Exploratory Data Analysis on US Accidents

- <u>Group -10</u>
- RENUGOPAL SIVAPRAKASAM
- SUSMITHA SAMALA
- JAYSYAM DESIREDDY
- SWEATHA SUBRAMANIAN

INDEX

1. **Goals and objectives**
- ✓ Motivation
- ✓ Significance
- ✓ Objectives
- ✓ Features

2. **Introduction**
3. **Background**
- ✓ Related work of topic with linked references

4. **Our Solution**
- ✓ Architecture Diagram with Explanation
- ✓ Workflow diagram with explanation

5. **Dataset**
- ✓ Detailed description of Dataset
- ✓ Detail Design of Features with diagram

6. Analysis of data
- ✓ Data Pre-processing
- ✓ Graph model with explanation

7. Implementation
- ✓ Pseudocode
- ✓ Integration is clearly performed, shown, and explained.
- ✓ Explanation of implementation

8. Results
- ✓ Diagrams for results with detailed explanation

9. Project Management

10. Implementation status report
- ✓ Work completed
- ✓ Description
- ✓ Responsibility (Task, Person)
- ✓ Contributions (percentage)
- ✓ Issues/Concerns
- ✓ Uniqueness of project

11. References/Bibliography

Motivation

- The main motivation for this project is to improve public safety and prevent automobile accidents.
- In order to do this, we need to better understand the reasons that contribute to the accident and its severity.
- This information can be used by authorities to understand patterns and allocate proper emergency services.
- This Exploratory Data Analysis can throw light on predicting the severity of accidents which can help in arranging timely adequate medical help.

Significance

- EDA and predictive analytics is important for decision making which can **save lives and improve road safety**
- **Impact on community** this project can have when the relevant insights are shared with the policymakers and authorities fostering a collective effort to create safer environment.
- **Efficient traffic management practices** by understanding patterns and factors influencing accidents.
- The significance of using Pyspark as an integral component of our architecture is to have **scalability and to boost performance**.
- To be able to predict severity of an accident and take precautionary measures is also an important outcome of this project.

Objectives

- Our main objective is to prevent road accidents by predicting their severity based on location, time and weather.
- We perform exploratory data analysis on huge datasets to generate actionable insights.
- In order to utilize big data tools like Pyspark, Hadoop, Solr, Lucene, Hive, etc to process huge amounts of data and identify accident patterns.
- Performance optimization in pyspark to process data faster than traditional hadoop approach.

Understanding the data patterns and reporting to government agencies.

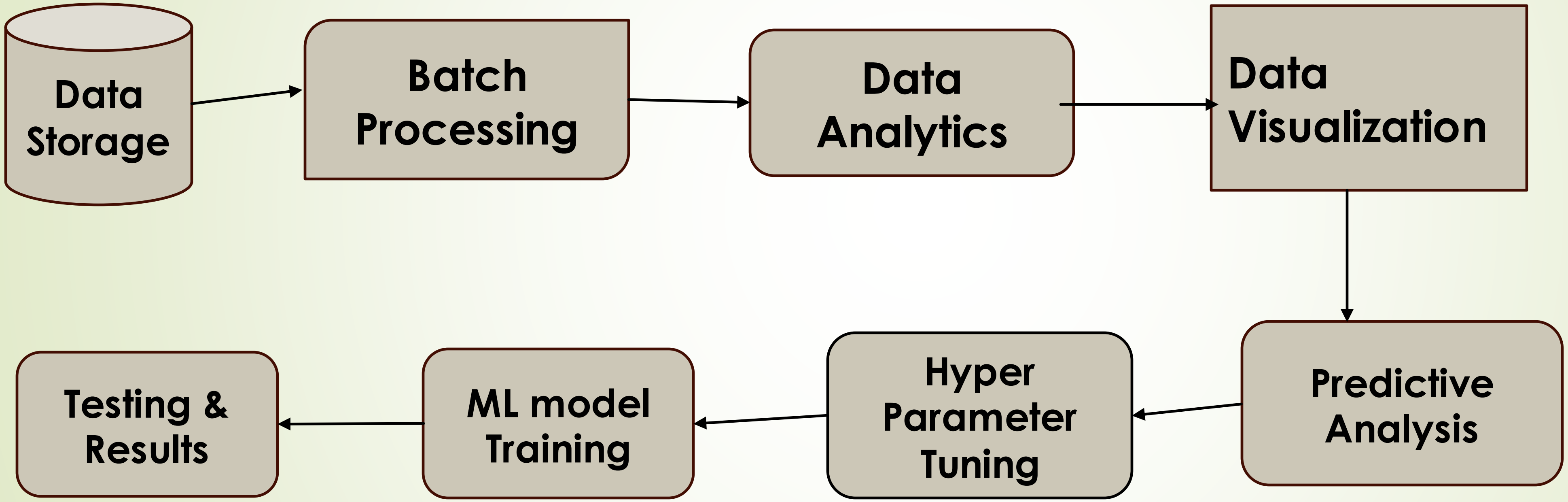
FEATURES

- Google Cloud Platform has been chosen as the cloud provider for this project mainly because of the excellent data processing services and processing capabilities.
- GCP's Dataproc is Google's version of HDFS file system which allows seamless integration with other big data tools.
- Pyspark used to perform in memory processing thereby increasing the speed multifold compared to the traditional apache hadoop.
- We integrate Hive with Pyspark to store as tables and process large volumes of data.
- Solr is used to index the data and query it faster.
- Visualize the data using Matplotlib
- Predictive analytics to find the severity of accidents.

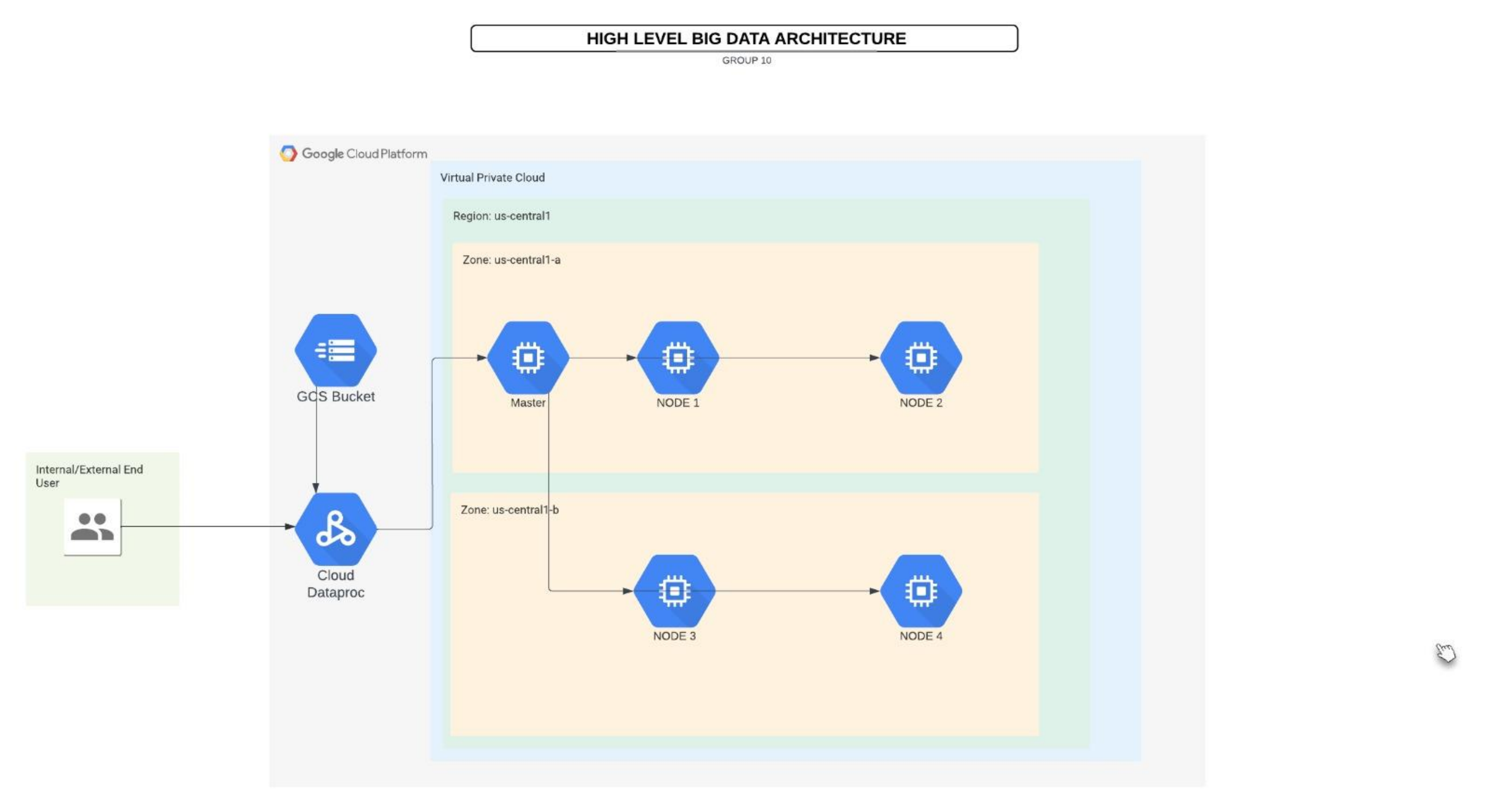
BACKGROUND WORK

- **US Accidents Dataset**
  This data comprises of data about accidents in US from 2016 to 2023, sourced from Kaggle. This dataset consists of over 7 Billion data points and 46 unique features.
- Previous work by Lahiru S. Boyagoda and Lakshika S. Nawarathna on "Analysis and Prediction of Severity of United States Countrywide Car Accidents Based on Machine Learning Techniques". (link)
- Machine learning for accident prediction in the paper - Predicting Crash Injury Severity with Machine Learning Algorithm Synergized with Clustering Technique. (link)
- Spark and Hadoop comparative study.
- Data Visualization using multiple libraries like Matplotlib, Plotly, Geoplot.

