Assignment 2
Spark Machine Learning

Theodore Fitch
Department of Data Analytics, University of Maryland Global Campus

DATA 650: Big Data Analytics

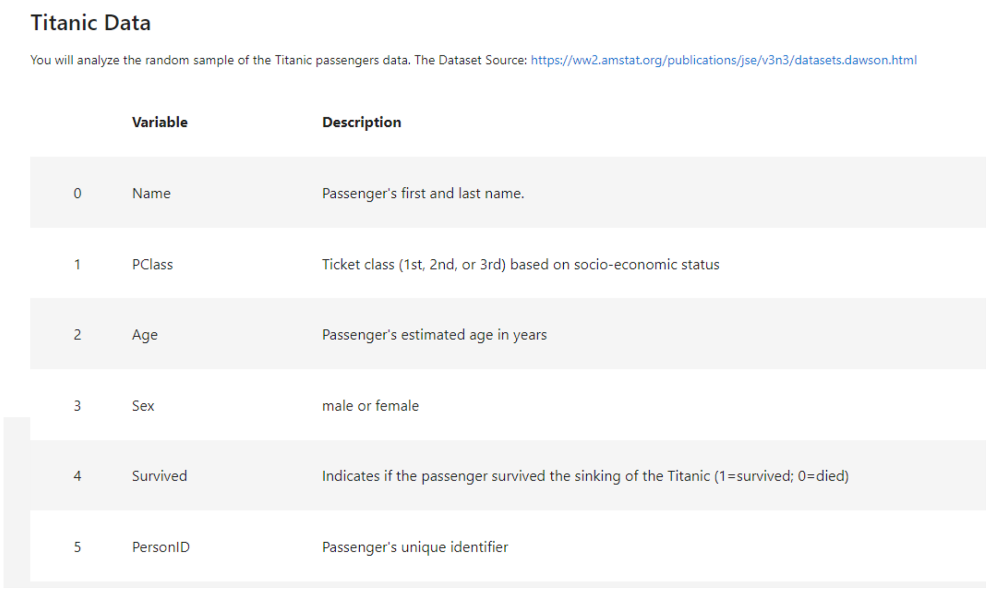
Dr. Ozan Ozcan

June 25th, Summer 2024

<u>Part I – Titanic Logistic Regression:</u>

<u>Discuss the importance of data exploration and visualization prior to running the logistic regression method.</u>

Before running any kind of predictive analysis or let alone before analyzing data, the dataset must be understood. This includes understanding if the data needs to be cleaned or prepared in any way, if any transformations on variables are appropriate, to understand each variable type and structure, and to understand basic descriptive statistics of each variable. This is necessary for a plethora of reasons. One of them is outliers; if outliers exist in the dataset, it may be appropriate to remove them in order to make the model more accurate. A predictive model training with outliers in the training dataset will be less accurate on test data. The data must also be examined for missing values. The data must also be checked for class imbalance (especially of a target variable). There are several simple visualizations that can help one understand correlations (like correlation matrices or heatmaps), or histograms/box plots can help identify outliers. Furthermore, understanding the dataset and values will help one understand how to model the data appropriately. In this case, a dataset is being analyzed which aims to predict survivorship of the Titanic passengers. There are 6 variables with the target variable to predict being Survived (where 1 meaning they survived and 0 meaning they died).



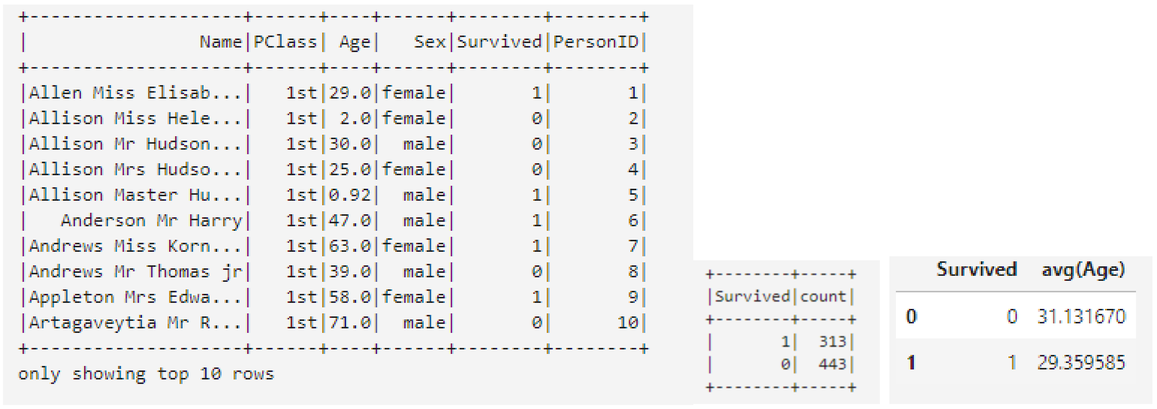
Titanic Data

You will analyze the random sample of the Titanic passengers data. The Dataset Source: https://ww2.amstat.org/publications/jse/v3n3/datasets.dawson.html

	Variable	Description
0	Name	Passenger's first and last name.
1	PClass	Ticket class (1st, 2nd, or 3rd) based on socio-economic status
2	Age	Passenger's estimated age in years
3	Sex	male or female
4	Survived	Indicates if the passenger survived the sinking of the Titanic (1=survived; 0=died)
5	PersonID	Passenger's unique identifier

Figure 1. Titanic dataset variables show 6 variables with 1 target variable being survivorship.

After loading libraries, the data was loaded from a CSV file, parsed into an object, and then loaded into a dataframe. Some basic descriptive statistics were then generated:



```
+--------------------+------+----+------+--------+--------+
|                Name|PClass| Age|   Sex|Survived|PersonID|
+--------------------+------+----+------+--------+--------+
|Allen Miss Elisab...|   1st|29.0|female|       1|       1|
|Allison Miss Hele...|   1st| 2.0|female|       0|       2|
|Allison Mr Hudson...|   1st|30.0|  male|       0|       3|
|Allison Mrs Hudso...|   1st|25.0|female|       0|       4|
|Allison Master Hu...|   1st|0.92|  male|       1|       5|
|   Anderson Mr Harry|   1st|47.0|  male|       1|       6|
|Andrews Miss Korn...|   1st|63.0|female|       1|       7|
|Andrews Mr Thomas jr|   1st|39.0|  male|       0|       8|
|Appleton Mrs Edwa...|   1st|58.0|female|       1|       9|
|Artagaveytia Mr R...|   1st|71.0|  male|       0|      10|
+--------------------+------+----+------+--------+--------+
only showing top 10 rows
```

```
+--------+-----+
|Survived|count|
+--------+-----+
|       1|  313|
|       0|  443|
+--------+-----+
```

	Survived	avg(Age)
0	0	31.131670
1	1	29.359585

Figure 2. Titanic dataset of survivors and victims dataframe's first 10 rows (left), count by survivorship (middle), and average age of survivorship (right).

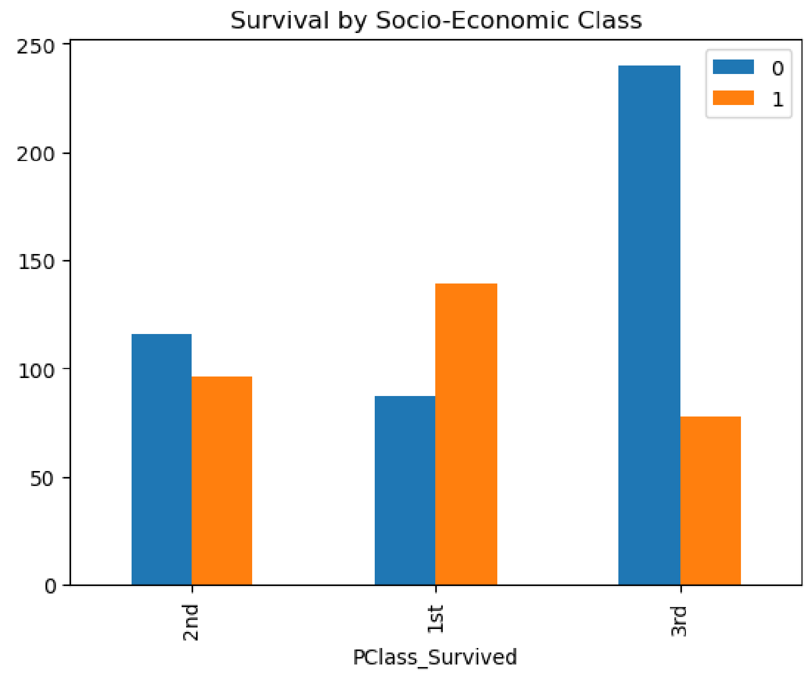


Figure 3. Titanic dataset survivors and victims count by socio-economic class.

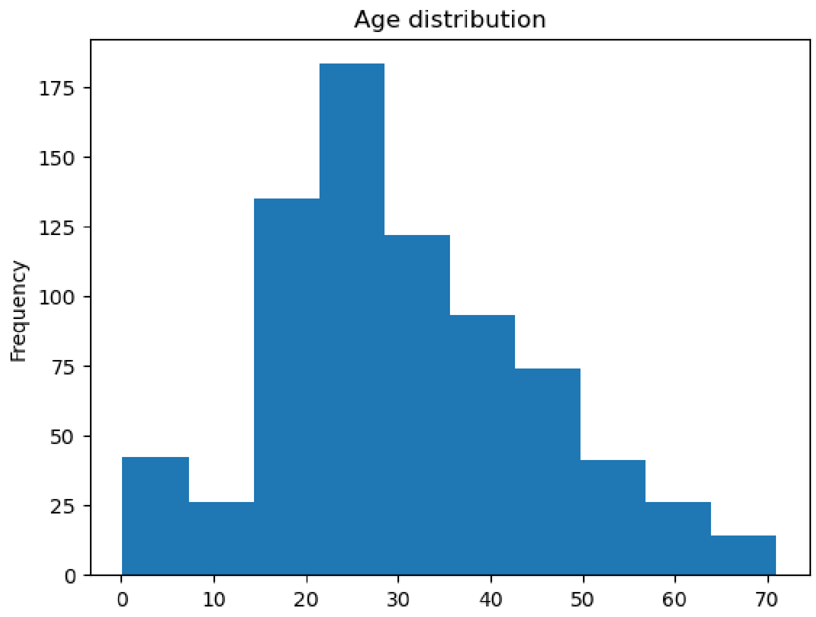


Figure 4. Titanic dataset survivors and victims count histogram by age.

