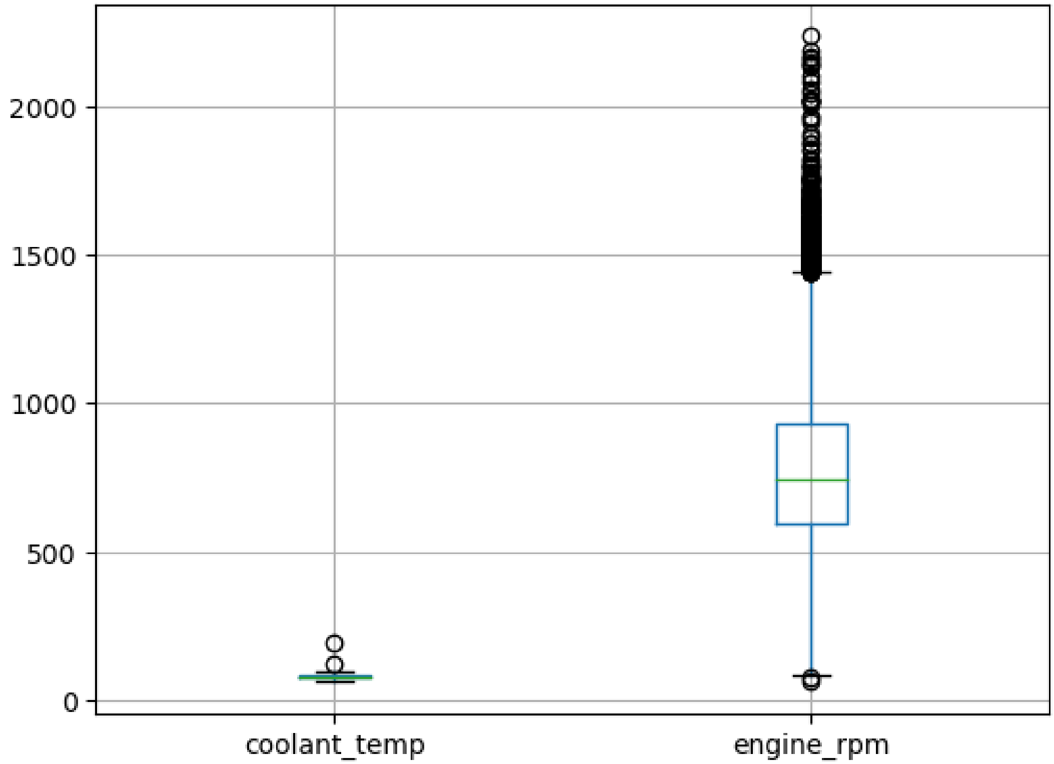


3.0.5 Step 5 - Create a Box Plot

The scatterplot created above for coolant_temp vs engine_rpm reveals potential outliers. Therefore, we are going to look at a boxlot of each of those two variables for closer examination.

[178]:
```
# create a box plot of coolant_temp and engine_rpm variables
pdf1.boxplot(column=['coolant_temp', 'engine_rpm'])
```

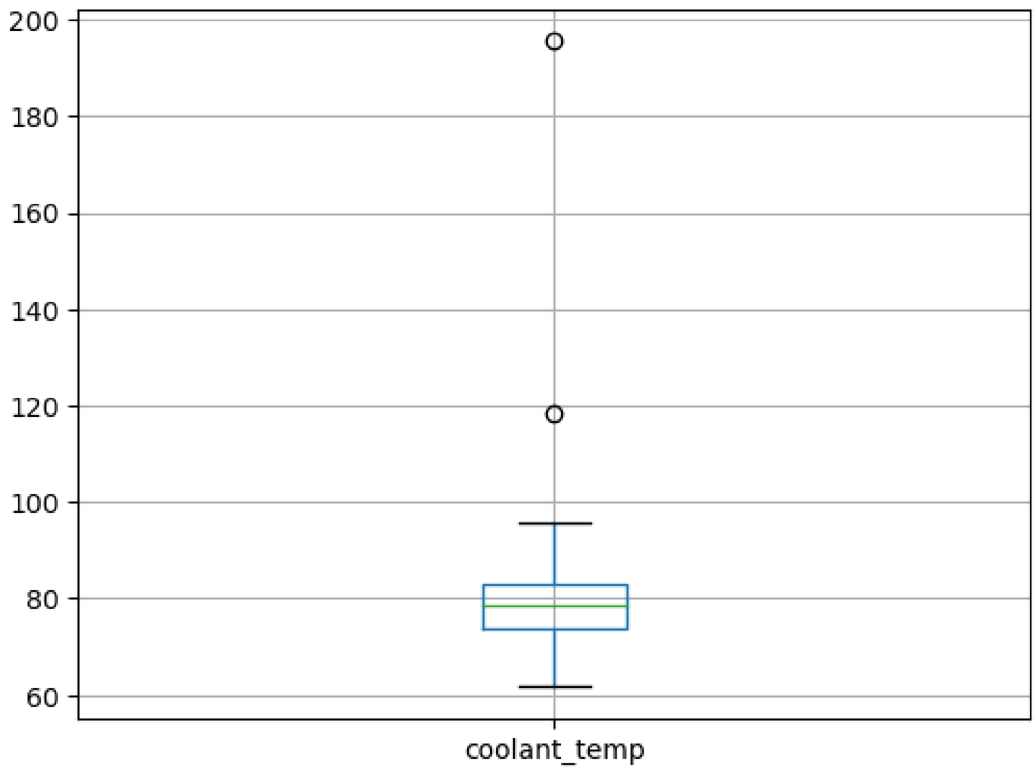
[178]: <Axes: >



Given the different scales for the two variables we are going to create a separate boxplot for coolant temp so we may have a better view of the related data.

[179]:
```
# create a box plot of the coolant_temp variable
pdf1.boxplot(column=['coolant_temp'])
```

[179]: <Axes: >



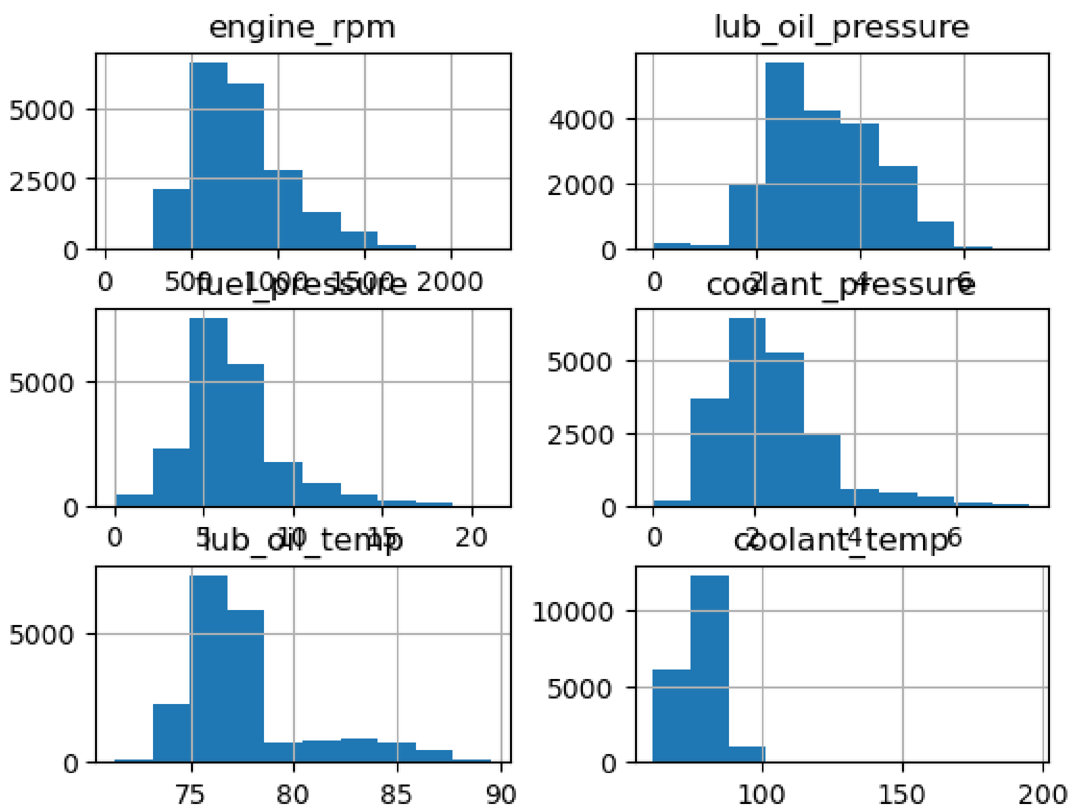
Indeed the scatter plots do reveal deeper information on these two variables - The coolant_temp variable shows the two outliers. It appears there is relatively little skewness in data with the mean/centrality of the data being just under 80 - The engine_rpm variable does show that a lot of the datapoints lie outside and mostly to the right (top) of the "wisker" and with a "long tail" - meaning right skewed, with the median(centrality) of the data being closer to the box lower values and the upper wisker is long. - Further data wrangling and transformation may be needed in order to correct possible issues after consulting with subject matter domain experts on vehicle engine operations data range, etc.

