

STW7082CEM: Big Data Management and Data Visualization

Submitted To
Shiddrtha Neupane

Submitted by

Samin kc
240581

Table contents

Introduction	2
2. Tools and Technologies: Facilitating Scalable Predictive Analytics.....	2
2.1 Distributed Data Processing and Machine Learning.....	3
2.2 Core Programming & Data Management	3
2.3 Visualization & Interpretability	3
2.4 Development Environments	4
3. Dataset and Analysis Goals: Establishing a Foundation for Predictive Healthcare	4
3.1 Contextual Significance.....	5
3.2 Research Objectives.....	6
3.3 Exploration of Health Patterns and Risk Factors.....	6
3.4 Supervised Classification Modeling	6
3.5 Development of Scalable and Explainable Systems.....	6
4. Data Pre-processing	7
4.1 Data Loading Using Apache Spark.....	7
4.2 Null Handling, Deduplication, and Schema Validation	8
4.3 Categorical Encoding using String Indexer and OneHotEncoder.....	9
4.4 Feature engineering.....	10
4.5 GBTCClassifier	12
4.6 Output.....	13
Exploratory Data Analysis and Visualization.....	15
5.1 Introduction	15
5.2 Dataset Overview.....	15
5.3 Data Preparation for EDA.....	16
5.4 Univariate Insights	16
5.5 Bivariate and Multivariate Relationships.....	17
5.6 Visualization Techniques.....	18
5.7 Key Findings	22
8. Discussion.....	23
9. Conclusion.....	24
References	25
Appendix	27
Link.....	36

Introduction

The rising prevalence of chronic diseases such as diabetes and cardiovascular conditions continues to pose a significant global health challenge. Although many of these illnesses are preventable or manageable through early intervention, healthcare systems often remain reactive rather than proactive. With the growing availability of electronic health records, wearable technologies, and other digital sources, a vast amount of health data is now being collected continuously.

This project explores the use of Big Data technologies and machine learning to shift healthcare toward early detection and prevention. By leveraging tools such as Apache Spark and PySpark, large-scale healthcare datasets can be processed efficiently, enabling the development of predictive models to identify individuals at risk. Classification techniques like Logistic Regression and Random Forest are applied, alongside data visualization tools such as Tableau.

The ultimate goal is to build a scalable, data-driven system that supports clinical decision-making and promotes personalized, preventive healthcare interventions.

2. Tools and Technologies: Facilitating Scalable Predictive Analytics

Building a data-driven preventive healthcare system requires a solid technology base to sustain large-scale data, complex analytical pipelines, and distributed machine learning. The selected tools break through key scalability, interoperability, and computational barriers to achieve reliable predictive analytics to detect diseases at early stages.

2.1 Distributed Data Processing and Machine Learning

Apache Spark: A system for distributed processing that is intended to facilitate mass-scale data processing, Spark enables the parallel execution of ETL (Extract, Transform, Load) pipelines and iterative machine learning operations. Fault tolerance and in-memory processing make Spark especially suitable to processing diverse health data, for example, EHRs (Electronic Health Records), wearable device stream data, and genome data.

PySpark (MLlib): Spark's Python API interacts with MLlib, Spark's scalable machine-learning library that enables distributed model training for gradient-boosted trees (GBTs), logistic regression, and clustering algorithms and others. This is necessary in order to construct high-performing risk-prediction models with no single-node computational bottlenecks.

2.2 Core Programming & Data Management

Python: Statistical analysis' dominant language (SciPy) and model prototyping' language (Scikit-learn and TensorFlow). Its extensive ecosystem allows one to move easily from exploratory tasks to deep deep-learning workflows.

SQL: Essential to querying and aggregation of structured healthcare data from relational stores (e.g., BigQuery, PostgreSQL). SQL fine-tunes the ETL pipelines to enable rapid extraction of patient cohorts and time-oriented trends in healthcare

2.3 Visualization & Interpretability

Tableau/Power BI: These transform predictive results to interactive dashboards with risk stratification, correlation of biomarkers and outcomes of intervention in focus. Visual analytics connects data science and clinical practice and facilitates evidence-based decision-making.

Matplotlib/Seaborn: Packages in Python for generating explanatory plots (e.g., SHAP plots, ROC curves) to describe model behavior and provide clinical relevance.

2.4 Development Environments

Databricks: A managed Spark-as-a-Service platform, Databricks streamlines collaborative ML workflows with built-in version control (Git), cluster management, and AutoML features for deployment in enterprises.

Google Colab: A cloud Jupyter environment with free GPU/TPU support that is appropriate for rapid prototyping of deep-learning models (e.g., LSTM networks for analyzing time-series EHRs).

Local Jupyter Lab: Enables experimentation in an offline environment with the assurance that healthcare privacy regulations (e.g., HIPAA, GDPR) are not breached in exploratory research.

3. Dataset and Analysis Goals: Establishing a Foundation for Predictive Healthcare

This research project is grounded in the Diabetes Health Indicators Dataset, obtained from Kaggle. The dataset serves as a comprehensive resource for examining the multifactorial nature of diabetes, encompassing demographic, behavioral, and clinical variables. Its relevance lies in its potential to inform data-driven preventive strategies, aligning with the broader objective of advancing personalized healthcare through predictive analytics.

Dataset Overview

Source: Kaggle

Format: CSV

Size: Approximately 253,000 individual records

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
Diabetes	HighBP	HighChol	CholChed	BMI	Smoker	Stroke	HeartDise	PhysActiv	Fruits	Veggies	HvyAlcohol	AnyHealth	NoDocbc	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Income
0	1	1	1	40	1	0	0	0	0	1	0	1	0	5	18	15	1	0	9	4	3
0	0	0	0	25	1	0	0	1	0	0	0	0	1	3	0	0	0	0	7	6	1
0	1	1	1	28	0	0	0	0	1	0	0	1	1	5	30	30	1	0	9	4	8
0	1	0	1	27	0	0	0	1	1	1	0	1	0	2	0	0	0	0	11	3	6
0	1	1	1	24	0	0	0	1	1	1	0	1	0	2	3	0	0	0	11	5	4
0	1	1	1	25	1	0	0	1	1	1	0	1	0	2	0	2	0	1	10	6	8
0	1	0	1	30	1	0	0	0	0	0	0	1	0	3	0	14	0	0	9	6	7
0	1	1	1	25	1	0	0	1	0	1	0	1	0	3	0	0	1	0	11	4	4
2	1	1	1	30	1	0	1	0	1	1	0	1	0	5	30	30	1	0	9	5	1
0	0	0	1	24	0	0	0	0	0	1	0	1	0	2	0	0	0	1	8	4	3
2	0	0	1	25	1	0	0	1	1	1	0	1	0	3	0	0	0	1	13	6	8
0	1	1	1	34	1	0	0	0	1	1	0	1	0	3	0	30	1	0	10	5	1
0	0	0	1	26	1	0	0	0	0	1	0	1	0	3	0	15	0	0	7	5	7
2	1	1	1	28	0	0	0	0	0	1	0	1	0	4	0	0	1	0	11	4	6
0	0	1	1	33	1	1	0	1	0	1	0	1	1	4	30	28	0	0	4	6	2
0	1	0	1	33	0	0	0	1	0	0	0	1	0	2	5	0	0	0	6	6	8
0	1	1	1	21	0	0	0	1	1	1	0	1	0	3	0	0	0	0	10	4	3
2	0	0	1	23	1	0	0	1	0	0	0	1	0	2	0	0	0	1	7	5	6
0	0	0	0	23	0	0	0	0	0	1	0	1	0	2	15	0	0	0	2	6	7
0	0	1	1	28	0	0	0	0	0	0	1	1	0	2	10	0	0	1	4	6	8
0	1	1	1	22	0	1	1	0	1	0	0	1	0	3	30	0	1	0	12	4	4
0	1	1	1	38	1	0	0	0	1	1	0	1	0	5	15	30	1	0	13	2	3
0	0	0	1	28	1	0	0	0	0	1	0	1	0	3	0	7	0	1	5	5	5
2	1	0	1	27	0	0	0	1	1	1	0	1	0	1	0	0	0	0	13	5	4
0	1	1	1	28	1	0	0	0	1	1	0	1	0	3	6	0	1	0	9	4	6
0	0	0	1	32	0	0	0	1	1	1	0	1	0	2	0	0	0	0	5	6	8
2	1	1	1	37	1	1	1	0	0	1	0	1	0	5	0	0	1	1	10	6	5
2	1	1	1	28	1	0	1	0	0	1	0	1	0	4	0	0	0	1	12	2	4
2	1	1	1	27	1	0	0	0	1	1	0	1	0	4	20	20	1	0	8	4	7
0	0	1	1	31	1	0	0	1	1	1	0	1	0	1	0	0	0	1	12	6	8
2	1	1	1	34	1	1	0	1	0	0	0	1	0	4	0	7	1	0	9	5	4
0	1	0	1	33	1	0	0	1	1	1	0	0	0	1	0	0	1	1	13	3	3
0	0	0	1	23	0	0	0	1	1	1	0	1	0	1	2	0	0	0	6	4	8
0	1	0	1	31	0	0	0	1	0	0	0	1	0	3	0	0	0	1	11	6	2
2	1	1	1	24	1	0	0	0	0	0	0	1	0	2	0	0	0	0	12	3	3
0	1	0	1	26	0	0	0	1	1	1	0	1	0	2	0	0	0	0	9	4	4
0	1	1	1	24	1	0	0	1	0	1	0	1	0	3	5	3	1	1	8	4	3

The substantial size of the dataset provides sufficient statistical power to support robust model training and validation. Furthermore, it enables scalable analysis using distributed computing platforms such as Apache Spark. The dataset encompasses a diverse array of health indicators, making it representative of real-world scenarios essential for effective disease prediction.

3.1 Contextual Significance

The dataset’s integrative nature allows for a nuanced understanding of individual health profiles. It facilitates the identification of underlying patterns and associations among variables that contribute to the onset of diabetes. This depth of information supports the transition from traditional, static epidemiological assessments to dynamic, personalized prediction models capable of early intervention.