For example, we can see that a majority of the victims of the Titanic were 3rd class tickets (Figure 3). This aligns with the common knowledge that people with lower class tickets were stuck in the bottom levels of the ship as it sunk. However, it's not common knowledge what the age distribution of Titanic passengers was. Using a histogram, we can clearly see it appears somewhat normal, with a vast majority of passengers being between 15-40. Furthermore, because there are around 100 more victims than survivors, there are methods that could be applied to help balance the dataset (however it is not needed in this case because this is not a heavy imbalance).

What do the Titanic data exploration results reveal about the relationships between the likelihood of survival and passenger data?

The EDA on the Titanic dataset shows several key insights. Most importantly, we can see clearly that 1st class passengers were the only class where there were more survivors than victims. 2nd class only had a few less survivors than victims, whereas 3rd class has clear imbalance of more than 2:1 victims to survivors (Figure 3). While the histogram only shows all passengers without differentiation, it is common knowledge that children were prioritized during evacuation (Figure 4). Thus, we see the average age of the survivors is lower than the average age of the victims (Figure 2). While the bar chart wasn't made regarding passenger sex, it is also common knowledge that women were also prioritized during the evacuation. Thus, we should likely see a distribution of more male victims than survivors, while there are fewer female victims and survivors. Of the 2 other variables, they should have essentially 0 effect on predicting survivorship since they are an arbitrary ID and the person's name. Neither of these would have any effect on survivorship.

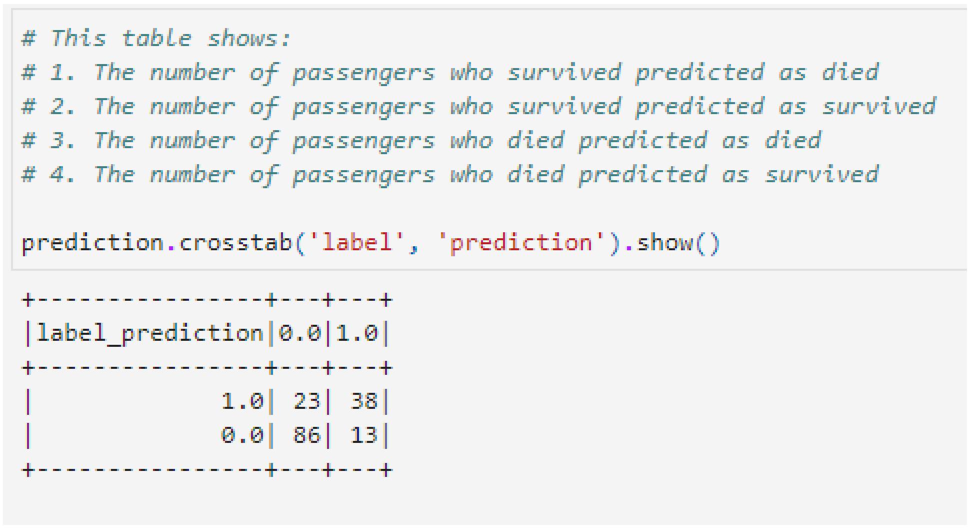
<u>Discuss the logistic regression method results, including the classification accuracy for training and test set.</u>

The logistic regression method had the dataset split into training and test data (with an 80:20 split, or 596 values to 160 records). The goal of the method is to use a logistic regression model in order to predict whether each value is more likely a survivor or a victim. The model requires the data to be split so that the model can be trained (on the training dataset) and then it is evaluated using the test dataset. There are 5 primary metrics used to evaluate the models (Srivastava, 2024):

- Accuracy: the proportion of the total number of correct predictions that were correct
- Error: the proportion of the total number of correct predictions that were incorrect
- Precision: the proportion of positive cases that were correctly identified
- Recall: the proportion of actual positive cases which are correctly identified
- F1: the harmonic mean of precision and recall

There are many situations where precision is the most important metric. When it is critical to identify the target variable (like when predicting survivorship, or heart disease, or churn), we would primarily use precision in order to identify the positive values of the target variable. While we are attempting to predict survivorship for this dataset, we care equally about survivorship and non-survivorship in this case. Thus, accuracy is the most important measure since we care about how the model is performing overall; but each of the metrics give insight into how the model performs. There are also a few other tools to used evalute the model. This includes a confusion matrix when shows the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). This gives an easy insight into the raw numbers of the model and how the above metrics are calculated. Lastly, the area under the PR (precision-recall) curve shows the precision plotted against the recall, while the area under the ROC (receiver operator curve) shows the true positive rate plotted against the false positive rate. A higher area under the curve

(AUC) is better for both of these plots/metrics. A metric of 0.5 would be the baseline, while the closer to 1 the metric is, the better the model is performing (Srivastava, 2024).



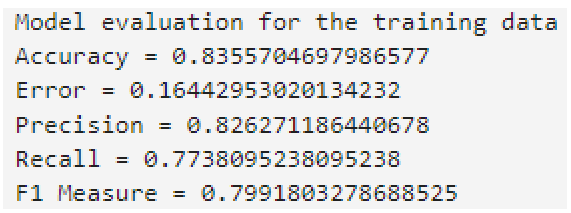
```
# This table shows:
# 1. The number of passengers who survived predicted as died
# 2. The number of passengers who survived predicted as survived
# 3. The number of passengers who died predicted as died
# 4. The number of passengers who died predicted as survived

prediction.crosstab('label', 'prediction').show()
```

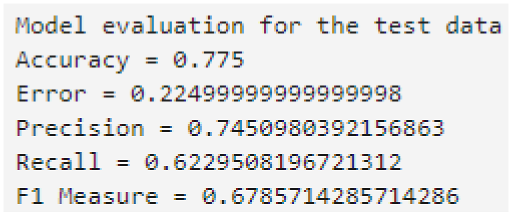
```
+----------------+---+---+
|label_prediction|0.0|1.0|
+----------------+---+---+
|             1.0| 23| 38|
|             0.0| 86| 13|
+----------------+---+---+
```

Figure 5. Titanic dataset logistic regression method confusion matrix.

The confusion matrix shows each of the values from the test data for TN, FN, TP, and FP. We can see from the above that there are 38 survivors correctly predicted, while the model accidentally predicted 13 values as survivors when they were not (false positives). On the other hand, there were 86 victims correctly predicted, while the model accidentally predicted 23 values as victims when they were not (false negatives). Therefore, this model had a 77.5% accuracy. While this is much better than randomly guessing (since the baseline of a binary target variable is 50%), there is still much room for improvement. A better model should be able to achieve a 90-100% accuracy rate.



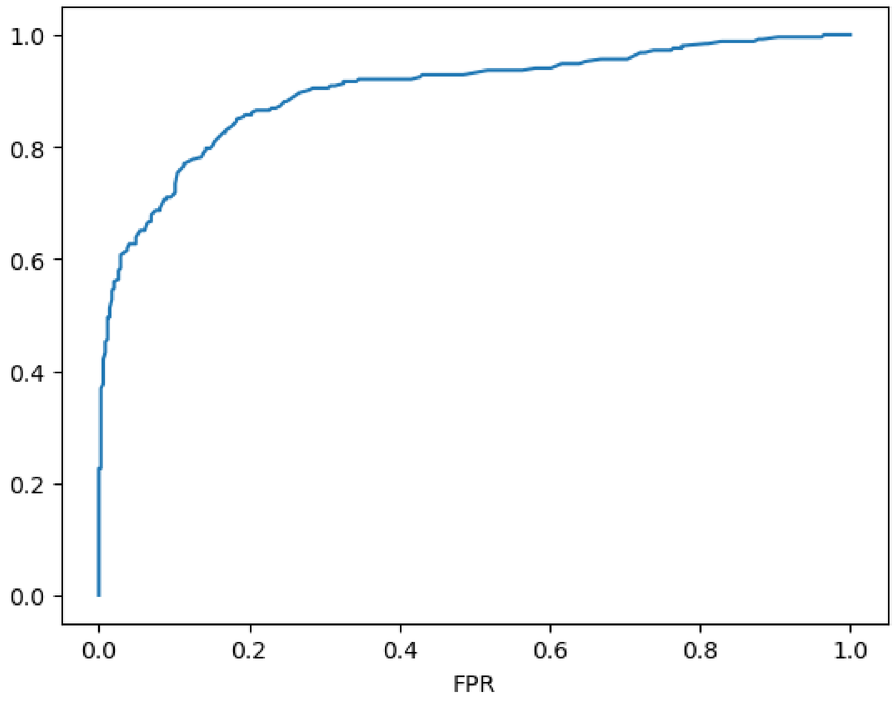
```
Model evaluation for the training data
Accuracy = 0.8355704697986577
Error = 0.16442953020134232
Precision = 0.826271186440678
Recall = 0.7738095238095238
F1 Measure = 0.7991803278688525
```

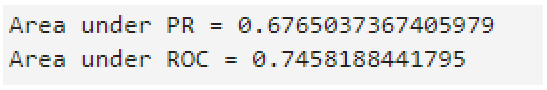


```
Model evaluation for the test data
Accuracy = 0.775
Error = 0.22499999999999998
Precision = 0.7450980392156863
Recall = 0.6229508196721312
F1 Measure = 0.6785714285714286
```

Figure 6. Titanic dataset logistic regression method showing training and test dataset model evaluation metrics.

Moving along to the 5 major metrics to evaluate this model, we can glean a few other insights. First, comparing the 83.6% accuracy of the training data versus the 77.5% accuracy of the test data, shows some overfitting of this model to the training data. While it is only a 6% drop, and it should be expected that the model performs slightly worse on the test data, ideally a model shouldn't drop this far in accuracy from training to test data. The error rate (which is 1-accuracy) is a simple reflection of that same thing. Precision shows a similar trend to accuracy where it decreases from 82.6% in training to 74.5% in test. Thus, overall this model is decent at identifying the survivors. The recall shows the deepest drop from 77.4% in training to an abysmal 62.3% in test data. As mentioned, 50% is baseline so this is only slightly better than baseline. The F1 score similarly decreases from 79.9% to 67.9% which decreases proportionally with recall since it is a composite metric using recall and precision. We can thus see how the higher false negative rate from the confusion matrix has affected the F1 and recall metrics. Having a higher false negative rate shows that the model is being overly conservative when classifying positives.





```
Area under PR = 0.6765037367405979
Area under ROC = 0.7458188441795
```

Figure 7. Titanic dataset logistic regression method area under the PR and ROC curve values, along with the plot for the ROC.