Architecture Diagram



GCP Architecture



Architecture

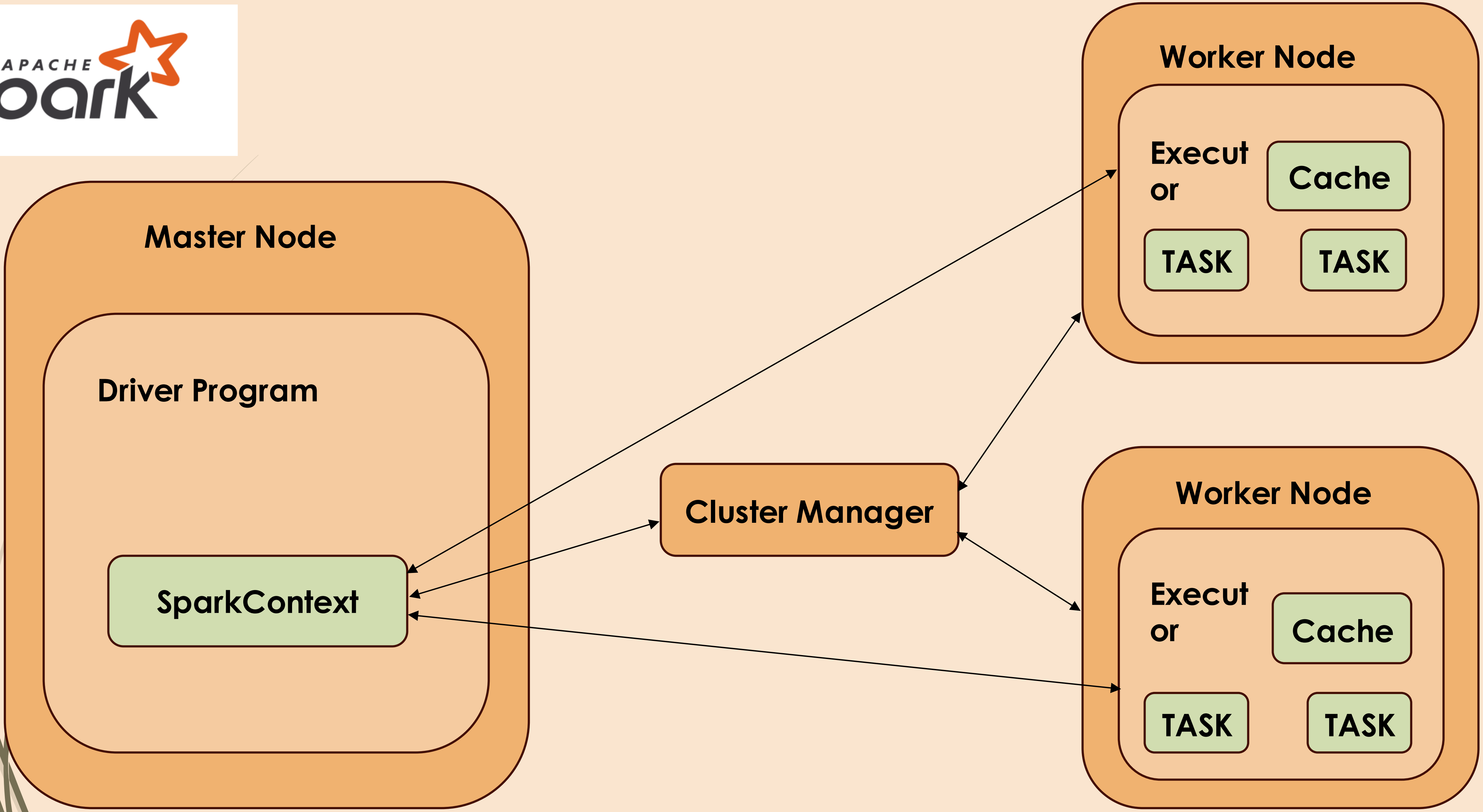
- Google Cloud Platform (GCP) Infrastructure
  Dataproc Cluster: The core of the architecture involves utilizing Google Cloud Dataproc to create and manage clusters. Dataproc provides a fully managed Apache Spark and Hadoop service, allowing for scalable and efficient data processing.
- **PySpark** - For distributed data processing, PySpark, the Python API for Apache Spark, is used. It makes it easier to create scalable and parallelized data transformations and analytics that take advantage of the Dataproc cluster's processing capability.
- **Hive**, a Hadoop data warehouse and SQL-like query language, is integrated into the design. It offers organized querying of the dataset, allowing for the creation of tables and the execution of complicated queries, hence improving data retrieval and analysis performance.
- **Solr** for Full-Text Search and Indexing

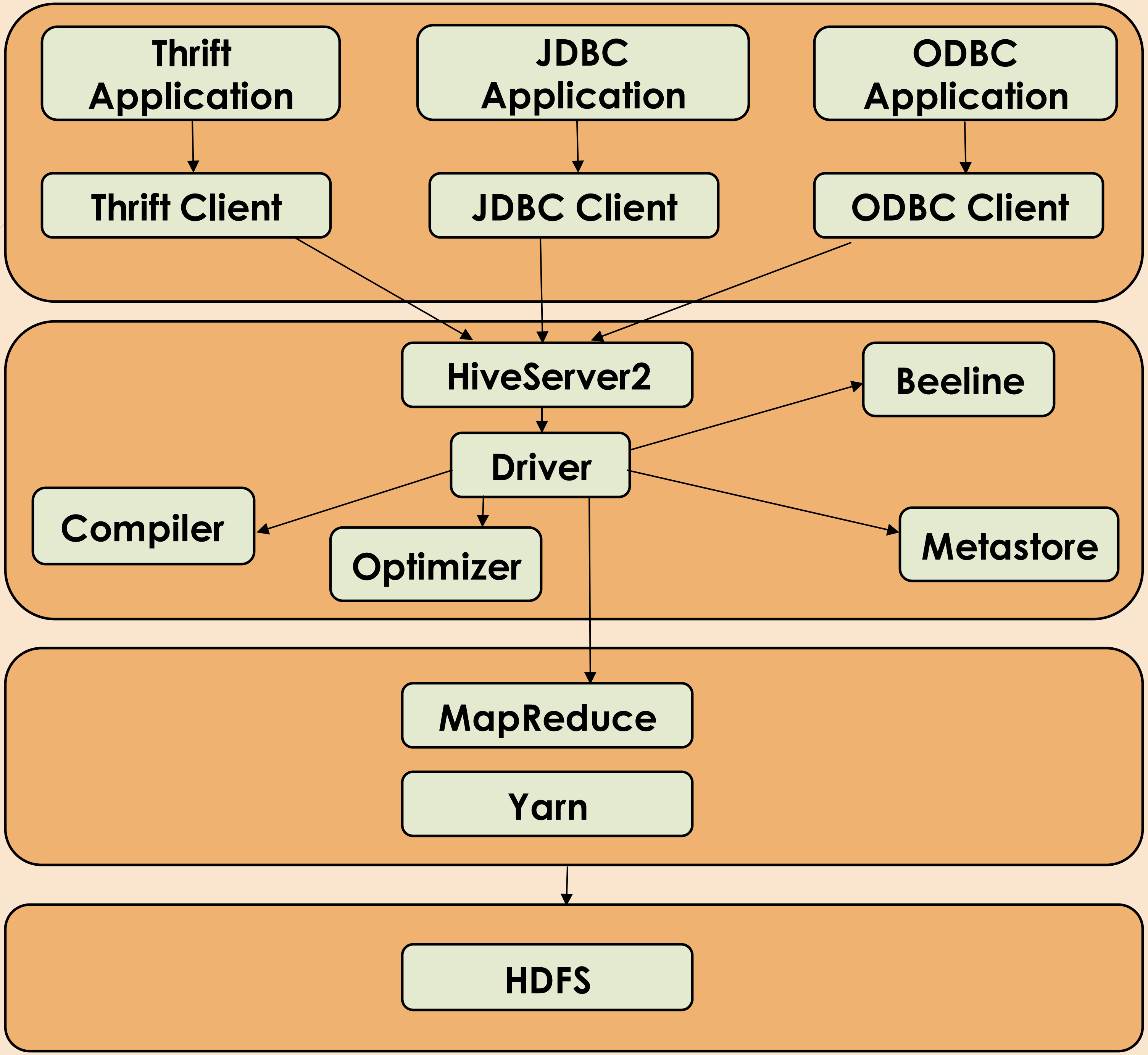
Architecture

- **Googler Cloud Storage** for storing intermediate results, works in tandem with dataproc for storing the dataset.
- **Identity and Access Management (IAM)** for providing authorized access to users to manage services.
- **Autoscaling policy** – set as a policy to scale up VM's by creating new nodes.
- **Visualizations using Jupyter Notebook** - this provides a user friendly way to work on the pyspark- python code.
- **Apache Solr** on GCP DataProc which provides rich indexing and querying capabilities

Workflow

DATA STORAGE
HDFS by Google
Dataproc
Google Cloud Storage
DATA PROCESSING
PySpark
HIVE
Search and Indexing
Solr
Lucene
Data Visualization
matplotlib
plotly