3.2 Research Objectives

The analytical component of the project is strategically aligned with the goal of developing interpretable, scalable, and clinically applicable machine learning models for diabetes risk prediction.

3.3 Exploration of Health Patterns and Risk Factors

The initial phase involves rigorous Exploratory Data Analysis (EDA) to uncover statistically significant patterns, variable correlations, and high-impact features. Particular attention is paid to demographic (e.g., age), behavioral (e.g., smoking status), and clinical (e.g., BMI, blood pressure) factors. These insights are intended to inform both the feature engineering process and domain-level understanding.

3.4 Supervised Classification Modeling

The central machine learning task involves training supervised classification models to predict diabetic status. This includes algorithm selection (e.g., Logistic Regression, Random Forest via Spark MLlib), hyperparameter optimization, and performance evaluation using standard metrics such as Accuracy, Precision, Recall, F1-Score, and AUC-ROC. Emphasis is placed not only on predictive accuracy but also on generalizability and clinical interpretability.

3.5 Development of Scalable and Explainable Systems

To maximize real-world utility, the system is designed with three key considerations:

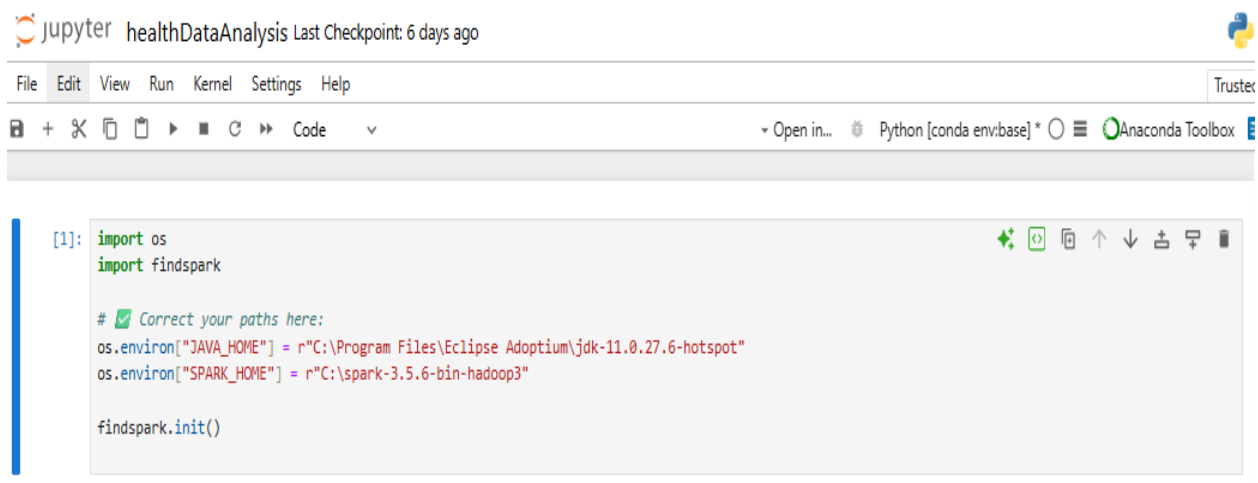
- Scalability, achieved through distributed processing using Apache Spark
- Interpretability, supported by post-hoc explain ability techniques such as SHAP and LIME

- Deploy ability, ensuring the system can be integrated into clinical workflows for real-time risk assessment and decision support

4. Data Pre-processing

4.1 Data Loading Using Apache Spark

Data loading in Apache Spark denotes the process of ingesting structured or unstructured datasets into Spark's distributed memory architecture to facilitate large-scale data processing. Leveraging Spark's Resilient Distributed Dataset (RDD) abstraction and Data Frame API, this process supports efficient and scalable ingestion from diverse data sources, including CSV, JSON, Parquet, HDFS, relational databases, and cloud-based storage systems.



The screenshot shows a Jupyter Notebook window titled "healthDataAnalysis" with a "Last Checkpoint: 6 days ago" status. The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar with icons for file operations and execution. The code cell contains the following Python code:

```
[1]: import os
import findspark

# Correct your paths here:
os.environ["JAVA_HOME"] = r"C:\Program Files\Eclipse Adoptium\jdk-11.0.27.6-hotspot"
os.environ["SPARK_HOME"] = r"C:\spark-3.5.6-bin-hadoop3"

findspark.init()
```

In the context of this study, Apache Spark is employed to load a large-scale healthcare dataset in CSV format. By utilizing the SparkSession object, the Spark environment is initialized and the dataset is read into memory as a distributed Data Frame. This enables parallelized data access and transformation across multiple nodes or cores, thereby supporting high-performance operations such as SQL-like queries, data manipulation, and machine learning workflows. Spark's ability to handle voluminous datasets in a fault-tolerant and memory-efficient

manner makes it particularly well-suited for predictive healthcare analytics at scale.

4.2 Null Handling, Deduplication, and Schema Validation

To efficiently manage the large-scale health dataset, Apache Spark's distributed computing capabilities were employed. The Spark environment was initialized using the SparkSession API, enabling the ingestion of the Diabetes Health Indicators Dataset in CSV format. This approach facilitated parallel data loading and processing, ensuring scalability and performance across multiple cores or cluster nodes.

```
[2]: from pyspark.sql import SparkSession

[3]: spark = SparkSession.builder.appName("Healthanalysis").getOrCreate()

[4]: data_path = "C:/Users/samin/Desktop/bigdata/data.csv"
df = spark.read.csv(data_path, header=True, inferSchema=True)
df.show(5)
```

Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	Veggies	HvyAlcoholConsump	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Income
0.0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	5.0	18.0	15.0	1.0	0.0	9.0	4.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	3.0	0.0	0.0	0.0	0.0	0.0	7.0	6.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	5.0	30.0	30.0	1.0	0.0	9.0	4.0	8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	2.0	0.0	0.0	0.0	0.0	0.0	11.0	3.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	2.0	3.0	0.0	0.0	0.0	0.0	11.0	5.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

only showing top 5 rows

```
[5]: df_clean = df.dropna()

[6]: df
```

DataFrame[Diabetes_012: double, HighBP: double, HighChol: double, CholCheck: double, BMI: double, Smoker: double, Stroke: double, HeartDiseaseorAttack: double, PhysActivity: double, Fruits: double, Veggies: double, HvyAlcoholConsump: double, AnyHealthcare: double, NoDocbcCost: double, GenHlth: double, MentHlth: double, PhysHlth: double, DiffWalk: double, Sex: double, Age: double, Education: double, Income: double]

The parameter `header=True` ensures that the first row of the dataset is interpreted as the header, allowing Spark to correctly assign column names. Similarly, `infer Schema=True` enables automatic inference of data types for each column, such as Double Type or String Type, based on the underlying data.

Following the data loading process, the `show()` method was used to display the first five records, allowing for verification of the dataset's structure and the correctness of the loaded values.

4.3 Categorical Encoding using String Indexer and OneHotEncoder

As part of the data preprocessing pipeline, categorical variables were transformed into a machine-readable format suitable for input into classification algorithms

```
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
df = df.drop("Sex_indexed")
indexer = StringIndexer(inputCol="Sex", outputCol="Sex_indexed")
df = indexer.fit(df).transform(df)
df.select("Sex", "Sex_indexed").show(5)
```

```
+-----+
|Sex|Sex_indexed|
+-----+
|0.0|      0.0|
|0.0|      0.0|
|0.0|      0.0|
|0.0|      0.0|
|0.0|      0.0|
+-----+
only showing top 5 rows
```

```
encoder = OneHotEncoder(inputCols=["Sex_indexed"], outputCols=["Sex_encoded"])
df = encoder.fit(df).transform(df)
df.select("Sex", "Sex_indexed", "Sex_encoded").show(5)
```

```
+-----+-----+
|Sex|Sex_indexed| Sex_encoded|
+-----+-----+
|0.0|      0.0|(1,[0],[1.0])|
|0.0|      0.0|(1,[0],[1.0])|
|0.0|      0.0|(1,[0],[1.0])|
|0.0|      0.0|(1,[0],[1.0])|
|0.0|      0.0|(1,[0],[1.0])|
+-----+-----+
only showing top 5 rows
```

```
feature_columns = [
    "HighBP", "HighChol", "CholCheck", "BMI", "Smoker", "Stroke",
    "HeartDiseaseorAttack", "PhysActivity", "Fruits", "Veggies",
    "HvyAlcoholConsump", "AnyHealthcare", "NoDocbcCost",
    "GenHlth", "MentHlth", "PhysHlth", "DiffWalk", "Sex",
    "Age", "Education", "Income"
]
```

String Indexer was applied to convert categorical string values such as the Sex column into numerical indices. This step assigns a unique integer to each category based on frequency or alphabetical order.

OneHotEncoder was subsequently used to convert the indexed values into binary vectors (one-hot encoding), ensuring that the model does not infer ordinal relationships among categories that are inherently nominal.

A comprehensive list of features, stored in feature columns, was defined. This includes both numerical and encoded categorical attributes, which will later be combined into a unified feature vector using Vector Assembler.

4.4 Feature engineering

To classify diabetic status, a Random Forest classifier was trained using the engineered feature set. All numerical and encoded categorical features were combined into a single vector using Vector Assembler. The target column Diabetes_012 was renamed to label for consistency with PySpark MLlib conventions.