Lastly, as we evaluate the PR and ROC values we can see that they are both above the baseline of 50%. But at only 67.7% (PR) and 74.5% (ROC), we would say this model only performs at a Fair level following a general rule of thumb (Srivastava, 2024). This along with the accuracy and recall metrics above clearly show that this model can act with some level of predictive power; however, there is much room for improvement and optimization.

Is logistic regression suitable for this problem? Why or Why not?

Logistic regression is a suitable method for this problem however there may be better options. First, the target variable is binary which is a requirement of the logistic regression model type. Either the passenger survived or did not. The method models data along a logistic curve where the value either aligns with the asymptote at 1 or at 0. Secondly, the data types being used at continuous and categorical both of which logistic regression can handle. While regression technically requires all variable inputs to be numeric, it can convert categorical values into dummy variables to encode them and incorporate them into the model.

There are some major limitations with this model and those mostly pertain to the dataset. First, there are no outliers skewing the data or the model. Second, a quick examination through the data shows there are no missing data. Missing data could cause lower accuracy rates. The target variable is not heavily imbalanced which is good and thus the dataset doesn't require sampling to become more accurate. Since overfitting occurred, it may be due to the dataset split of 80% into training data. Perhaps using more data in the test dataset and less in training would cause the model to not overtrain and become more robust. While this dataset is fairly clean (with no missing values or major outliers or need for transformation before analysis), it is also quite simple. There could be countless other factors which may have influence on survivorship. Having a larger dataset would allow for a more accurate model since other variables could be used to predict survivorship. And in turn, having more variables would allow different types of models to be used which could yield more accurate results. For example, if more variables like income, port of entry, hometown, etc. were added to the dataset, a decision tree, random forest, or gradient boosting model might be able to produce tremendously accurate results.

What alternative machine learning methods could be suitable for this problem? Consider at least 2 alternative methods.

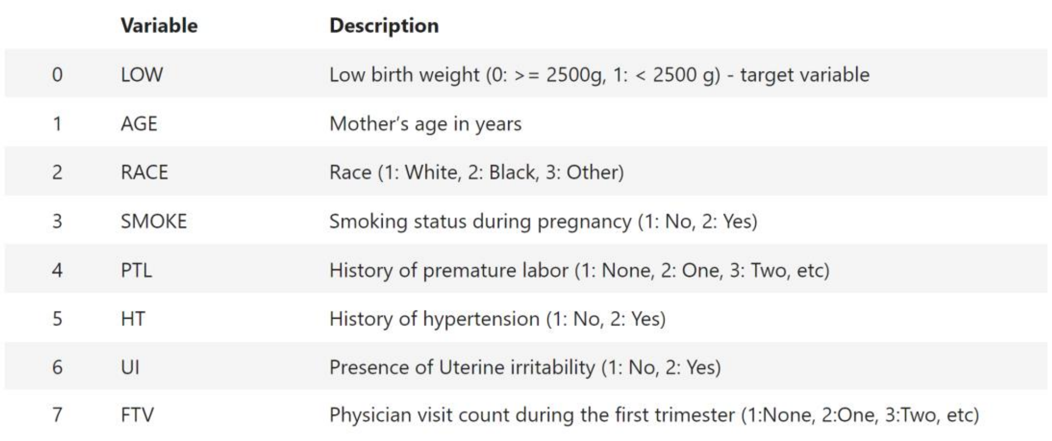
There are several alternative methods that could be suitable for this problem of predicting survivorship of Titanic passengers using machine learning models. First, a random forest could be quite effective. It is an ensemble learning method which combines multiple decision trees to improve accuracy and control for overfitting (which the current models suffers from)(Srivastava, 2022)(Databricks, n.d.). Each tree is built on a subset of the data and a random subset of the features. While there is definitely a big drawback in that interpretability decreases, the feature importance of each variable can still be determined. Another drawback is that these models are very computationally expensive. That said, they are very robust models offering the least amount of overfitting possible. They can also handle non-linear relationships between the features and the target variable. This matters if there are complex relationships between the data. This would be the best model of choice especially with a larger, more complex variable dataset.

A gradient boosting model may also offer a better option than the logistic regression model. It is another ensemble model which builds trees sequentially (Srivastava, 2022) (Databricks, n.d.). Each new tree attempts to correct the errors made by the previous trees, which results in a model that is highly accurate and capable of capturing complex patterns in the data. Gradient boosting models are similar to random forests in that they are both highly accurate and can handle non-linearity adeptly. Gradient boosting models (also similarly) are computationally expensive and less interpretable. They are also prone towards overfitting if not tuned correctly. Thus, in order to create a more accurate model for the Titanic dataset, I would look first at a random forest model and then at a gradient boosting model. Though, it’s important to note that there are many other models that may also be used for this problem like a neural network, support vector machine, or AutoML. The former two are also very accurate, powerful models while the latter generates many models quickly/automatically in order to find the most accurate model type for the dataset in question (Madireddy et al., 2018).

Part II – Logistic Regression for Low Birth Weight Dataset:

Introduction:

A second predictive modeling study was performed using Spark on a dataset provided by Hosmer et al. (2013). This dataset was on infants having a low birth weight and some of the possible predictors of this issue. This is an incredibly important metric used in public health as it can give insight to how a population's maternal health is (WHO, n.d.). It serves as a proxy indicator of long-term maternal malnutrition, general maternal health, and poor healthcare during pregnancy. It is officially designated at < 2,500 grams (5.5 pounds). It can be caused by prematurity, uterine growth restriction, or both (WHO, n.d.). Thus, it is used both descriptively (as a public health outcome indicator) and prescriptively (using known variables to ensure healthy infant weight like maternal nutrition, proper pregnancy healthcare, etc.). The dataset has 1 target variable (LOW) and 7 predictor variables: age, race, smoking status, history of premature labor, history of hypertension, presence of uterine irritability, and physician visit count during 1st trimester (there is also an ID variable which is effectively ignored for this study). Thus, it is expected that most of these variables would have a clear, apparent relationship in predicting birth weight. A mother who is non-smoking, younger, doesn't have a history of premature labor, etc. would be expected to have an infant with a healthy birth weight. This study thus aims to explore what variables contribute most to this outcome and make a predictive model with high fidelity in determining whether a mother is likely to have an infant with healthy birth weight or low birth weight.

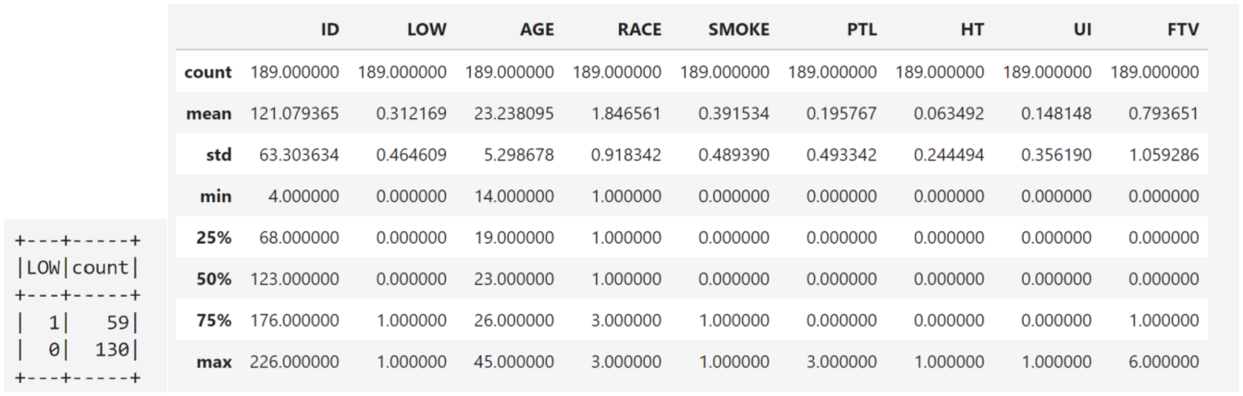


	Variable	Description
0	LOW	Low birth weight (0: >= 2500g, 1: < 2500 g) - target variable
1	AGE	Mother’s age in years
2	RACE	Race (1: White, 2: Black, 3: Other)
3	SMOKE	Smoking status during pregnancy (1: No, 2: Yes)
4	PTL	History of premature labor (1: None, 2: One, 3: Two, etc)
5	HT	History of hypertension (1: No, 2: Yes)
6	UI	Presence of Uterine irritability (1: No, 2: Yes)
7	FTV	Physician visit count during the first trimester (1:None, 2:One, 3:Two, etc)

Figure 8. Infant birth weight dataset variables (note: PTL variable description should read (0: None, 1: One, 2: Two, etc.). This was printed incorrectly by the author).

Exploratory Data Analysis (EDA):

The dataset was explored in a variety of ways before any predictive modeling occurred. First, the dataset was small with only 189 entries split between 130 healthy weight infants and 59 low weight infants.



```
+---+-----+
|LOW|count|
+---+-----+
|  1|   59|
|  0|  130|
+---+-----+
```

	ID	LOW	AGE	RACE	SMOKE	PTL	HT	UI	FTV
count	189.000000	189.000000	189.000000	189.000000	189.000000	189.000000	189.000000	189.000000	189.000000
mean	121.079365	0.312169	23.238095	1.846561	0.391534	0.195767	0.063492	0.148148	0.793651
std	63.303634	0.464609	5.298678	0.918342	0.489390	0.493342	0.244494	0.356190	1.059286
min	4.000000	0.000000	14.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	68.000000	0.000000	19.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	123.000000	0.000000	23.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	176.000000	1.000000	26.000000	3.000000	1.000000	0.000000	0.000000	0.000000	1.000000
max	226.000000	1.000000	45.000000	3.000000	1.000000	3.000000	1.000000	1.000000	6.000000

Figure 9. Infant birth weight dataset target variable counts (left) and descriptive statistics (right).

The descriptive statistics instantly give us the snapshot of size of the dataset, the ranges of each (which only applies to age as a quantitative variable but confirms the others are binary/categorical), and the spread of each variable. Next, a visualization was made to see how the target variable was split across race. We can clearly see that proportionally white mothers (1) are affected least, followed by other mothers (3), followed lastly by black mothers (2). Black mothers appear to be affected nearly at a 1:1 ratio while white mothers are affected at a less than 2:1 ratio of healthy weight to low weight. This indicates that on average black mothers are not getting the healthcare/nutrition they need as much as white mothers.