3.0.6 - Step 6 - Create Histogram plots and Correlation Matrix of the variables

Let us look at the histogram of the first 6 columns of the pandas dataframe

[180]:
```
# lets plot the histograms
pdf1[['engine_rpm', 'lub_oil_pressure', 'fuel_pressure', 'coolant_pressure',␣
↪'lub_oil_temp', 'coolant_temp']].hist()
```

[180]:
```
array([[<Axes: title={'center': 'engine_rpm'}>,
        <Axes: title={'center': 'lub_oil_pressure'}>],
       [<Axes: title={'center': 'fuel_pressure'}>,
        <Axes: title={'center': 'coolant_pressure'}>],
       [<Axes: title={'center': 'lub_oil_temp'}>,
        <Axes: title={'center': 'coolant_temp'}>]], dtype=object)
```



Lets look at the correlation matrix among the first 6 columns of the pandas dataframe

[181]:
```
# select the first 6 columns to include in the correlation matrix
pdf1_subset1 = pdf1[['engine_rpm', 'lub_oil_pressure', 'fuel_pressure',␣
↪'coolant_pressure', 'lub_oil_temp', 'coolant_temp']]

# calculate the correlation matrix
corr_matrix = pdf1_subset1.corr()

print(corr_matrix)
```

```
                  engine_rpm  lub_oil_pressure  fuel_pressure  \
engine_rpm          1.000000          0.025254      -0.001564
lub_oil_pressure    0.025254          1.000000       0.043127
fuel_pressure      -0.001564          0.043127       1.000000
coolant_pressure   -0.025053         -0.009591       0.033327
lub_oil_temp        0.052303         -0.007958      -0.025037
coolant_temp        0.029091         -0.060808      -0.042949

                  coolant_pressure  lub_oil_temp  coolant_temp
engine_rpm               -0.025053      0.052303      0.029091
lub_oil_pressure         -0.009591     -0.007958     -0.060808
```

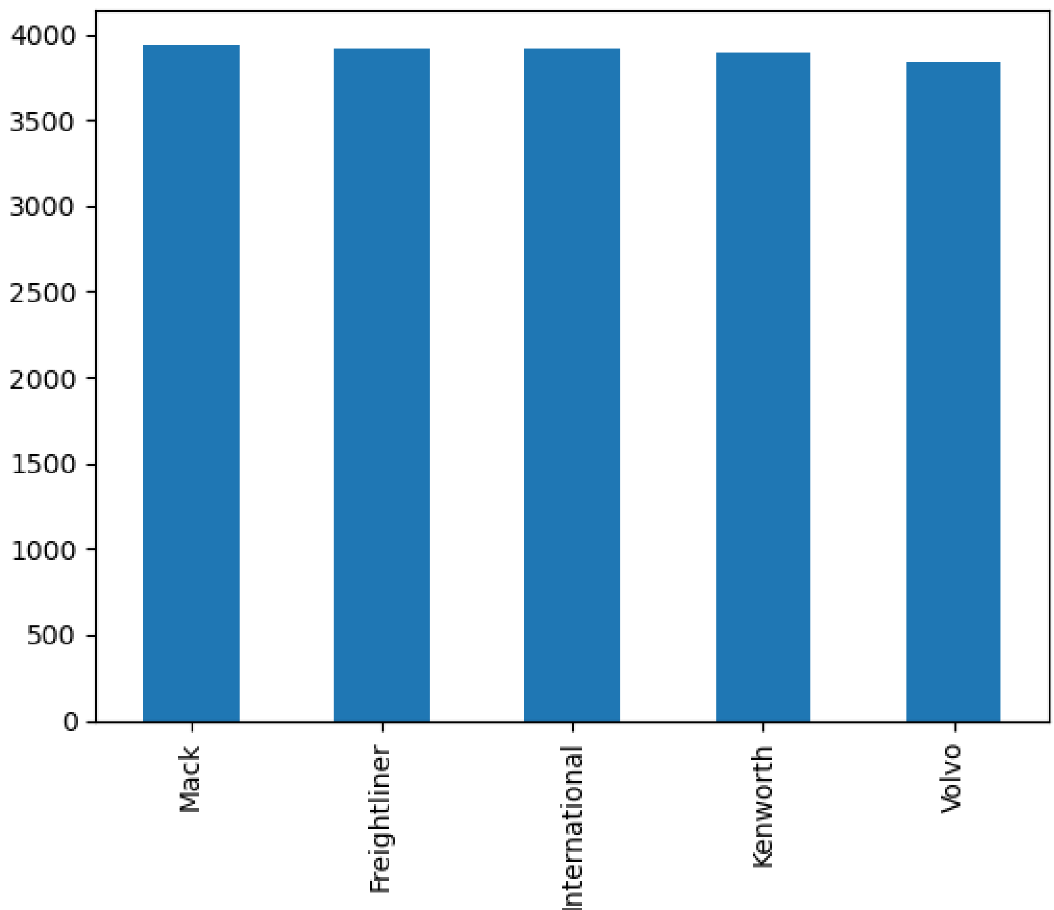
```
fuel_pressure              0.033327    -0.025037    -0.042949
coolant_pressure           1.000000    -0.020339     0.033413
lub_oil_temp              -0.020339     1.000000     0.073396
coolant_temp               0.033413     0.073396     1.000000
```

Let's plot the manufacturers and count

[182]:
```python
from sklearn.linear_model import LogisticRegression
```

[183]:
```python
pdf1.manufacturers.value_counts().plot.bar()
```

[183]: <Axes: >



3.1 Part 4 - Building Logsitic Regression Models

We can see that the 5 commercial truck vehicle engine manufactueres that we randomly distributed against the original Kaggle Automotive Vehicle Engine Health Dataset is relatively evenly distributed in terms of counts.

3.1.1 Step 1 - Logistic Regression Exploration with manufacturers as target

- We are going to explore the dataframe further to see if there is a logistic regression relationship that may possible be established with some relatively high degree of predictability between a feature set (lub_oil_pressure, fuel_pressure, coolant_pressure, lub_oil_temp and coolant_temp) and a target (manufacturers).
- A question we may be interested in - Is there a possibility, for example, that through machine learning we may gain insights of whether a unique combination of values in the feature set signal a leaning towards one particular make of truck engines over the others.

[184]:
```
target = pdf1["manufacturers"]
```

[185]:
```
features = pdf1[["lub_oil_pressure", "fuel_pressure", "coolant_pressure",␣
↪"lub_oil_temp", "coolant_temp"]]
```

Build and train a classifier - Step 1 - Create Logistic Regression Model - Step 2 - Train/Fit the model

[186]:
```
classifier = LogisticRegression()
```

[187]:
```
classifier.fit(features,target)
```

[187]:
```
LogisticRegression()
```

Evaluate the model

[188]:
```
#Higher the score, better the model.
classifier.score(features,target)
```

[188]:
```
0.20563668972585192
```

The above shows a relatively low score, which means it is not a very good model of vehicle prediction. Nevertheless, for completeness we will try and make a prediction based on the following values - lub_oil_pressure = 3.75689 - fuel_pressure = 7.078912 - coolant_pressure = 1.984560 - lub_oil_temp = 78.348910 - coolant_temp = 77.896715

Before making the prediction lets get a summary statics for all numeric variables

[189]:
```
pdf1.describe()
```

[189]:
```
          engine_rpm  lub_oil_pressure  fuel_pressure  coolant_pressure  \
count  19515.000000      19515.000000   19515.000000      19515.000000
mean     791.139175          3.303988       6.656169          2.335031
std      267.541144          1.021625       2.760988          1.035967
min       61.000000          0.003384       0.003187          0.002483
25%      593.000000          2.518866       4.917632          1.600334
50%      746.000000          3.162117       6.202104          2.166883
75%      934.000000          4.055081       7.745124          2.848597
max     2239.000000          7.265566      21.138326          7.478505
```

```
       lub_oil_temp  coolant_temp  engine_condition
count  19515.000000  19515.000000      19515.000000
mean      77.643254     78.426461          0.630387
std        3.110442      6.207129          0.482712
min       71.321974     61.673325          0.000000
25%       75.725868     73.895421          0.000000
50%       76.817401     78.345955          1.000000
75%       78.072537     82.914990          1.000000
max       89.580796    195.527912          1.000000
```

[190]:
```
classifier.predict([[3.75689,7.078912,1.984560,78.348910,77.896715]])
```

```
E:\anaconda\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does
not have valid feature names, but LogisticRegression was fitted with feature
names
  warnings.warn(
```

[190]:
```
array(['Mack'], dtype=object)
```

Based on our machine learning model and the classfier predictor values we supplied, the results show that the possible manufacture is a Mack

We are going to do another logistic regression evaluation - this time we are going to keep the “target” as manufacturer, but use feature set as only the “engine rpm”

[191]:
```
target2 = pdf1["manufacturers"]
```

[192]:
```
features2 = pdf1[["engine_rpm"]]
```

[193]:
```
classifier2 = LogisticRegression()
```

[194]:
```
classifier2.fit(features2,target2)
```

[194]:
```
LogisticRegression()
```

[195]:
```
#Higher the score, better the model.
classifier2.score(features2,target2)
```

[195]:
```
0.202152190622598
```

[196]:
```
classifier2.predict([[2225]])
```

```
E:\anaconda\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does
not have valid feature names, but LogisticRegression was fitted with feature
names
  warnings.warn(
```

[196]:
```
array(['Mack'], dtype=object)
```

Based on our machine learning model using RPM as the classfier predictor value we supplied, the result show that the possible manufacture is a Mack engine make.

3.1.2 Step 2 - Initial analysis on these first sets of Logistic Regression Models

For both lostic regression models demonostrated above, the models were poor predictors. Keep in mind the following: - The Automotive Vehicles Engine Health Dataset is not related to real data ascertained from the specific named vehicle engine manufactures and the dataset is also not necessarily related to commercial vehicle engine use case under real worl operations. - The Engine Manufacturer dataset used was from data randomly created and populated to match each record in the Kaggle's Automotive Vehicles Enginer Health Dataset.