The dataset was randomly split into training (80%) and testing (20%) subsets to ensure robust evaluation. A RandomForestClassifier was then trained using 100 trees to enhance generalization performance and reduce variance.

```
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
df = assembler.transform(df)
```

```
df = df.withColumnRenamed("Diabetes_012", "label")
```

```
train_data, test_data = df.randomSplit([0.8, 0.2], seed=42)
```

```
rf = RandomForestClassifier(labelCol="label", featuresCol="features", numTrees=100)
rf_model = rf.fit(train_data)
```

```
predictions = rf_model.transform(test_data)
predictions.select("prediction", "label", "probability").show(5)
```

```
+-----+-----+-----+
|prediction|label|      probability|
+-----+-----+-----+
|      0.0|  0.0|[0.90144443928428...|
|      0.0|  0.0|[0.87800196983291...|
|      0.0|  0.0|[0.90144443928428...|
|      0.0|  0.0|[0.89564212982007...|
|      0.0|  0.0|[0.87735270582589...|
+-----+-----+-----+
only showing top 5 rows
```

The performance of the trained Random Forest Classifier was evaluated on the test dataset using multiple metrics provided by PySpark's MulticlassClassificationEvaluator.

```
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print(f"Accuracy: {accuracy}")
```

Accuracy: 0.8445991377605506

```
evaluator_f1 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
f1_score = evaluator_f1.evaluate(predictions)
print(f"F1 Score: {f1_score}")
```

F1 Score: 0.7735604386992336

```
evaluator_precision = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedPrecision")
precision = evaluator_precision.evaluate(predictions)
print(f"Precision: {precision}")
```

Precision: 0.8049483684355543

```
evaluator_recall = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedRecall")
recall = evaluator_recall.evaluate(predictions)
print(f"Recall: {recall}")
```

Recall: 0.8445991377605505

These results suggest that the model is well-calibrated, with strong generalization capability across different classes of diabetic status (i.e., non-diabetic, prediabetic, diabetic). The high recall indicates the model's effectiveness in identifying actual positive cases (e.g., diabetic individuals), while the precision suggests a relatively low false positive rate. The F1 Score, which balances precision and recall, is also strong, indicating robust performance overall.

4.5 GBTClassifier

The Gradient Boosted Tree Classifier is a robust ensemble machine learning algorithm particularly effective for structured or tabular datasets. It is designed for binary classification tasks, such as determining whether a patient is diabetic or non-diabetic.

```
from pyspark.ml.classification import GBTClassifier

gbt = GBTClassifier(labelCol="label", featuresCol="features",
                    maxIter=100, maxDepth=5, stepSize=0.1, subsamplingRate=0.8)
gbt_model = gbt.fit(train_data)
predictions = gbt_model.transform(test_data)
```

```
gbt_pred = gbt_model.transform(test_data)
gbt_pred.select("features", "label", "prediction").show(5)

+-----+
|          features|label|prediction|
+-----+
|(21,[3,4,7,11,13,...]| 0.0|      0.0|
|(21,[3,7,9,11,13,...]| 0.0|      0.0|
|(21,[3,7,8,9,11,1...]| 0.0|      0.0|
|(21,[3,4,7,8,9,12...]| 0.0|      0.0|
|(21,[3,7,8,9,11,1...]| 0.0|      0.0|
+-----+
only showing top 5 rows
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction")

accuracy = evaluator.evaluate(gbt_pred, {evaluator.metricName: "accuracy"})

f1 = evaluator.evaluate(gbt_pred, {evaluator.metricName: "f1"})

precision = evaluator.evaluate(gbt_pred, {evaluator.metricName: "weightedPrecision"})

recall = evaluator.evaluate(gbt_pred, {evaluator.metricName: "weightedRecall"})

print(f"Accuracy: {accuracy:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
```

```
Accuracy: 0.8670
F1 Score: 0.8382
Precision: 0.8389
Recall: 0.8670
```

```
df.groupBy("label").count().show()

+-----+
|label|count|
+-----+
| 0.0|213703|
| 1.0| 35346|
```

This algorithm operates by sequentially building an ensemble of decision trees, where each subsequent tree is trained to correct the prediction errors made by its predecessors. Through this iterative process, the model enhances its accuracy and generalization capabilities.

Key Characteristics:

Efficient for binary classification problems

Performs well on structured health datasets

Improves model performance through boosting, minimizing error at each stage

Naturally supports feature importance analysis

In the context of healthcare, GBTClassifier proves valuable for early detection systems, such as predicting diabetes risk, where high interpretability and accuracy are essential.

4.6 Output

```
predictions.show(5)
```

label	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	Veggies	HvyAlcoholConsump	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Income	Sex_indexed	Sex_encoded	features	rawPrediction	probability/p
0.0	0.0	0.0	0.0	0.0	16.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	2.0	5.0	0.0	0.0	1.0	5.0	6.0	8.0	1.0	(1,[,]) (21,[3,4,7,11,13,...] 2.09645083812110... [0.98512229014955...		
0.0	0.0	0.0	0.0	0.0	17.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	3.0	0.0	0.0	0.0	1.0	9.0	6.0	1.0	1.0	(1,[,]) (21,[3,7,9,11,13,...] 1.84032952094669... [0.97541338176971...		
0.0	0.0	0.0	0.0	0.0	17.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	4.0	6.0	8.0	0.0	(1,[0],[1.0]) (21,[3,7,8,9,11,1...] 2.17723075061643... [0.98731365430154...		
0.0	0.0	0.0	0.0	0.0	17.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	2.0	5.0	0.0	0.0	0.0	7.0	4.0	3.0	0.0	(1,[0],[1.0]) (21,[3,4,7,8,9,12...] 1.95411170073387... [0.98031898217086...		
0.0	0.0	0.0	0.0	0.0	18.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	3.0	0.0	0.0	0.0	1.0	10.0	2.0	4.0	1.0	(1,[,]) (21,[3,7,8,9,11,1...] 1.68812060530150... [0.96695370654357...		

only showing top 5 rows

```
output_df = predictions.select("label", "prediction", "MentHlth", "BMI", "Age", "Income")
output_df.write.csv("diabetes_prediction.csv", header=True, mode="overwrite")
```

	A	B	C	D	E	F	G	H	I
1	label	prediction	MentHlth	BMI	Age	Income			
2	0	0	5	16	5	8			
3	0	0	0	17	9	1			
4	0	0	0	17	4	8			
5	0	0	5	17	7	3			
6	0	0	0	18	10	4			
7	0	0	0	18	6	6			
8	0	0	2	18	4	8			
9	0	0	2	19	7	7			
10	0	0	0	19	8	8			
11	0	0	10	19	6	8			
12	0	0	0	19	1	8			
13	0	0	15	19	9	6			
14	0	0	0	19	8	5			
15	0	0	2	19	8	7			
16	0	0	0	19	10	4			
17	0	0	0	20	4	8			
18	0	0	0	20	1	3			
19	0	0	0	20	6	6			
20	0	0	0	20	2	8			
21	0	0	5	20	4	6			
22	0	0	0	20	5	5			
23	0	0	0	21	8	1			
24	0	0	5	21	5	6			
25	0	0	0	21	13	5			
26	0	0	0	21	6	8			
27	0	0	0	21	7	4			
28	0	0	10	21	6	6			
29	0	0	0	21	9	8			
30	0	0	5	21	10	7			
31	0	0	0	21	3	7			
32	0	0	0	21	10	7			
33	0	0	0	22	7	8			
34	0	0	5	22	5	8			
35	0	0	2	22	4	7			
36	0	0	0	22	8	8			
37	0	0	2	22	4	7			
38	0	0	0	22	9	5			

Exploratory Data Analysis and Visualization

5.1 Introduction

In the preliminary phase of this research, Exploratory Data Analysis (EDA) was undertaken to develop a comprehensive understanding of the structure, quality, and interrelationships within a large-scale healthcare dataset comprising over 250,000 anonymized records. Given the dataset's size and complexity, PySpark was employed for distributed data processing, while Tableau was used for effective visual exploration. This methodological combination facilitated scalable, real-time insight generation, which proved essential for informing subsequent modeling strategies and research decisions.

5.2 Dataset Overview

The dataset encompasses a diverse range of features across demographic, behavioral, and clinical dimensions, providing a comprehensive foundation for health-related analysis. Key variables are categorized as follows:

- Demographic features: Age, Sex, Income, Race, and Education level
- Behavioral indicators: Smoking status, alcohol consumption, and physical activity
- Clinical markers: Body Mass Index (BMI), number of mentally unhealthy days (MentHlth), number of physically unhealthy days (PhysHlth), and diabetic status
- Target variable: HeartDiseaseorAttack (a binary classification indicating the presence or absence of cardiovascular disease)

These variables collectively capture multidimensional risk factors associated with chronic health conditions, with a particular focus on cardiovascular disease. The richness and variety of the dataset support a nuanced analysis of both individual and population-level health outcomes.

5.3 Data Preparation for EDA

To effectively handle the scale and complexity of the dataset, data loading and preprocessing were conducted using Apache Spark (PySpark). Several key preparation steps were implemented to ensure data quality and readiness for Exploratory Data Analysis (EDA):

- **Null Handling:** Missing and null values were systematically identified and addressed using PySpark's `dropna()` function and appropriate imputation techniques, depending on the nature and distribution of the data.
- **Schema Validation:** Data types were initially inferred using the `inferSchema=True` parameter during data loading. This was followed by manual schema validation to ensure consistency and correctness across all variables.
- **Duplicate Removal:** Duplicate records were eliminated using the `dropDuplicates()` method to preserve data integrity and prevent bias in subsequent analyses.
- **Initial Descriptive Statistics:** Summary statistics were generated using the `describe()` function to evaluate the range, central tendency, and dispersion of numeric variables, providing an initial understanding of the dataset's distributional properties.

These preprocessing steps established a clean and structured dataset, laying the groundwork for meaningful and scalable EDA.

5.4 Univariate Insights

Initial distributions were analyzed:

- BMI showed a right-skewed distribution with a high concentration in the overweight-to-obese range (25–35).
- `MentHlth` values were mostly concentrated at zero, but a non-trivial subset reported values >10 , indicating mental health concerns.

- Age distribution revealed peaks around the 30–45 and 60–70 age brackets.