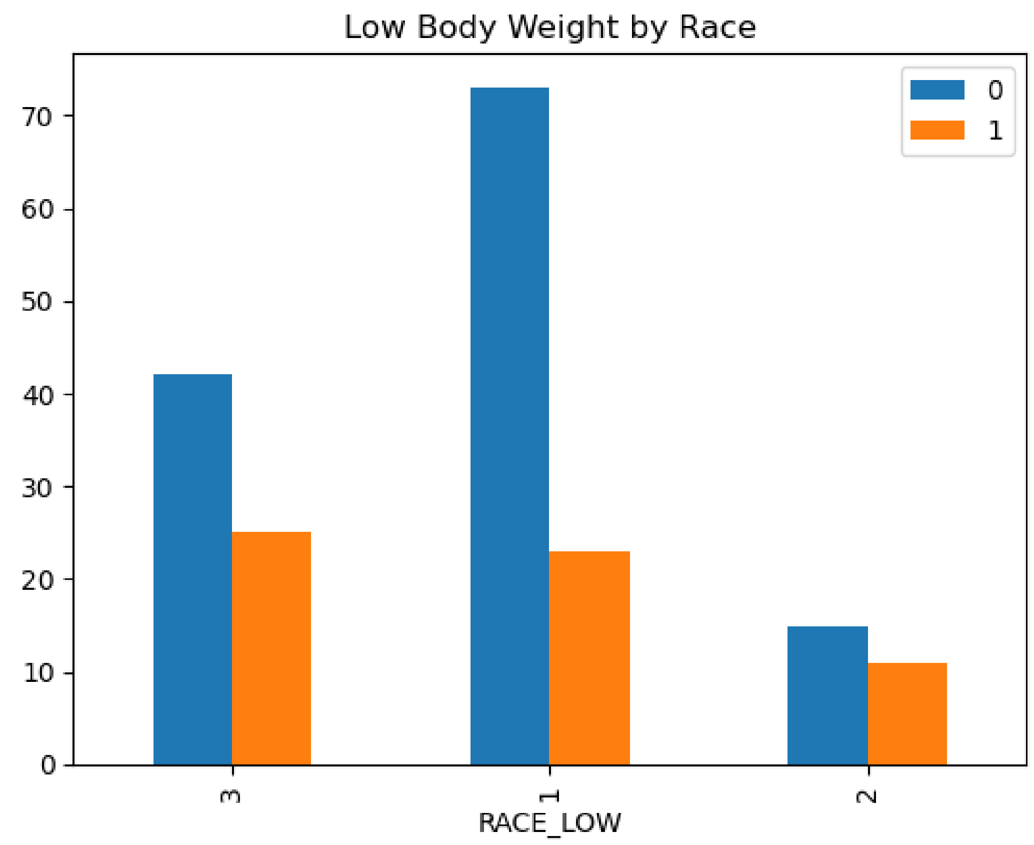
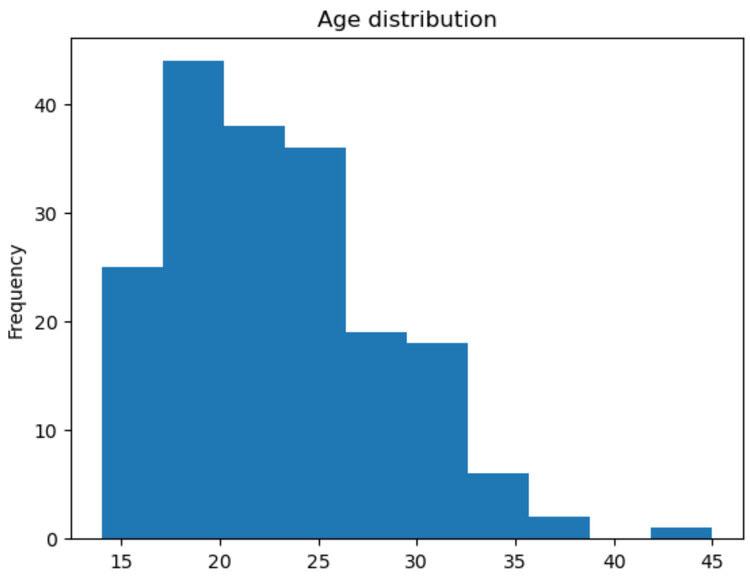
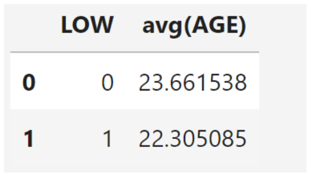


Figure 10. Infant birth weight dataset target variable counts per race (3: other; 1: white; 2: black).

Next, the age variable was briefly explored. We can see through the histogram plot that there is one outlier on the high end at 45 years. This will be left in for the purpose of this analysis. The majority of mothers fall between 15-30 years old with a few being on the higher end. The average age per the target variable shows that the average age of low birth weight mothers is slightly younger (22.3 years) than healthy weight mothers (23.7). This could be an indicator that younger mothers on average have a harder time getting needed healthcare / nutrition than an older mother (who might have more resources at her disposal).





	LOW	avg(AGE)
0	0	23.661538
1	1	22.305085

Figure 11. Infant birth weight dataset histogram of age (left) and average age per target variable (right).

Next a histogram was made for every variable. The ID should be ignored. We see the target variable is distributed approximately 2:1 of healthy to low. The age histogram we've discussed already. Interestingly, we see an approximately ratio of 3:1:2 of white mothers to black mothers to other mothers in this dataset. We can also observe that the smoking distribution is around 3:2. The PTL and FTV distributions are exactly how we might expect them with many in the first category (no history of premature labor and number of physician visits in the first trimester) and then tapering off quickly. Lastly, the hypertension and uterine irritability variables show heavy imbalance at almost a 10:1 ratio.

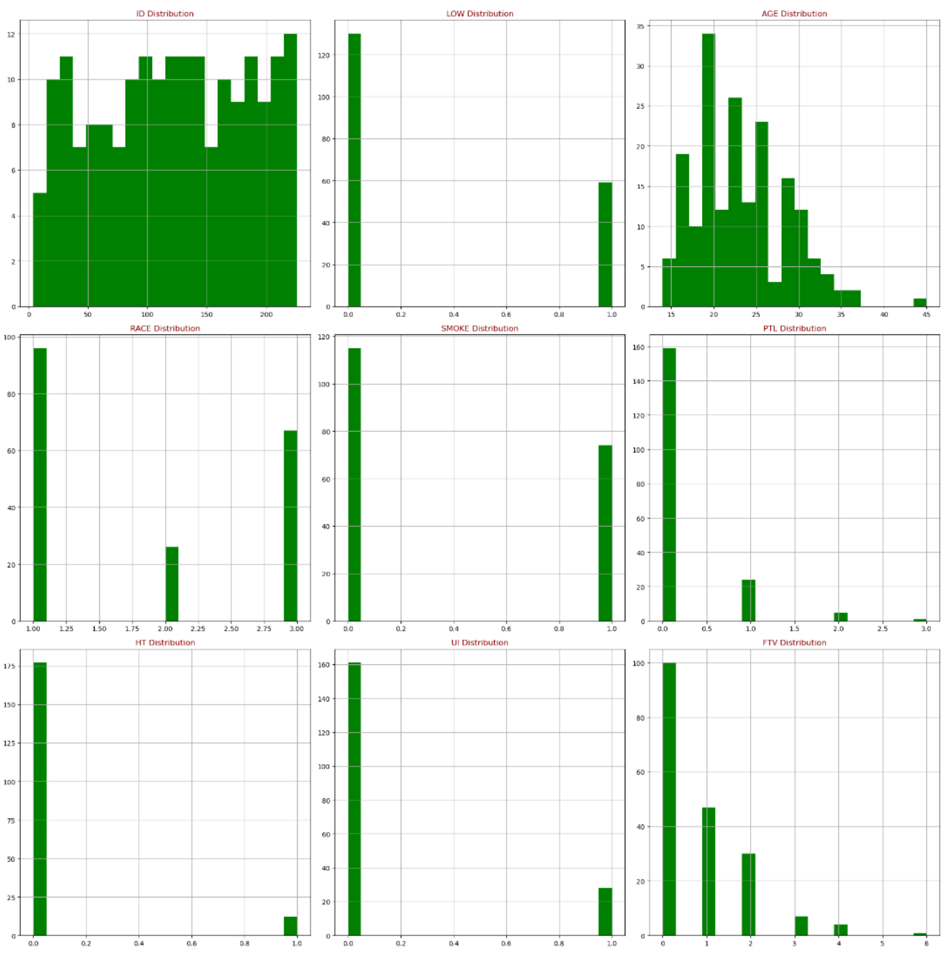


Figure 12. Infant birth weight dataset histograms for every variable.

Next a correlation matrix of all variables was created to identify any interesting relationships. No direct interesting relationships were observed using this plot. This type of plot is generally best suited when a target variable has more quantitative predictors (whereas this dataset has mostly categorical and binary variables). This type of plot could show interesting relationships between variables, correlations, and identify if variables are redundant. This plot does show one interesting feature that there is another outlier for FTV at a value of 6 (this will also be left in the dataset for the purpose of this analysis).