Despite the poor predictor models, we should not be too disheartened as there are some important basis for the models. - In the USA, almost all commercial trucks hauling freight across State lines are required to and have ELDs installed. These ELDs, do provide real world operating data that can be ascertained for the variables used in the The Automatve Vehicles Engine Health Dataset for each of the 5 named vehicle engine manufacturers. - With real ELD datasets, the model developed here maybe very useful in helping the 98% of the commercial trucking companies (mentioned above) for new and replacement trucks select the vehicle engine manufacturer that best meet their use case. - ELD providers could use the models to develop predictive maintenance plans derived from running against this data, generating alerts or recommendations for maintenance or repair for their clients. - Even without real world ELD datasets, one may use these model for useful what if scenarios in machine learning training.

3.1.3 Step 3 - Further analysis on new set of Logistic Regression with engine condition as target

We are now going to explore further to see how good a logistic regression relationship is with some relatively high degree of predictability between the feature set (lub_oil_pressure, fuel_pressure, coolant_pressure, lub_oil_temp and coolant_temp) and our target (engine_condition). Is there a possibility, for example, that through machine learning we may gain insights of how well the feature set can be used to accurately predict engine conditition.

Sub Step 1 - Get Required libraries

[197]:
```
#import functions for train test split

from sklearn.model_selection import train_test_split



# functions for metrics

from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

Sub Step 2 - Let's plot the types and count of engine_condition

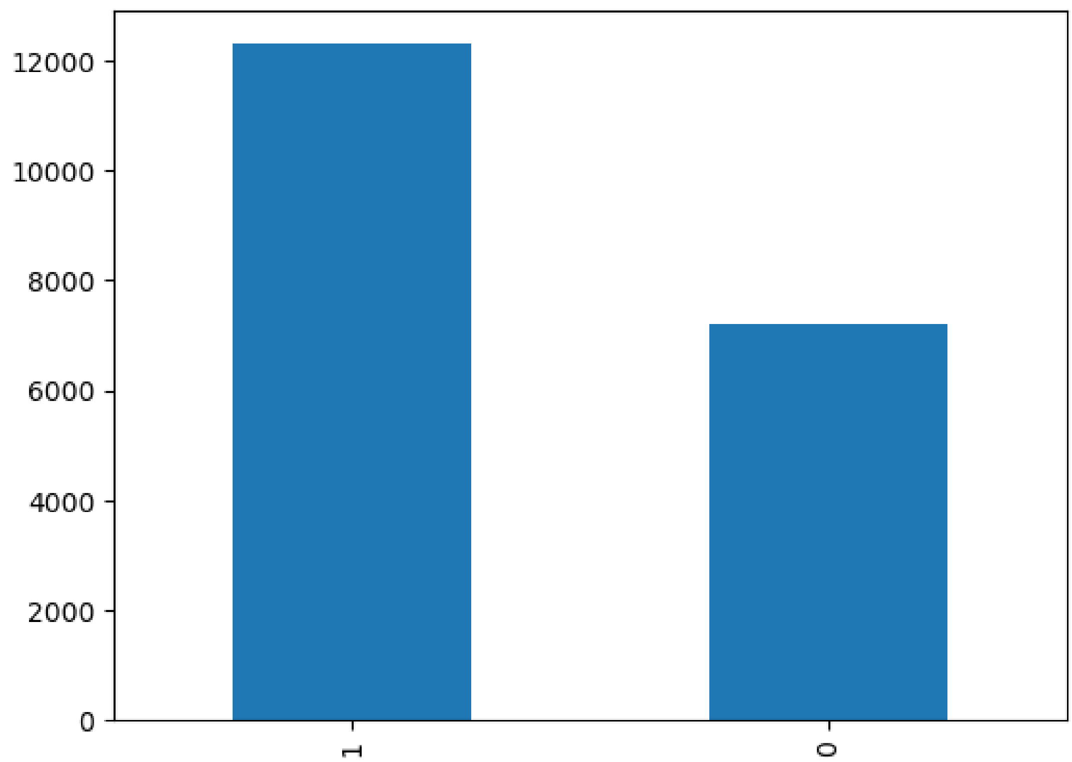
[198]:
```
pdf1.engine_condition.value_counts()
```

[198]:
```
1    12302
0     7213
```

```
Name: engine_condition, dtype: int64
```

[199]:
```
pdf1.engine_condition.value_counts().plot.bar()
```

[199]:
```
<Axes: >
```



There are more 12,302 counts of engine condition being good (1) and more than 7,213 of engine condition being bad (0) in this dataset.

Sub Step 3 - Identify the target column and the data columns First we identify the target. Target is the value that our machine learning model needs to classify

[200]:
```
Y = pdf1["engine_condition"]
```

We identify the features next. Features are the input values our machine learning model learns from

[201]:
```
X = pdf1[["lub_oil_pressure", "fuel_pressure", "coolant_pressure",␣
↪"lub_oil_temp", "coolant_temp"]]
```

Sub Step 4 - Split the data set We split the data set in the ratio of 70:30. 70% training data, 30% testing data.

[202]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30,␣
↪random_state=40)
```

Sub Step 5 - Build and train a classifier Create a Logistic Regression model

[203]:
```
classifier3 = LogisticRegression()
```

Train/Fit the model on training data

[204]:
```
classifier3.fit(X_train,Y_train)
```

[204]:
```
LogisticRegression()
```

Sub Step 6 - Calculate approporiate metrices

[205]:
```
#Higher the score, better the model.
xytest = classifier3.score(X_test,Y_test)
```

[206]:
```
xytest
```

[206]:
```
0.6356959863364645
```

To compute the detailed metrics we need two values, the original value and a predicted value.

[207]:
```
original_values = Y_test
predicted_values = classifier3.predict(X_test)
```

Precision

[208]:
```
precisionscore = precision_score(original_values, predicted_values) # Higher␣
↪the value the better the model
```

[209]:
```
precisionscore
```

[209]:
```
0.6429347826086956
```

Recall

[210]:
```
recallscore = recall_score(original_values, predicted_values) # Higher the␣
↪value the better the model
```

[211]:
```
recallscore
```

[211]:
```
0.9563459983831851
```

F1 Score

[212]:
```
f1score = f1_score(original_values, predicted_values) # Higher the value the␣
↪better the model
```

[213]:
```
f1score
```

[213]: 0.7689307767305817

Confusion Matrix

[214]:
```
confusematrix = confusion_matrix(original_values, predicted_values) # can be␣
↪used to manually calculate various met
```

[215]:
```
confusematrix
```

[215]:
```
array([[ 173, 1971],
       [ 162, 3549]], dtype=int64)
```

3.2 Part 5 - Examining Evaluation Metrics

3.2.1 - Step 1 - Summarize

[218]:
```
print("Evaluation - Important Metrices")

print("X_Test, Y_Test = ", xytest)
print("Precision Score = ", precisionscore)
print("Recall Score = ", recallscore)
print("F1 Score = ", f1score)
print("Confusion Matrix = ", confusematrix)
```

```
Evaluation - Important Metrices
X_Test, Y_Test =  0.6356959863364645
Precision Score =  0.6429347826086956
Recall Score =  0.9563459983831851
F1 Score =  0.7689307767305817
Confusion Matrix =  [[ 173 1971]
 [ 162 3549]]
```

3.2.2 Step 2 - Analysis

The metrics as shown above are used as a score method in scikit-learn to evaluate the performance of our model on a test dataset with our target "engine_condition" being the focus of our classification by our feature set conisting of all other variables (except the "manufacturers").

- The quality of a machine learning model is often measured by its performance on a test dataset. In this project for our ML model, we first want to examine the qaulitity of the original dataset that we got from Kaggle, without the impact of our dataset of manufacturers, which we randomly created.
- The score method returns the coefficient of determination (R-squared) value, which is a measure of how well the model fits the data. R-squared value ranges from 0 to 1, with higher values indicating better model performance. The X_Test, Y_Test value of 0.636 we got above indicates that our model is performing well.
- The precision score is a metric to help us understand how well our model correctly predicts positive observations. Essentially, it takes the total number of correctly predicted positive observations and divides by the total number of predicted positive observations. The precision

score of 0.6429 we got from our model tells us that from the data set 64.29% of the time the model is accurately predicting the engine_condition based on our feature set.

- The recall score metric measures how well your model correctly predicted all possible positive observations. Therefore, it takes the total number of correctly predicted positive data points and divides it by the total number of all data points that it positive, whether or not the model predicted it correctly. The recall score of 0.9563 we got from our model tells us that from the data set, 95.63% of the time the model gets the positive prediction on the engine_condition correctly.
- The F1 score is a weighted average metric of the precision and recall metrices. It helps us understand how many times the model got it wrong, whether positively or negatively during its prediction. the lower this value is the higher the rate of false positives and false negatives. In our score above this number was 0.7689, which means 76.89% of the time the model is NOT predicting false positives and false negatives.
- The confusion metrics summarizes in a table foramt all the other other metrics. It is often used to measure the performance of classification models, which aim to predict a categorical label for each input instance. The matrix displays the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) produced by the model on the test data. The embedded image below provides a good pictural view of the results of our model. Please note that the numbers in the image may NOT be the same numbers as in the table above. However, the numbers will be pretty close. The difference can be accounted for due to fact that each time the code is compiled and the randomness employed in the process, the numbers may come up slighly different. However, the result does not change the conclusion.

[276]:
```
# import image module
from IPython.display import Image
from IPython.core.display import HTML
```

[277]:
```
PATH = 'E:\\MLCapstone\\engine_data\\'
Image(filename = PATH + 'confusion_matrix.jpg', width=300, height=300)
```

[277]:

		Actual (True) Values	
		Good Engine Condition	Not Good Engine Condition
Predicted Values	Good Engine Condition	171 (correct model	1,903 (wrong positive
	Not Good Engine Condition	192 (wrong negative	3,589 (correct model

3.2.3 Step 3 - Summary on Initial ETL Process

- The steps taken and tasks completed so far, including the creation of statistics and visualization on our Data Set was to help us identify what variables (columns) to use in our ML modeling for target and feature set. Additionaly, it helped us to have identified potential data quality issues and the potential feature transformations necessary.
- The next sections of this project will switch focus to developing and interpreting a machine learning model suitable for our use case and based on the datasets we have examined above.