Master Node
Driver Program
SparkContext
Cluster Manager
Worker Node
Execut
or
Cache
TASK
TASK
Worker Node
Execut
or
Cache
TASK
TASK


Thrift Application
JDBC Application
ODBC Application
Thrift Client
JDBC Client
ODBC Client
HiveServer2
Beeline
Driver
Compiler
Optimizer
Metastore
MapReduce
Yarn
HDFS

REQUEST HANDLERS

/admin | /select | /spell

RESPONSE WRITERS

XML | Binary | JSON

UPDATE HANDLERS

XML | CSV | Binary

SEARCH COMPONENTS

Query
Highlighting
Spelling
Statistics
Faceting
Debug

Distributed Search

SCHEMA

CONFIG

UPDATE PROCESSO
RS

Signature
Logging
Indexing
Query
Parsing
Analysis

Extracting
Request
Handler
PDF/WORD

Apache
Tika

Data Import Handler
(SQL/RSS)

Faceting | Filtering | Search | Caching | Highlighting

INDEX REPLICATION

Apache Lucene

Core Search
IndexReader/
Searcher

Text Analysis

Indexing
IndexWriter

Detailed Description of the US Accidents Dataset:

- The US Accidents dataset is a comprehensive dataset that typically includes detailed information about traffic accidents that have occurred across the 49 United States.
- This dataset is usually compiled from various sources, including law enforcement agencies, traffic cameras, and other monitoring systems.
- The accident data were gathered between February 2016 and March 2023.

1. ID - The dataset has a unique ID for every single accident that has happened.
2. Source - The information about the accident report can be taken from the police reports, news outlets, traffic cameras, etc.
3. Severity – The severity of the accident can be measured on a numerical scale.
4. Time and Date when the accident occurred are crucial for analyzing the timings of the accident.
5. To know where the accidents happened it can be tracked using the Location details.
6. Start_Lat, Start_Lan - It gives the location where the accident started.

7. End_Lat, End_Lan – It gives the end location of the accident.
8. The exact address details of the accident spot can be taken from the Street, City, County, State, Zipcode, Country.
9. Timezone, Airport_Code - For regional analysis, time zone information and nearby airport codes are included.
10. Weather Conditions:
    - Weather_Timestamp: The time of the weather observation.
    - Temperature(F), Wind_Chill(F): Temperature and wind chill factors.
    - Humidity(%), Pressure(in), Visibility(mi): Humidity levels, atmospheric pressure, and visibility.
    - Wind_Direction, Wind_Speed(mph): Wind direction and speed.
    - Precipitation(in): Rainfall or snowfall amounts.
    - Weather_Condition: General weather conditions (e.g., cloudy, clear, rain, fog)

    11. Distance(mi) - The length of the road that was impacted by the collision, measured in miles.

12. Description - A textual description of the accident.

Preliminary Analysis of data

DATA CLEANING

We look for missing values first

In [3]:
```
from pyspark import pandas as pd_10
```

```
/usr/lib/spark/python/pyspark/pandas/__init__.py:49: UserWarning: 'PYARROW_IGNORE_TIMEZONE' environment variable wa
s not set. It is required to set this environment variable to '1' in both driver and executor sides if you use pyar
row>=2.0.0. pandas-on-Spark will set it for you but it does not work if there is a Spark context already launched.
  warnings.warn(
```

In [4]:
```
from pyspark.sql import functions as ps_10
```

In [5]:
```
op = 'Severity'

# find different column types and seggregate to set right defaults
string_cols =  [col[0] for col in df.dtypes if col[1] == "string" ]

num_cols = [col[0] for col in df.dtypes if col[1] == "int" or col[1] == "double" or col[1] == "float" ]

# output could be null
num_cols.remove(op)

bool_cols = [col[0] for col in df.dtypes if col[1] == "boolean"]

print("String columns - ", string_cols)
print("Numeric columns - ", num_cols)
print("Boolean columns - ", bool_cols)
```

```
String columns -  ['ID', 'Source', 'Description', 'Street', 'City', 'County', 'State', 'Zipcode', 'Country', 'Timez
one', 'Airport_Code', 'Wind_Direction', 'Weather_Condition', 'Sunrise_Sunset', 'Civil_Twilight', 'Nautical_Twiligh
t', 'Astronomical_Twilight']
Numeric columns -  ['Start_Lat', 'Start_Lng', 'End_Lat', 'End_Lng', 'Distance(mi)', 'Temperature(F)', 'Wind_Chill
(F)', 'Humidity(%)', 'Pressure(in)', 'Visibility(mi)', 'Wind_Speed(mph)', 'Precipitation(in)']
Boolean columns -  ['Amenity', 'Bump', 'Crossing', 'Give_Way', 'Junction', 'No_Exit', 'Railway', 'Roundabout', 'Sta
tion', 'Stop', 'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop']
```

```
In [6]: # now we initialize empty rows for each kind of datatype

        df = df.fillna("not_available", string_cols)

        df = df.fillna(0 , num_cols)

        # for bool column, we really can't initialize with 0 or 1
```

```
In [12]: # Now we count the no of nulls to verify -
         col_vs_nulls = {col:df.filter(ps_10.isnull(df[col[0]])).count() for col in df.dtypes }
         print(col_vs_nulls)
```

```
{('ID', 'string'): 0, ('Source', 'string'): 0, ('Severity', 'int'): 0, ('Start_Time', 'timestamp'): 0, ('End_Time',
'timestamp'): 0, ('Start_Lat', 'double'): 0, ('Start_Lng', 'double'): 0, ('End_Lat', 'double'): 0, ('End_Lng', 'dou
ble'): 0, ('Distance(mi)', 'double'): 0, ('Description', 'string'): 0, ('Street', 'string'): 0, ('City', 'string'):
0, ('County', 'string'): 0, ('State', 'string'): 0, ('Zipcode', 'string'): 0, ('Country', 'string'): 0, ('Timezon
e', 'string'): 0, ('Airport_Code', 'string'): 0, ('Weather_Timestamp', 'timestamp'): 120228, ('Temperature(F)', 'do
uble'): 0, ('Wind_Chill(F)', 'double'): 0, ('Humidity(%)', 'double'): 0, ('Pressure(in)', 'double'): 0, ('Visibilit
y(mi)', 'double'): 0, ('Wind_Direction', 'string'): 0, ('Wind_Speed(mph)', 'double'): 0, ('Precipitation(in)', 'dou
ble'): 0, ('Weather_Condition', 'string'): 0, ('Amenity', 'boolean'): 0, ('Bump', 'boolean'): 0, ('Crossing', 'bool
ean'): 0, ('Give_Way', 'boolean'): 0, ('Junction', 'boolean'): 0, ('No_Exit', 'boolean'): 0, ('Railway', 'boolea
n'): 0, ('Roundabout', 'boolean'): 0, ('Station', 'boolean'): 0, ('Stop', 'boolean'): 0, ('Traffic_Calming', 'boole
an'): 0, ('Traffic_Signal', 'boolean'): 0, ('Turning_Loop', 'boolean'): 0, ('Sunrise_Sunset', 'string'): 0, ('Civil
_Twilight', 'string'): 0, ('Nautical_Twilight', 'string'): 0, ('Astronomical_Twilight', 'string'): 0}
```

INTEGRATION OF GCS DATAPROC/HDFS AND PYSPARK

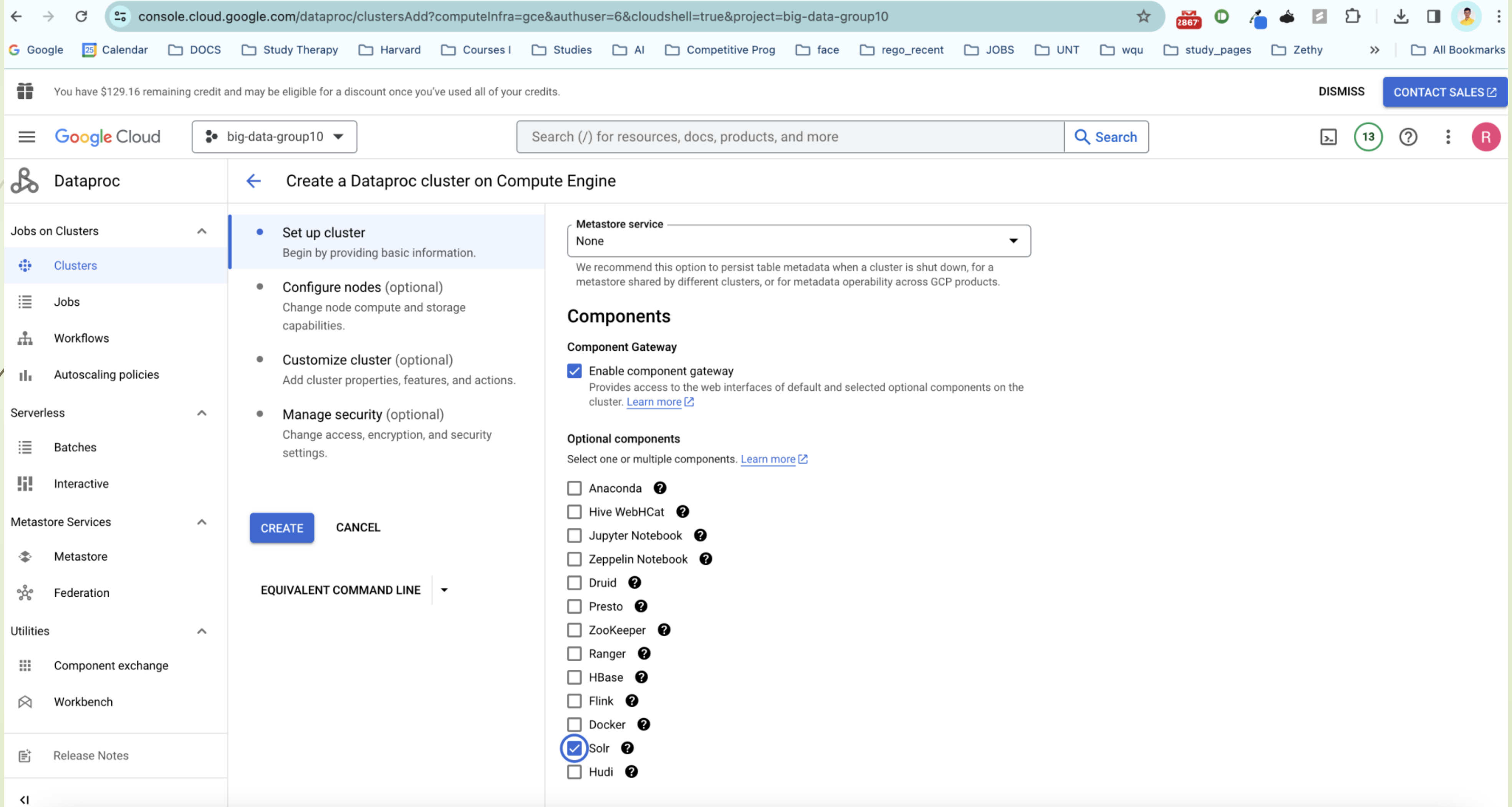
- In GCP, since we have multiple nodes, HDFS works in a different way.
- Data is typically stored in GCS buckets
- During processing, dataproc clusters read data from GCS and write results back to GCS.
- Although Dataproc clusters store data on GCS, they offer an interface that is compatible with HDFS. This implies that Dataproc clusters can utilize the same commands and APIs for the Hadoop Distributed File System (HDFS) as a regular Hadoop cluster.
- Fault Tolerant storage