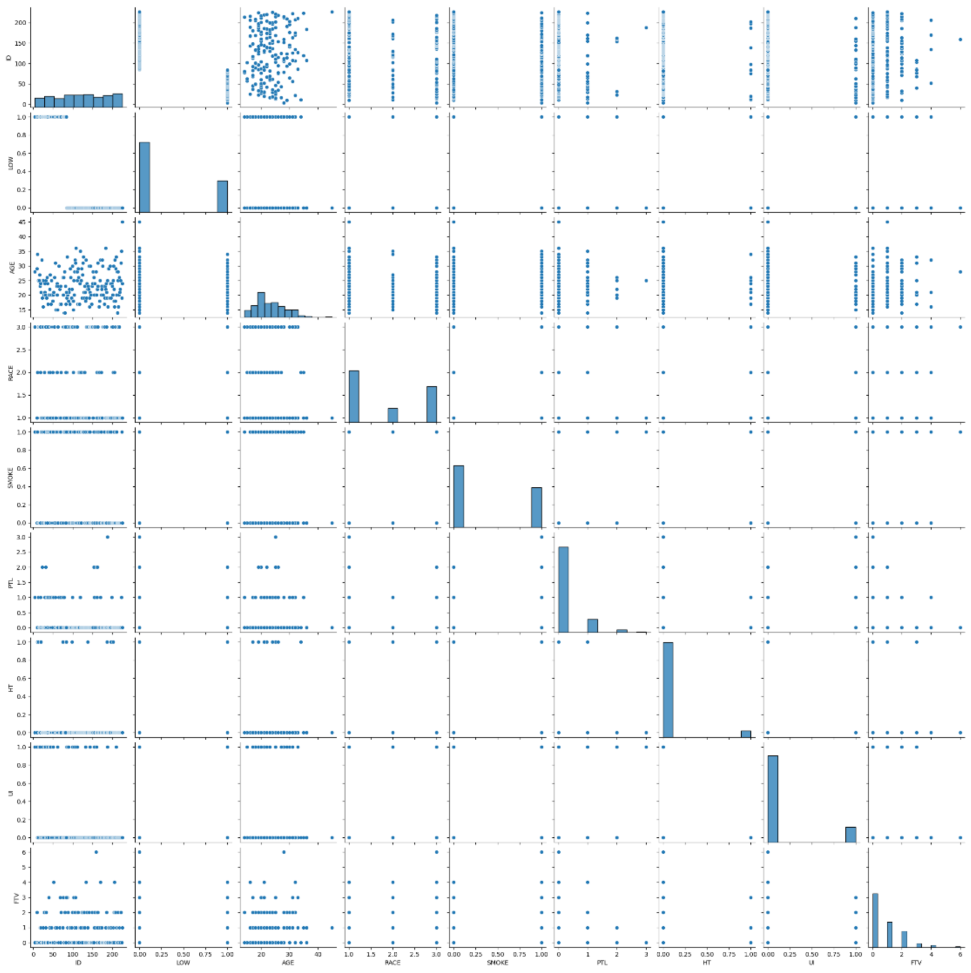


Figure 13. Infant birth weight dataset correlation matrix of every variable.

Next, a heatmap was generated to observe potential correlations between variables quantitatively. This again verifies what we previously mentioned from the correlation matrix that there are no correlations between variables. The strongest that occurs at -0.81 is between the ID and the target variable (which is because all of the low birth weight values use earlier IDs whereas the later IDs are all healthy birth weights). The highest correlation value after this is -0.34 between race and smoking but this value is low enough to be negligible.

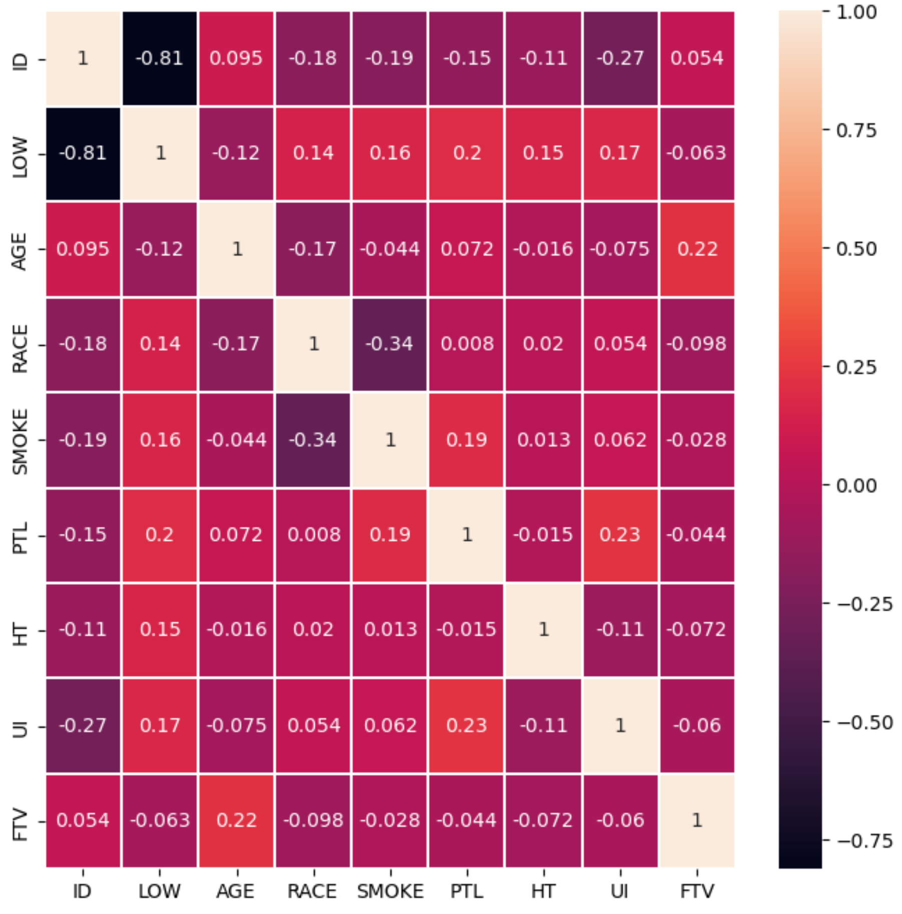
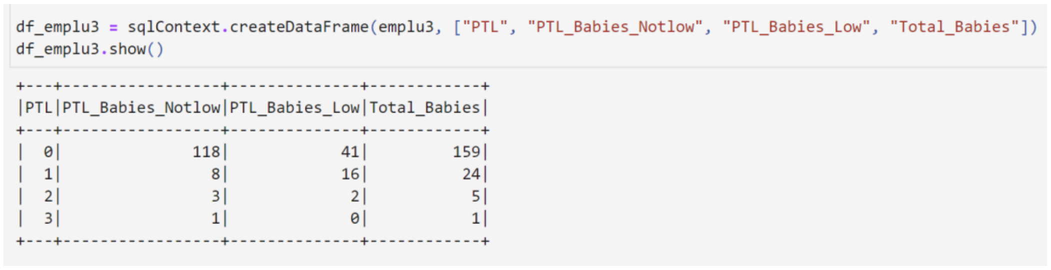


Figure 14. Infant birth weight dataset heatmap of every variable.

Finally, in order to demonstrate the usage of Spark SQL in this analysis, a table was generated showing the target variable and total value counts per history of premature labor. It seems apparent that there’s not a clear pattern between each of the categories of PTL (premature labor) and the target variable. Though it is interesting to note that if a mother has had 1 prior premature labor, then the likelihood of the next one being a low birthweight is 2:1 whereas every other category is 1:2. But it is important to note that 6 total values for a PTL of 2 and 3, whereas a PTL of 1 has 24 values, and 0 has 159 values. Thus, it could be that with a larger dataset, we would see the relationship become apparent. It would seem self-evident that PTL should be a predictor of low birth weight.



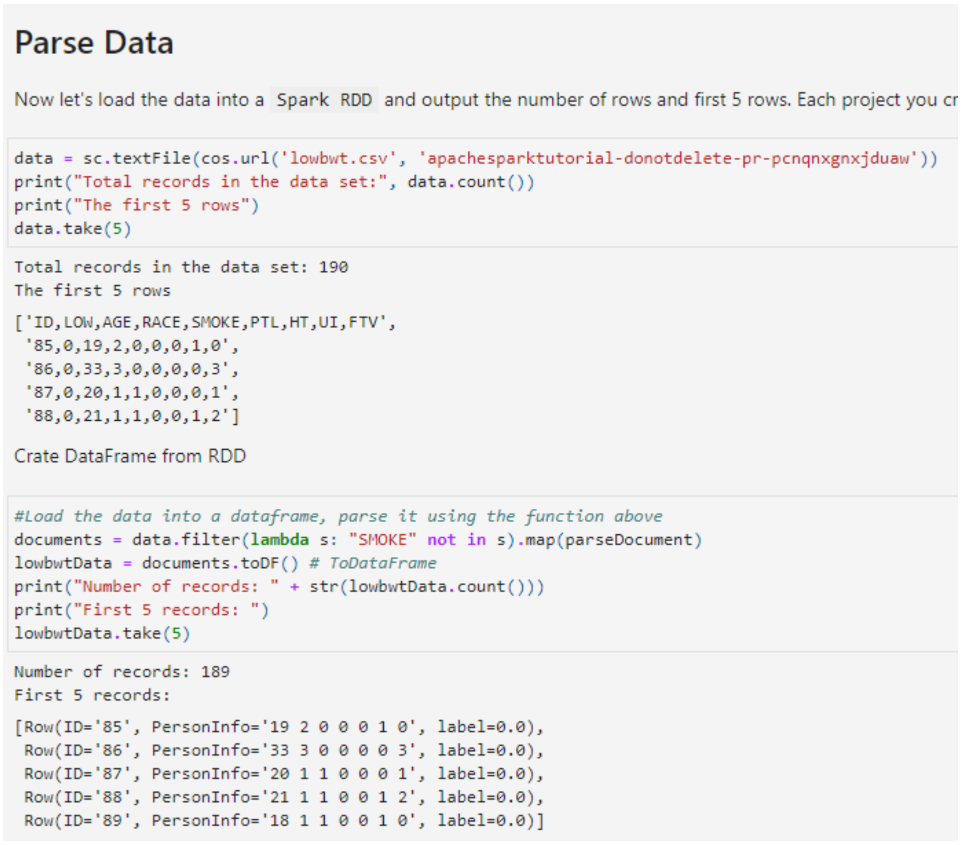
```
df_emplu3 = sqlContext.createDataFrame(emplu3, ["PTL", "PTL_Babies_Notlow", "PTL_Babies_Low", "Total_Babies"])
df_emplu3.show()

+---+-----------------+--------------+------------+
|PTL|PTL_Babies_Notlow|PTL_Babies_Low|Total_Babies|
+---+-----------------+--------------+------------+
|  0|              118|            41|         159|
|  1|                8|            16|          24|
|  2|                3|             2|           5|
|  3|                1|             0|           1|
+---+-----------------+--------------+------------+
```

Figure 15. Infant birth weight dataset heatmap of every variable.