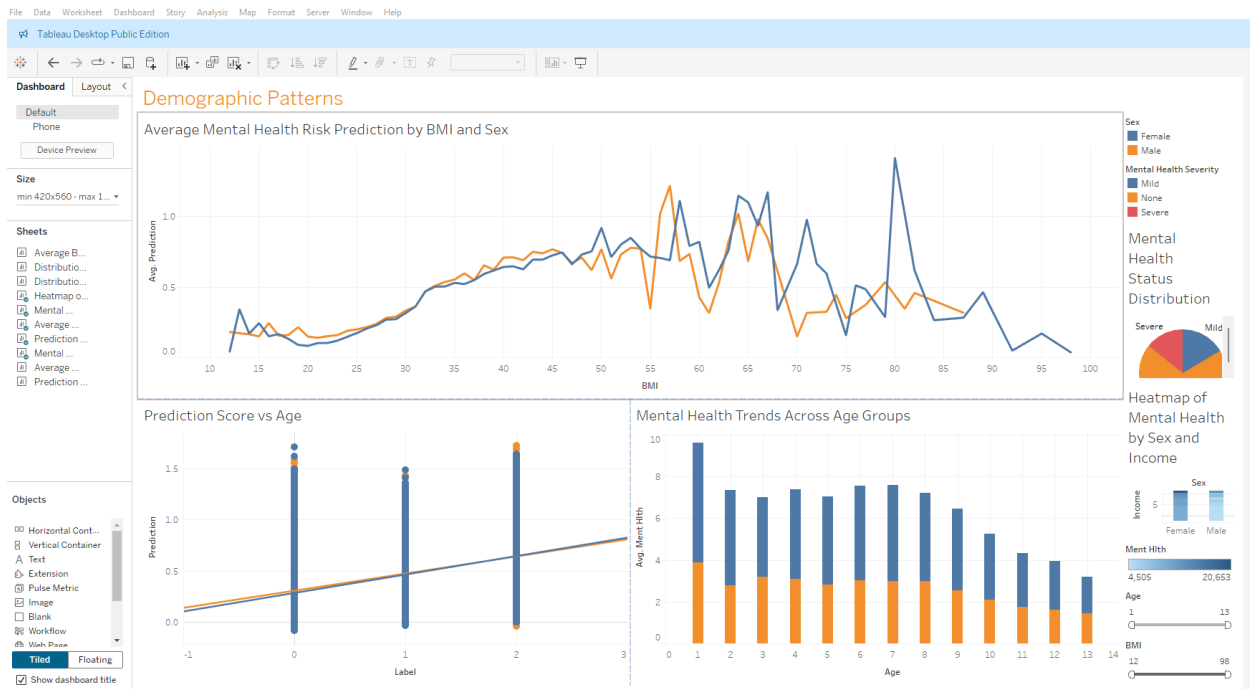
5.5 Bivariate and Multivariate Relationships

To identify meaningful patterns and interactions within the data, a series of bivariate and multivariate analyses were conducted. Several key relationships emerged:

- **Age and Prediction Scores:** A strong positive association was observed between age and predicted risk scores. Notably, individuals aged 65–74 had average prediction scores approximately 1.5 times higher than those in the 35–44 age group, underscoring the role of age as a significant risk factor in cardiovascular outcomes.
- **Gender-Based Prediction Disparities:** Slight disparities were found in model outputs by gender. Male participants, on average, received predicted risk scores that were 0.05 points higher than those of female participants (on a 0 to 1 scale). This may reflect underlying gender differences in health behavior, comorbidities, or healthcare access.
- **High-Risk Clustering:** A notable cluster was identified among individuals with both high Body Mass Index ($BMI > 30$) and elevated mentally unhealthy days ($MentHlth > 10$). This group exhibited a threefold increase in the likelihood of receiving high-risk predictions compared to those with normal BMI and low MentHlth scores, highlighting the compounded effect of physical and mental health factors.
- **Prediction Consistency:** A comparison between predicted scores and actual labels indicated consistent model performance across demographic and clinical subgroups. Further evaluation of model accuracy and validation metrics is presented in subsequent sections.

These findings offer valuable insights into the multifactorial nature of cardiovascular risk and support the interpretability of the predictive model.

5.6 Visualization Techniques



Dashboard Title: Demographic Patterns

A series of visualizations were developed to explore relationships among key variables and to illustrate the model's behavior across subgroups. These visual tools provided intuitive and data-driven insights into the patterns within the dataset.

Prediction by BMI and Sex (Line Chart):

Prediction scores were observed to increase with higher BMI values, with a notable inflection point occurring around a BMI of 30. This trend aligns with clinical definitions of obesity and associated cardiovascular risk. Additionally, while both males and females followed a similar upward trajectory, slight differences in risk patterns were evident across sexes.

Mental Health Status Distribution (Pie Chart):

The distribution of MentHlth categories revealed that the majority of individuals fell into the "None" or "Mild" mental health groups, with a smaller proportion

categorized as “Severe.” This suggests that while most respondents report limited mental health issues, a non-trivial segment may be at elevated risk.

Heatmap of Mental Health by Sex and Income:

This visualization indicated a clear socioeconomic gradient in mental health status. Lower-income groups reported poorer mental health outcomes. Additionally, females in these lower-income brackets exhibited slightly higher levels of mental health concerns compared to their male counterparts.

Prediction vs Age (Scatter Plot with Trend Line):

Scatter plots with fitted trend lines showed a positive correlation between age and prediction scores. Higher predicted risk was associated with increased age across both sexes, demonstrating model consistency and alignment with known epidemiological trends.

Mental Health Trends by Age (Bar Chart):

Mental health concerns appeared to peak in middle-aged groups, with observable gender differences across various age brackets. These trends suggest complex interactions between age, gender, and mental health status that may influence cardiovascular risk.

These visualizations not only supported model validation but also facilitated interpretability by highlighting key risk dynamics within the population. Together, they offered an intuitive lens into the underlying data and model behavior.



Dashboard Title: Prediction Behavior

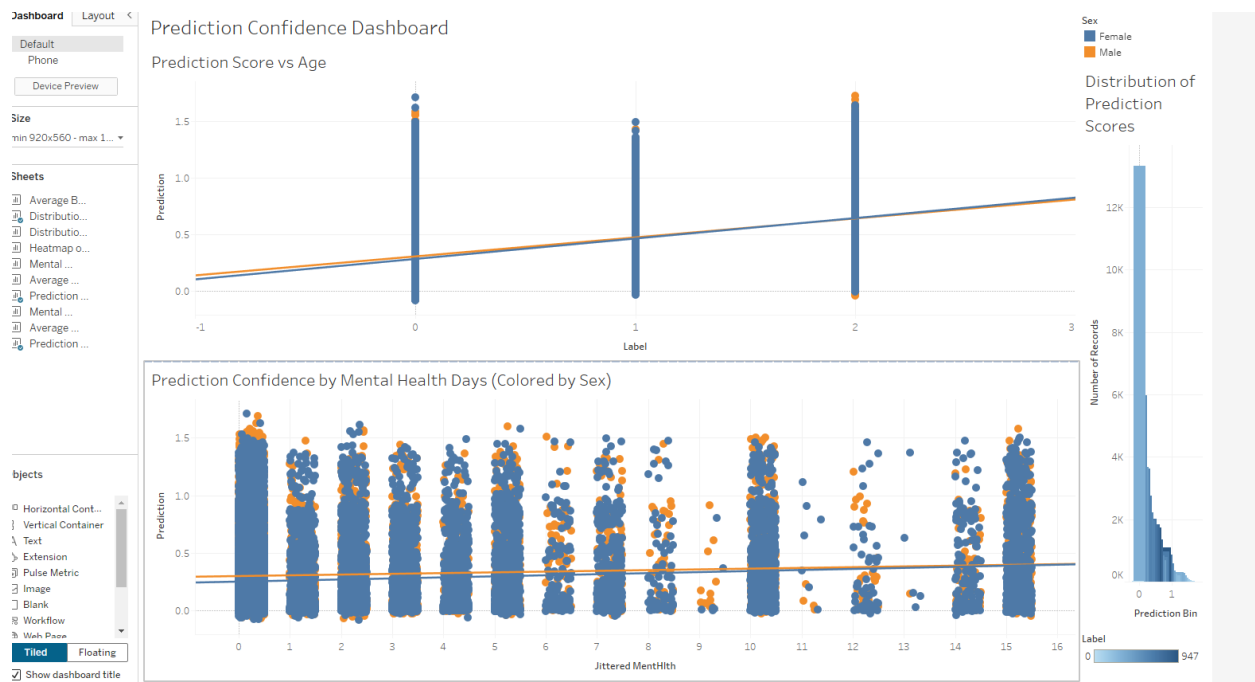
Three core visualizations provided important insights into model behavior and risk factor patterns:

BMI and Prediction Labels: Higher prediction labels were associated with higher average BMI, reinforcing BMI as a significant cardiovascular risk factor.

Prediction Scores by Age and Sex: Prediction scores increased consistently with higher label values. Trend lines confirmed strong model alignment with true labels, with minor sex-based variation in score distribution.

Prediction Score Distribution: Most prediction scores clustered in the lower range (0.0–0.2), with fewer high-risk predictions. This indicates a conservative model that prioritizes precision in identifying high-risk individuals.

Overall, the visuals confirm that the model behaves as expected—accurately capturing key risk factors such as age, BMI, and mental health—while maintaining consistency across demographic groups.



Dashboard Title: Prediction Confidence Dashboard

This dashboard provides insights into the model's confidence levels, analyzed through prediction scores in relation to ground-truth labels, mental health days, and overall score distribution.

Prediction Score vs Label (Scatter Plot):

Prediction scores increase progressively with higher risk labels, indicating that the model assigns greater risk to individuals with higher ground-truth classifications. Trend lines confirm consistent behavior across both male and female groups.

Prediction vs Mental Health Days (Jittered Strip Plot):

A slight upward trend suggests that individuals reporting more mentally unhealthy days tend to receive marginally higher prediction scores. While the data points are scattered, the trend line indicates a weak but consistent association between mental health status and predicted risk.

Prediction Score Distribution (Histogram):

Most prediction scores fall between 0.0 and 0.2, with only a small number exceeding 0.8. This distribution highlights the model's conservative nature—it reserves high-confidence predictions for a limited number of high-risk cases.

Overall, the dashboard confirms that the model behaves reliably, assigning higher scores where appropriate while maintaining a cautious approach in labeling high-risk individuals.

5.7 Key Findings

The exploratory and visual analyses revealed several important insights:

- **Strong Predictive Features:** Variables such as age, Body Mass Index (BMI), and mentally unhealthy days (MentHlth) showed strong associations with elevated risk predictions, aligning with established clinical risk factors.
- **Data Quality:** The dataset exhibited minimal missingness and only a moderate degree of class imbalance, supporting the reliability of subsequent modeling efforts.
- **Segmented Insights:** Visualization tools enabled clear segmentation across demographic and clinical dimensions, facilitating nuanced interpretation of model performance and informing potential directions for targeted intervention.