3.3 Part 6 - Create a Machine Learning Baseline Pipeline

3.3.1 Task 1 - Get the PySpark Dataframe created earlier and drop a columns not needed

[219]:
```
# Drop multiple columns
df5 = df4.drop("manufacturers")
```

[220]:
```
df5.show(5)
```

```
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+
|engine_rpm|lub_oil_pressure|fuel_pressure|coolant_pressure|lub_oil_temp|coolant
_temp|engine_condition|
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+
|       811|     5.452593264|  5.079589925|     2.680202261| 76.54589478|
90.08560545|               0|
|       558|     4.443161459|  5.610184593|     3.112912175| 76.66414132|
```

```
67.08213633|               1|
|      1234|     2.981909274|  12.63864452|     2.102405305| 77.37659228|
74.82047801|               1|
|       469|     4.328405379|  5.216832349|     6.625636587| 76.09339431|
78.95197818|               1|
|       770|     2.961174741|  2.240656493|      2.31300994| 74.53125761|
80.90553092|               0|
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+
only showing top 5 rows
```

3.3.2 Task 2 - Import the required functions and define the VectorAssembler pipeline stage

Stage 1 - Assemble the input columns into a single column "features".

[221]:
```python
from pyspark.ml.feature import VectorAssembler

vectorAssembler =␣
↪VectorAssembler(inputCols=['engine_rpm','lub_oil_pressure','fuel_pressure','coolant_pressur
↪'coolant_temp'], outputCol="features3")
```

3.3.3 Task 3 - Instantiate a classifier from SparkML package and assign it to the classifier variable

[222]:
```python
from pyspark.ml.classification import GBTClassifier

classifier = GBTClassifier(labelCol='engine_condition',␣
↪featuresCol='features3', maxIter=10)
```

3.3.4 Task 3 - Import the required ML function, build, train and evaluate

Stage 2 - build pipeline

[223]:
```python
from pyspark.ml import Pipeline
pipeline = Pipeline(stages=[vectorAssembler, classifier])
```

Stage 3 - train model - we are goig to use the pyspark dataframe df4 that was created earlier.

[224]:
```python
model = pipeline.fit(df5)
```

Stage 4 - Predict and show model

[225]:
```python
prediction = model.transform(df5)
```

[226]:
```python
prediction.show()
```

```
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+---------------------+---------------------+---------------
```

```
-----+----------+
|engine_rpm|lub_oil_pressure|fuel_pressure|coolant_pressure|lub_oil_temp|coolant
_temp|engine_condition|           features3|       rawPrediction|
probability|prediction|
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+--------------------+--------------------+---------------
-----+----------+
|       811|     5.452593264|  5.079589925|     2.680202261| 76.54589478|
90.08560545|
0|[811.0,5.45259326…|[-0.1566322391932…|[0.42231811737574…|       1.0|
|       558|     4.443161459|  5.610184593|     3.112912175| 76.66414132|
67.08213633|
1|[558.0,4.44316145…|[-0.7745298305666…|[0.17522212380758…|       1.0|
|      1234|     2.981909274|  12.63864452|     2.102405305| 77.37659228|
74.82047801|
1|[1234.0,2.9819092…|[-0.2546668366317…|[0.37534974105868…|       1.0|
|       469|     4.328405379|  5.216832349|     6.625636587| 76.09339431|
78.95197818|
1|[469.0,4.32840537…|[-0.6025968818877…|[0.23055256365510…|       1.0|
|       770|     2.961174741|  2.240656493|      2.31300994| 74.53125761|
80.90553092|
0|[770.0,2.96117474…|[-0.4534527581038…|[0.28763348149155…|       1.0|
|       619|     3.934368926|  7.698462273|     2.278726856| 86.06308412|
88.42011639|
1|[619.0,3.93436892…|[-0.5815342383639…|[0.23811017412303…|       1.0|
|      1606|     4.398524811|  5.503495054|     1.675167803| 76.97447105|
87.88562744|
1|[1606.0,4.3985248…|[0.02701098980326…|[0.51350221135247…|       0.0|
|       728|     2.483021887|  6.562360645|      1.72643451| 74.17618041|
76.73653723|
1|[728.0,2.48302188…|[-0.2986042300134…|[0.35498261349994…|       1.0|
|       524|     2.271233227|  5.599198784|     3.374839294| 75.19588881|
80.25312165|
0|[524.0,2.27123322…|[-0.6955209443379…|[0.19924147734788…|       1.0|
|       916|     2.331835634|  6.329489495|     2.220629546| 75.21374294|
84.35683965|
0|[916.0,2.33183563…|[-0.0767660328487…|[0.46169220361430…|       1.0|
|       581|     2.532502622|  5.211893945|     1.020389138| 84.03211401|
74.97293084|
0|[581.0,2.53250262…|[-0.2945063211436…|[0.35686143206712…|       1.0|
|       696|     3.229960878|  13.50813113|     2.711880305| 74.91004261|
76.23301589|
1|[696.0,3.22996087…|[-0.6658679916133…|[0.20887235821302…|       1.0|
|       523|     2.768377533|  10.41567564|     1.603632275| 76.31189436|
74.70228332|
1|[523.0,2.76837753…|[-0.8107587475434…|[0.16499569538413…|       1.0|
|       848|     4.143512581|   2.86629221|     2.521902353| 86.77647934|
77.87564948|
```

```
1|[848.0,4.14351258…|[0.27761884648550…|[0.63534991989666…|       0.0|
|       823|      2.080203637|  7.236541038|     2.573919401| 77.34109804|
81.94345215|
0|[823.0,2.08020363…|[-0.1700408719210…|[0.41578962064471…|       1.0|
|      1031|      4.899692719|  4.732332599|     1.347354638| 78.53583698|
77.58817477|
0|[1031.0,4.8996927…|[0.11252563400248…|[0.55602654665327…|       0.0|
|       723|      2.359133956|  10.50397534|     2.719321943| 76.05518118|
74.4970153|
0|[723.0,2.35913395…|[-0.2979441662545…|[0.35528494087582…|       1.0|
|       814|      3.159342477|  5.551390389|     3.180498063| 77.95522031|
85.68445037|
0|[814.0,3.15934247…|[-0.1013328156201…|[0.44950630280335…|       1.0|
|       819|      2.383461976|  6.738287649|     3.231114826| 77.41499188|
78.06126422|
1|[819.0,2.38346197…|[-0.1677796718431…|[0.41688856723167…|       1.0|
|       680|      0.808593578|  3.780912195|     0.947805614| 77.76832404|
71.91843029|
0|[680.0,0.80859357…|[0.35183392122330…|[0.66900047779774…|       0.0|
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+--------------------+--------------------+---------------
-----+----------+
only showing top 20 rows
```

Stage 5 - Evaluate Model

[227]:
```python
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
binEval = MulticlassClassificationEvaluator().setMetricName("accuracy") .
 ↪setPredictionCol("prediction").setLabelCol("engine_condition")

binEval.evaluate(prediction)
```

[227]:
```
0.6811683320522675
```

- The evaluation results of 68.11683% from the model suggest the model is a farily good model to use to predict engine condition based on the set of variables features.
- Now that we have built an acceptable ML model we can move to the essence of the Project

4 Part 7 - Create The Desired ML Pipeline - The Essence of the Project

One of the main objectives of the project is to develop a ML model that takes into consideration the feature set vehicle engine manufacturers. However, to do this we will need to convert the categorial variable of "manufactuers" to an integer type. To do this we will use the One-Hot encode approach.

- First, we will use a StringIndexer to index the "manufacturers" categorical variables into numbers. This does not require a specific order. Essentially, we are mapping the strings values to numbers, and keeps track of it as metadata attached to the DataFrame.

- Second, we will use One-hot encoding to map the categorical feature, now represented as a label index, to a binary vector with at most a single one-value indicating the presence of a specific feature value from among the set of all feature values. This encoding will allow our use a Regression ML.

4.0.1 Step 1 - Import required ML functions, create a String Indexer and then One Hot Encode

[228]:
```
from pyspark.ml.feature import OneHotEncoder, StringIndexer

# Using the df4 PySpark Dataframe, the colunm to encode is 'manufacturers'
stringInd = StringIndexer(inputCols=['manufacturers'],␣
 ↪outputCols=['manufacturers_si'])
onehotencoded = OneHotEncoder(inputCols=['manufacturers_si'],␣
 ↪outputCols=['manufacturers_ohe'])
```

4.0.2 Step 2 - Create Vector Assembler

[229]:
```
vectorAssembler2 =␣
 ↪VectorAssembler(inputCols=['engine_rpm','lub_oil_pressure','fuel_pressure','coolant_pressur
 ↪'coolant_temp'] +['manufacturers_ohe'], outputCol="features4")
```

4.0.3 Step 3 - Build Classifer

[230]:
```
classifier3 = GBTClassifier(labelCol='engine_condition',␣
 ↪featuresCol='features4', maxIter=10)
```

4.0.4 Step 4 - Build the Pipeline

[231]:
```
pipeline2 = Pipeline(stages=[stringInd, onehotencoded, vectorAssembler2,␣
 ↪classifier3])
```