Method:

There were 3 predictive models generated to model the data: a logistic regression, a naïve bayes, and a decision tree. This was performed in order to see which model might have the best accuracy/recall on the dataset. The data was put into a Spark RDD, then into a dataframe, and then was parsed using a unique function. The parser only kept the 7 discussed variables in the PersonInfo variable and effectively ignored ID. The parser also was programmed to remove the header row by looking for “SMOKE” and removing any row which contained it (which was only the header row). Unlike the previous notebook, the data split was changed from 80:20 to 70:30. This was because we previously saw overfitting occurring. Especially because the dataset is so small, it’s important to adjust this dataset to prevent overfitting (or for the model to overguess on the more abundant target variable).



Parse Data

Now let's load the data into a Spark RDD and output the number of rows and first 5 rows. Each project you cr

```
data = sc.textFile(cos.url('lowbwt.csv', 'apachesparktutorial-donotdelete-pr-pcnqnxgnxjduaw'))
print("Total records in the data set:", data.count())
print("The first 5 rows")
data.take(5)
```

```
Total records in the data set: 190
The first 5 rows
['ID,LOW,AGE,RACE,SMOKE,PTL,HT,UI,FTV',
 '85,0,19,2,0,0,0,1,0',
 '86,0,33,3,0,0,0,0,3',
 '87,0,20,1,1,0,0,0,1',
 '88,0,21,1,1,0,0,1,2']
```

Crate DataFrame from RDD

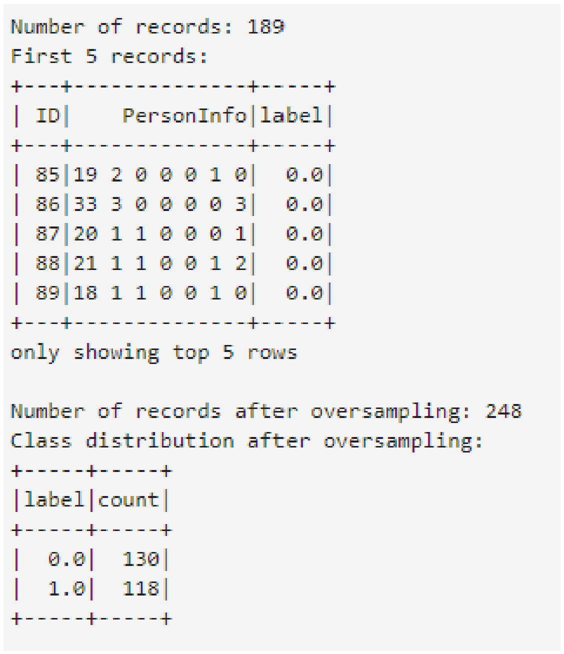
```
#Load the data into a dataframe, parse it using the function above
documents = data.filter(lambda s: "SMOKE" not in s).map(parseDocument)
lowbwtData = documents.toDF() # ToDataFrame
print("Number of records: " + str(lowbwtData.count()))
print("First 5 records: ")
lowbwtData.take(5)
```

```
Number of records: 189
First 5 records:
[Row(ID='85', PersonInfo='19 2 0 0 0 1 0', label=0.0),
 Row(ID='86', PersonInfo='33 3 0 0 0 0 3', label=0.0),
 Row(ID='87', PersonInfo='20 1 1 0 0 0 1', label=0.0),
 Row(ID='88', PersonInfo='21 1 1 0 0 1 2', label=0.0),
 Row(ID='89', PersonInfo='18 1 1 0 0 1 0', label=0.0)]
```

Figure 16. Infant birth weight dataset put into a Spark RDD, then a dataframe, and then parsed using a function.

The 3 models discussed were run twice. First they were run using a typical 70:30 data split. Then they were run again with the minority class of the target variable oversampled. Because there was a class imbalance of nearly 2:1 for the target variable, it was suspected this could contribute to the model being weaker. Thus, a function was written to duplicate a portion of the minority class values so that the final dataset would be more proportional. An attempted approach was to use SMOTE, or Synthetic Minority Oversampling Technique (Brownlee, 2021). This technique synthetically creates new data of the minority class target variable. Unfortunately, the attempt wasn’t successful. So instead, the approach taken was to duplicate each minority class target variable value. The final outcome after duplication and oversampling was 248 total records, 130 healthy weight infants, and 118 low weight infants (as opposed to the original 59).

That way even with no “new data” the model will be able to train more thoroughly on the data where there are low birth weights. Again, this is to prevent the model from simply defaulting to “guessing” at the majority target variable which results in exceptionally low recall. Since it is important to determine the minority class target variable (instead of correctly identifying the majority class), this approach was taken. It’s important to note that this code was inserted as an entire block after the dataframe from RDD creation occurs. That way this code can be commented out to produce the first 3 models, while the code should be kept in order to produce the second 3 models.



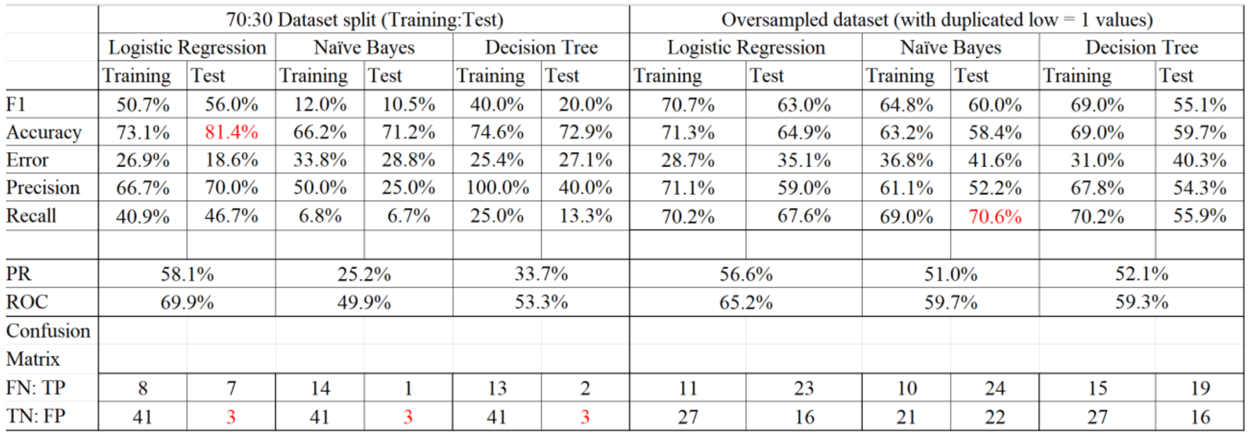
```
Number of records: 189
First 5 records:
+---+-------------+-----+
| ID|   PersonInfo|label|
+---+-------------+-----+
| 85|19 2 0 0 0 1 0|  0.0|
| 86|33 3 0 0 0 0 3|  0.0|
| 87|20 1 1 0 0 0 1|  0.0|
| 88|21 1 1 0 0 1 2|  0.0|
| 89|18 1 1 0 0 1 0|  0.0|
+---+-------------+-----+
only showing top 5 rows

Number of records after oversampling: 248
Class distribution after oversampling:
+-----+-----+
|label|count|
+-----+-----+
|  0.0|  130|
|  1.0|  118|
+-----+-----+
```

Figure 17. Infant birth weight dataset after oversampling the minority class target variable (by duplicating the records of the low weight entries).

Results:

Thus, in total 6 models were generated with 3 using the original dataset (with a 70:30 split) and 3 using a dataset where each target variable at LOW = 1 was duplicated to balance the target variable. As previously mentioned, the models will be primarily measured against accuracy and (most importantly) recall. Since recall (or sensitivity) is a measure of how well the model identifies the low birth weight, this measure is most important. It matters more that the infants with health risks are identified than identifying infants with no health risk. Each metric was displayed because they show a different aspect of each model.



	70:30 Dataset split (Training:Test)						Oversampled dataset (with duplicated low = 1 values)					
	Logistic Regression		Naïve Bayes		Decision Tree		Logistic Regression		Naïve Bayes		Decision Tree	
	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test
F1	50.7%	56.0%	12.0%	10.5%	40.0%	20.0%	70.7%	63.0%	64.8%	60.0%	69.0%	55.1%
Accuracy	73.1%	81.4%	66.2%	71.2%	74.6%	72.9%	71.3%	64.9%	63.2%	58.4%	69.0%	59.7%
Error	26.9%	18.6%	33.8%	28.8%	25.4%	27.1%	28.7%	35.1%	36.8%	41.6%	31.0%	40.3%
Precision	66.7%	70.0%	50.0%	25.0%	100.0%	40.0%	71.1%	59.0%	61.1%	52.2%	67.8%	54.3%
Recall	40.9%	46.7%	6.8%	6.7%	25.0%	13.3%	70.2%	67.6%	69.0%	70.6%	70.2%	55.9%
PR	58.1%		25.2%		33.7%		56.6%		51.0%		52.1%	
ROC	69.9%		49.9%		53.3%		65.2%		59.7%		59.3%	
Confusion Matrix												
FN: TP	8	7	14	1	13	2	11	23	10	24	15	19
TN: FP	41	3	41	3	41	3	27	16	21	22	27	16

Figure 18. Infant birth weight dataset 6 model results showing the 7 accuracy measures and confusion matrix for each model. Models from henceforth will be referred to in shorthand as follows: LR, NB, DT, OLB, ONB, and ODT.

The results neatly summarized in the above table paint a clear picture. Each The overall accuracy results are higher on the original dataset; with Test Accuracy of 81%, 71%, and 73% for LR, NB, and DT. They all performed quite well and were far above the baseline of random chance guessing. However, each of those models also had abysmal recall scores: 47%, 7%, and 13% respectively. This is far, far below the baseline of random chance guessing and is unacceptable to use. Thus, the earlier assumption was correct that the models may have poor recall since they were overguessing the majority class of the target variable. The oversampling technique was therefore needed since the first three models performed so poorly on the target variable. We can also see from the confusion matrices that there were only 7, 1, and 2 TP (true positive) predictions made for the LR, NB, and DT models – meaning the LR model performed best in identifying a total of 7 low birth weight infants. It's interesting to note that there were exactly 3 FPs (false positive, meaning identified as low birth weight but actually being healthy weight) for each of these models. Perhaps these are 3 outliers (previously discussed) which are difficult for the models to classify. Further investigation would have to be performed. It is important to also note that the 3 confusion matrices are only generated for the test data (not training data), and that these values are the ones used to make all of the other accuracy metrics.

