Explore the use of Big Data Machine Learning Systems

CSP 554: Big Data Technologies

Vaishnavi Manjunath (A20446043) vmanjunath@hawk.iit.edu
Vidya Sudharshana (A20472468) vsudharshana@hawk.iit.edu
Kavya Ravella (A20444960) kravella@hawk.iit.edu
Jiaxing Wu (A20398273) jwu96@hawk.iit.edu

Contributions:

Kavya Ravella - For Airline Satisfaction data, worked on data loading, profiling, cleaning, preprocessing. Implemented a data pipeline to process data using Spark on an EMR cluster. Processed the 4 ML models with Spark. Reviewed final report.

Jiaxing Wu - Processed the 4 ML model with Tensorflow, added plots, writing to final report, reviewing final report

Vaishnavi Manjunath - For Iris dataset, Spark SQL cleaning, Spark MLlib clustering algorithms, plots, final report writing, code organisation of our GitHub repository [15], conclusion

Vidya Sudharshana - For Airline Satisfaction data, worked on data loading, profiling, cleaning, preprocessing and processing ML models. Wrote and reviewed the final report.

Overview:

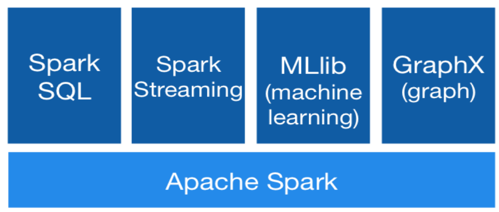
Technologies Used:

1. **Apache Spark [1]**

Apache Spark is a lightning-fast unified analytics engine for big data and machine learning. Apache Spark, built on top of Hadoop's MapReduce framework and released as an open-source project in 2010, is a distributed data processing engine with the purpose of improving the efficiency and ease-of-use of MapReduce. Spark differs from its predecessor in the below areas.

Processing - Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.

Generality - Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.



Ease of Use - Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala, Python, R, and SQL shells.

Runs Everywhere - You can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, on Mesos, or on Kubernetes. Access data in HDFS, Alluxio, Apache Cassandra, Apache HBase, Apache Hive, and hundreds of other data sources.

2. Spark SQL [2]

Spark SQL is a Spark module for structured data processing. It provides a programming abstraction called DataFrames and can also act as a distributed SQL query engine. It also provides powerful integration with the rest of the Spark ecosystem (e.g., integrating SQL query processing with machine learning). Spark SQL supports:

- Importing from and writing data to a variety of sources: Hive, Avro, Parquet, ORC, JSON, and JDBC
- HiveQL syntax, allowing access to existing Hive warehouses

- A server mode for JDBC and ODBC connectivity for business intelligence

Unlike the basic Spark RDD (Resilient Distributed Datasets) API, the interfaces provided by Spark SQL provide Spark with more information about the structure of both the data and the computation being performed. Internally, Spark SQL uses this extra information to perform extra optimizations. Spark SQL does this by providing:

- The Data Source API loads and stores structured data, with support for Hive, Avro, Parquet, JSON, JDBC, etc.
- The Dataframe API can perform both on external sources and on Spark's distributed collections. The dataframe API operates lazily so that it can perform relational optimizations. The Dataframe API is similar to Spark's existing RDD abstraction and is indeed an RDD but with the inclusion of a schema

The Catalyst API allows users to add new data sources, optimization rules, and data structures that are commonly used in machine learning. The cost-based optimization that the API provides makes querying in Spark SQL much faster than querying through traditional Spark RDD [3].

3. Spark MLlib [4]

Spark MLlib is the machine learning component of the Spark environment and offers many of the benefits inherent in Spark, including scalability, performance, ease of use, and compatibility across different platforms. Spark MLLib seamlessly integrates with other Spark components such as Spark SQL, Spark Streaming, and DataFrames. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:

- ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
- Featurization: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
- Persistence: saving and loading algorithms, models, and Pipelines
- Utilities: linear algebra, statistics, data handling, etc.

Because ML algorithms are typically iterative in nature, using Spark's in-memory capability tends to be more efficient. The addition of other libraries in the Spark environment, such as Spark SQL, Spark Streaming, and GraphX, which provide a range of tools to simplify machine learning projects, also makes MLlib highly appealing to data scientists. One such tool is MLlib's linear algebra package, Breeze, which depends on netlib-java for optimized numerical processing. If any of the native libraries are not available at runtime, you will see a warning message and a pure JVM implementation will be used instead [4]. Along with ML algorithms, MLlib provides support in basic statistics, hypothesis testing, linear algebra factorizations, feature extraction, transformations, model evaluation, and parameter tuning.

4. Tensorflow [7]

Tensorflow is an open-source machine learning library that was released by Google. Tensorflow has a particular focus on training and deep neural networks. Tensorflow uses n dimension lists called "tensors" to represent data. Then, tensors are put into the "flow" of some mathematical progress. To allow for fast mathematical operations, Tensorflow applications are written in python but the mathematical operations are translated from python to C++. Moreover, Tensorflow allows users to work with deep neural works in 5 simple steps:

- Collect dataset
- Build a neural network model
- Train the neural network model
- Evaluate the neural network model
- Predict with the neural network model

Since Tensorflow is particularly focused on the interface of deep neural networks, machine learning features like decision trees and random forests are not supported. It is very good for building and training deep neural networks but not so good with machine learning models.

5. Scikit-learn

Scikit-learn is one of the big data machine learning libraries. It has various classification, regression, and clustering algorithms. It started as a Google Summer of Code project by David Cournapeau. Scikit Learn was developed with NumPy for its high-performance linear algebra and array operations Being open-sourced, Scikit Learn is well kept and frequently updated. One advantage that Scikit Learn has over the other big data machine learning libraries i s that it is a high-level library. Scikit-learn

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

2.2 Comparison Chart

Although there are plenty of Machine learning technologies, this report is only focused on the three that we have implemented in our project: TensorFlow, Scikit-learn and Spark MLlib.

	TensorFlow	**Scikit-learn**	**Spark MLlib**
Supported Programming	Python	Python	Scala/Java/Python

Language			
Range of algorithms	Limited, Good for Deep learning algorithms	Good	Limited to distributed only
Speed (small - medium data size)	Medium-High	Medium	Medium
Scalability for Big Data	Very limited, only scales vertically	Very limited, only scales vertically	Excellent
Data Source Integration	Very Good	Very Good	Very Good
Visualization tools	Very Good	Very Good	Limited and depends on other partners
Learning Curve	High	Small	Average
IDEs	Jupyter / PyCharm	Eclipse / PyCharm / Jupyter	Eclipse / PyCharm / Jupyter

Objective:

The goal of this project is to utilize two datasets to compare the predictive power of Spark MLlib models against similar models from Scikit-learn and Tensorflow. To achieve a consistent comparison across the three ML technologies, data was cleaned equivalently using Spark SQL/Spark DataFrames for Spark ML models, pandas for Scikit-learn and Tensorflow models. Both datasets are fit with up to four models, with the exact number of models depending on whether the dataset necessitates binary classification or multiclass classification. The fit models were then asked to make categorical predictions of the test dataset from which accuracy, weighted recall, and weighted precision metrics are collected. Finally, the resulting metrics are compared across the three ML pipelines, two datasets, and four models.