INTEGRATION OF SPARK AND HIVE

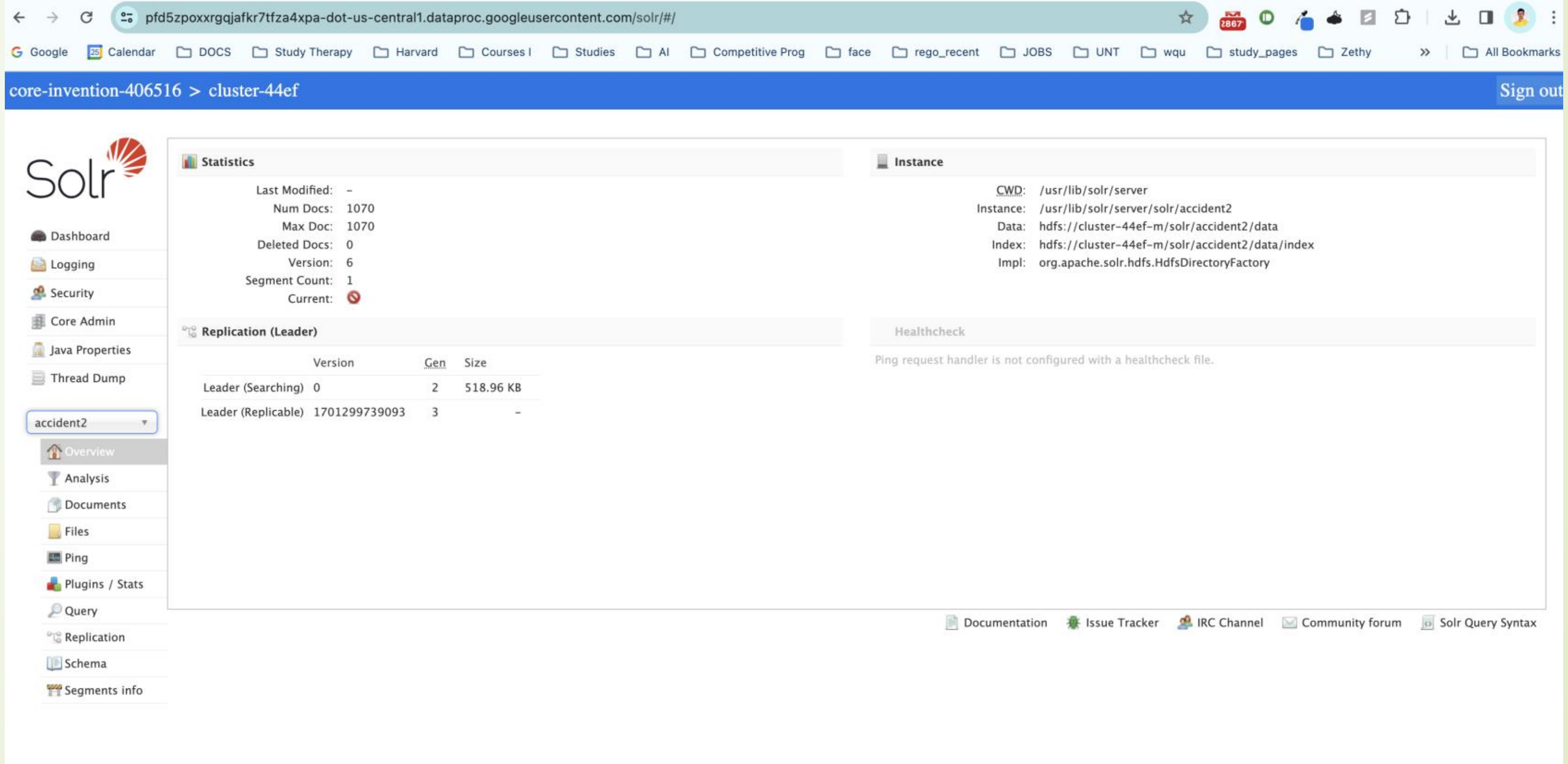
- In the current architecture, we are using the dataset from Dataproc – GCS bucket to create table in Hive then query using Pyspark sql.

```
In [1]: from pyspark import SparkContext, SparkConf
        from pyspark.conf import SparkConf
        from pyspark.sql import SparkSession, HiveContext

In [2]: sparkSession = (SparkSession
                        .builder
                        .appName('example-pyspark-read-and-write-from-hive')
                        .config("hive.metastore.uris", "thrift://localhost:9083", conf=SparkConf())
                        .enableHiveSupport()
                        .getOrCreate()
                        )

        23/11/29 15:53:25 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take eff
        ect.

In [3]:
        data_path = "gs://group_10_big_data/US_Accidents_March23.csv"
        accidnets_df = spark.read.csv(data_path, header=True, inferSchema=True)

In [4]: # Write to HIVE TABLE
        accidnets_df.write.saveAsTable('accidents')

        ivysettings.xml file not found in HIVE_HOME or HIVE_CONF_DIR,/etc/hive/conf.dist/ivysettings.xml will be used
```