For these first 3 models, the F1 scores were also quite low with results of 56%, 11%, and 20%. This is not surprising as the F1 score is the harmonic mean of recall and precision – meaning if recall/precision are low, then F1 will also be low. Since Error is simple 1-Accuracy, further discussion is just redundant to the Accuracy discussion. The PR and ROC values are meant to demonstrate how well the models are distinguishing between positive and negative classes (ROC) or performing as a function of both precision and recall (PR). The ROC for all 3 models are close to baseline with only LR (with the higher accuracy rate) being at 70% (20% better than the baseline of 50%, but far from the max of 100%). The other 2 being near 50% indicate that these models are essentially as good as guessing. Again, the PR values are all on the lower side further reflecting these models' poor performance (with a range of 0-100%). Lastly, a ROC Curve Plot was generated to show the curve of TPR (True Positive Rate) vs FPR (False Positive Rate). The faster this curve rises and reaches a horizontal asymptote, the better it shows the model is performing. The values displayed for ROC are the Area Under the Curve, meaning the more area, the better the models performed (since the TPR increases rapidly)

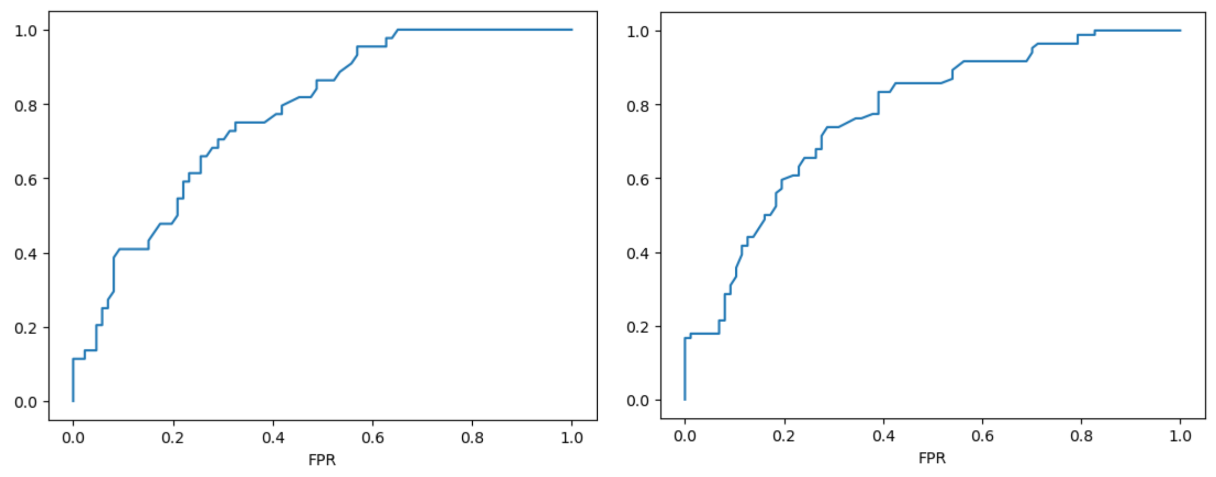


Figure 19. Infant birth weight dataset ROC curve plot for the LR model (left) and OLR model (right).

It is key to note that the best results of Accuracy occurred on the LR where the model performed better on the test data than the training data. This clearly indicates that LR was not overfit and is a robust model. On the other hand, NB and DT both decreased in recall and only NB increased in accuracy on the test data. It's important to note since a model that performs better on test data than training data is robust and is likely to continue performing well in the future. However, recall is far too low on the first 3 models and thus, discussion will continue to the second 3 models.

The oversample models, OLR, ONB, and ODT, all performed with worse average accuracy but, critically, with better recall. Respectively, they have accuracy of 65%, 58%, and 60% but recall of 68%, 71%, and 56% (as a reminder the LR model had the highest of the original dataset at a recall of 47%). While a "random guess" would typically be 50%, since the target variable was imbalanced, an informed random guess would actually be 68.8% (130 (healthy infants) / 189 total infants). On the other hand, only 31.2% of the dataset were low weight infants. If one were to randomly guess all infants were low weight, there would be a 100% recall rate but very low model accuracy. That is to say, the ONB model have a 71% recall means this model lifts the chances from “random guess” by 21% and from the “informed random guess” by 40%. This is by far the best performing model assuming that recall is the most important metric. If it’s determined that the healthy infants are more important to determine, then accuracy can be used as the best metric which would indicate that LR is the best model.

The OLR and ODT both saw some decreases in accuracy and recall from the training dataset to the test dataset indicating some overfitting. However, the ONB did see a slight increase in recall (and decrease in accuracy) indicating some level of robustness. The F1 scores are much higher on average and this is due to the recall values being lifted so much higher (causing F1 to increase as well). The confusion matrices are included though no major patterns or insight are observed in them. The respectively show the TP values of 23, 24, and 19 across the OLR, ONB, and ODT models. Lastly, the PR and ROC values both saw significant lift using this oversampling method. On average, it was 3% higher for the ROC values and 13% higher for the PR values. The ROC values are still only in the high 55%-65%, which is not high, but it is still an improvement over the last method.

Discussion:

Overall, 6 models were generated, with the evidence of the accuracy measures being clear that the original dataset models (LR, NB, and DT) performed better in accuracy whereas the oversampled models (OLR, ONB, and ODT) performed better in recall. The best models were LR (for accuracy, in being able to identify healthy weight infants) and ONB (for recall, in being able to identify low weight infants). Logistic regression is absolutely a suitable method in order to model this dataset. Logistic regression aims to predict a target binary variable. While it requires having numeric inputs, all of the class inputs from this dataset and easily turned into dummy variables. It is also an interpretable model (unlike certain blackbox models) so that it can be easily understood how each variable affects the outcome. The results of this study showed that the LR and OLR models were much better than guessing randomly. While it may have its limitations and the Naïve Bayes model had a higher recall, the logistic regression model clearly is effective at modeling this data. Although the models never hit a >90% accuracy, this is much more likely due to the size of the dataset. This dataset is small with only 189 entries. In addition, the heavy skew of the target variable makes it difficult to train the models effectively. This makes it difficult to accurately model the data especially when applying a dataset of only 189 values to many, many other future records. In addition, because the target variable is imbalanced, the test and training datasets will have different ratios of low/healthy weight values in them; and because the 3 sets of models were run using different training/test datasets, they may have performed slightly differently due to random chance. The oversampling technique used clearly appeared to work. While it is possible that the model could see repeat observations of the same values in the training dataset, the test of this would be to look for overfitting. Since we did not see major signs of overfitting from OLR, ONB, and ODT, this was not an issue. It appeared the duplicate oversampling technique was effective in balancing the dataset for the target variable and training the models effectively. It seems to have directly caused the recall to increase on the test data 21%, 64%, 43% for each of the models (LR to OLR and so on). Those increases, especially seeing a 64% increase in recall on the ONB model, clearly demonstrate the effectiveness of this technique.

There are multiple ways that these models could be made more accurate. First, a larger dataset is most important. Having more entries allows the models to become more robust. Also, having a dataset with more predictive variables is important to allow the models to become more accurate. Especially when it comes to classifying outliers, it becomes even more useful to have more variables. Second, outliers (as previously discussed) could be removed from the dataset. This would make the models more accurate since they are not going to be trained on outlier data. The values with high maternal age, or high FTV. While these were not far out of the typical range of other values, they were still high and would allow the models to become more accurate and likely have higher recall. Third, the training to test dataset ratio could be adjusted. A 70:30 split was used; however, perhaps a 60:40 or 50:50 split would be better. Fourth, there are parameters in the models that could be adjusted. For example, the logistic regression model was run using maxIter=50, regParam=0.01 meaning the model had a maximum of 50 iterations and the latter prevents overfitting by adding a penalty to the model's complexity (Apache Spark, n.d.). Adding iterations and adjusting the penalty factor may allow the model to become more accurate.

Two methods in addition to the logistic regression were already explained, performed, and discussed. Two other methods that would be worth testing on this dataset would be a random forest and a gradient boosting model, two types of ensemble models. These are both computationally expensive but they can thus be highly accurate. While there are many types of models out there, some are considered blackbox where it cannot be explained how each parameter helps the model reach its conclusion (like neural networks). In contrast, the random forest and gradient boosting methods are both interpretable, which is particularly important when predicting something related to public health. One cannot sway the outcomes if one doesn't understand the inputs. A random forest combines the outputs of multiple decision tress into a single output, whereas a gradient boosting model makes multiple decision trees in sequence improving the method each time (Databricks, n.d.). Both are robust to overfitting and would likely be able to handle the class imbalance well. Future research on this dataset should thus include: tuning the current model parameters; attempting different training / test dataset splits; using the SMOTE approach; or attempting new model types.

Part III – Research Questions:

What is overfitting? What is the impact of overfitting on model performance? Discuss at least 2 approaches to avoid overfitting the model.

Some discussion on this has already occurred throughout this text in regard to overfitting as we saw several instances of this through several models. In summary, overfitting is when a model fits itself too well to the training dataset, and thus performs poorly on the test dataset. This can occur in a variety of ways but some of the common ones are: when a dataset is quite small, when the data is messy and has outliers, and when there is a high number of variables. The first and most important impact is the drop in accuracy of the models. This is observed when the key accuracy metrics drop from training to test datasets. Another is that the models become overly complex. This causes them to also become more computationally expensive and use more resources. This can happen because they are attempting to explain obscure patterns in the dataset (like outliers). The best models need to find the balance between considering all the variables it observes in the training dataset, and making the pattern apply robustly across the test dataset.

To prevent overfitting multiple approaches can be taken. Cross validation can occur such that the dataset is split into multiple subsections, and the model can be trained and tested on each of them to generate the highest level of accuracy. This is especially helpful if a dataset is small. K-fold cross validation is an example where the data is split into k subsets, then the model is trained on k-1 of the sub-datasets and tested on the final one, and then this occurs k times. In this manner, every piece of the data is used to both train and test the model (AITechStudio, 2024). Another approach is regularization which was demonstrated in the logistic regression model.

This adds a penalty to the loss function of the model so that any overcomplexity is discouraged (AITechStudio, 2024). Two examples are Lasso, which adds the absolute values of the coefficients as a penalty to the loss function, and Ridge, which adds the squared values of the coefficients as a penalty to the loss function.

Discuss 5 (five) key differences between HDFS and Object Storage.