Datasets:

A total of two datasets were analyzed in this project. One dataset necessitated classification models (Need to write the other dataset model). In general, all datasets were put through a similar pipeline for cleaning. First, the raw data was loaded into either Spark DataFrames/Spark SQL, pandas. Second, missing values were dropped or imputed where appropriate. Next, categorical variables were one-hot encoded. Finally, the cleaned data were summarized and written out in order to be read in for model fitting and evaluation. The following paragraphs provide details of each dataset along with specific data cleaning and imputation techniques.

Iris Dataset:

The Iris dataset [5] provides flower measurements that can be used to predict the species of iris. The raw dataset was already cleaned and contained no missing values.

Airline Passenger satisfaction Dataset:

The Airline Passenger satisfaction dataset [13] contains an airline passenger satisfaction survey and finds out factors which are highly correlated to a satisfied passenger and predict passenger satisfaction.

Dataset	Models	Records	Numeric Features	Categorical Features
Iris	Multiclass Classification (3 classes - “setosa", "versicolor", "virginica”)	150	4	0
Airline Passenger Satisfaction	Classification	129880	36	10

Airline Passenger satisfaction Data Preprocessing

Data preprocessing includes dealing with missing values, selecting relevant features, transforming dataset and creating new features in such a manner that datasets are suitable for Machine Learning models.

```
>>> df = df.select('Age', 'Gender', 'Customer Type', 'Type of Travel', 'Class', 'Flight Distance', 'Ease of Online booking', 'Food and drink', 'Online boardin
g', 'Seat comfort', 'Inflight service', 'Cleanliness', 'Departure Delay in Minutes', 'Arrival Delay in Minutes','satisfaction')
```

We calculated the number of missing values from each column and found 310 missing values in the dataset for column ‘Arrival Delay in Minutes’.

```
>>> from pyspark.sql.functions import isnan, when, count, col
>>> df.select([count(when(col(c).isNull(),c)).alias(c) for c in df.columns]).show()
| id|Gender|Customer Type|Age|Type of Travel|Class|Flight Distance|Inflight wifi service|Departure/Arrival time convenient|Ease of Online booking|Gate locatio
n|Food and drink|Online boarding|Seat comfort|Inflight entertainment|On-board service|Leg room service|Baggage handling|Checkin service|Inflight service|Clean
liness|Departure Delay in Minutes|Arrival Delay in Minutes|satisfaction|
|  0|     0|            0|  0|             0|    0|              0|                    0|                                0|                     0|
0|             0|              0|           0|                     0|               0|               0|               0|              0|               0|
     0|                         0|                     310|           0|
```

As observed, ‘Arrival Delay in Minutes’ column has missing values. We had multiple options to deal with missing columns, like dropping them or substituting with mean value or filling them with zeros and so on. We chose to substitute the missing values by the mean value.

```
>>> from pyspark.sql.functions import mean as _mean, stddev as _stddev, col
>>> df_stats = df.select(_mean(col('Arrival Delay in Minutes')).alias('mean'),_stddev(col('Arrival Delay in Minutes')).alias('std')).collect()
>>> mean = df_stats[0]['mean']
>>> std = df_stats[0]['std']
>>> df = df.fillna(mean, subset=['Arrival Delay in Minutes'])
```

```
>>> df.select([count(when(col(c).isNull(),c)).alias(c) for c in df.columns]).show()
```

Age	Gender	Customer Type	Class	Type of Travel	Flight Distance	Departure Delay in Minutes	Arrival Delay in Minutes	Ease of Online booking	Food and Drink	Online boarding	Seat comfort	Inflight Service	Cleanliness
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Once our dataset is cleaned, the next step is to process the dataset to create a model using MLlib. MLlib standardizes APIs for machine learning algorithms to make it easier to combine multiple algorithms into a single pipeline, or workflow. Pipeline is used for preparing the dataset features suitable for machine learning models.

```
>>> from pyspark.ml import Pipeline
>>> pipeline = Pipeline(stages = stages)
>>> pipelineModel = pipeline.fit(df)
>>> df = pipelineModel.transform(df)
>>> selectedCols = ['features'] + cols
>>> df = df.select(selectedCols)
```

```
>>> df = df.selectExpr("features","satisfaction as label","Age", "Gender", "`Customer Type`", "`Type of Travel`", "Class", "`Flight Distance`", "`Ease of Online booking`", "`Food and drink`", "`Online boarding`", "`Seat comfort`", "`Inflight service`", "Cleanliness", "`Departure Delay in Minutes`", "`Arrival Delay in Minutes`","satisfaction")
```

After input is transformed through pipeline this is how our data looks:

```
>>> df.show(5)
```

features	label	Age	Gender	Customer Type	Type of Travel	Class	Flight Distance	Ease of Online booking	Food and drink	Online boarding	Seat comfort	Inflight service	Cleanliness	Departure Delay in Minutes	Arrival Delay in Minutes	satisfaction
(39,[1,5,11,16,21...	1	13	Male	Loyal Customer	Personal Travel	Eco Plus	460	3	5	3	5	5	5	25	18	1
(39,[2,4,5,14,16,...	1	25	Male	disloyal Customer	Business travel	Business	235	3	1	3	1	4	1	1	6	1
(39,[0,1,2,4,6,11...	0	26	Female	Loyal Customer	Business travel	Business	1142	2	5	5	5	4	5	0	0	0
(39,[0,1,2,4,9,13...	1	25	Female	Loyal Customer	Business travel	Business	562	5	2	2	2	4	2	11	9	1
(39,[1,2,4,5,10,1...	0	61	Male	Loyal Customer	Business travel	Business	214	3	4	5	5	3	3	0	0	0

only showing top 5 rows

Models:

Iris Dataset:

After our dataset was cleaned, the records were partitioned by a 75 - 25 train test split for model fitting and evaluation. Models were implemented on Spark MLlib, Scikit-learn and Tensorflow. All cross validation models use 3 fold cross validation and all parameters not explicitly listed below were defaulted. The default parameters can be found in Spark MLlib [5], Scikit-learn [6], and Tensorflow [7]. While hyperparameters for training and cross validation were kept as consistent as possible across all three implementations, it should be noted that default parameters and varying implementations may contribute to different result metrics from the three packages. The following paragraphs describe specific models and their parameters as implemented across all three packages.

Logistic Regression Classification by Cross Validation [8]

The logistic regression model was cross validated with respect to both the regression parameter and elastic net parameter. The meaning of both parameters is the same as for the linear regression model. As in linear regression, the regression parameter (lambda) was tested with values of 0, ¼, and ½ ,and the elastic net parameter (alpha) was tested with values 0, ¼, and ½.

Decision Tree Classification by Cross Validation [9]

The decision tree model was cross validated with respect to both the max depth of the tree and max bins of the tree. The max depth of the tree was tested with values 5, 10, and 15 while the max bins were tested with values 8, 16, 32.