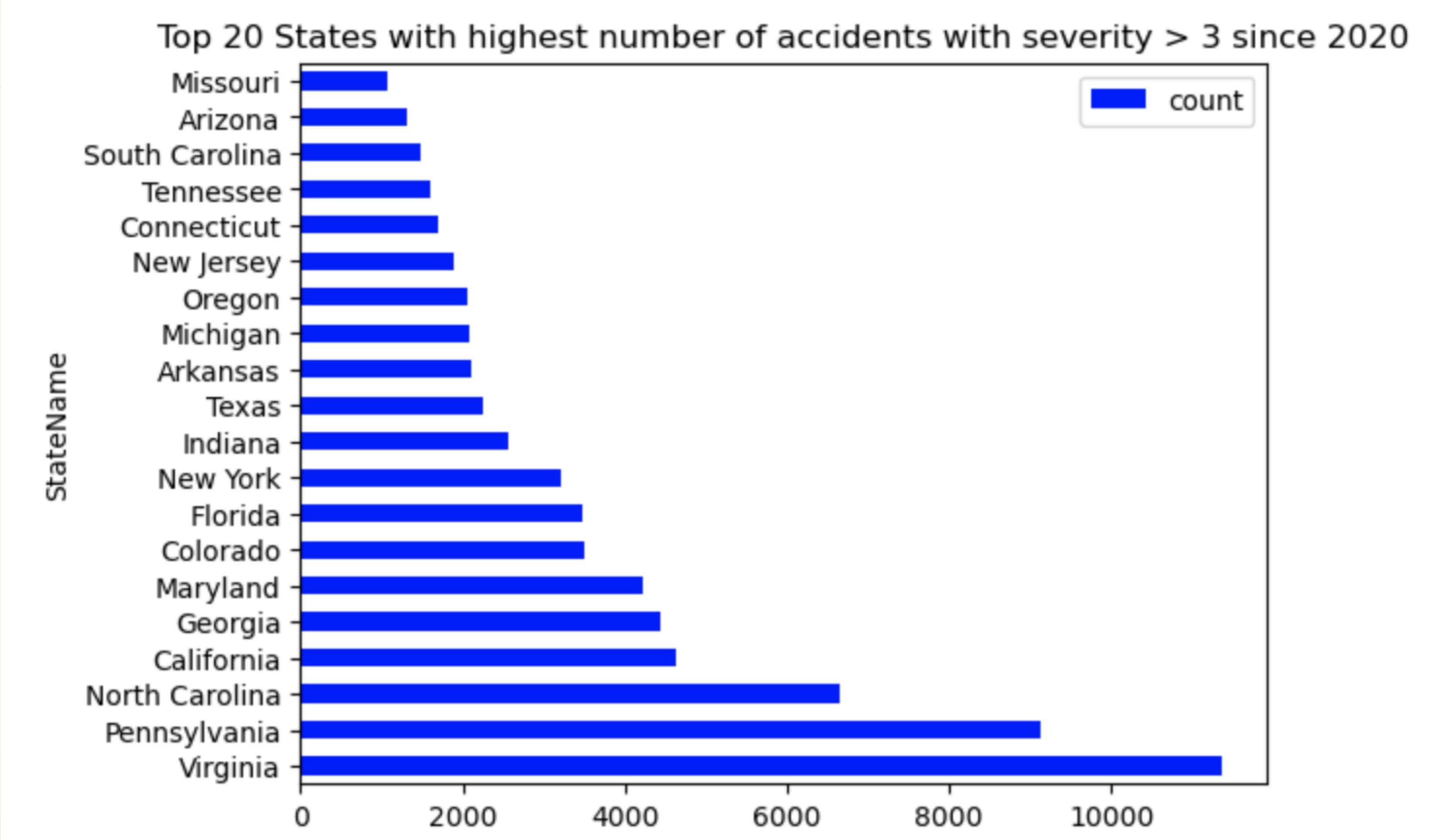
INTEGRATION OF SPARK AND SOLR

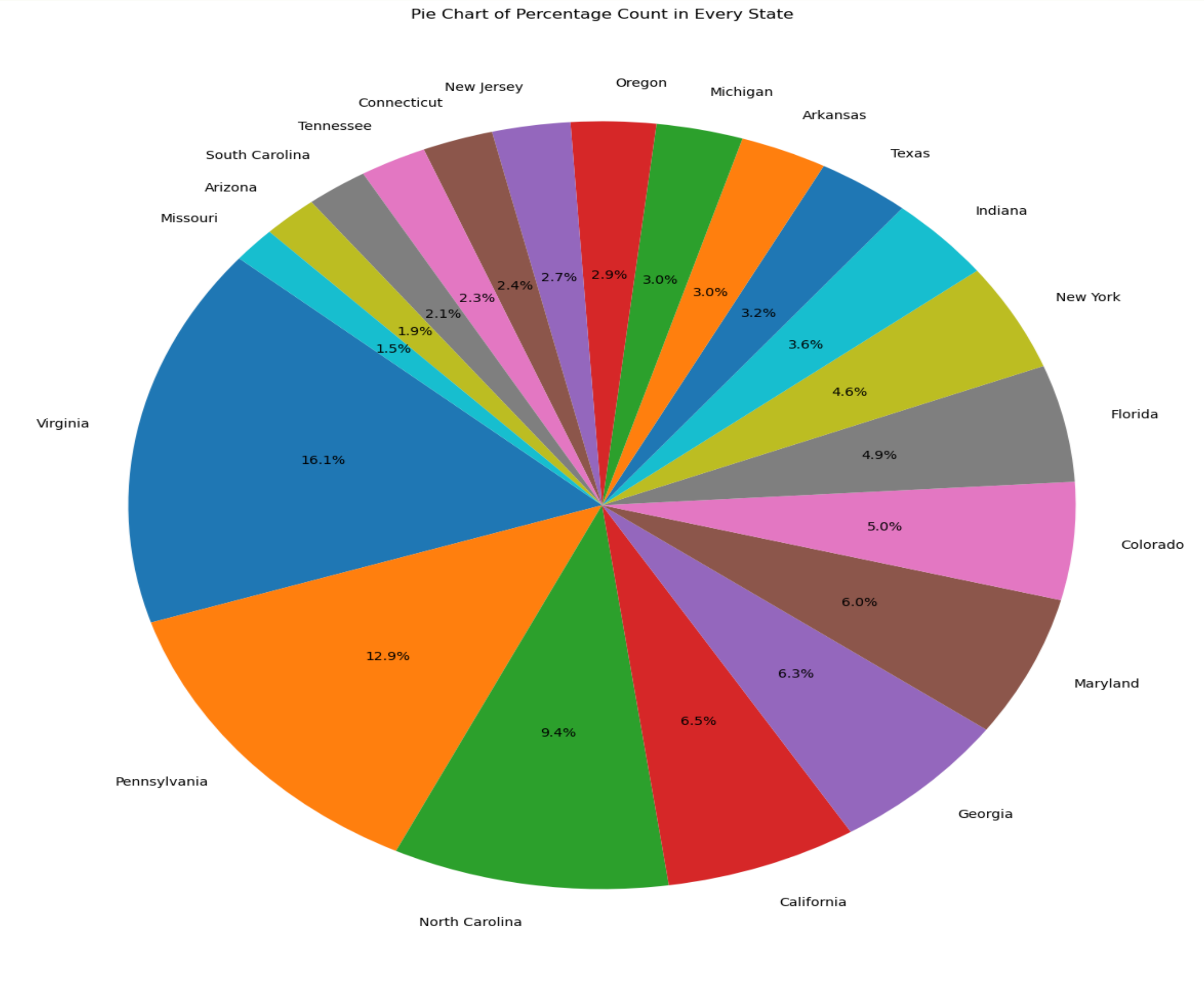


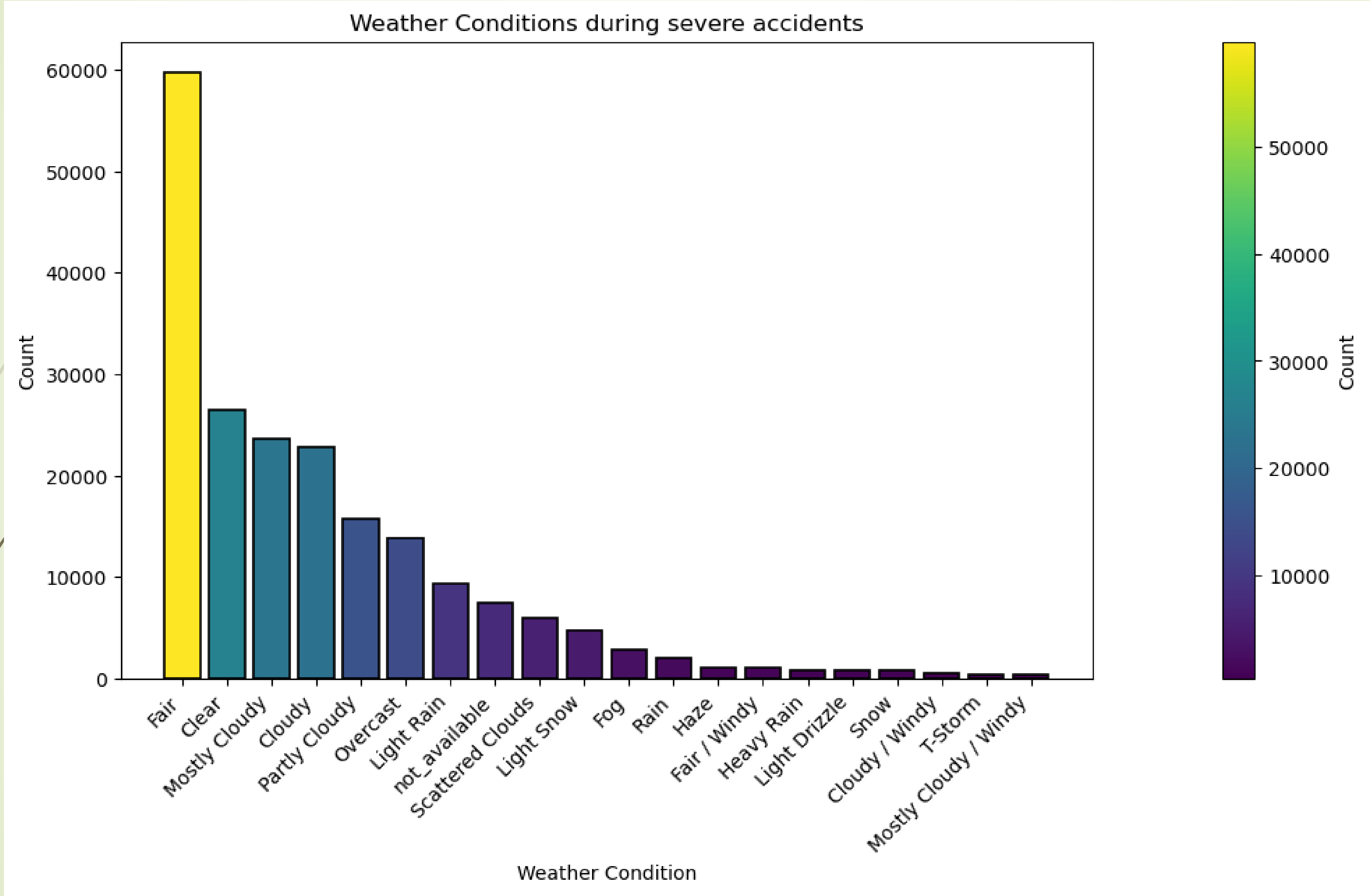
INTEGRATION OF SPARK AND SOLR

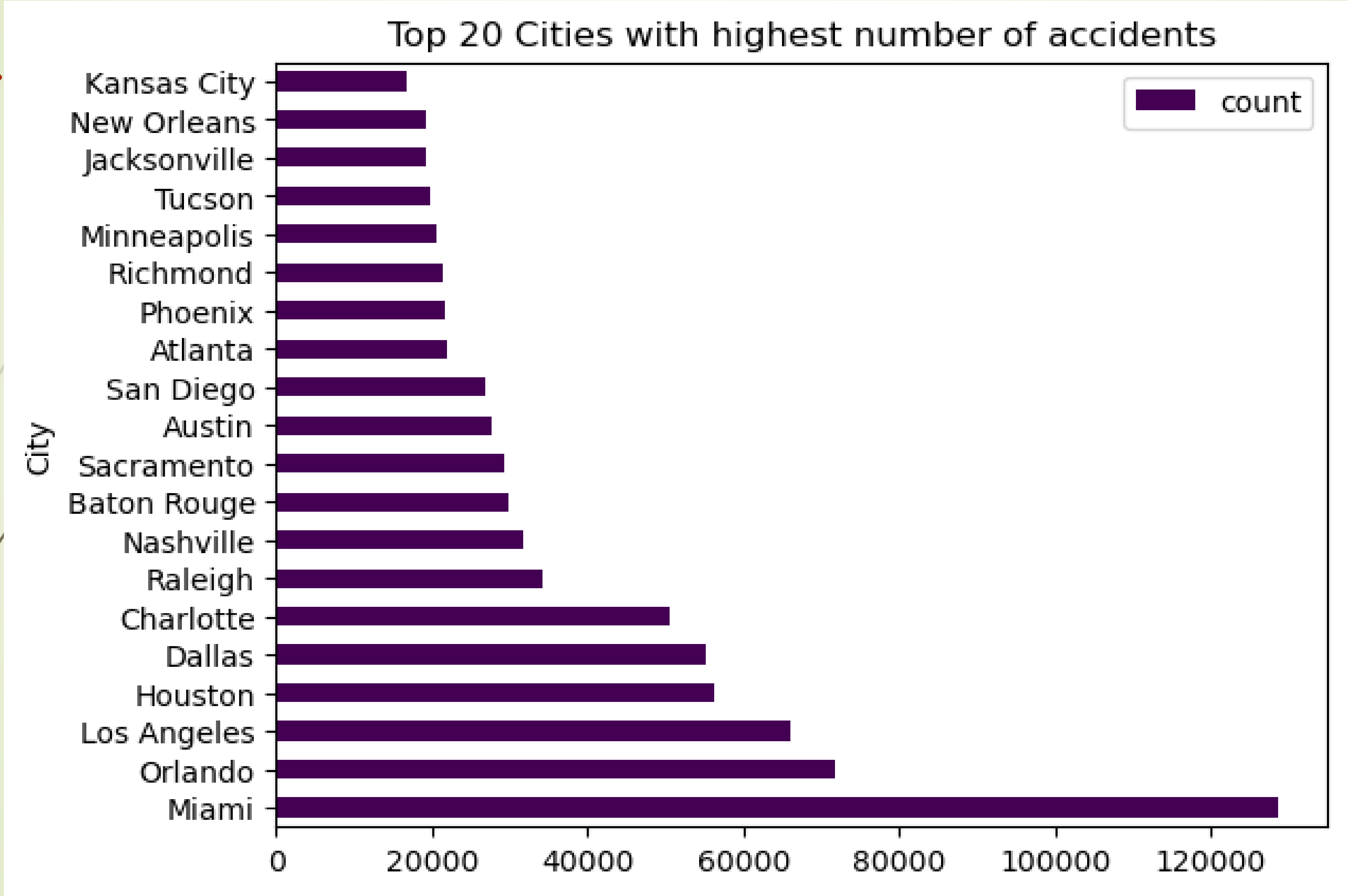


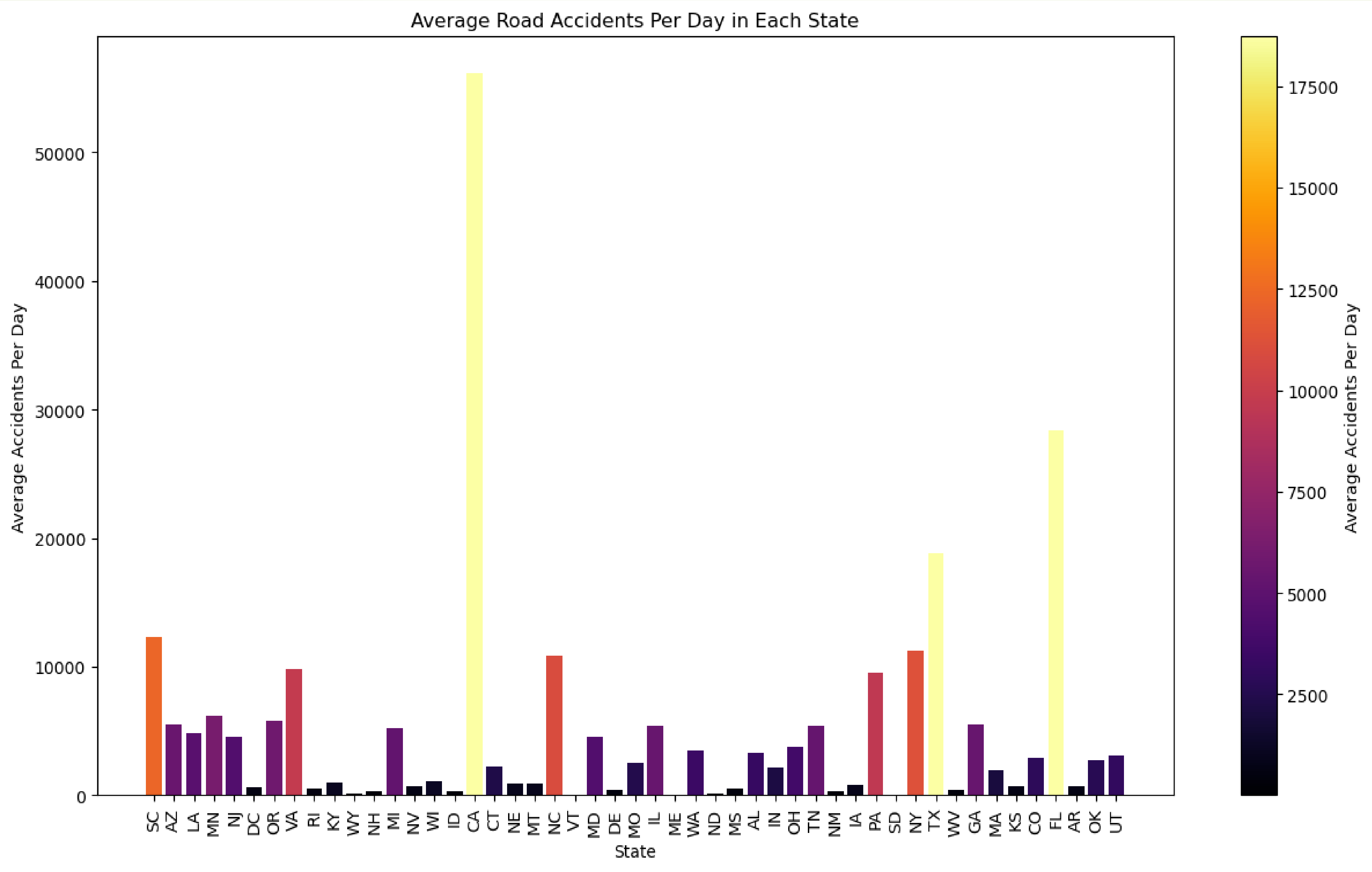
EXPLORATORY DATA ANALYSIS




Pie Chart of Percentage Count in Every State
Oregon
2.9%
Michigan
3.0%
Arkansas
3.0%
Texas
3.2%
Indiana
3.6%
New York
4.6%
Florida
4.9%
Colorado
5.0%
Maryland
6.0%
Georgia
6.3%
California
6.5%
North Carolina
9.4%
Pennsylvania
12.9%
Virginia
16.1%
Missouri
1.5%
Arizona
1.9%
South Carolina
2.1%
Tennessee
2.3%
Connecticut
2.4%
New Jersey
2.7%