4.0.5 Step 5 - Fit Model

[232]:
```
model2 = pipeline2.fit(df4)
```

4.0.6 Step 6 - Model predict

[233]:
```
prediction2 = model2.transform(df4)
```

4.0.7 Step 7 - Show Model

[234]:
```
prediction2.show()
```

```
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+-------------+----------------+-----------------+--------
------------+--------------------+--------------------+----------+
|engine_rpm|lub_oil_pressure|fuel_pressure|coolant_pressure|lub_oil_temp|coolant
```

```
_temp|engine_condition|manufacturers|manufacturers_si|manufacturers_ohe|
features4|       rawPrediction|         probability|prediction|
+----------+----------------+-------------+----------------+-----------+-------
-----+----------------+-------------+----------------+-----------------+--------
------------+--------------------+--------------------+----------+
|       811|     5.452593264|  5.079589925|     2.680202261| 76.54589478|
90.08560545|               0|         Mack|             0.0|
(4,[0],[1.0])|[811.0,5.45259326…|[-0.0939635775154…|[0.45315599447345…|
1.0|
|       558|     4.443161459|  5.610184593|     3.112912175| 76.66414132|
67.08213633|               1|         Mack|             0.0|
(4,[0],[1.0])|[558.0,4.44316145…|[-0.7752405899007…|[0.17501678171693…|
1.0|
|      1234|     2.981909274|  12.63864452|     2.102405305| 77.37659228|
74.82047801|               1|         Mack|             0.0|
(4,[0],[1.0])|[1234.0,2.9819092…|[-0.2561211847953…|[0.37466800904622…|
1.0|
|       469|     4.328405379|  5.216832349|     6.625636587| 76.09339431|
78.95197818|               1|        Volvo|             4.0|
(4,[],[])|[469.0,4.32840537…|[-0.6284157901509…|[0.22151980019201…|
1.0|
|       770|     2.961174741|  2.240656493|      2.31300994| 74.53125761|
80.90553092|               0|        Volvo|             4.0|
(4,[],[])|[770.0,2.96117474…|[-0.4550226251100…|[0.28699057774230…|
1.0|
|       619|     3.934368926|  7.698462273|     2.278726856| 86.06308412|
88.42011639|               1|        Volvo|             4.0|
(4,[],[])|[619.0,3.93436892…|[-0.5819925660840…|[0.23794392017351…|
1.0|
|      1606|     4.398524811|  5.503495054|     1.675167803| 76.97447105|
87.88562744|               1| Freightliner|             1.0|
(4,[1],[1.0])|[1606.0,4.3985248…|[0.02622614495400…|[0.51311006686682…|
0.0|
|       728|     2.483021887|  6.562360645|      1.72643451| 74.17618041|
76.73653723|               1| Freightliner|             1.0|
(4,[1],[1.0])|[728.0,2.48302188…|[-0.2997156008401…|[0.35447383664444…|
1.0|
|       524|     2.271233227|  5.599198784|     3.374839294| 75.19588881|
80.25312165|               0| Freightliner|             1.0|
(4,[1],[1.0])|[524.0,2.27123322…|[-0.6913526882042…|[0.20057485588147…|
1.0|
|       916|     2.331835634|  6.329489495|     2.220629546| 75.21374294|
84.35683965|               0| Freightliner|             1.0|
(4,[1],[1.0])|[916.0,2.33183563…|[-0.0782358771995…|[0.46096167818735…|
1.0|
|       581|     2.532502622|  5.211893945|     1.020389138| 84.03211401|
74.97293084|               0|     Kenworth|             3.0|
(4,[3],[1.0])|[581.0,2.53250262…|[-0.2960559613013…|[0.35615042804617…|
```

```
1.0|
|       696|     3.229960878|  13.50813113|     2.711880305| 74.91004261|
76.23301589|               1|     Kenworth|             3.0|
(4,[3],[1.0])|[696.0,3.22996087…|[-0.6670929382819…|[0.20846781506813…|
1.0|
|       523|     2.768377533|  10.41567564|     1.603632275| 76.31189436|
74.70228332|               1|     Kenworth|             3.0|
(4,[3],[1.0])|[523.0,2.76837753…|[-0.8113195471694…|[0.16484122833773…|
1.0|
|       848|     4.143512581|   2.86629221|     2.521902353| 86.77647934|
77.87564948|               1|International|             2.0|
(4,[2],[1.0])|[848.0,4.14351258…|[0.27698388091531…|[0.63505565120227…|
0.0|
|       823|     2.080203637|  7.236541038|     2.573919401| 77.34109804|
81.94345215|               0|International|             2.0|
(4,[2],[1.0])|[823.0,2.08020363…|[-0.1714848200568…|[0.41508829681266…|
1.0|
|      1031|     4.899692719|  4.732332599|     1.347354638| 78.53583698|
77.58817477|               0|International|             2.0|
(4,[2],[1.0])|[1031.0,4.8996927…|[0.11177278947867…|[0.55565481942190…|
0.0|
|       723|     2.359133956|  10.50397534|     2.719321943| 76.05518118|
74.4970153|               0|International|             2.0|
(4,[2],[1.0])|[723.0,2.35913395…|[-0.2994335339198…|[0.35460293354968…|
1.0|
|       814|     3.159342477|  5.551390389|     3.180498063| 77.95522031|
85.68445037|               0|         Mack|             0.0|
(4,[0],[1.0])|[814.0,3.15934247…|[-0.1027666906360…|[0.44879678017693…|
1.0|
|       819|     2.383461976|  6.738287649|     3.231114826| 77.41499188|
78.06126422|               1|         Mack|             0.0|
(4,[0],[1.0])|[819.0,2.38346197…|[-0.1693694273076…|[0.41611585684630…|
1.0|
|       680|     0.808593578|  3.780912195|     0.947805614| 77.76832404|
71.91843029|               0|         Mack|             0.0|
(4,[0],[1.0])|[680.0,0.80859357…|[0.35028916947971…|[0.66831598488583…|
0.0|
+----------+----------------+------------+----------------+-----------+-------
-----+----------------+------------+----------------+----------------+--------
------------+--------------------+--------------------+----------+
only showing top 20 rows
```

4.0.8 Step 8 - Evaluate the Model

[235]:
```
binEval2 = MulticlassClassificationEvaluator().setMetricName("accuracy") .
 ↪setPredictionCol("prediction").setLabelCol("engine_condition")

binEval2.evaluate(prediction2)
```

[235]:
```
0.6822444273635665
```

4.0.9 Step 9 - Interpretting the Model

Upon adding the "manufacturers" variable by converting this categorial variable from string to Integer through One Hot Encoding into a new pipeline model, the evaluation results was improvded a little to 68.224443%.

5 Part 8 - Training the Model

5.0.1 Task 1 - Split the data

[236]:
```
# Split the data into training and testing sets with 70:30 split.
# set the value of seed to 42


(trainingData, testingData) = df4.randomSplit([0.7, 0.3], seed=42)
```

5.0.2 Task 2 - Fit the pipeline to the training data

[237]:
```
# Fit the pipeline using the training data

pipelineModel = pipeline2.fit(trainingData)
```