Random Forest Classification by Cross Validation [10]

The random forest model was cross validated with respect to both the max depth of each tree and the number of trees in the forest. The max depth of the tree was tested with values 5, 10, and 15 and the number of trees was tested with values 10, 15, and 20.

Naive Bayes Classification by Cross Validation [11]

The naive bayes model was cross validated with respect to the smoothing parameter which was tested with values ½, 1, and 2.

Airline Passenger Satisfaction Dataset:

We have the dataset ready for training the ML model. We split the dataset into a training and testing set with 80:20 ratio as below. We train the model using Logistic Regression, Random Forest, Decision Tree, Gradient Boosted Tree classifiers and attempt to evaluate their results in the following sections. ROC, receiver operator characteristic, is an evaluation matrix for the above classifiers. It is a probability curve plotting the true positive rate against the false positive rate and helps to separate the signal from noise. The area under the curve, AUC, is a measure of the ability of a classifier to distinguish between classes and depicts the summary of the ROC curve. Precision is the ratio of True positives and all the positives. Recall is a measure of the model correctly identifying True positives.

```
>>> train, test = df.randomSplit([0.8, 0.2], seed = 2018)
>>> print("Training Dataset Count: " + str(train.count()))
Training Dataset Count: 83236
>>> print("Test Dataset Count: " + str(test.count()))
Test Dataset Count: 20737
```

Logistic Regression [8]

Logistic Regression is used to predict the probability of a categorical dependent variable i.e., features column. The dependent variable is a binary variable (0 or 1). The model builds a regression model to predict the probability that a given data belongs to a features category.

```
>>> from pyspark.ml.classification import LogisticRegression
>>> lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=10)
>>> lrModel = lr.fit(train)
```

```
>>> import numpy as np
>>> beta = np.sort(lrModel.coefficients)
>>> print(beta)
[-3.82730017e+00 -2.52618484e+00 -1.73176222e+00 -6.94224140e-01
 -2.30549680e-01 -1.68316650e-05  5.22445570e-04  3.37187314e-03
  6.74644944e-03  5.77630110e-02  8.62442732e-02  1.08459646e-01
  1.30965431e-01  2.41827552e-01  2.80059787e-01  2.97335582e-01
  3.32724602e-01  3.71102247e-01  3.92360400e-01  4.14466745e-01
  4.51111978e-01  5.70666217e-01  6.05857386e-01  6.78550506e-01
  7.27586938e-01  7.62361823e-01  9.84597551e-01  1.00067737e+00
  1.09630104e+00  1.32992008e+00  1.35701057e+00  1.43447989e+00
  1.44519454e+00  1.51438075e+00  1.59338185e+00  1.65829307e+00
  2.21105139e+00  2.33368380e+00  2.43215048e+00]
```

```
>>> roc = trainingSummary.roc
>>> print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))
Training set areaUnderROC: 0.9497078706951031
```

```
>>> from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>> evaluator = BinaryClassificationEvaluator()
>>> print('Test Area Under ROC', evaluator.evaluate(predictions))
Test Area Under ROC 0.9512744216892577
```

```
>>> from pyspark.ml.evaluation import MulticlassClassificationEvaluator
>>> evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
>>> print('Accuracy', evaluator.evaluate(predictions))
Accuracy 0.8856150841491055
```

Decision Tree Classifier [9]

Decision Tree classifier is used to build the model and fit it with the train data. We have fixed the maximum depth to 3. This parameter is chosen after trying various values for maximum depth and considering the optimal one.

```
>>> from pyspark.ml.classification import DecisionTreeClassifier
>>> dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth = 3)
>>> dtModel = dt.fit(train)
>>> predictions = dtModel.transform(test)
>>> predictions.select('label', 'rawPrediction', 'prediction', 'probability').show(10)
+-----+----------------+----------+--------------------+
|label|   rawPrediction|prediction|         probability|
+-----+----------------+----------+--------------------+
|    0|[12062.0,2852.0]|       0.0|[0.80877028295561...|
|    0|[12062.0,2852.0]|       0.0|[0.80877028295561...|
|    0|[12062.0,2852.0]|       0.0|[0.80877028295561...|
|    0|[12062.0,2852.0]|       0.0|[0.80877028295561...|
|    0|[12062.0,2852.0]|       0.0|[0.80877028295561...|
|    0|[12062.0,2852.0]|       0.0|[0.80877028295561...|
|    0|[12062.0,2852.0]|       0.0|[0.80877028295561...|
|    0|[12062.0,2852.0]|       0.0|[0.80877028295561...|
|    0|[12062.0,2852.0]|       0.0|[0.80877028295561...|
|    0|[12062.0,2852.0]|       0.0|[0.80877028295561...|
+-----+----------------+----------+--------------------+
only showing top 10 rows
```

```
>>> evaluator = BinaryClassificationEvaluator()
>>> print("Test Area Under ROC: " + str(evaluator.evaluate(predictions, {evaluator.metricName: "areaUnderROC"})))
Test Area Under ROC: 0.8553736616929443
```

```
>>> evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
>>> print('Accuracy', evaluator.evaluate(predictions))
Accuracy 0.8393692433813956
```

Random Forest Classifier [10]

Random Forest is a classification algorithm that uses bagging and features randomness when building each individual tree to try to create a decorrelated forest of trees. We will build the model and fit it with the train data.

```
>>> from pyspark.ml.classification import RandomForestClassifier
>>> rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label')
>>> rfModel = rf.fit(train)
>>> predictions = rfModel.transform(test)
>>> predictions.select('label', 'rawPrediction', 'prediction', 'probability').show(10)
+-----+--------------------+----------+--------------------+
|label|       rawPrediction|prediction|         probability|
+-----+--------------------+----------+--------------------+
|    0|[14.8102366136110...|       0.0|[0.74051183068055...|
|    0|[15.7031782625540...|       0.0|[0.78515891312770...|
|    0|[15.7031782625540...|       0.0|[0.78515891312770...|
|    0|[15.7031782625540...|       0.0|[0.78515891312770...|
|    0|[15.7031782625540...|       0.0|[0.78515891312770...|
|    0|[14.8102366136110...|       0.0|[0.74051183068055...|
|    0|[14.8102366136110...|       0.0|[0.74051183068055...|
|    0|[15.7031782625540...|       0.0|[0.78515891312770...|
|    0|[15.7031782625540...|       0.0|[0.78515891312770...|
|    0|[15.7031782625540...|       0.0|[0.78515891312770...|
+-----+--------------------+----------+--------------------+
only showing top 10 rows
```

```
>>> evaluator = BinaryClassificationEvaluator()
>>> print("Test Area Under ROC: " + str(evaluator.evaluate(predictions, {evaluator.metricName: "areaUnderROC"})))
Test Area Under ROC: 0.941533129491009
```

```
>>> evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
>>> print('Accuracy', evaluator.evaluate(predictions))
Accuracy 0.877224285094276
```

Gradient Boosted Tree Classifier [13]