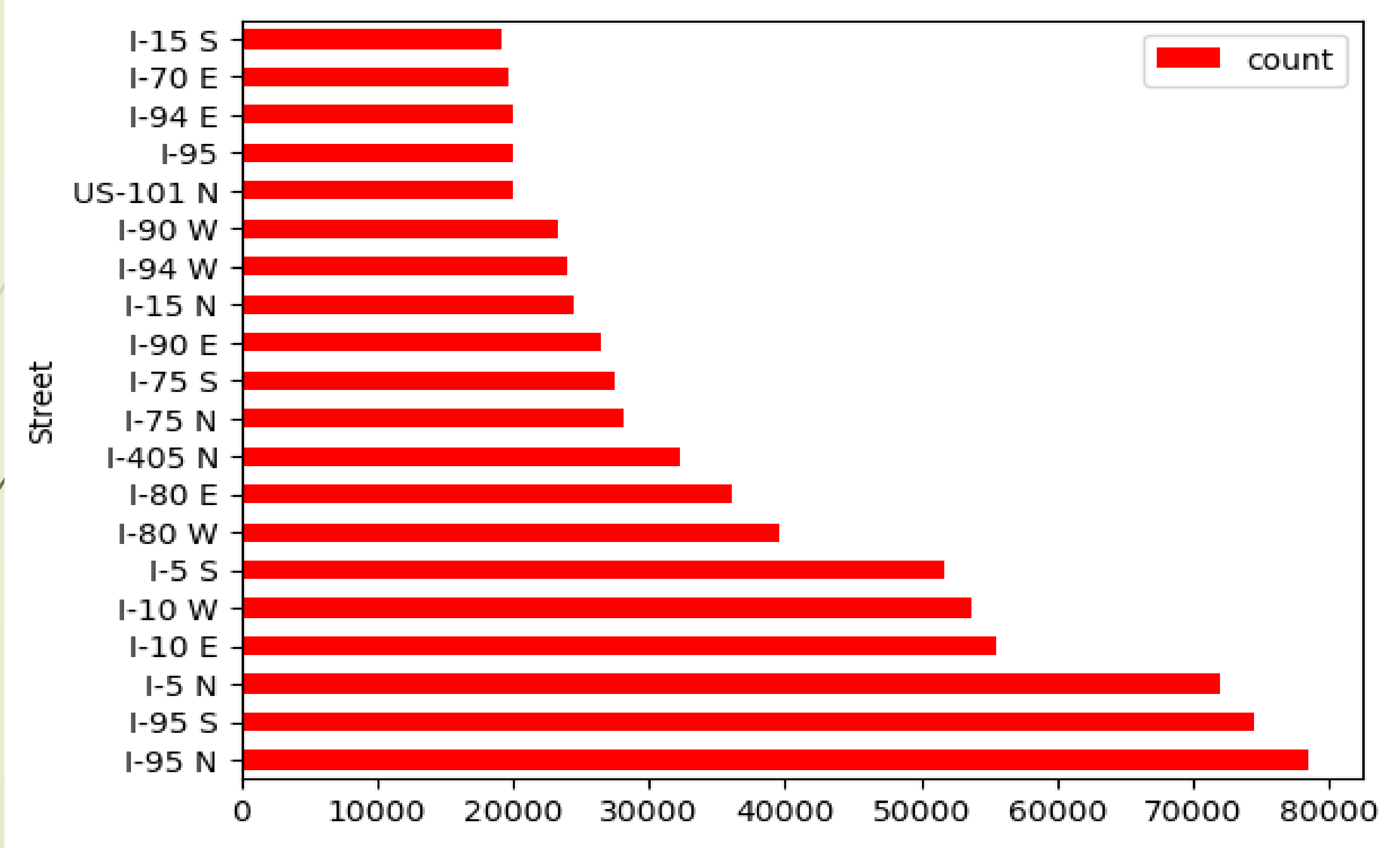

Weather Conditions during severe accidents
Count
60000
50000
40000
30000
20000
10000
0
Fair
Clear
Mostly Cloudy
Cloudy
Partly Cloudy
Overcast
Light Rain
not_available
Scattered Clouds
Light Snow
Fog
Rain
Haze
Fair / Windy
Heavy Rain
Light Drizzle
Snow
Cloudy / Windy
T-Storm
Mostly Cloudy / Windy
Weather Condition
Count
50000
40000
30000
20000
10000

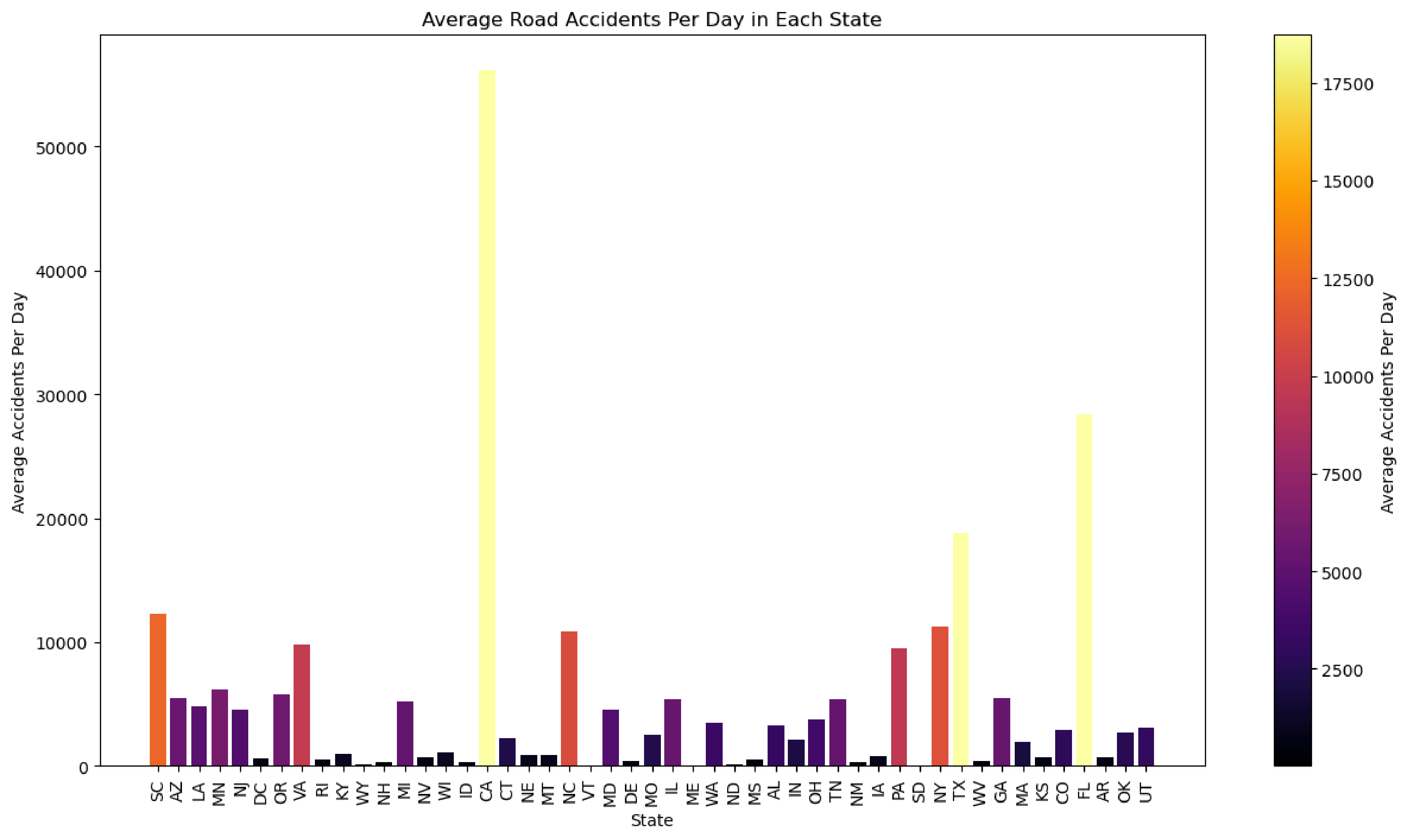

Top 20 Cities with highest number of accidents
count
City
Kansas City
New Orleans
Jacksonville
Tucson
Minneapolis
Richmond
Phoenix
Atlanta
San Diego
Austin
Sacramento
Baton Rouge
Nashville
Raleigh
Charlotte
Dallas
Houston
Los Angeles
Orlando
Miami
0
20000
40000
60000
80000
100000
120000



Average Road Accidents Per Day in Each State
Average Accidents Per Day
State
50000
40000
30000
20000
10000
0
SC AZ LA MN NJ DC OR VA RI KY WY NH MI NV WI ID CA CT NE MT NC VT MD DE MO IL ME WA ND MS AL IN OH TN NM IA PA SD NY TX WV GA MA KS CO FL AR OK UT
17500
15000
12500
10000
7500
5000
2500

Top 20 dangerous Streets for accidents




Average Road Accidents Per Day in Each State
Average Accidents Per Day
50000
40000
30000
20000
10000
0
SC AZ LA MN NJ DC OR VA RI KY WY NH MI NV WI ID CA CT NE MT NC VT MD DE MO IL ME WA ND MS AL IN OH TN NM IA PA SD NY TX WV GA MA KS CO FL AR OK UT
State
17500
15000
12500
10000
7500
5000
2500
Average Accidents Per Day

PREDICTION OF ACCIDENT SEVERITY

jupyter v4_stats_severity_v4 (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | PySpark

Run | Code | nbdiff

In [21]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_timestamp, dayofweek, hour
from pyspark.ml.feature import VectorAssembler, StringIndexer, OneHotEncoder, StandardScaler
from pyspark.ml.classification import RandomForestClassifier, LogisticRegression, GBTClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
```

In [22]:
```
print(spark)
```

```
<pyspark.sql.session.SparkSession object at 0x7f3cbffc96f0>
```

In [23]:
```
data_path = "gs://group_10_big_data/US_Accidents_March23.csv"
df = spark.read.csv(data_path, header=True, inferSchema=True)
```

In [24]:
```
# Data preprocessing
df = df.withColumn("Start_Time", to_timestamp(col("Start_Time")))
df = df.withColumn("End_Time", to_timestamp(col("End_Time")))
df = df.withColumn("Day_of_Week", dayofweek(col("Start_Time")))
df = df.withColumn("Hour_of_Day", hour(col("Start_Time")))

# Handle null values
df = df.na.drop()

indexer = StringIndexer(inputCol="Weather_Condition", outputCol="Weather_Index").setHandleInvalid("keep")
encoder = OneHotEncoder(inputCols=["Weather_Index"], outputCols=["Weather_Cond"])

assembler = VectorAssembler(
    inputCols=["Start_Lat", "Start_Lng", "End_Lat", "End_Lng", "Distance(mi)", "Day_of_Week", "Hour_of_Day", "Weathe
    outputCol="features",
    handleInvalid="skip")