5.0.3 Task 3 - Predict Model with Training data

[238]:
```
tdprediction1 = pipelineModel.transform(trainingData)
```

5.0.4 Task 4 - Show the Model with the Training Data

[239]:
```
tdprediction1.show()
```

```
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+-------------+----------------+-----------------+--------
------------+--------------------+--------------------+----------+
|engine_rpm|lub_oil_pressure|fuel_pressure|coolant_pressure|lub_oil_temp|coolant
_temp|engine_condition|manufacturers|manufacturers_si|manufacturers_ohe|
features4|       rawPrediction|         probability|prediction|
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+-------------+----------------+-----------------+--------
------------+--------------------+--------------------+----------+
```

```
|       348|      2.29528494|  14.20575915|     2.926311152| 76.65707907|
73.40836881|               1| Freightliner|            4.0|
(4,[],[])|[348.0,2.29528494…|[-1.1773683580345…|[0.08669000870866…|
1.0|
|       351|     3.863922586|  6.392786638|     2.213875211|  76.5604572|
71.36323417|               1|     Kenworth|            2.0|
(4,[2],[1.0])|[351.0,3.86392258…|[-1.1311483598765…|[0.09429404009006…|
1.0|
|       367|     3.601815275|   6.74004355|     1.854593765| 74.83093464|
82.10152938|               1|        Volvo|            3.0|
(4,[3],[1.0])|[367.0,3.60181527…|[-1.0488363478655…|[0.10932322871920…|
1.0|
|       370|     1.895986462|  16.51134782|     3.551033906| 82.65443468|
76.1284858|               1|         Mack|            1.0|
(4,[1],[1.0])|[370.0,1.89598646…|[-1.1307103168381…|[0.09436888678198…|
1.0|
|       370|     2.382148744|  4.373560376|     3.020646985| 76.73040964|
84.84346506|               1|     Kenworth|            2.0|
(4,[2],[1.0])|[370.0,2.38214874…|[-1.0129096309365…|[0.11651860618877…|
1.0|
|       374|     3.696694909|  7.410403641|     1.250102206| 77.49503091|
80.05310435|               1|     Kenworth|            2.0|
(4,[2],[1.0])|[374.0,3.69669490…|[-1.2000412090156…|[0.08316641192292…|
1.0|
|       387|     1.429110412|  8.161765578|     2.553869784| 77.66315357|
78.19249521|               1|        Volvo|            3.0|
(4,[3],[1.0])|[387.0,1.42911041…|[-1.1454454276496…|[0.09188019151884…|
1.0|
|       387|      2.41422254|   5.28609528|     3.203941661| 74.05471378|
82.30946933|               1|         Mack|            1.0|
(4,[1],[1.0])|[387.0,2.41422254…|[-1.0318522394246…|[0.11267492715324…|
1.0|
|       388|     2.661658328|  5.714024977|     1.286889548| 76.32742911|
70.15362259|               1|        Volvo|            3.0|
(4,[3],[1.0])|[388.0,2.66165832…|[-1.0901115568399…|[0.10154057140829…|
1.0|
|       392|     0.003384113|  5.385946667|     1.505215636| 73.66027815|
81.26645814|               1| Freightliner|            4.0|
(4,[],[])|[392.0,0.00338411…|[-0.9715057832183…|[0.12531737723074…|
1.0|
|       394|     5.046976309|  4.780149587|      1.10885257|  75.7769797|
82.65763167|               1|        Volvo|            3.0|
(4,[3],[1.0])|[394.0,5.04697630…|[-0.9748917538535…|[0.12457696638670…|
1.0|
|       398|     4.794103656|  9.862642817|     1.525782872| 82.92430375|
87.42792024|               1| Freightliner|            4.0|
(4,[],[])|[398.0,4.79410365…|[-1.0677038854396…|[0.10570270593691…|
1.0|
```

```
|       406|      5.164862822|  6.786440771|      5.604405978| 76.60528593|
73.66097884|                1| Freightliner|             4.0|
(4,[],[])|[406.0,5.16486282…|[-0.8597044872946…|[0.15194730712541…|
1.0|
|       409|      3.932652326|  1.512916108|      1.251806999| 74.19968598|
81.3467327|                1|International|             0.0|
(4,[0],[1.0])|[409.0,3.93265232…|[-0.8697026697623…|[0.14938848321393…|
1.0|
|       416|      2.657514597|  6.856036223|      6.300309285| 72.58472965|
82.88622071|                1|        Volvo|             3.0|
(4,[3],[1.0])|[416.0,2.65751459…|[-0.8857702298737…|[0.14535086231732…|
1.0|
|       418|      2.538428379|  2.945069073|      2.603207655| 78.01496098|
76.49333619|                0| Freightliner|             4.0|
(4,[],[])|[418.0,2.53842837…|[-0.3385455291477…|[0.33691085717336…|
1.0|
|       420|      2.601225572|  2.632606367|      2.173108753|  74.6780238|
88.06133059|                1|International|             0.0|
(4,[0],[1.0])|[420.0,2.60122557…|[-0.8134334454198…|[0.16426001795278…|
1.0|
|       422|       4.94215732|    10.063178|      1.778199142| 77.45671911|
81.85560843|                1|International|             0.0|
(4,[0],[1.0])|[422.0,4.94215732…|[-1.0456898835313…|[0.10993748976663…|
1.0|
|       425|       3.00647749|  2.774749692|      1.339009448| 77.80423513|
73.64821115|                1| Freightliner|             4.0|
(4,[],[])|[425.0,3.00647749…|[-0.3825144303539…|[0.31755544325501…|
1.0|
|       427|       3.65661683|  9.633174204|      1.423482398| 77.25776565|
66.44921806|                1|     Kenworth|             2.0|
(4,[2],[1.0])|[427.0,3.65661683…|[-1.1985813130437…|[0.08338931648896…|
1.0|
+----------+----------------+-------------+----------------+------------+-------
-----+----------------+-------------+----------------+-----------------+--------
------------+--------------------+--------------------+----------+
only showing top 20 rows
```

5.0.5 Task 5 - Evaluate the Trained Model

[240]:
```
binEval3 = MulticlassClassificationEvaluator().setMetricName("accuracy") .
 ↪setPredictionCol("prediction").setLabelCol("engine_condition")

binEval2.evaluate(tdprediction1)
```

[240]:
```
0.6835590505315275
```

For the new pipeline model based off the training data, the evaluation results shows 68.355905%.

6 Part 9 - Predict with Testing Data and Evaluate the Model

6.0.1 Task 1 - Predict with the model using Testing Data

[241]:
```
# Make predictions on testing data

tndpredictions2 = pipelineModel.transform(testingData)
```

[242]:
```
binEval3 = MulticlassClassificationEvaluator().setMetricName("accuracy") .
↪setPredictionCol("prediction").setLabelCol("engine_condition")

binEval2.evaluate(tndpredictions2)
```

[242]:
```
0.6644179207749524
```

For the new pipeline model based off the testing data, the evaluation results shows 66.444179%

6.0.2 Task 2 - Print the MSE

[243]:
```
#your code goes here

from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(predictionCol="prediction",␣
↪labelCol="engine_condition", metricName="mse")
mse = evaluator.evaluate(tndpredictions2)
print(mse)
```

```
0.3355820792250475
```

6.0.3 Task 3 - Print the MAE

[244]:
```
#your code goes here

evaluator = RegressionEvaluator(predictionCol="prediction",␣
↪labelCol="engine_condition", metricName="mae")
mae = evaluator.evaluate(tndpredictions2)
print(mae)
```

```
0.33558207922504757
```

6.0.4 Task 4 - Print the R-Squared(R2)

[245]:
```
#your code goes here

evaluator = RegressionEvaluator(predictionCol="prediction",␣
↪labelCol="engine_condition", metricName="r2")
r2 = evaluator.evaluate(tndpredictions2)
print(r2)
```

```
-0.4349868850361458
```

6.0.5 Task 5 - Summarize ML Model with Testing Data

Run the code cell below. Use the answers here to answer the final evaluation quiz in the next section. If the code throws up any errors, go back and review the code you have written.

[246]:
```python
print("Task 5 - Summary")

print("Mean Squared Error = ", round(mse,2))
print("Mean Absolute Error = ", round(mae,2))
print("R Squared = ", round(r2,2))
```

```
Task 5 - Summary
Mean Squared Error =  0.34
Mean Absolute Error =  0.34
R Squared =  -0.43
```

7 Part 10 - Persist the Model

7.0.1 Task 1 - Save the model to the path "IBM_ML_Capstone_Final_Project"

[251]:
```python
# Save the pipeline model as "Final_Project"
pipelineModel.write().overwrite().save("IBM_ML_Capstone_Final_Project")
```

7.0.2 Task 2 - Load the model from the path "IBM_ML_Capstone_Final_Project"

[252]:
```python
# Load the pipeline model you have created in the previous step
loadedPipelineModel = PipelineModel.load("IBM_ML_Capstone_Final_Project")
```

7.0.3 Task 3 - Make predictions using the loaded model on the testdata

[253]:
```python
# Use the loaded pipeline model and make predictions using testingData
predictions4 = loadedPipelineModel.transform(testingData)
```

7.0.4 Task 4 - Show the predictions

[254]:
```python
#show top 5 rows from the predections dataframe. Display only the label column␣
↪and predictions
predictions4_display = predictions4.select("engine_condition","prediction")
```

[255]:
```python
predictions4_display.show(100)
```

```
+----------------+----------+
|engine_condition|prediction|
+----------------+----------+
|               1|       1.0|
|               1|       1.0|
```

```
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    0|          1.0|
|                    0|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    0|          1.0|
|                    0|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    0|          1.0|
|                    1|          1.0|
|                    0|          1.0|
|                    0|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    0|          1.0|
|                    1|          1.0|
|                    1|          1.0|
|                    1|          1.0|
```

```
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                0|       1.0|
|                1|       1.0|
|                0|       1.0|
|                1|       1.0|
|                1|       1.0|
|                0|       1.0|
|                0|       1.0|
|                0|       1.0|
|                0|       1.0|
|                1|       1.0|
|                1|       1.0|
|                0|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                0|       1.0|
|                1|       1.0|
|                0|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       0.0|
|                0|       0.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                0|       1.0|
|                1|       1.0|
|                0|       1.0|
|                1|       1.0|
|                0|       1.0|
|                1|       1.0|
|                0|       1.0|
|                0|       1.0|
|                0|       1.0|
|                1|       1.0|
```

```
|               1|       1.0|
|               1|       1.0|
+----------------+----------+
only showing top 100 rows
```

8 Part 11 - Decode the One-Hot-Encode Prediction

8.0.1 Method 1 - Attempt

[264]:
```python
from pyspark.ml.feature import IndexToString
# Extract the one-hot encoded vector
coconverter = IndexToString(inputCol="prediction", outputCol="prediction_label")
df_converted = coconverter.transform(predictions4_display)
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
Cell In[264], line 4
      2 # Extract the one-hot encoded vector
      3 coconverter = IndexToString(inputCol="prediction",␣
 ↪outputCol="prediction_label")
----> 4 df_converted = coconverter.transform(predictions4_display)

File E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\ml\base.py:262, in␣
 ↪Transformer.transform(self, dataset, params)
    260         return self.copy(params)._transform(dataset)
    261     else:
--> 262         return self._transform(dataset)
    263 else:
    264     raise TypeError("Params must be a param map but got %s." %␣
 ↪type(params))

File E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\ml\wrapper.py:398, in␣
 ↪JavaTransformer._transform(self, dataset)
    395 assert self._java_obj is not None
    397 self._transfer_params_to_java()
--> 398 return DataFrame(self._java_obj.transform(dataset._jdf), dataset.
 ↪sparkSession)

File E:\Spark\spark-3.5.0-bin-hadoop3\python\lib\py4j-0.10.9.7-src.
 ↪zip\py4j\java_gateway.py:1322, in JavaMember.__call__(self, *args)
   1316 command = proto.CALL_COMMAND_NAME +\
   1317     self.command_header +\
   1318     args_command +\
   1319     proto.END_COMMAND_PART
   1321 answer = self.gateway_client.send_command(command)
-> 1322 return_value = get_return_value(
```

```
   1323     answer, self.gateway_client, self.target_id, self.name)
   1325 for temp_arg in temp_args:
   1326     if hasattr(temp_arg, "_detach"):

File E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\errors\exceptions\captured.py:179, in capture_sql_exception.<locals>.deco(*a, **kw)
    177 def deco(*a: Any, **kw: Any) -> Any:
    178     try:
--> 179         return f(*a, **kw)
    180     except Py4JJavaError as e:
    181         converted = convert_exception(e.java_exception)

File E:\Spark\spark-3.5.0-bin-hadoop3\python\lib\py4j-0.10.9.7-src.zip\py4j\protocol.py:326, in get_return_value(answer, gateway_client, target_id, name)
    324 value = OUTPUT_CONVERTER[type](answer[2:], gateway_client)
    325 if answer[1] == REFERENCE_TYPE:
--> 326     raise Py4JJavaError(
    327         "An error occurred while calling {0}{1}{2}.\n".
    328         format(target_id, ".", name), value)
    329 else:
    330     raise Py4JError(
    331         "An error occurred while calling {0}{1}{2}. Trace:\n{3}\n".
    332         format(target_id, ".", name, value))

Py4JJavaError: An error occurred while calling o4060.transform.
: java.util.NoSuchElementException: None.get

        at scala.None$.get(Option.scala:529)

        at scala.None$.get(Option.scala:527)

        at org.apache.spark.ml.feature.IndexToString.transform(StringIndexer.scala:605)

        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:76)

        at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:52)

        at java.base/java.lang.reflect.Method.invoke(Method.java:578)

        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
```

```
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:374)

        at py4j.Gateway.invoke(Gateway.java:282)

        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)

        at py4j.commands.CallCommand.execute(CallCommand.java:79)

        at py4j.ClientServerConnection.waitForCommands(ClientServerConnection.
↪java:182)

        at py4j.ClientServerConnection.run(ClientServerConnection.java:106)

        at java.base/java.lang.Thread.run(Thread.java:1589)
```