The Gradient-Boosted Tree builds trees sequentially and employs gradient descent algorithms to minimize errors in sequential models. Gradient boosting models are becoming popular because of their effectiveness at classifying complex datasets. We will build the model and fit it with the train data.

```
>>> from pyspark.ml.classification import GBTClassifier
>>> gbt = GBTClassifier(maxIter=10)
>>> gbtModel = gbt.fit(train)
>>> predictions = gbtModel.transform(test)
>>> predictions.select('label', 'rawPrediction', 'prediction', 'probability').show(10)
+-----+--------------------+----------+--------------------+
|label|       rawPrediction|prediction|         probability|
+-----+--------------------+----------+--------------------+
|    0|[0.84452128710240...|       0.0|[0.84409820757021...|
|    0|[0.84452128710240...|       0.0|[0.84409820757021...|
|    0|[0.84452128710240...|       0.0|[0.84409820757021...|
|    0|[0.84452128710240...|       0.0|[0.84409820757021...|
|    0|[0.84452128710240...|       0.0|[0.84409820757021...|
|    0|[0.84452128710240...|       0.0|[0.84409820757021...|
|    0|[0.84452128710240...|       0.0|[0.84409820757021...|
|    0|[0.84452128710240...|       0.0|[0.84409820757021...|
|    0|[0.84452128710240...|       0.0|[0.84409820757021...|
|    0|[0.84452128710240...|       0.0|[0.84409820757021...|
+-----+--------------------+----------+--------------------+
only showing top 10 rows
```

```
>>> evaluator = BinaryClassificationEvaluator()
>>> print("Test Area Under ROC: " + str(evaluator.evaluate(predictions, {evaluator.metricName: "areaUnderROC"})))
Test Area Under ROC: 0.9619261821491389
```

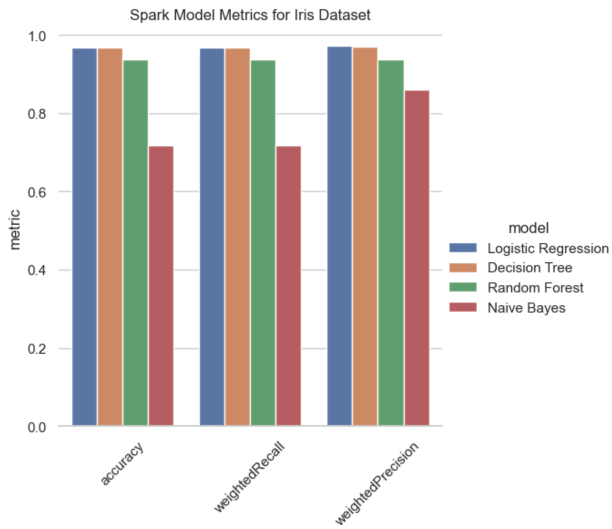
```
>>> evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
>>> print('Accuracy', evaluator.evaluate(predictions))
Accuracy 0.8979601678159811
```

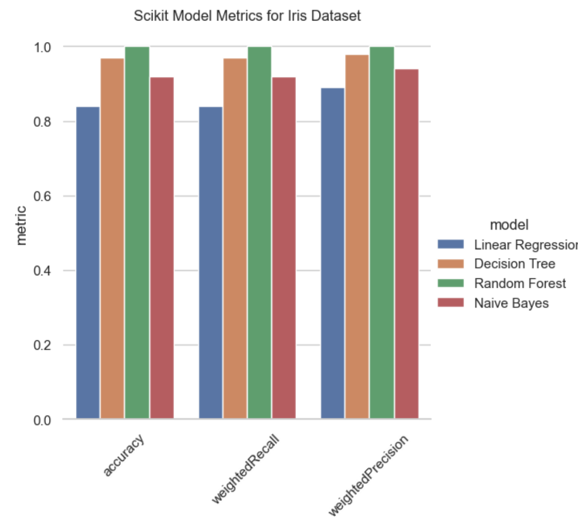
The below table shows the results of all ML models:

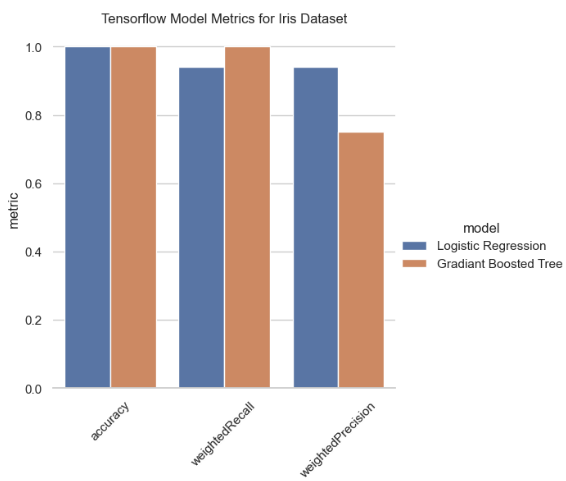
Metrics	**Logistic Regression**	**Decision Tree**	**Random Forest**	**Gradient Boosted Tree**
Accuracy	0.8856	0.8393	0.8772	0.8979
Test Area under ROC	0.9512	0.8553	0.9415	0.9619

Results:

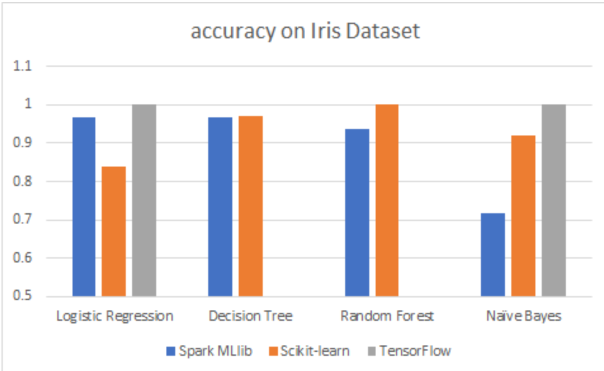
The following three bar charts represent the results of each Spark MLlib, Scikit-learn and TensorFlow algorithm on the Iris dataset.

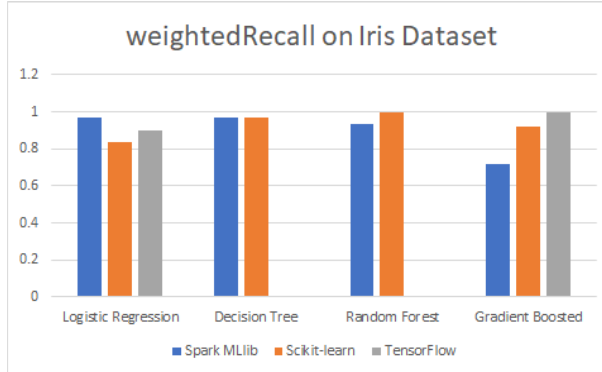


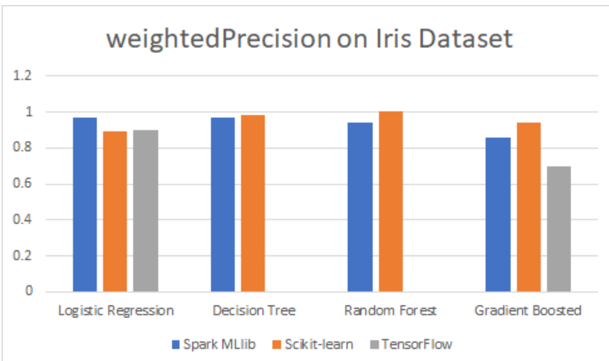




The following graphs below compare the results across the three ML technologies. We selected the Iris dataset to show the performance differences between.