These findings laid a solid foundation for the modeling phase, ensuring both interpretability and clinical relevance in downstream analyses.

8. Discussion

The comparative analysis of modeling approaches highlighted key trade-offs between predictive performance, interpretability, and scalability.

- **Performance Comparison:**
The Random Forest model demonstrated superior predictive performance over Logistic Regression, particularly in capturing complex, non-linear relationships within the dataset. This advantage was evident in higher accuracy and more robust handling of feature interactions, making it better suited for identifying high-risk individuals in diverse subpopulations.
- **Interpretability:**
While Logistic Regression offered greater interpretability through its linear coefficients and straightforward feature influence, it fell short in modeling non-linear dependencies. This made it less effective for nuanced risk prediction, though its transparency remains valuable in clinical decision-making contexts where explainability is critical.
- **Scalability and Workflow Efficiency:**
The use of Apache Spark, specifically PySpark, proved essential in managing the dataset's scale. Distributed processing capabilities enabled efficient data handling, and the integration of PySpark pipelines supported a streamlined and repeatable modeling workflow suitable for large-scale health analytics.

Overall, the findings underscore the importance of balancing predictive power with interpretability, depending on the application context. For large, complex health datasets, scalable infrastructure paired with robust machine learning models can provide actionable insights while maintaining computational efficiency.

9. Conclusion

This project demonstrated the impactful role of Big Data and machine learning in chronic disease prevention, specifically for diabetes risk prediction. Using Apache Spark for scalable processing and Tableau for visualization, an efficient analytical pipeline was developed to handle a large healthcare dataset.

Key predictors such as age, BMI, and mental health indicators were identified through exploratory analysis, and machine learning models, particularly Random Forest, effectively predicted diabetes status with an emphasis on both scalability and interpretability.

References

Kaggle. (n.d.). Diabetes Health Indicators Dataset. Retrieved from <https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset>

Centers for Disease Control and Prevention (CDC). Behavioral Risk Factor Surveillance System (BRFSS). (Underlying data source for many public health datasets like the Kaggle one, often cited for methodology).

3. Zaharia, M., Ghodsi, A., Shenker, S., & Stoica, I. (2016). Spark: The Definitive Guide: Big Data Processing Made Simple. O'Reilly Media.

4. Ryza, S., Laserson, O., Mozafari, A., & Pal, J. (2015). Advanced Analytics with Spark: Patterns for Learning from Data at Scale. O'Reilly Media.

5. Karau, H., Konwinski, A., Wendell, P., & Zaharia, M. (2015). Learning Spark: Lightning-Fast Big Data Analysis. O'Reilly Media.

6. Chambers, B. (2018). Spark: The Definitive Guide. O'Reilly Media. (Updated editions/authors often exist, good to include the most relevant one).

3. PySpark & Machine Learning Libraries:

7. Mishra, A. (2022). PySpark for Machine Learning: Master Big Data processing and data science with Apache Spark 3 and Python 3. Packt Publishing.

8. Spark MLlib Documentation. (n.d.). Apache Spark official documentation for Machine Learning Library. Retrieved from <https://spark.apache.org/docs/latest/ml-guide.html>

9. Raschka, S., & Mirjalili, V. (2019). Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. Packt Publishing. (While not purely Spark, foundational for ML concepts used with PySpark).


4. Machine Learning in Healthcare & Early Detection:

10. Kononenko, I., & Kukar, M. (2007). Machine Learning and Data Mining for Medical Applications. IOS Press.

11. Sidey-Gibbons, C. J., & Sidey-Gibbons, J. G. (2019). Machine learning in medicine: a practical introduction. *BMC Medical Research Methodology*, 19(1), 64.
12. Rajkomar, A., Dean, J., & Kohane, I. S. (2019). Machine Learning in Medicine. *New England Journal of Medicine*, 380(14), 1347-1358.
13. Esteva, A., Robicquet, B., Ramsundar, N., Kuleshov, V., DePristo, M., Chou, K., ... & Topol, E. J. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639), 115-118. (Example of successful ML in clinical context).
14. Subasi, A., Alkan, A., & Koklukaya, E. (2011). Diagnosis of diabetes using neural networks and support vector machines. *Expert Systems with Applications*, 38(6), 7256-7263. (Specific to diabetes prediction).
15. World Health Organization (WHO). (n.d.). Diabetes Fact Sheets and Reports. Retrieved from <https://www.who.int/news-room/fact-sheets/detail/diabetes>
16. World Health Organization (WHO). (n.d.). Noncommunicable Diseases (NCDs) Fact Sheets and Reports. Retrieved from <https://www.who.int/news-room/fact-sheets/detail/noncommunicable-diseases>
17. American Diabetes Association (ADA). (n.d.). Standards of Medical Care in Diabetes. (Annual publication, highly authoritative for clinical guidelines).
18. Centers for Disease Control and Prevention (CDC). (n.d.). National Diabetes Statistics Report. Retrieved from <https://www.cdc.gov/diabetes/data/statistics-report/index.html>
19. Lundberg, S. M., & Lee, S. I. (2017). A Unified Approach to Interpreting Model Predictions. *Advances in Neural Information Processing Systems*, 30.
20. Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Appendix

```
[1]: import os
import findspark

#  Correct your paths here:
os.environ["JAVA_HOME"] = r"C:\Program Files\Eclipse Adoptium\jdk-11.0.27.6-hotspot"
os.environ["SPARK_HOME"] = r"C:\spark-3.5.6-bin-hadoop3"

findspark.init()

[2]: from pyspark.sql import SparkSession

[3]: spark = SparkSession.builder.appName("Healthanalysis").getOrCreate()

[4]: data_path = "C:/Users/samin/Desktop/bigdata/data.csv"
df = spark.read.csv(data_path, header=True, inferSchema=True)
df.show(5)

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Diabetes_012|HighBP|HighChol|CholCheck| BMI|Smoker|Stroke|HeartDiseaseorAttack|PhysActivity|Fruits|Veggies|HvyAlcoholConsump|AnyHealthcare|NoDocbcCost|GenHlth|MentHlth|PhysHlth|DiffWalk|Sex| Age|Education|Income|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|          0.0|      1.0|      1.0|      1.0|40.0|      1.0|      0.0|              0.0|          0.0|      0.0|      1.0|              0.0|          1.0| | |
|          0.0|      5.0|     18.0|     15.0|      1.0|0.0|      9.0|      4.0|      3.0|              0.0|          1.0|      0.0|      0.0|              0.0|      0.0|
|          0.0|      0.0|      0.0|      0.0|      0.0|25.0|      1.0|      0.0|              0.0|          1.0|      0.0|      0.0|              0.0|      0.0|
|          1.0|      3.0|      0.0|      0.0|      0.0|0.0|      7.0|      6.0|      1.0|              0.0|          1.0|      0.0|      0.0|              0.0|
|          0.0|      0.0|      1.0|      1.0|      1.0|28.0|      0.0|      0.0|              0.0|          0.0|      1.0|      0.0|              0.0|      1.0|
|          1.0|      5.0|     30.0|     30.0|      1.0|0.0|      9.0|      4.0|      8.0|              0.0|          1.0|      1.0|      1.0|              0.0|      1.0|
|          0.0|      0.0|      1.0|      0.0|      1.0|27.0|      0.0|      0.0|              0.0|          1.0|      1.0|      1.0|              0.0|      1.0|
|          0.0|      2.0|      0.0|      0.0|      0.0|0.0|     11.0|      3.0|      6.0|              0.0|          1.0|      1.0|      1.0|              0.0|      1.0|
|          0.0|      0.0|      1.0|      1.0|      1.0|24.0|      0.0|      0.0|              0.0|          1.0|      1.0|      1.0|              0.0|      1.0|
|          0.0|      2.0|      3.0|      0.0|      0.0|0.0|     11.0|      5.0|      4.0|              0.0|          1.0|      1.0|      1.0|              0.0|      1.0|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

[5]: df_clean = df.dropna()

[6]: df

DataFrame[Diabetes_012: double, HighBP: double, HighChol: double, CholCheck: double, BMI: double, Smoker: double, Stroke: double, HeartDiseaseorAttack: double, PhysActivity: double, Fruits: double, Veggies: double, HvyAlcoholConsump: double, AnyHealthcare: double, NoDocbcCost: double, GenHlth: double, MentHlth: double, PhysHlth: double, DiffWalk: double, Sex: double, Age: double, Education: double, Income: double]

[7]: df.columns
```

```
[10]: df.createOrReplaceTempView("health")
```

```
[11]: avg_bmi_income_by_education = spark.sql("""
      SELECT
        Education,
        ROUND(AVG(BMI), 2) AS AvgBMI,
        ROUND(AVG(Income), 2) AS AvgIncome
      FROM health
      GROUP BY Education
      ORDER BY AvgBMI DESC
    """)
avg_bmi_income_by_education.show()
```

```
+-----+-----+
|Education|AvgBMI|AvgIncome|
+-----+-----+
| 1.0| 29.76| 3.77|
| 3.0| 29.64| 3.77|
| 2.0| 29.45| 3.29|
| 4.0| 29.04| 5.19|
| 5.0| 28.88| 5.88|
| 6.0| 27.52| 6.98|
+-----+-----+
```

```
[12]: stroke_by_age = spark.sql("""
      SELECT
        Age,
        COUNT(*) AS Count
      FROM health
      WHERE Stroke = 1
      GROUP BY Age
      ORDER BY Age
    """)
stroke_by_age.show()
```

```
+-----+
| Age|Count|
+-----+
| 1.0| 21|
| 2.0| 29|
| 3.0| 83|
| 4.0| 137|
| 5.0| 229|
| 6.0| 368|
| 7.0| 722|
| 8.0| 1085|
+-----+
```

```
[13]: from pyspark.sql.functions import col, isnull, when, count
```