8.0.2 Method 2 - Attempt

[256]:

```
# Decode the one-hot encoded prediction
from pyspark.rdd import RDD
decoded = predictions4_display.select("engine_condition","prediction").rdd.
↪map(lambda x: (x[0], x[1].toArray())).toDF(["engine_condition",␣
↪"binaryVector"])
decoded.show()
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
Cell In[256], line 3
      1 # Decode the one-hot encoded prediction
      2 from pyspark.rdd import RDD
----> 3 decoded = predictions4_display.select("engine_condition","prediction").
↪rdd.map(lambda x: (x[0], x[1].toArray())).toDF(["engine_condition",␣
↪"binaryVector"])
      4 decoded.show()

File E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\sql\session.py:122, in␣
↪_monkey_patch_RDD.<locals>.toDF(self, schema, sampleRatio)
     87 @no_type_check
     88 def toDF(self, schema=None, sampleRatio=None):
     89     """
     90     Converts current :class:`RDD` into a :class:`DataFrame`
     91
    (…)
    120     +---+
    121     """
--> 122     return sparkSession.createDataFrame(self, schema, sampleRatio)
```

```
File E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\sql\session.py:1443, in SparkSession.createDataFrame(self, data, schema, samplingRatio, verifySchema)
   1438 if has_pandas and isinstance(data, pd.DataFrame):
   1439     # Create a DataFrame from pandas DataFrame.
   1440     return super(SparkSession, self).createDataFrame(  # type: ignore[call-overload]
   1441         data, schema, samplingRatio, verifySchema
   1442     )
-> 1443 return self._create_dataframe(
   1444     data, schema, samplingRatio, verifySchema  # type: ignore[arg-type]
   1445 )

File E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\sql\session.py:1483, in SparkSession._create_dataframe(self, data, schema, samplingRatio, verifySchema)
   1480         return obj
   1482 if isinstance(data, RDD):
-> 1483     rdd, struct = self._createFromRDD(data.map(prepare), schema, samplingRatio)
   1484 else:
   1485     rdd, struct = self._createFromLocal(map(prepare, data), schema)

File E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\sql\session.py:1056, in SparkSession._createFromRDD(self, rdd, schema, samplingRatio)
   1052 """
   1053 Create an RDD for DataFrame from an existing RDD, returns the RDD and schema.
   1054 """
   1055 if schema is None or isinstance(schema, (list, tuple)):
-> 1056     struct = self._inferSchema(rdd, samplingRatio, names=schema)
   1057     converter = _create_converter(struct)
   1058     tupled_rdd = rdd.map(converter)

File E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\sql\session.py:996, in SparkSession._inferSchema(self, rdd, samplingRatio, names)
    975 def _inferSchema(
    976     self,
    977     rdd: RDD[Any],
    978     samplingRatio: Optional[float] = None,
    979     names: Optional[List[str]] = None,
    980 ) -> StructType:
    981     """
    982     Infer schema from an RDD of Row, dict, or tuple.
    983
   (…)
    994     :class:`pyspark.sql.types.StructType`
    995     """
--> 996     first = rdd.first()
```

```
    997     if isinstance(first, Sized) and len(first) == 0:
    998         raise PySparkValueError(
    999             error_class="CANNOT_INFER_EMPTY_SCHEMA",
   1000             message_parameters={},
   1001         )

File E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\rdd.py:2888, in RDD.first(self)
   2862 def first(self: "RDD[T]") -> T:
   2863     """
   2864     Return the first element in this RDD.
   2865 
   (…)
   2886     ValueError: RDD is empty
   2887     """
-> 2888     rs = self.take(1)
   2889     if rs:
   2890         return rs[0]

File E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\rdd.py:2855, in RDD.take(self, num)
   2852         taken += 1
   2854 p = range(partsScanned, min(partsScanned + numPartsToTry, totalParts))
-> 2855 res = self.context.runJob(self, takeUpToNumLeft, p)
   2857 items += res
   2858 partsScanned += numPartsToTry

File E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\context.py:2510, in SparkContext.runJob(self, rdd, partitionFunc, partitions, allowLocal)
   2508 mappedRDD = rdd.mapPartitions(partitionFunc)
   2509 assert self._jvm is not None
-> 2510 sock_info = self._jvm.PythonRDD.runJob(self._jsc.sc(), mappedRDD._jrdd, partitions)
   2511 return list(_load_from_socket(sock_info, mappedRDD._jrdd_deserializer))

File E:\Spark\spark-3.5.0-bin-hadoop3\python\lib\py4j-0.10.9.7-src.zip\py4j\java_gateway.py:1322, in JavaMember.__call__(self, *args)
   1316 command = proto.CALL_COMMAND_NAME +\
   1317     self.command_header +\
   1318     args_command +\
   1319     proto.END_COMMAND_PART
   1321 answer = self.gateway_client.send_command(command)
-> 1322 return_value = get_return_value(
   1323     answer, self.gateway_client, self.target_id, self.name)
   1325 for temp_arg in temp_args:
   1326     if hasattr(temp_arg, "_detach"):
```

```
File E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\errors\exceptions\captured
↪py:179, in capture_sql_exception.<locals>.deco(*a, **kw)
    177 def deco(*a: Any, **kw: Any) -> Any:
    178     try:
--> 179         return f(*a, **kw)
    180     except Py4JJavaError as e:
    181         converted = convert_exception(e.java_exception)

File E:\Spark\spark-3.5.0-bin-hadoop3\python\lib\py4j-0.10.9.7-src.
↪zip\py4j\protocol.py:326, in get_return_value(answer, gateway_client,␣
↪target_id, name)
    324 value = OUTPUT_CONVERTER[type](answer[2:], gateway_client)
    325 if answer[1] == REFERENCE_TYPE:
--> 326     raise Py4JJavaError(
    327         "An error occurred while calling {0}{1}{2}.\n".
    328         format(target_id, ".", name), value)
    329 else:
    330     raise Py4JError(
    331         "An error occurred while calling {0}{1}{2}. Trace:\n{3}\n".
    332         format(target_id, ".", name, value))

Py4JJavaError: An error occurred while calling z:org.apache.spark.api.python.
↪PythonRDD.runJob.
: org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in␣
↪stage 1099.0 failed 1 times, most recent failure: Lost task 0.0 in stage 1099
↪0 (TID 16213) (DESKTOP-03FMA23 executor driver): org.apache.spark.api.python.
↪PythonException: Traceback (most recent call last):
  File "E:\Spark\spark-3.5.0-bin-hadoop3\python\lib\pyspark.zip\pyspark\worker.
↪py", line 1247, in main
  File "E:\Spark\spark-3.5.0-bin-hadoop3\python\lib\pyspark.zip\pyspark\worker.
↪py", line 1239, in process
  File "E:\Spark\spark-3.5.0-bin-hadoop3\python\lib\pyspark.
↪zip\pyspark\serializers.py", line 274, in dump_stream
    vs = list(itertools.islice(iterator, batch))
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\rdd.py", line 2849, in␣
↪takeUpToNumLeft
    yield next(iterator)
          ^^^^^^^^^^^^^^

  File "E:\Spark\spark-3.5.0-bin-hadoop3\python\lib\pyspark.zip\pyspark\util.
↪py", line 83, in wrapper
    return f(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^

  File "C:\Users\josep\AppData\Local\Temp\ipykernel_7056\4151713344.py", line 3 ␣
↪in <lambda>
AttributeError: 'float' object has no attribute 'toArray'
```

```
        at org.apache.spark.api.python.BasePythonRunner$ReaderIterator.
↪handlePythonException(PythonRunner.scala:572)

        at org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.
↪scala:784)

        at org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.
↪scala:766)

        at org.apache.spark.api.python.BasePythonRunner$ReaderIterator.
↪hasNext(PythonRunner.scala:525)

        at org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator
↪scala:37)

        at scala.collection.Iterator.foreach(Iterator.scala:943)

        at scala.collection.Iterator.foreach$(Iterator.scala:943)

        at org.apache.spark.InterruptibleIterator.foreach(InterruptibleIterator
↪scala:28)

        at scala.collection.generic.Growable.$plus$plus$eq(Growable.scala:62)

        at scala.collection.generic.Growable.$plus$plus$eq$(Growable.scala:53)

        at scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala
↪105)

        at scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala
↪49)

        at scala.collection.TraversableOnce.to(TraversableOnce.scala:366)

        at scala.collection.TraversableOnce.to$(TraversableOnce.scala:364)

        at org.apache.spark.InterruptibleIterator.to(InterruptibleIterator.scala:
↪28)

        at scala.collection.TraversableOnce.toBuffer(TraversableOnce.scala:358)

        at scala.collection.TraversableOnce.toBuffer$(TraversableOnce.scala:358)

        at org.apache.spark.InterruptibleIterator.toBuffer(InterruptibleIterator.
↪scala:28)

        at scala.collection.TraversableOnce.toArray(TraversableOnce.scala:345)
```

```
        at scala.collection.TraversableOnce.toArray$(TraversableOnce.scala:339)

        at org.apache.spark.InterruptibleIterator.toArray(InterruptibleIterator
↪scala:28)

        at org.apache.spark.api.python.PythonRDD$.$anonfun$runJob$1(PythonRDD.
↪scala:181)