HDFS (Hadoop Distributed File System) and Object Storage both store data, but they fundamentally have different objectives. From a high-level, HDFS is more suitable for traditional big data processing tasks on structured/semi-structured data (Ashtari, 2022). It’s primarily for performing calculations and file manipulation, which allows it to fulfil the requirements needed for big data analytics. On the other hand, Object Storage excels at handling large volumes of unstructured data (like videos or images) with flexible access patterns. We’ll examine 6 major categories where these two differ: structure, access patterns, scalability, data integrity, resource efficiency, and latency.

HDFS organizes its data into hierarchies of files and directories (Ashtari, 2022). Each file is split into blocks and the blocks are spread across various DataNodes. This sort of structure makes HDFS excel at handling big data analytics problems because the data is processed sequentially. If there is a failure, the sequence can be used (previous and next steps) to infer what the data should be. HDFS is deeply integrated in the Hadoop ecosystem with other tools available for data analysis like MapReduce, Hive, and Pig. In contrast, Object Storage stores data as objects in a flat namespace which consists of the data, the metadata, and a unique identifier. This unique structure is highly scalable and flexible. Thus, Object Storage is ideal for multimedia files, backups, and logs. Each object is able to have extensive metadata making them more searchable and giving context to them.

HDFS is optimized for diverse access patterns where large datasets are read and written sequentially (Ashtari, 2022). It’s thus best suited for workloads involving batch processing and big data analytics. Random access is possible though not optimal since HDFS is designed for streaming large data files. On the other hand, Object storage is optimized for diverse access patterns, including both sequential and random access. It supports high levels of concurrent access and is designed to handle a variety of workloads, such as serving web content, storing backups, and providing scalable storage for cloud-native applications. It is also suitable for applications requiring frequent access to data with varying sizes and types. Linking back to the previous paragraph, Object Storage being excellent at multimedia and concurrent access should bring to mind accessing photos on popular social media sites like LinkedIn or Facebook.

HDFS scales horizontally by adding more DataNodes to the cluster (Ashtari, 2022). However, its scalability is limited by the performance and capacity of the NameNode, which stores all the metadata. As the cluster grows, the NameNode can become a bottleneck. Recent developments like HDFS Federation and high availability configurations help mitigate this; however, the rule of thumb is that the NameNode is the limiting factor (IBM, 2024). Meanwhile, Object Storage is designed for massive, virtually unlimited scalability. It distributes both data and metadata across multiple nodes, allowing for seamless expansion by simply adding more storage nodes. This distributed architecture enables Object Storage to handle exabytes of data without significant performance degradation.

HDFS ensures data integrity and durability through replication. Each data block is replicated across multiple DataNodes, typically three by default. This redundancy provides fault tolerance, ensuring data availability even if several nodes fail. Additionally, HDFS performs periodic checksums to detect and, if needed, correct data corruption. Object Storage typically employs erasure coding for data integrity and durability (Leone, 2019). Erasure coding breaks data into fragments and stores these fragments with redundancy across multiple nodes. This method is more storage efficient compared to replication while still providing high durability and fault tolerance. Object Storage systems also include mechanisms for data verification and self-healing.

HDFS can be resource-intensive due to its reliance on replication for fault tolerance, leading to higher storage costs (Smith, 2024). Each piece of data is stored multiple times (as previously mentioned, typically 3), which adds extra expense. Additionally, managing and maintaining an HDFS cluster requires significant administrative overhead, including ensuring NameNode high availability and handling hardware failures. Object Storage tends to be more cost-efficient, especially for large-scale and long-term storage needs (Smith, 2024). Erasure coding reduces the storage overhead compared to replication, lowering costs (Fullestop, 2015). Many Object Storage solutions are available as managed services (e.g. Amazon S3, Google Cloud Storage, etc.), which offload the maintenance and scaling responsibilities to the provider, further reducing operational costs (Fullestop, 2015).

Lastly, HDFS generally exhibits higher latency, particularly for metadata operations and random access patterns (Google Cloud, 2018). Again, the centralized NameNode can become a bottleneck under heavy loads, adding to the latency. HDFS is optimized for high-throughput batch processing, not for low-latency operations, making it less suitable for interactive or real-time applications. Object Storage is designed for low-latency access, especially for read operations(Google Cloud, 2018). Its distributed architecture allows for parallel data retrieval, reducing latency. Many Object Storage systems also use caching and content delivery networks (CDNs) to further minimize access times. However, the performance can vary depending on the specific implementation and network conditions, with some cloud-based solutions offering options for enhanced performance tiers.

We may use R, Python, Scala, and Java programming languages with Spark. Discuss the pros and cons of each language.

There are a variety of languages that can be used within Spark including R, Python, Scala, and Java. Each have their own purpose and place, and their strengths and weaknesses should be considered before beginning a project using one of them. R is a language that was specifically designed for statistics and data analytics (Enugurthi, 2024). It offers a wide range of tools and packages supporting this function. It has the ability to make visualizations with ggplot2 and it has a strong community to help with troubleshooting and offer support. R typically is on the slower side which means it can have performance issues when it comes to big data. While the structure is somewhat similar to Python, it has a significant learning curve if one is not familiar with it (as I and many others have experienced). Lastly, the SparkR is less feature rich compared to PySpark and Scala. This is important for the packages already developed compared to what still needs to be developed.

Next, Java has some significant drawbacks but significant strengths to balance them. It is a verbose language where more code is generally needed to perform the same functions as a shorter code in Python or Scala (Enugurthi, 2024). It’s a more complex language making it harder to learn and maintain. On the other hand, Java has a very high-performance ceiling making it an ideal language for big data analysis. Compared to R, it is very mature and comes with countless tools, packages, and libraries for one to use in their project. Java's stability and backward compatibility make it a reliable choice for enterprise applications. Lastly, its strong type system helps catch errors at compile-time.

Python, as demonstrated through this project, is very human readable and simple (Enugurthi, 2024). It’s easy to read the functions and often have a sense of what the code is doing. Thus comparatively, it’s a much easier language to learn. Python has a vast ecosystem of libraries ranging from visualization (Matplotlib) to analysis (Pandas, NumPy) to machine learning (scikit learn). It is one of the most widely used languages in the data science world meaning it also has a vast community support network. It’s much easier to troubleshoot common issues when people have already had that same issue and have generated solutions to it. The biggest drawback of Python is that it is on the slower side performance-wise. This is due to its interpreted nature and Global Interpreter Lock (GIL), which can impact performance for very large-scale data processing tasks. Furthermore, Python abstracts away some low-level details of Spark, which can limit optimization and fine-tuning.

Scala, on the other hand, is statically typed, offering better performance and more efficient memory management compared to Python (Enugurthi, 2024). Since Spark itself is written in Scala, this language provides the most natural integration with Spark, with new features often available in Scala first. Scala’s strong type system helps catch errors at compile-time, reducing runtime errors, and its support for functional programming paradigms leads to more concise and expressive code for data processing tasks. However, Scala has a steeper learning curve compared to Python, especially for those unfamiliar with functional programming concepts. Additionally, while it is still growing, the Scala community and resources are not as extensive as Python's. This can make troubleshooting much more difficult to find solutions and tools already existing.

Therefore, one should consider each languages strengths and limitations before choosing it for a project. Python is the best programming language in Spark for general ease of use, a rich ecosystem, and integration with data science libraries. This makes it suitable for most data processing and machine learning tasks. It also comes with potential performance limitations. Scala is ideal for performance, direct integration with Spark, and leveraging functional programming, making it suitable for those seeking deep integration and efficiency. Java is preferred for performance, stability, and a mature ecosystem, making it suitable for large-scale enterprise applications despite being more verbose and complex. R excels in statistical analysis and visualization, making it suitable for data analysis tasks, although it has a lower ceiling on performance and is less feature-rich for Spark integration (Enugurthi, 2024). The choice of language ultimately depends on the specific requirements of one’s project, the expertise of one’s team, and the particular strengths one needs from the language.

References:

AITechStudio. (2024). *Cross-Validation & Regularization*. AITechStudio. https://aitech.studio/aie/regularization/

Apache Spark. (n.d.). *Spark Programming Guide*. Apache Spark. https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html

Ashtari, H. (2022). *What Is Hadoop Distributed File System (HDFS)? Working, Architecture, and Commands*. Spiceworks. https://www.spiceworks.com/tech/big-data/articles/hadoop-distributed-file-system-hdfs/

Brownlee, J. (2021). *SMOTE for Imbalanced Classification with Python*. Machine Learning Mastery. https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Databricks. (n.d.). *Machine Learning Models*. Databricks. https://www.databricks.com/glossary/machine-learning-models

Enugurthi, R. (2024). *Scala Vs Python Vs R Vs Java - Which language is better for Spark & Why?* KnowledgeHut. https://www.knowledgehut.com/blog/programming/scala-vs-python-vs-r-vs-java

Fullestop. (2015). *Data Storage Techniques: Object versus the HDFS*. Fullestop. https://www.fullestop.com/blog/data-storage-techniques-object-versus-the-hdfs

Google Cloud. (2018). *HDFS vs. Cloud Storage: Pros, cons and migration tips*. Google Cloud. https://cloud.google.com/blog/products/storage-data-transfer/hdfs-vs-cloud-storage-pros-cons-and-migration-tips

Hosmer, D.W., Lemeshow, S. and Sturdivant, R.X. (2013) *Applied Logistic Regression, 3rd ed.*, New York: Wiley

IBM. (2024). *HFDS Federation.* IBM. https://www.ibm.com/docs/en/spectrum-symphony/7.3.0?topic=features-hdfs-federation

Leone, R. (2019). *Object Storage - How It Works? (2/3)*. Scaleway Blog. https://www.scaleway.com/en/blog/object-storage-how-it-works/

Madireddy, M., Ferruzza, G., & Sharma, A. (2018) *An Overview of AutoML and Available Technology*. Medium. https://medium.com/an-overview-of-automl-and-available-technology/an-overview-of-automl-and-available-technology-a6d3e4650c7c

Smith, T. (2024). *What's the difference between cloud object storage vs HDFS*. Starburst. https://www.starburst.io/blog/cloud-object-storage-vs-hdfs/

Srivastava, T. (2024). *12 Important Model Evaluation Metrics for Machine Learning Everyone Should Know*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

Srivastava, T. (2022). *Support Vector Machine - simplified*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2014/10/support-vector-machine-simplified/

WHO. (n.d.) *Low birth weight*. WHO. https://www.who.int/data/nutrition/nlis/info/low-birth-weight