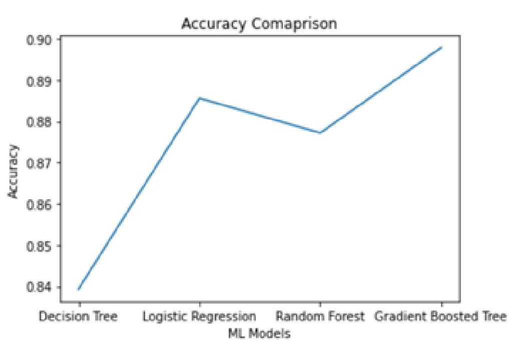


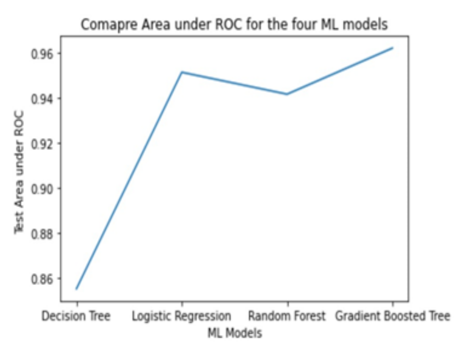




For the Iris dataset, the performances across the different models, as measured by the three criteria, are fairly similar, with not one technology consistently outperforming the other two. One exception comes when implementing Naive Bayes, in which R caret outperforms the other two consistently across all classification datasets. This difference in performance can be explained by the different hyperparameter defaults set in each technology. As such, some of the set defaults do not return adequate results depending on the datasets. All three technologies allow for ways to access hyperparameter tuning so ease of tuning is similar across all three. Also we noticed that tensorflow has limitations in implementing ML algorithms like Decision Tree and Random Forest algorithm directly, to implement we have to use external modules like scikit-learn in order to build the tensorflow ML models.

The below graphs illustrate the performance of Airline Passenger Satisfaction dataset on different ML algorithms across Spark MLlib:





One great tuning tool that improves MLlib’s ease of use is Apache MLflow, an open-source platform for tracking machine learning model tuning that can be implemented in Python and R. Automated MLflow is supported by MLlib, giving it an edge, as users can run tuning code and have hyperparameters and metrics automatically logged. Explicit API logs to MLflow must be made if automated MLflow is not supported [14].

Conclusion:

There are many advantages to using Spark for machine learning. We discussed several tools in the Spark environment that add additional functionality and ease of implementation when used alongside MLlib. We then compared Spark MLlib algorithms against the same algorithms in TensorFlow and Scikit-learn and saw that performance was comparable on the datasets. While it runs comparatively well on the small datasets, Spark’s true advantage can be seen on large scale datasets, where the runtime will outpace the other two technologies and scaling is considerably cheaper.

The Spark environment and its work with machine learning are still fairly new. But with its current popularity and its still growing user base, more and more research and contributions will be put forth to further develop new Spark tools and to refine existing tools. We expect that one of the weaknesses of MLlib, its limited number of ML models, will be addressed in the future through either a direct expansion of MLlib or through new Spark open-source projects.

References:

[1] Apache Spark
https://spark.apache.org/

[2] Apache Spark SQL
https://spark.apache.org/docs/latest/sql-programming-guide.html

[3] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. 2015. Spark SQL: Relational Data Processing in Spark. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD ’15). Association for Computing Machinery, New York, NY, USA, 1383–1394.
https://dl.acm.org/doi/10.1145/2723372.2742797

[4] Apache Spark MLlib
https://spark.apache.org/docs/latest/ml-guide.html

[5] Iris Data Set. UCI Machine Learning Repository.
http://archive.ics.uci.edu/ml/datasets/iris

[5] MLlib: Main Guide - Spark 2.4.5 Documentation. Machine Learning Library (MLlib) Guide.

https://spark.apache.org/docs/latest/ml-guide.html

[6] Scikit learn: User Guide.
https://scikit-learn.org/stable/user_guide.html

[7] TensorFLow: Guide
https://www.tensorflow.org/guide

[8] Logistic Regression Classification by Cross Validation
https://medium.com/@lily_su/logistic-regression-accuracy-cross-validation-58d9eb58d6e6

[9] Decision Tree Classification by Cross Validation
http://www.cs.toronto.edu/~fidler/teaching/2015/slides/CSC411/tutorial3_CrossVal-DTs.pdf

[10] Random Forest Classification by Cross Validation
https://medium.com/@hjhuney/implementing-a-random-forest-classification-model-in-python-583891c99652

[11] Naive Bayes Classification by Cross Validation
https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn

[12] Airline Passenger Satisfaction Dataset
https://www.kaggle.com/teejmahal20/airline-passenger-satisfaction

[13] Gradient Boosted Tree Classifier
https://spark.apache.org/docs/latest/ml-classification-regression.html#gradient-boosted-tree-classifier

[14] Apache Spark MLlib and automated MLflow tracking.
https://docs.databricks.com/applications/machine-learning/automl/mllib-mlflow-integration.html

[15] Github repository for our project code
https://github.com/Vaishu1319/CSP554-Final-Project

Appendix:

Dataset contains train.csv and test.csv, used for training the machine learning model and testing the model respectively.

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.7-amzn-1
      /_/

Using Python version 3.7.9 (default, Feb 18 2021 03:10:35)
SparkSession available as 'spark'.
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder.appName('ml-airline-satisfaction').getOrCreate()
>>> df = spark.read.csv("/user/hadoop/train.csv", header = True, inferSchema = True)
```

Figure 3.1: Dataframe and Spark initialization

Data profiling

Contents of the dataset is as follows

Gender	:	Gender of the passenger
Age	:	Age of the passenger
Type of travel	:	Purpose of flight
Class	:	Travel class
Flight distance	:	Flight journey distance
Inflight WiFi service	:	Satisfaction level of the inflight wifi service (0: Not Applicable: 1-5)
Time Convenience	:	Satisfaction level of Departure/Arrival time convenient
Ease of online booking	:	Satisfaction level of online booking
Gate location	:	Satisfaction level of Gate location
Food and drink	:	Satisfaction level of Food and drink
Online boarding	:	Satisfaction level of online boarding
Seat comfort	:	Satisfaction level of Seat comfort
Inflight entertainment	:	Satisfaction level of inflight entertainment
On-board service	:	Satisfaction level of On-board service
Leg room service	:	Satisfaction level of Leg room service
Baggage handling	:	Satisfaction level of baggage handling
Check-in service	:	Satisfaction level of Check-in service
Inflight service	:	Satisfaction level of inflight service
Arrival Delay in Minutes	:	Minutes delayed when Arrival
Satisfaction	:	Airline satisfaction level(Satisfaction, neutral or dissatisfaction)
Cleanliness	:	Satisfaction level of Cleanliness
Departure Delay in Minutes	:	Minutes delayed when departure

Using the command printSchema(), schema can be printed as below