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
```

```
In [25]: # Initialize multiple classifiers
classifiers = [
    RandomForestClassifier(labelCol="Severity", featuresCol="scaledFeatures"),
#     LogisticRegression(labelCol="Severity", featuresCol="scaledFeatures"),
    # GBTClassifier(labelCol="Severity", featuresCol="scaledFeatures")
]
```

```
In [26]: 
# Split the data into training and test sets
train_data, test_data = df.randomSplit([0.7, 0.3])
```

```
In [27]: 
# Loop through classifiers
for classifier in classifiers:
    # Define the pipeline
    pipeline = Pipeline(stages=[indexer, encoder, assembler, scaler, classifier])

    # Define the parameter grid for hyperparameter tuning
    paramGrid = (ParamGridBuilder()
        .addGrid(classifier.maxIter, [10, 20]) if not isinstance(classifier, RandomForestClassifier)
        else ParamGridBuilder().addGrid(classifier.numTrees, [10, 20, 30])
        ).build()

    # CrossValidator for model tuning
    crossval = CrossValidator(estimator=pipeline,
                              estimatorParamMaps=paramGrid,
                              evaluator=MulticlassClassificationEvaluator(labelCol="Severity", predictionCol="predic
                              numFolds=3)

    # Fit the model
    cvModel = crossval.fit(train_data)

    # Make predictions on the test data
    predictions = cvModel.transform(test_data)

    # Evaluate the model
    evaluator = MulticlassClassificationEvaluator(labelCol="Severity", predictionCol="prediction", metricName="accur
    accuracy = evaluator.evaluate(predictions)
    #print(f"Model Accuracy for {classifier._class.name_}: {accuracy}")
    print(f"Model Accuracy : {accuracy}")
```

```
    #print(f"Model Accuracy for {classifier._class.name_}: {accuracy}")
    print(f"Model Accuracy : {accuracy}")
```

```
[Stage 548:==================================================>    (21 + 2) / 23]
```

```
Model Accuracy : 0.9420933721532795
```

In [ ]:

In [28]:

```
# from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# accuracy_evaluator = MulticlassClassificationEvaluator(labelCol="Severity", predictionCol="prediction", metricName
# accuracy = accuracy_evaluator.evaluate(predictions)
# print(f"Model Accuracy : {accuracy}")

# Calculate Precision and Recall
precision_evaluator = MulticlassClassificationEvaluator(labelCol="Severity", predictionCol="prediction", metricName=
recall_evaluator = MulticlassClassificationEvaluator(labelCol="Severity", predictionCol="prediction", metricName="we
precision = precision_evaluator.evaluate(predictions)
recall = recall_evaluator.evaluate(predictions)

print(f"Model Precision: {precision}")
print(f"Model Recall: {recall}")
```

```
[Stage 552:====================================================>  (22 + 1) / 23]
```

```
Model Precision: 0.8875399218551375
Model Recall: 0.9420933721532795
```

In [29]:

```
# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall if precision + recall != 0 else 1)

print(f"Model F1 Score: {f1_score}")
```

```
Model F1 Score: 0.9140033477083392
```

In [29]:
```
# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall if precision + recall != 0 else 1)

print(f"Model F1 Score: {f1_score}")
```

```
Model F1 Score: 0.9140033477083392
```

In [30]:
```
import pandas as pd
predictions_pd = predictions.select(['probability', 'Severity']).toPandas()

# Extract the probabilities for the positive class (assuming binary classification)
predictions_pd['probability'] = predictions_pd['probability'].apply(lambda x: x[1])
```

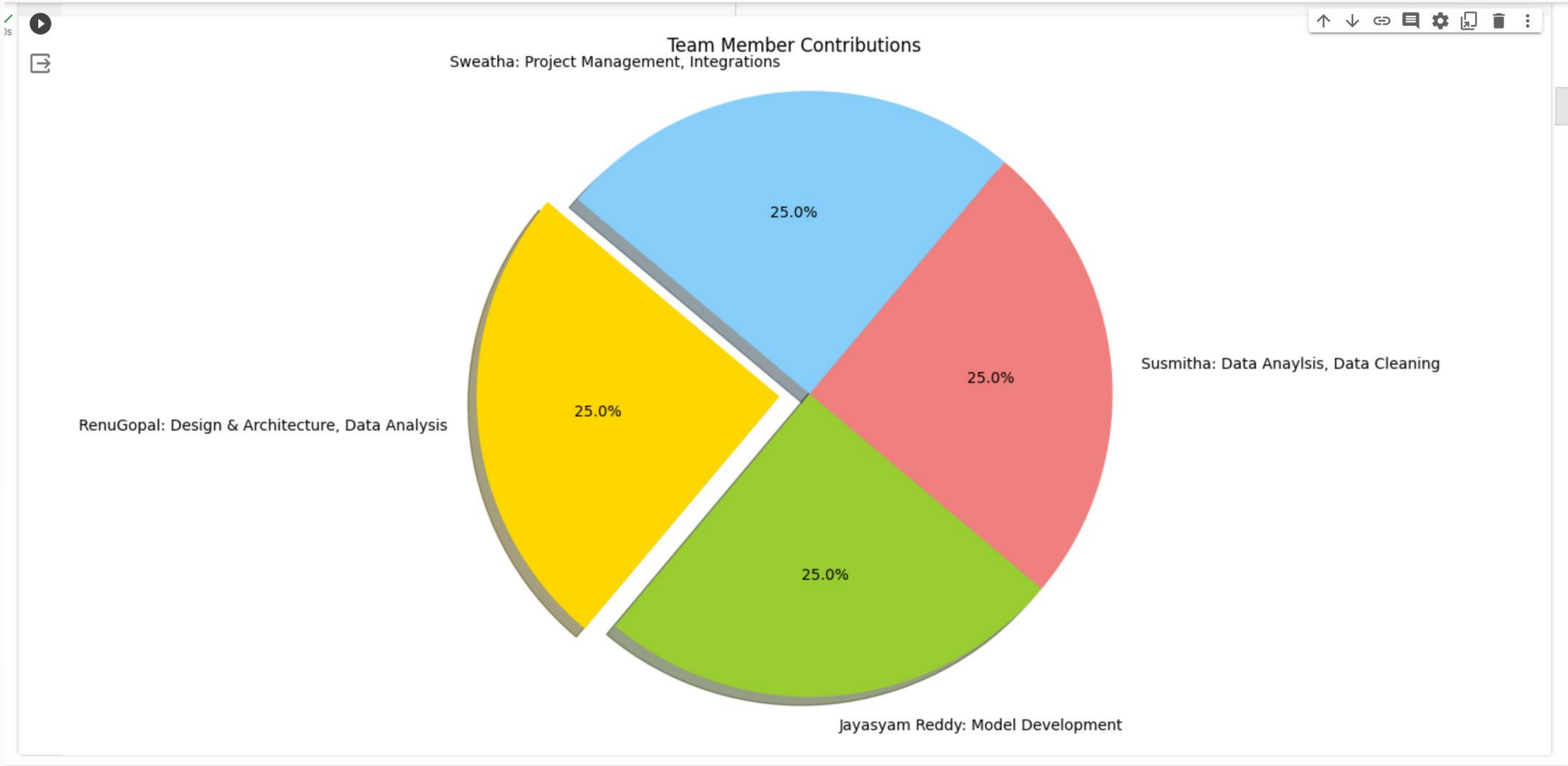
In [20]:
```
# Convert the 'Severity' levels to a binary problem
y_true_binary = (y_true == 2).astype(int)

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_true_binary, y_scores)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

ISSUES/CHALLENGES

- Idea initially was to develop on DataProc Kube Clusters which will make it even more scalable and resilient. But GCP has restrictions on Free tier accounts.
- Idea was to use Serverless Cluster on DataProc as well, this also required more credits.
- The entire experimentation process on GCP cloud used around 350$ worth free credits.
- The VM Dataproc cluster running Jupyter Notebook's Kernel died when we run huge machine learning models, kernel had to be restarted multiple times.
- The higher config for the cluster is limited to Businesses, we have use the highest possible configs allowed for an individual working on GCP.
- AWS or other cloud providers did not provide free credits.
- The size of the data 3GB is huge compared to other projects which are run like a POC

UNIQUE POINTS

- Size of the Dataset – 7B data points
- Pyspark, MLib and custom GCP architecture.
- Multiple integrations with tools like Hive, Solr.
- Highly Scalable with the current allowed infrastructure. (added auto scaling policy on gcp)
- Accuracy of over 94% to predict severity of accident using Random forest Classifier.
- State of the art – business ready architecture and infrastructure utilized.



Team contributions

References

PySpark: https://spark.apache.org/docs/latest/api/python/index.html
Hadoop HDFS: https://www.ibm.com/topics/hdfs
Cassandra: https://en.wikipedia.org/wiki/Cassandra
Scikit-Learn: https://scikit-learn.org/
MLlib: https://spark.apache.org/mllib/
Matplotlib: https://matplotlib.org/
Plotly: https://plotly.com/
GeoPlot: https://residentmario.github.io/geoplot/
Folium: https://python-visualization.github.io/folium/
XGBoost: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
https://www.kaggle.com/datasets/sobhanmoosavi/us-accidents
https://www.kaggle.com/datasets/daveianhickey/2000-16-traffic-flow-england-scotland-wales
https://www.kaggle.com/datasets/silicon99/dft-accident-data
https://ieeexplore.ieee.org/document/9719675
https://ieeexplore.ieee.org/document/9993371

Hatwell, J., Gaber, M.M. & Azad, R.M.A. CHIRPS: Explaining random forest classification. *Artif Intell Rev* **53**, 5747–5788 (2020). https://doi.org/10.1007/s10462-020-09833-6

Thank You!