        at org.apache.spark.SparkContext.$anonfun$runJob$5(SparkContext.scala:
↪2438)

        at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:93)

        at org.apache.spark.TaskContext.runTaskWithListeners(TaskContext.scala:
↪161)

        at org.apache.spark.scheduler.Task.run(Task.scala:141)

        at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$4(Executor.
↪scala:620)

        at org.apache.spark.util.SparkErrorUtils.
↪tryWithSafeFinally(SparkErrorUtils.scala:64)

        at org.apache.spark.util.SparkErrorUtils.
↪tryWithSafeFinally$(SparkErrorUtils.scala:61)

        at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:94)

        at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:623)

        at java.base/java.util.concurrent.ThreadPoolExecutor.
↪runWorker(ThreadPoolExecutor.java:1144)

        at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.
↪run(ThreadPoolExecutor.java:642)

        at java.base/java.lang.Thread.run(Thread.java:1589)


Driver stacktrace:

        at org.apache.spark.scheduler.DAGScheduler.
↪failJobAndIndependentStages(DAGScheduler.scala:2844)

        at org.apache.spark.scheduler.DAGScheduler.
↪$anonfun$abortStage$2(DAGScheduler.scala:2780)
```

```
        at org.apache.spark.scheduler.DAGScheduler.
↪$anonfun$abortStage$2$adapted(DAGScheduler.scala:2779)

        at scala.collection.mutable.ResizableArray.foreach(ResizableArray.scala:
↪62)

        at scala.collection.mutable.ResizableArray.foreach$(ResizableArray.scala:
↪55)

        at scala.collection.mutable.ArrayBuffer.foreach(ArrayBuffer.scala:49)

        at org.apache.spark.scheduler.DAGScheduler.abortStage(DAGScheduler.scala:
↪2779)

        at org.apache.spark.scheduler.DAGScheduler.
↪$anonfun$handleTaskSetFailed$1(DAGScheduler.scala:1242)

        at org.apache.spark.scheduler.DAGScheduler.
↪$anonfun$handleTaskSetFailed$1$adapted(DAGScheduler.scala:1242)

        at scala.Option.foreach(Option.scala:407)

        at org.apache.spark.scheduler.DAGScheduler.
↪handleTaskSetFailed(DAGScheduler.scala:1242)

        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.
↪doOnReceive(DAGScheduler.scala:3048)

        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.
↪onReceive(DAGScheduler.scala:2982)

        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.
↪onReceive(DAGScheduler.scala:2971)

        at org.apache.spark.util.EventLoop$$anon$1.run(EventLoop.scala:49)

        at org.apache.spark.scheduler.DAGScheduler.runJob(DAGScheduler.scala:984)

        at org.apache.spark.SparkContext.runJob(SparkContext.scala:2398)

        at org.apache.spark.SparkContext.runJob(SparkContext.scala:2419)

        at org.apache.spark.SparkContext.runJob(SparkContext.scala:2438)

        at org.apache.spark.api.python.PythonRDD$.runJob(PythonRDD.scala:181)

        at org.apache.spark.api.python.PythonRDD.runJob(PythonRDD.scala)
```

```
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.
↪invoke0(Native Method)

        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.
↪invoke(NativeMethodAccessorImpl.java:76)

        at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.
↪invoke(DelegatingMethodAccessorImpl.java:52)

        at java.base/java.lang.reflect.Method.invoke(Method.java:578)

        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)

        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:374)

        at py4j.Gateway.invoke(Gateway.java:282)

        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)

        at py4j.commands.CallCommand.execute(CallCommand.java:79)

        at py4j.ClientServerConnection.waitForCommands(ClientServerConnection.
↪java:182)

        at py4j.ClientServerConnection.run(ClientServerConnection.java:106)

        at java.base/java.lang.Thread.run(Thread.java:1589)

Caused by: org.apache.spark.api.python.PythonException: Traceback (most recent␣
↪call last):
  File "E:\Spark\spark-3.5.0-bin-hadoop3\python\lib\pyspark.zip\pyspark\worker.
↪py", line 1247, in main
  File "E:\Spark\spark-3.5.0-bin-hadoop3\python\lib\pyspark.zip\pyspark\worker.
↪py", line 1239, in process
  File "E:\Spark\spark-3.5.0-bin-hadoop3\python\lib\pyspark.
↪zip\pyspark\serializers.py", line 274, in dump_stream
    vs = list(itertools.islice(iterator, batch))
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "E:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\rdd.py", line 2849, in␣
↪takeUpToNumLeft
    yield next(iterator)
          ^^^^^^^^^^^^^^

  File "E:\Spark\spark-3.5.0-bin-hadoop3\python\lib\pyspark.zip\pyspark\util.
↪py", line 83, in wrapper
    return f(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^

  File "C:\Users\josep\AppData\Local\Temp\ipykernel_7056\4151713344.py", line 3␣
↪in <lambda>
```

```
AttributeError: 'float' object has no attribute 'toArray'


        at org.apache.spark.api.python.BasePythonRunner$ReaderIterator.
↪handlePythonException(PythonRunner.scala:572)

        at org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.
↪scala:784)

        at org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.
↪scala:766)

        at org.apache.spark.api.python.BasePythonRunner$ReaderIterator.
↪hasNext(PythonRunner.scala:525)

        at org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator
↪scala:37)

        at scala.collection.Iterator.foreach(Iterator.scala:943)

        at scala.collection.Iterator.foreach$(Iterator.scala:943)

        at org.apache.spark.InterruptibleIterator.foreach(InterruptibleIterator
↪scala:28)

        at scala.collection.generic.Growable.$plus$plus$eq(Growable.scala:62)

        at scala.collection.generic.Growable.$plus$plus$eq$(Growable.scala:53)

        at scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala
↪105)

        at scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala
↪49)

        at scala.collection.TraversableOnce.to(TraversableOnce.scala:366)

        at scala.collection.TraversableOnce.to$(TraversableOnce.scala:364)

        at org.apache.spark.InterruptibleIterator.to(InterruptibleIterator.scala:
↪28)

        at scala.collection.TraversableOnce.toBuffer(TraversableOnce.scala:358)

        at scala.collection.TraversableOnce.toBuffer$(TraversableOnce.scala:358

        at org.apache.spark.InterruptibleIterator.toBuffer(InterruptibleIterato.
↪scala:28)
```

```
        at scala.collection.TraversableOnce.toArray(TraversableOnce.scala:345)

        at scala.collection.TraversableOnce.toArray$(TraversableOnce.scala:339)

        at org.apache.spark.InterruptibleIterator.toArray(InterruptibleIterator
↪scala:28)

        at org.apache.spark.api.python.PythonRDD$.$anonfun$runJob$1(PythonRDD.
↪scala:181)

        at org.apache.spark.SparkContext.$anonfun$runJob$5(SparkContext.scala:
↪2438)

        at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:93)

        at org.apache.spark.TaskContext.runTaskWithListeners(TaskContext.scala:
↪161)

        at org.apache.spark.scheduler.Task.run(Task.scala:141)

        at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$4(Executor.
↪scala:620)

        at org.apache.spark.util.SparkErrorUtils.
↪tryWithSafeFinally(SparkErrorUtils.scala:64)

        at org.apache.spark.util.SparkErrorUtils.
↪tryWithSafeFinally$(SparkErrorUtils.scala:61)

        at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:94)

        at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:623)

        at java.base/java.util.concurrent.ThreadPoolExecutor.
↪runWorker(ThreadPoolExecutor.java:1144)

        at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.
↪run(ThreadPoolExecutor.java:642)

        … 1 more
```

9 Part 12 - Conclusion

- Tiemac provides a Telematics and Fleet Management Solution for real time predictive analytics, data and business intelligence to measure, control and improve operational performance and profitability for carriers operating in the commercial over-the-road trucking sector
- This project demonstrates, this trained ML model, could be integrated into a larger system for monitoring the health of automotive engines. One of the Tiemac's goals is to use its CrewAccount ELD module to collect CAN-BUS J1939 data from sensors in commercial vehicles to collect real-time data on engine performance, which is then sent to its central server for analysis and use in its Tiemac Long Distance Load Intrchange Marketplace (TLDLIM). A predictive maintenance model, modeled off the approach use with this dataset, would then generate, among many other things, alerts or recommendations for maintenance or repair based on vehicle engine manufacturer of a truck. This from the train model suggest the model is a farily good model to use to predict engine condition based on the set of variables features.

9.1 Authors

Dr. Michael Treasure

9.2 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-10-22	1.0	Dr. Michael Treasure	Initial Version Created

Copyright © 2023 Tiemac Technologies, Inc. All rights reserved.