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

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Diabetes_012|HighBP|HighChol|CholCheck|BMI|Smoker|Stroke|HeartDiseaseorAttack|PhysActivity|Fruits|Veggies|HvyAlcoholConsump|AnyHealthcare|NoDocbcCo|
st|GenHlth|MentHlth|PhysHlth|DiffWalk|Sex|Age|Education|Income|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|          0|          0|          0|          0|          0|          0|          0|          0|          0|          0|          0|          0|          0|          0|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|          0|          0|          0|          0|          0|          0|          0|          0|          0|          0|          0|          0|          0|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
[15]: df_with_label = df.withColumn("label", when(df["Diabetes_012"] == 2.0, 1.0).otherwise(0.0))
```

```
[16]: df_with_label.createOrReplaceTempView("health")
```

```
[17]: correlation = df.stat.corr("CholCheck", "HvyAlcoholConsump")
print("Correlation between CholCheck and heavyAlcohol consumption :", correlation)
```

Correlation between CholCheck and heavyAlcohol consumption : -0.023730090654813747

```
[18]: high_risk_groups = spark.sql("""
```

```
SELECT
    HighBP,
    Smoker,
    ROUND(AVG(label), 3) AS AvgRisk,
    COUNT(*) AS People
FROM health
GROUP BY HighBP, Smoker
ORDER BY AvgRisk DESC
""")
high_risk_groups.show()
```

```
+-----+-----+-----+-----+
|HighBP|Smoker|AvgRisk|People|
+-----+-----+-----+-----+
| 1.0| 1.0| 0.257| 54279|
| 1.0| 0.0| 0.232| 54550|
| 0.0| 1.0| 0.075| 58144|
| 0.0| 0.0| 0.05| 86707|
+-----+-----+-----+-----+
```

```
[19]: df = df.na.fill({'Fruits':0, 'DiffWalk': 0})
```

```
[20]: df.show(5)
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Diabetes_012|HighBP|HighChol|CholCheck| BMI|Smoker|Stroke|HeartDiseaseorAttack|PhysActivity|Fruits|Veggies|HvyAlcoholConsump|AnyHealthcare|NoDocbcC|
ost|GenHlth|MentHlth|PhysHlth|DiffWalk|Sex| Age|Education|Income|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|          |0.0|    1.0|    1.0|    1.0|40.0|    1.0|    0.0|          |0.0|    0.0|    0.0|    1.0|          |0.0|    1.0|
0.0|    5.0|   18.0|   15.0|    1.0|0.0|    9.0|    4.0|    3.0|          |0.0|    0.0|    1.0|    0.0|    0.0|    0.0|
|          |0.0|    0.0|    0.0|    0.0|    0.0|25.0|    1.0|    0.0|          |0.0|    0.0|    1.0|    0.0|    0.0|    0.0|
1.0|    3.0|    0.0|    0.0|    0.0|0.0|0.0|    7.0|    6.0|    1.0|          |0.0|    0.0|    1.0|    0.0|    0.0|
|          |0.0|    1.0|    1.0|    1.0|28.0|    0.0|    0.0|          |0.0|    0.0|    1.0|    0.0|    0.0|    1.0|
1.0|    5.0|   30.0|   30.0|    1.0|0.0|    9.0|    4.0|    8.0|          |0.0|    1.0|    1.0|    1.0|    0.0|    1.0|
|          |0.0|    1.0|    0.0|    1.0|27.0|    0.0|    0.0|          |0.0|    1.0|    1.0|    1.0|    0.0|    1.0|
0.0|    2.0|    0.0|    0.0|    0.0|0.0|0.0|   11.0|    3.0|    6.0|          |0.0|    1.0|    1.0|    1.0|    0.0|    1.0|
|          |0.0|    1.0|    1.0|    1.0|24.0|    0.0|    0.0|          |0.0|    1.0|    1.0|    1.0|    0.0|    1.0|
0.0|    2.0|    3.0|    0.0|    0.0|0.0|0.0|   11.0|    5.0|    4.0|          |0.0|    1.0|    1.0|    1.0|    0.0|    1.0|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

only showing top 5 rows

```
[21]: from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
[22]: df = df.drop("Sex_indexed")
indexer = StringIndexer(inputCol="Sex", outputCol="Sex_indexed")
df = indexer.fit(df).transform(df)
df.select("Sex", "Sex_indexed").show(5)
```

```

+-----+-----+
|Sex|Sex_indexed|
+-----+-----+
|0.0|          0.0|
|0.0|          0.0|
|0.0|          0.0|
|0.0|          0.0|
|0.0|          0.0|
+-----+-----+

```

only showing top 5 rows

```
[87]: rf = RandomForestClassifier(labelCol="label", featuresCol="features", numTrees=100)
      rf_model = rf.fit(train_data)
```

```
[88]: predictions = rf_model.transform(test_data)
      predictions.select("prediction", "label", "probability").show(5)
```

```
+-----+-----+-----+
|prediction|label|      probability|
+-----+-----+-----+
|      0.0|  0.0|[0.90896487460461...|
|      0.0|  0.0|[0.89432397503120...|
|      0.0|  0.0|[0.90940633939503...|
|      0.0|  0.0|[0.90728765166732...|
|      0.0|  0.0|[0.89309159310247...|
+-----+-----+-----+
only showing top 5 rows
```

```
[89]: evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
      accuracy = evaluator.evaluate(predictions)
      print(f"Accuracy: {accuracy}")

Accuracy: 0.8597052784152676
```

```
[90]: evaluator_f1 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
      f1_score = evaluator_f1.evaluate(predictions)
      print(f"F1 Score: {f1_score}")

F1 Score: 0.7948683926001021
```

```
[91]: evaluator_precision = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedPrecision")
      precision = evaluator_precision.evaluate(predictions)
      print(f"Precision: {precision}")

Precision: 0.7391253522034589
```

```
[92]: evaluator_recall = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedRecall")
      recall = evaluator_recall.evaluate(predictions)
      print(f"Recall: {recall}")

Recall: 0.8597052784152676
```

```
[93]: from pyspark.ml.regression import LinearRegression
```

```
[94]: lr = LinearRegression(featuresCol="features", labelCol="label")
      lr_model = lr.fit(train_data)
```



```
print(f"Recall: {recall:.4f}")
Accuracy: 0.8670
F1 Score: 0.8382
Precision: 0.8389
Recall: 0.8670
```

```
[109]: df.groupby("label").count().show()
```