```
>>> df.printSchema()
root
 |-- id: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Customer Type: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Type of Travel: string (nullable = true)
 |-- Class: string (nullable = true)
 |-- Flight Distance: integer (nullable = true)
 |-- Inflight wifi service: integer (nullable = true)
 |-- Departure/Arrival time convenient: integer (nullable = true)
 |-- Ease of Online booking: integer (nullable = true)
 |-- Gate location: integer (nullable = true)
 |-- Food and drink: integer (nullable = true)
 |-- Online boarding: integer (nullable = true)
 |-- Seat comfort: integer (nullable = true)
 |-- Inflight entertainment: integer (nullable = true)
 |-- On-board service: integer (nullable = true)
 |-- Leg room service: integer (nullable = true)
 |-- Baggage handling: integer (nullable = true)
 |-- Checkin service: integer (nullable = true)
 |-- Inflight service: integer (nullable = true)
 |-- Cleanliness: integer (nullable = true)
 |-- Departure Delay in Minutes: integer (nullable = true)
 |-- Arrival Delay in Minutes: integer (nullable = true)
 |-- satisfaction: integer (nullable = true)
```

Below figure shows the distribution of data in the training dataset. It can be observed that there are 4 numerical, 10 categorical features in the data set and a target numerical feature. Shape of the dataset is 103904 x 24.

```
>>> df.show(5)
```

id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	Food and drink	Online boarding	Seat comfort	Inflight entertainment	On-board service	Leg room service	Baggage handling	Checkin service	Inflight service	Cleanliness	Departure Delay in Minutes	Arrival Delay in Minutes	satisfaction
70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	3	1	5	3	5	5	4	3	4	4	5	5	25	18	1
5047	Male	disloyal Customer	25	Business travel	Business	235	3	2	3	3	1	3	1	1	1	5	3	1	4	1	1	6	1
110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2	2	2	5	5	5	5	4	3	4	4	4	5	0	0	0
24026	Female	Loyal Customer	25	Business travel	Business	562	2	5	5	5	2	2	2	2	2	5	3	1	4	2	11	9	1
119299	Male	Loyal Customer	61	Business travel	Business	214	3	3	3	3	4	5	5	3	3	4	4	3	3	3	0	0	0

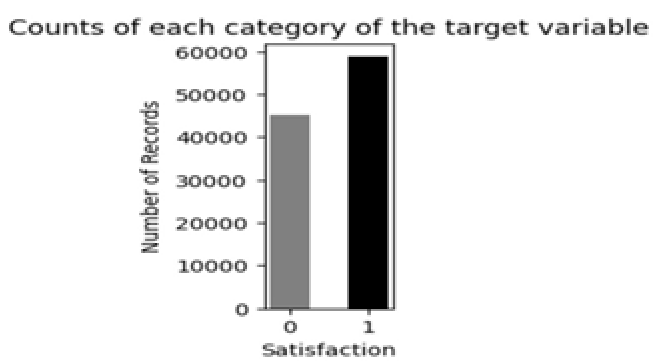
only showing top 5 rows

```
>>> rows = df.count()
>>> print('N rows = '+str(rows))
N rows = 103904
>>> columns = len(df.columns)
>>> print('N columns = '+str(columns))
N columns = 24
```

Count of rows and columns: 103904 and 24

```
>>> df.groupBy(df['satisfaction']).count().show()
+------------+-----+
|satisfaction|count|
+------------+-----+
|           1|58879|
|           0|45025|
+------------+-----+
```

Balance of target variable: 58879 satisfied customers and 45025 unsatisfied customers are present in the training dataset. Approximately 56% of passengers are satisfied and 43% are not satisfied with the airline.



Numerical Features:

Summary statistics for numeric features: Observing the above data of numerical features, we can expect to have outliers. Boxplot is used to detect the outliers. It used the median, 25th(Q1) and 75th(Q2) percentile. Difference between 75th and 25th percentiles is called InterQuartile Range (IQR).

LowerBound and UpperBound are calculated as (Q1 – 1.5 * IQR) and (Q2 + 1.5 * IQR). Values which are lesser than LowerBound and greater than UpperBound are considered as outliers.

```
>>> df.select(numeric_features).describe().show()
+-------+-----------------+-----------------+--------------------------+------------------------+
|summary|              Age|  Flight Distance|Departure Delay in Minutes|Arrival Delay in Minutes|
+-------+-----------------+-----------------+--------------------------+------------------------+
|  count|           103904|           103904|                    103904|                  103594|
|   mean|39.379706267323684|1189.4483754234677|        14.815618263012011|       15.178678301832152|
| stddev|15.114963699737796| 997.1472805289604|         38.23090058414182|       38.69868202096655|
|    min|                7|               31|                         0|                       0|
|    max|               85|             4983|                      1592|                    1584|
+-------+-----------------+-----------------+--------------------------+------------------------+
```

Outliers

Analyze percentiles and outliers on Numeric Features:

1. Calculating the outliers for Age Column

   25th and 75th percentile for Age is 27 and 51, respectively.
   IQR = 51 – 27 = 24
   LowerBound = Q1 – 1.5 * IQR = 27 – 1.5 * 24 = -9
   UpperBound = Q2 + 1.5 * IQR = 51 + 1.5 * 24 = 87

```
>>> df.selectExpr('percentile(Age, 0.25)').show()
+-------------------------------------+
|percentile(Age, CAST(0.25 AS DOUBLE), 1)|
+-------------------------------------+
|                                 27.0|
+-------------------------------------+

>>> df.selectExpr('percentile(Age, 0.75)').show()
+-------------------------------------+
|percentile(Age, CAST(0.75 AS DOUBLE), 1)|
+-------------------------------------+
|                                 51.0|
+-------------------------------------+
```

Outliers in ‘Age’ Column: There are no outliers in the Age column, as min value is 7 and max value is 85. Both fall with the range of -9 and 87.

```
df.filter((df['Age'] < -9) | (df['Age'] > 87)).show()

+---+------+-------------+---+--------------+-----+---------------+---------------------+---------------------------------+----------------------+-------------+--------------+---------------+---------
| id|Gender|Customer Type|Age|Type of Travel|Class|Flight Distance|Inflight wifi service|Departure/Arrival time convenient|Ease of Online booking|Gate location|Food and drink|Online boarding|Seat comf
+---+------+-------------+---+--------------+-----+---------------+---------------------+---------------------------------+----------------------+-------------+--------------+---------------+---------
+---+------+-------------+---+--------------+-----+---------------+---------------------+---------------------------------+----------------------+-------------+--------------+---------------+---------
```

```
df.filter((df['Age'] < -9) | (df['Age'] > 87)).count()

0
```

2. Calculating the outliers for Flight Distance Column

25th and 75th percentile for Flight Distance

IQR, LowerBound and UpperBound for ‘Flight Distance’ are 1329, -1579.5, 3736.5, respectively.