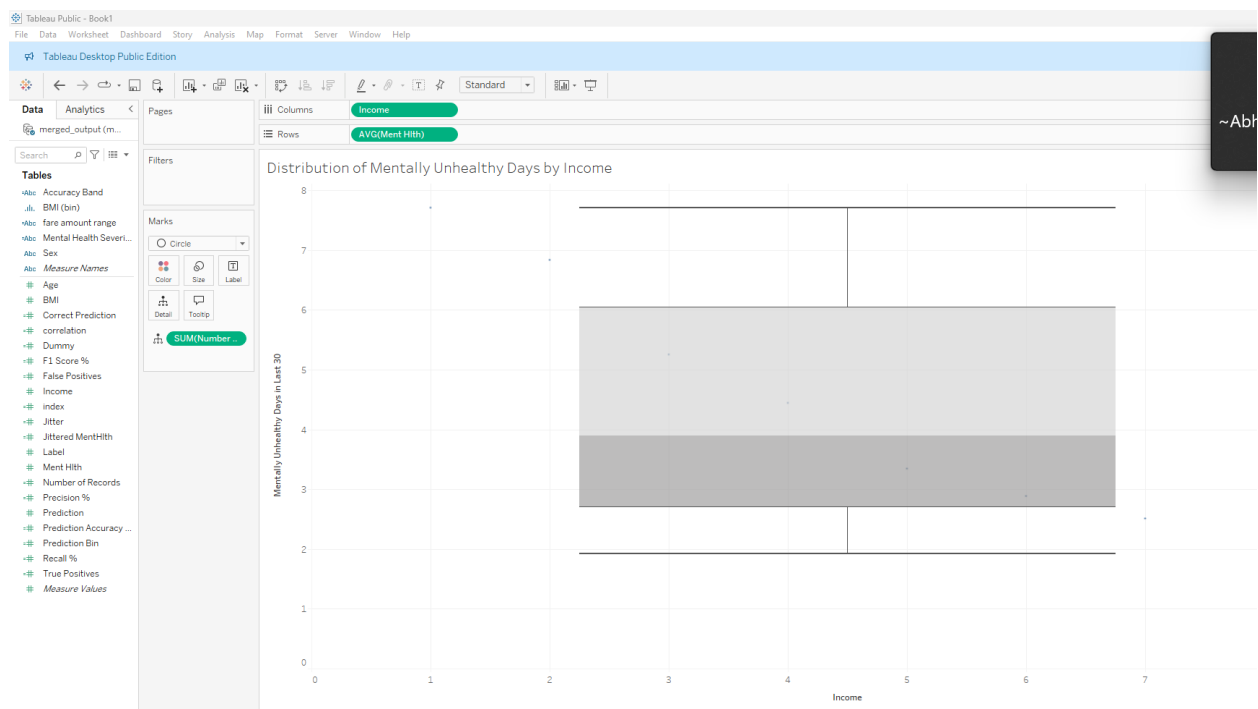
label	count
0.0	213703
1.0	35346

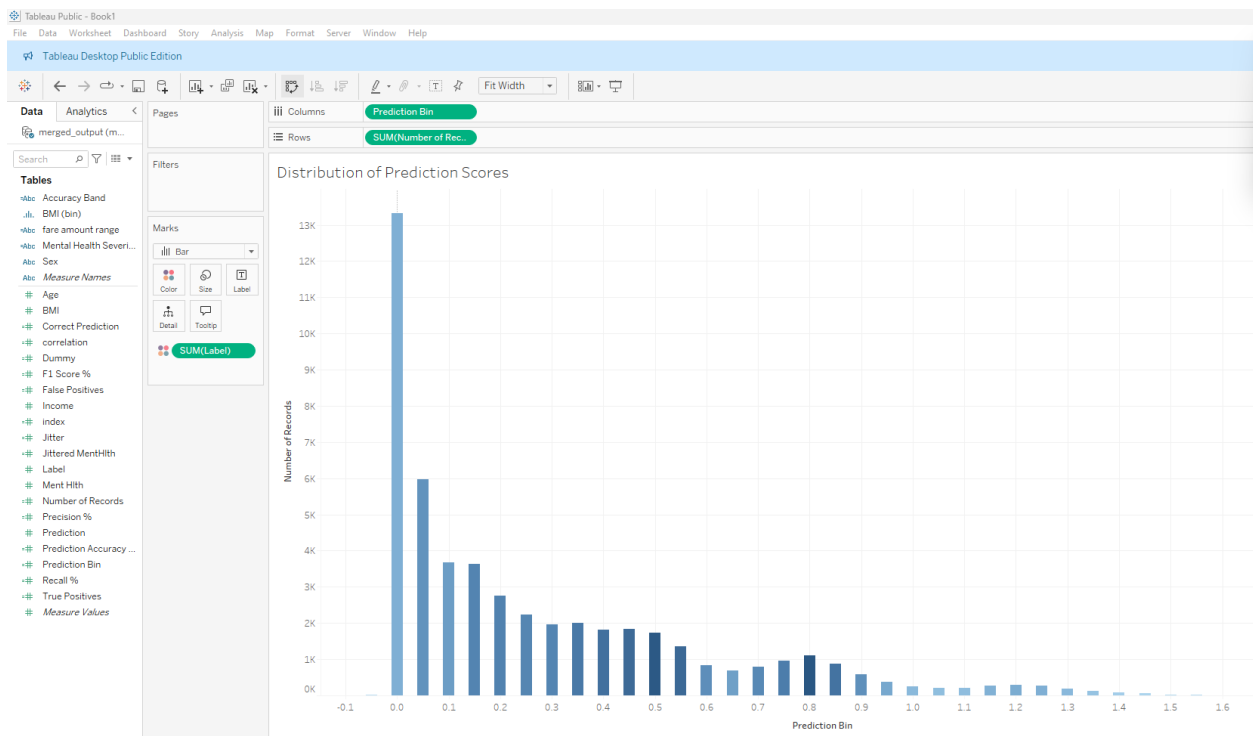
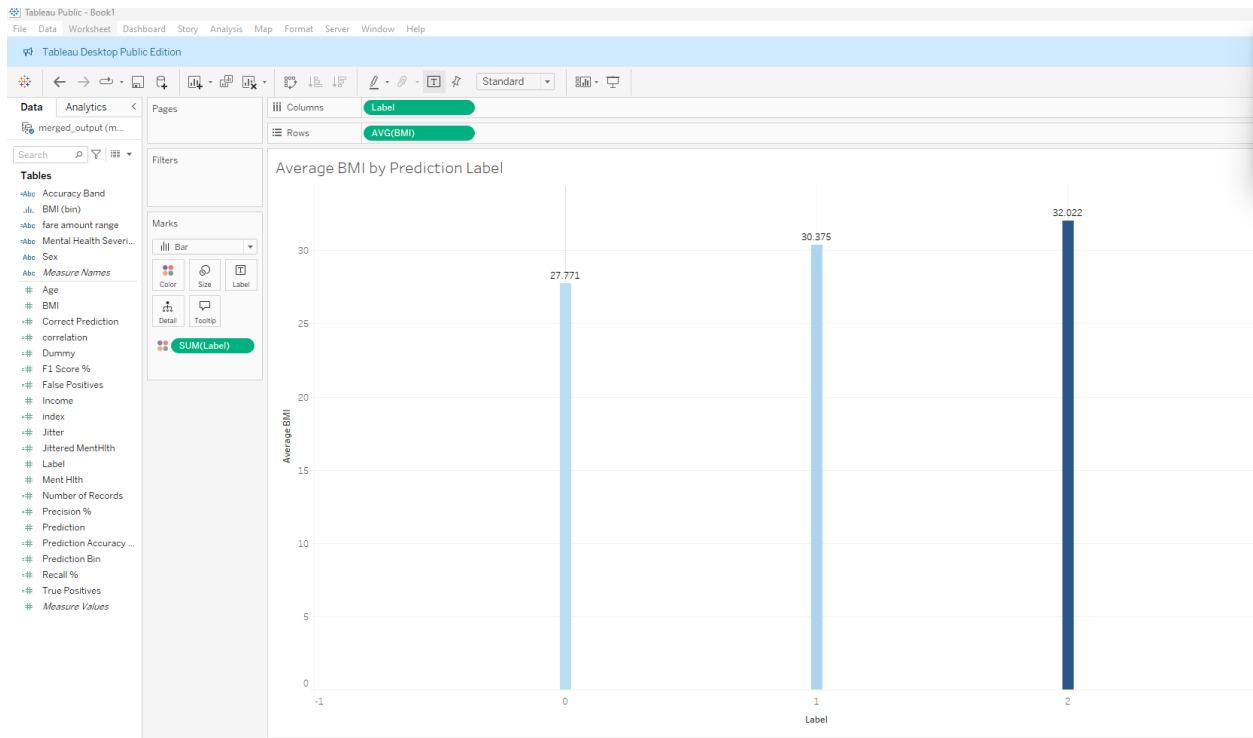
```
[107]: predictions.show(5)
```

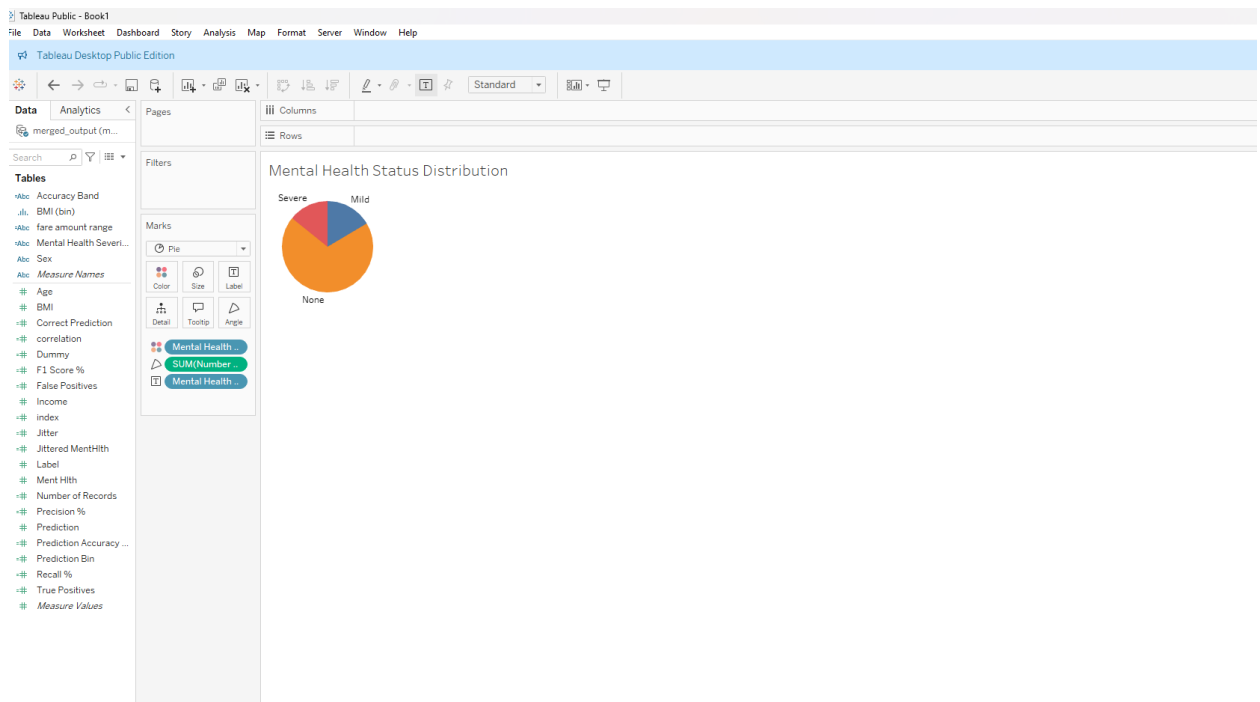
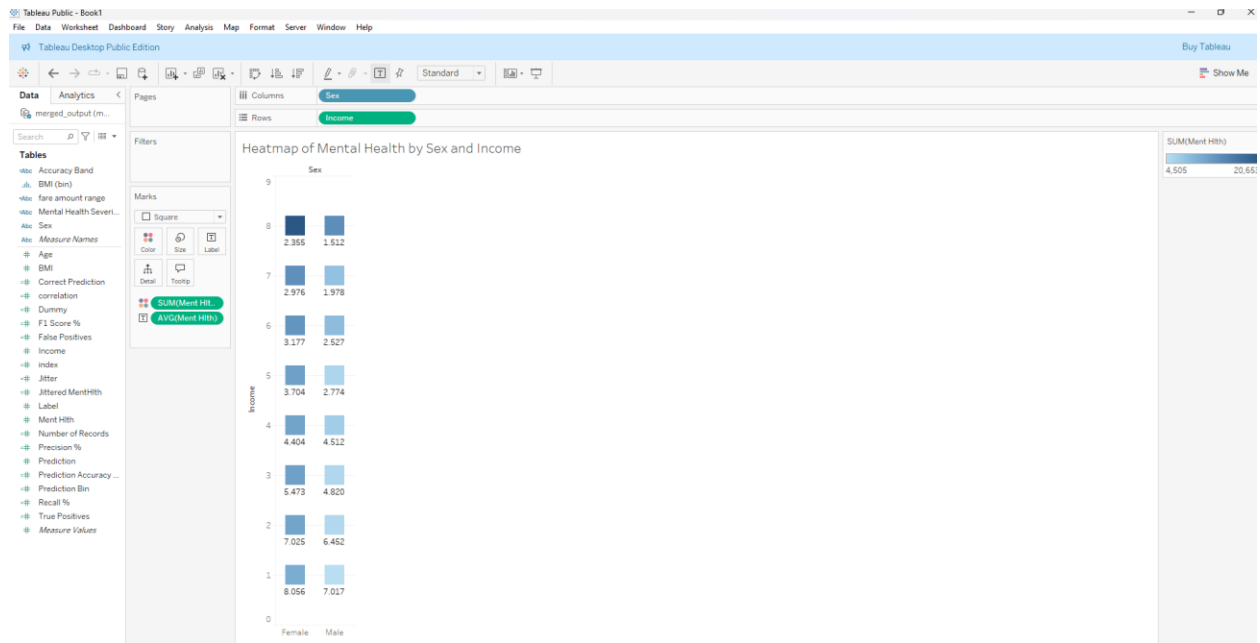
label	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	Veggies	HvyAlcoholConsump	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Income	Sex_indexed	Sex_encoded	features	rawPrediction	probability	p
0.0	0.0	0.0	0.0	0.0	16.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	2.0	5.0	0.0	0.0	1.0	5.0	6.0	8.0	1.0	(1,[,])	(21,[3,4,7,11,13,...]	[2.09645083812110...]	[0.98512229014955...]	
0.0	0.0	0.0	0.0	0.0	17.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	3.0	0.0	0.0	0.0	1.0	9.0	6.0	1.0	1.0	(1,[,])	(21,[3,7,9,11,13,...]	[1.84032952094669...]	[0.97541338176971...]	
0.0	0.0	0.0	0.0	0.0	17.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	4.0	6.0	8.0	0.0	(1,[0],[1.0])	(21,[3,7,8,9,11,1...]	[2.17723075061643...]	[0.98731365430154...]
0.0	0.0	0.0	0.0	0.0	17.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	2.0	5.0	0.0	0.0	0.0	0.0	7.0	4.0	3.0	0.0	(1,[0],[1.0])	(21,[3,4,7,8,9,12...]	[1.95411170073387...]	[0.98031898217086...]
0.0	0.0	0.0	0.0	0.0	18.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	3.0	0.0	0.0	0.0	1.0	10.0	2.0	4.0	1.0	(1,[,])	(21,[3,7,8,9,11,1...]	[1.68812060530150...]	[0.96695370654357...]	