```
>>> df.selectExpr('percentile(`Flight Distance`, 0.25)').show()
+-----------------------------------------------------+
|percentile(Flight Distance, CAST(0.25 AS DOUBLE), 1)|
+-----------------------------------------------------+
|                                                414.0|
+-----------------------------------------------------+

>>> df.selectExpr('percentile(`Flight Distance`, 0.75)').show()
+-----------------------------------------------------+
|percentile(Flight Distance, CAST(0.75 AS DOUBLE), 1)|
+-----------------------------------------------------+
|                                               1743.0|
+-----------------------------------------------------+
```

Outliers in ‘Flight Distance’ Column: There are 2291 outliers in the ‘Flight Distance’ column, as min value is 31 and max value is 4983

```
df.filter((df['Flight Distance'] < -1579.5) | (df['Flight Distance'] > 3736.5)).show()
```

id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gat
73302	Male	Loyal Customer	26	Business travel	Business	3960	1	1	1	
101275	Male	Loyal Customer	52	Business travel	Business	3747	5	5	5	
66800	Female	Loyal Customer	43	Business travel	Business	3854	5	5	5	
23328	Female	Loyal Customer	38	Business travel	Business	3753	2	2	2	
85109	Male	Loyal Customer	46	Business travel	Business	3995	4	4	4	
42926	Female	Loyal Customer	43	Business travel	Business	3946	3	3	3	
67702	Male	Loyal Customer	51	Business travel	Business	3991	5	5	5	
3875	Female	Loyal Customer	44	Business travel	Business	3767	1	1	1	
96830	Male	Loyal Customer	37	Business travel	Business	3744	5	5	5	
111496	Female	Loyal Customer	44	Business travel	Business	3979	4	4	4	
97335	Male	Loyal Customer	40	Business travel	Business	3764	1	1	1	
94609	Male	Loyal Customer	50	Business travel	Business	3960	4	4	4	
64701	Female	Loyal Customer	37	Business travel	Business	3969	1	3	3	
129624	Male	Loyal Customer	32	Business travel	Business	3802	0	0	0	
50393	Male	Loyal Customer	52	Business travel	Business	3807	1	3	3	
6274	Female	Loyal Customer	43	Business travel	Business	3967	3	4	4	
105516	Female	Loyal Customer	44	Business travel	Business	3954	1	1	1	
68153	Female	Loyal Customer	51	Business travel	Business	3993	1	1	1	
117009	Male	Loyal Customer	52	Business travel	Business	3885	2	2	2	
74085	Female	Loyal Customer	52	Business travel	Business	3895	4	4	4	

only showing top 20 rows

```
df.filter((df['Flight Distance'] < -1579.5) | (df['Flight Distance'] > 3736.5)).count()

2291
```

3. Calculating the outliers for Departure Delay Column

25th and 75th percentile for Departure Delay in Minutes

IQR, LowerBound and UpperBound for ‘Departure Delay in Minutes’ are 12, -18, 30, respectively.

```
>>> df.selectExpr('percentile(`Departure Delay in Minutes`, 0.25)').show()
+-------------------------------------------------------------+
|percentile(Departure Delay in Minutes, CAST(0.25 AS DOUBLE), 1)|
+-------------------------------------------------------------+
|                                                          0.0|
+-------------------------------------------------------------+

>>> df.selectExpr('percentile(`Departure Delay in Minutes`, 0.75)').show()
+-------------------------------------------------------------+
|percentile(Departure Delay in Minutes, CAST(0.75 AS DOUBLE), 1)|
+-------------------------------------------------------------+
|                                                         12.0|
+-------------------------------------------------------------+
```

Outliers in ‘Departure Delay in Minutes’ Column: There are 14529 outliers in the dataset. The min and max values are 0 and 1592, respectively.

```
>>> df.filter((df['Departure Delay in Minutes']< -18) | (df['Departure Delay in Minutes']> 30)).show()
```

id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	Food and drink	Online boarding	Seat comfort	Inflight entertainment	On-board service	Leg room service	Baggage handling	Checkin service	Inflight service	Cleanliness	Departure Delay in Minutes	Arrival Delay in Minutes	satisfaction
95709	Female	Loyal Customer	26	Personal Travel	Eco	453	3	2	3	2	2	3	2	2	4	3	2	2	1	2	43	35	1
71142	Female	Loyal Customer	26	Business travel	Business	2123	3	3	3	3	4	4	4	4	5	3	4	5	4	4	49	51	0
43510	Female	Loyal Customer	43	Personal Travel	Eco	752	3	5	3	5	5	4	5	3	3	3	5	3	3	4	52	29	1
105420	Female	disloyal Customer	23	Business travel	Eco	452	5	0	5	1	1	5	1	1	4	5	5	3	5	1	54	44	0
70594	Female	Loyal Customer	35	Business travel	Business	2611	4	5	4	4	4	4	4	3	3	4	5	4	3	4	109	120	0
7467	Female	Loyal Customer	45	Business travel	Business	3334	2	4	2	2	4	5	5	4	4	4	4	3	4	5	51	48	0
114501	Female	disloyal Customer	38	Business travel	Business	308	3	3	3	3	4	3	4	4	1	1	4	2	3	4	39	26	1
75930	Male	Loyal Customer	30	Business travel	Eco Plus	205	3	2	2	2	3	3	3	3	2	5	1	1	2	3	64	49	1
73302	Male	Loyal Customer	26	Business travel	Business	3960	1	1	1	1	4	4	4	4	4	2	5	4	4	4	45	48	0
88062	Male	Loyal Customer	47	Business travel	Business	406	2	2	2	2	3	5	5	4	4	4	4	5	4	4	44	50	0
118486	Female	disloyal Customer	41	Business travel	Business	304	5	5	5	3	2	5	2	2	3	4	4	4	4	2	31	19	0
6012	Male	Loyal Customer	30	Business travel	Business	1896	4	1	1	1	4	4	4	4	2	2	3	4	2	4	81	72	1
15130	Female	Loyal Customer	32	Business travel	Eco	1042	4	1	1	1	4	4	4	4	4	3	1	5	2	4	35	34	0
93076	Female	Loyal Customer	15	Personal Travel	Eco Plus	297	1	5	1	3	5	1	5	5	5	1	1	3	4	5	67	62	1
85494	Female	Loyal Customer	16	Personal Travel	Eco	332	2	5	2	1	3	2	4	3	5	4	5	4	4	3	40	52	1
23400	Male	Loyal Customer	20	Personal Travel	Eco	483	2	3	2	3	5	2	5	5	5	2	3	2	4	5	91	82	1
106007	Male	disloyal Customer	24	Business travel	Eco	791	3	3	3	3	5	3	5	5	2	1	3	1	3	5	39	6	1
112483	Male	Loyal Customer	13	Personal Travel	Eco	853	1	4	1	4	4	1	4	4	5	2	5	4	4	4	105	122	1
58896	Male	disloyal Customer	26	Business travel	Eco	888	2	2	2	3	5	2	5	5	4	3	3	2	4	5	51	30	1
13884	Male	Loyal Customer	49	Business travel	Eco	578	5	1	1	1	5	5	5	1	1	1	4	5	4	5	162	179	0