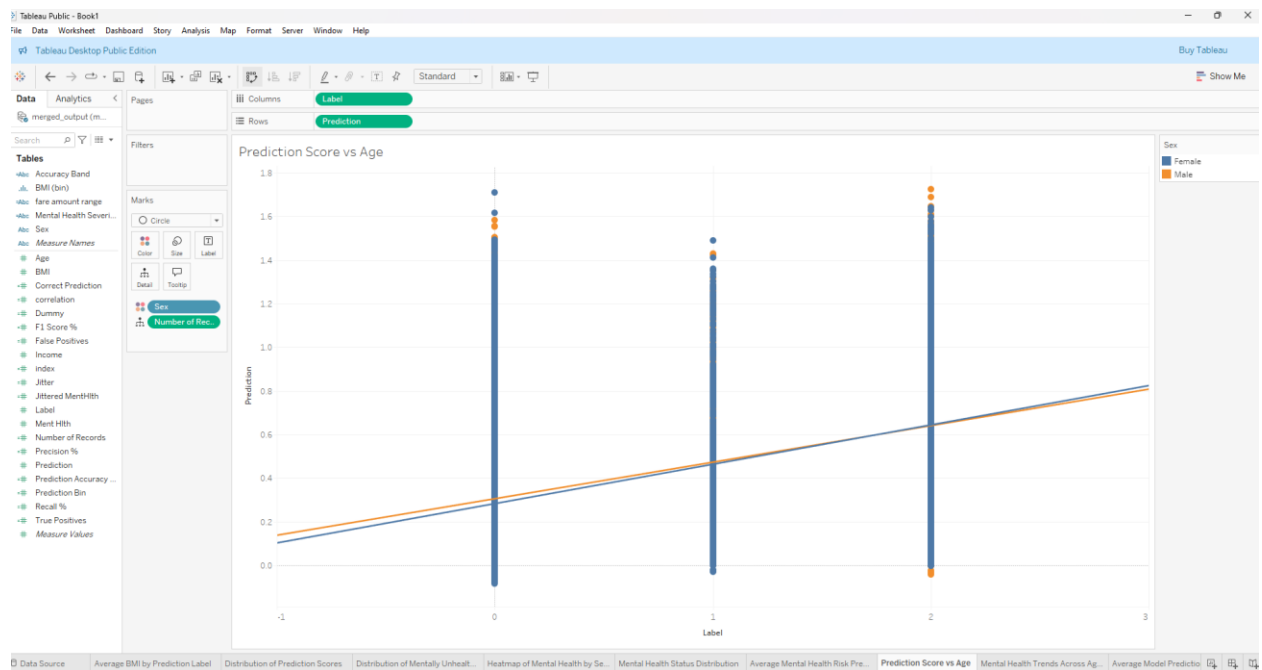
only showing top 5 rows

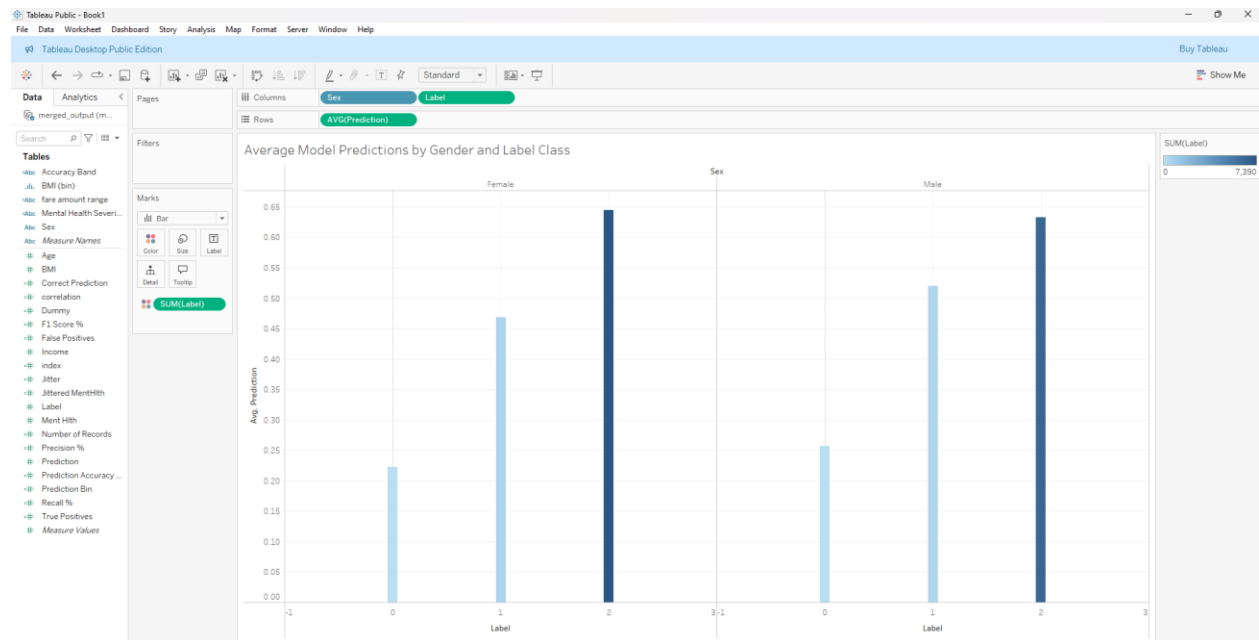
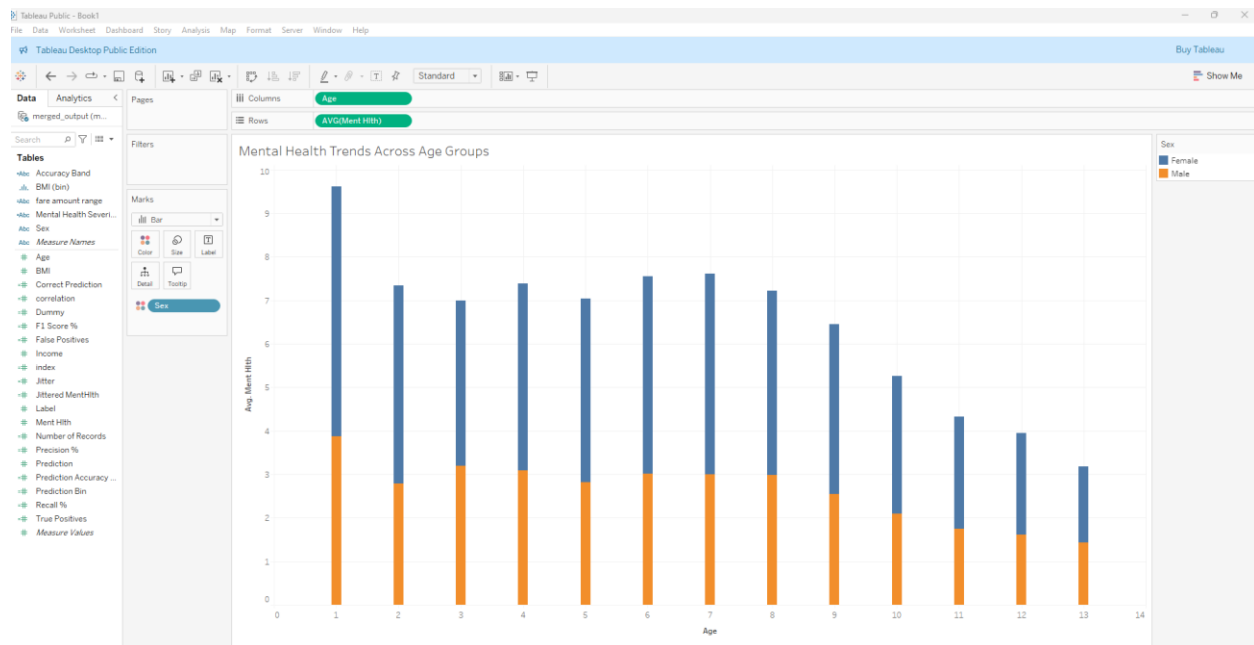
```
[108]: output_df = predictions.select("label", "prediction", "MentHlth", "BMI", "Age", "Income")
output_df.write.csv("diabetes_prediction.csv", header=True, mode="overwrite")
```











Link