```
only showing top 20 rows
```

```
>>> df.filter((df['Departure Delay in Minutes']< -18) | (df['Departure Delay in Minutes']> 30)).count()
14529
```

4. Calculating the outliers for Arrival Delay Column

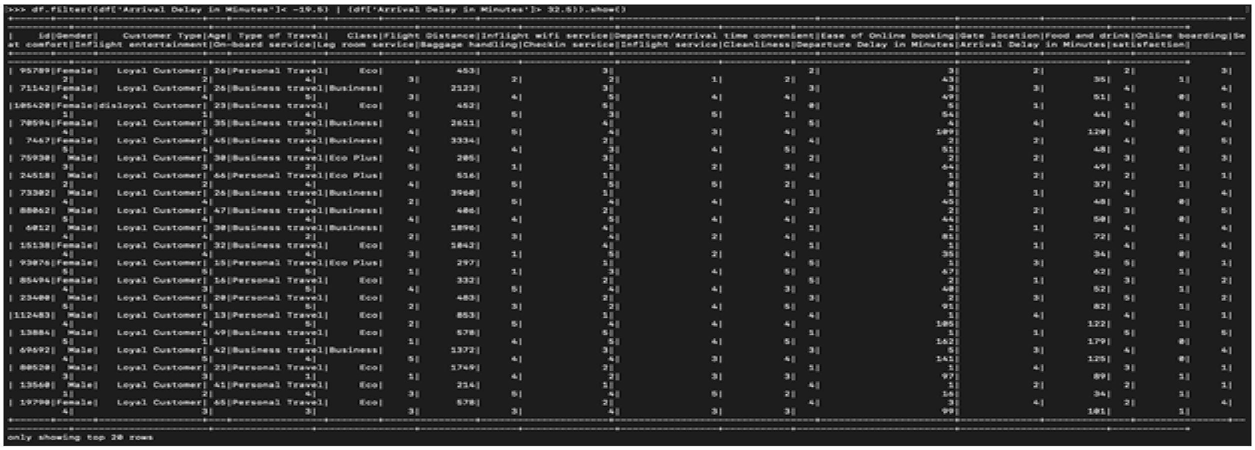
25th and 75th percentile for Arrival Delay in Minutes

IQR, LowerBound and UpperBound for ‘Arrival Delay in Minutes are 13, -19.5, 32.5, respectively.

```
>>> df.selectExpr('percentile(`Arrival Delay in Minutes`, 0.25)').show()
+-------------------------------------------------------------+
|percentile(Arrival Delay in Minutes, CAST(0.25 AS DOUBLE), 1)|
+-------------------------------------------------------------+
|                                                          0.0|
+-------------------------------------------------------------+

>>> df.selectExpr('percentile(`Arrival Delay in Minutes`, 0.75)').show()
+-------------------------------------------------------------+
|percentile(Arrival Delay in Minutes, CAST(0.75 AS DOUBLE), 1)|
+-------------------------------------------------------------+
|                                                         13.0|
+-------------------------------------------------------------+
```

Outliers in ‘Arrival Delay in Minutes’ Column: There are outliers in the ‘Arrival Delay in Minutes’ column. Min and Max values are 0 and 1584, respectively.




```
>>> df.filter((df['Arrival Delay in Minutes']< -19.5) | (df['Arrival Delay in Minutes']> 32.5)).count()
13954
```

Categorical Features

Categorical Features from the dataset are as follows:

- Gender
- Customer Type
- Class
- Type of Travel
- Ease of Online booking
- Food and Drink
- Online boarding
- Seat comfort
- Inflight service
- Cleanliness

For each of the above features we calculate the number of categories and count of each category in the following section.

Count Distinct Values: For Gender:

```
>>> df.groupBy(df['Gender']).count().show()
+------+-----+
|Gender|count|
+------+-----+
|  Male|51177|
|Female|52727|
+------+-----+

>>> df.groupBy(df['Gender']).count().count()
2
```

For Customer Type:

```
>>> df.groupBy(df['Customer Type']).count().show()
+-----------------+-----+
|    Customer Type|count|
+-----------------+-----+
|   Loyal Customer|84923|
|disloyal Customer|18981|
+-----------------+-----+

>>> df.groupBy(df['Customer Type']).count().count()
2
```

For Class:

```
>>> df.groupBy(df['Class']).count().show()
+--------+-----+
|   Class|count|
+--------+-----+
|Eco Plus| 7494|
|Business|49665|
|     Eco|46745|
+--------+-----+

>>> df.groupBy(df['Class']).count().count()
3
```

For Type of Travel:

```
>>> df.groupBy(df['Type of Travel']).count().show()
+---------------+-----+
| Type of Travel|count|
+---------------+-----+
|Personal Travel|32249|
|Business travel|71655|
+---------------+-----+

>>> df.groupBy(df['Type of Travel']).count().count()
2
```

For Ease of booking online:

```
>>> df.groupBy(df['Ease of Online booking']).count().show()
+----------------------+-----+
|Ease of Online booking|count|
+----------------------+-----+
|                     1|17525|
|                     3|24449|
|                     4|19571|
|                     5|13851|
|                     2|24021|
|                     0| 4487|
+----------------------+-----+

>>> df.groupBy(df['Ease of Online booking']).count().count()
6
```

For Food and drink:

```
>>> df.groupBy(df['Food and Drink']).count().show()
+--------------+-----+
|Food and Drink|count|
+--------------+-----+
|             1|12837|
|             3|22300|
|             4|24359|
|             5|22313|
|             2|21988|
|             0|  107|
+--------------+-----+

>>> df.groupBy(df['Food and Drink']).count().count()
6
```

For Online boarding:

```
>>> df.groupBy(df['Online boarding']).count().show()
+---------------+-----+
|Online boarding|count|
+---------------+-----+
|              1|10692|
|              3|21804|
|              4|30762|
|              5|20713|
|              2|17505|
|              0| 2428|
+---------------+-----+

>>> df.groupBy(df['Online boarding']).count().count()
6
```

For Seat Comfort:

```
>>> df.groupBy(df['Seat comfort']).count().show()
+------------+-----+
|Seat comfort|count|
+------------+-----+
|           1|12075|
|           3|18696|
|           4|31765|
|           5|26470|
|           2|14897|
|           0|    1|
+------------+-----+

>>> df.groupBy(df['Seat comfort']).count().count()
6
```

For Inflight Service:

```
>>> df.groupBy(df['Inflight service']).count().show()
+----------------+-----+
|Inflight service|count|
+----------------+-----+
|               1| 7084|
|               3|20299|
|               4|37945|
|               5|27116|
|               2|11457|
|               0|    3|
+----------------+-----+

>>> df.groupBy(df['Inflight service']).count().count()
6
```

For Cleanliness:

```
>>> df.groupBy(df['Cleanliness']).count().show()
+-----------+-----+
|Cleanliness|count|
+-----------+-----+
|          1|13318|
|          3|24574|
|          4|27179|
|          5|22689|
|          2|16132|
|          0|   12|
+-----------+-----+

>>> df.groupBy(df['Cleanliness']).count().count()
6
```

Distribution of various categorical features: