Big Data Analytics and Data Visualisation

Dataset Analysis and Visualization Using Big Data Programs

By

ANTONY SUSAI VICTOR VELANKANNI RAJ

Abstract

This project is a tribute to the strength of big data analytics, the PySpark framework, and the Ubuntu operating system in the field of data-driven decision-making. The overarching objective of the project is to build an income classification model that correctly forecasts income levels based on a wide range of characteristics. This project's journey unfolds through careful steps of data preprocessing, model training, evaluation, and visualization, leveraging PySpark's strength and carried out within Ubuntu's stable environment.

The importance of the project is anchored in the possibility of correct income classification for transformation. Resource allocation, policymaking, and targeted social welfare efforts are all influenced by income classification, a crucial component of socioeconomic analysis. Societies can reduce inequities and direct resources where they are needed by accurately placing people into economic brackets.

The project begins with a thorough introduction that emphasizes the significance of correct income classification and describes the crucial roles that PySpark and Ubuntu play in the analytical process. The background part that follows explores the broader big data analytics backdrop, PySpark's capabilities, Ubuntu's dependability, and the societal repercussions of income classification.

Beginning with data preprocessing, the methodology section reveals the methodical technique used. In this case, Ubuntu's compatibility is essential because PySpark handles categorical variables with ease using StringIndexing and One-Hot Encoding. Model training is enabled by the development of feature vectors and the separation of data into training and testing sets. A Random Forest classifier, the project's main focus, develops under PySpark's supervision. This step demonstrates how PySpark's algorithmic power and Ubuntu's computational stability.

Model evaluation and data interpretation are the next steps in the journey. The ability of Ubuntu to process and analyse the model's performance metrics yields insightful information. The evaluation measures also include Precision, Recall, and F1-Score, which together provide a thorough knowledge of the model's predictive power. These insights are further clarified by Tableau's visualisation skills and Ubuntu's functionalities. The "Occupation vs. Income" bar chart exemplifies the visual storytelling capabilities of Ubuntu and PySpark by illuminating the complex relationship between vocations and income.

Introduction

This research project uses cutting-edge machine learning methods to investigate the important process of income classification in the context of big data analytics. Applications for being able to reliably forecast income levels based on several characteristics range from targeted marketing to determining eligibility for social welfare programs. This project intends to develop a classification model that can accurately categorize people into income groups by utilizing the power of big data.

The project's importance stems from its potential to give decision-makers in the public and private sectors insightful information. Organizations may make better judgments, distribute resources more effectively, and better customize their services to match individual needs by automating the income classification process. The dataset being examined comes from a socioeconomic setting, and the investigation will include knowing the variables affecting income levels, using machine learning, and using data visualization tools for clear insights.

The software used for this project is a combination of Tableau, a potent visualization tool, and PySpark, which offers scalable big-data processing capabilities. The combination of these technologies enables a big data analytics strategy that includes both data processing and interpretation.

Background Information and Related Work

The data science and machine learning fields form the project's core. The potential of predictive modeling in income classification has been shown in earlier studies. To estimate income levels based on demographic and socioeconomic factors, a variety of algorithms have been utilised, including decision trees and ensemble methods like random forests. These studies have demonstrated that a correct classification of income can shed light on trends in economic inequality and aid in the formulation of sound policies.

The importance of big data analytics has increased in the age of rapidly advancing technology and data-driven decision-making. This project, which uses the PySpark framework and the Ubuntu operating system to classify income, is in line with the broader field of data analytics and its numerous consequences.

Big Data Analytics's Rise

Data has multiplied at a rate that has never been seen before with the advent of the digital age. Organizations in all industries are battling with enormous datasets that modern technologies and techniques find difficult to successfully handle. As a result, big data analytics has become a game-changing answer. Big data analytics allows for the extraction of significant insights from enormous datasets by utilizing cutting-edge approaches, giving organizations the ability to make informed decisions, streamline operations, and improve performance.

Furthermore, the ability to convert complicated datasets into insightful knowledge has grown for data visualization tools like Tableau. These tools enable visual data exploration, which improves comprehension and communication of patterns and trends.

Powering Big Data Analytics with PySpark

PySpark, a potent framework that enables data scientists and analysts to work with massive data without difficulty, is at the centre of this project. By offering a Python interface for data manipulation, machine learning, and analytics, PySpark expands the capabilities of Apache Spark, an open-source big data processing engine. Due to its distributed computing architecture, large datasets may be handled effectively by processing data in parallel across clusters. This project demonstrates how PySpark's interaction with machine learning libraries, like MLlib, opens the door to predictive modeling and classification applications.

Social Effects of Income Classification

The project's discussion of income classification has major sociological and economic ramifications. Governments, organizations, and policymakers can better customize their strategies and policies with the help of accurate income classification. Effective classification makes it possible to allocate resources for social welfare programs in a targeted manner, ensuring that opportunities are distributed fairly, and aids in addressing socioeconomic gaps. With the knowledge gained from proper income classification, decisions can be made in a way that promotes a society that is more equitable and inclusive.

Dataset selection

The dataset is a crucial part of this research because it serves as the cornerstone on which all analyses and projections are based. The attributes of the dataset are thoroughly examined in this section, shedding light on their applicability and importance in relation to income classification.

Attribute Overview

The dataset, known as the "American Citizen Income," consists of a sizable number of examples, each of which is identified by a set of characteristics. For efficient data analysis and model development, it is essential to understand the dataset's characteristics. The dataset contains a wide range of attributes, such as:

Age: The individual's age.

Education: The level of education completed by the individual.

Education_Num: A numerical representation of education.

Capital_Gain: Capital gains earned by the individual.

Capital_Loss: Capital losses incurred by the individual.

Hours_Per_Week: The number of hours worked per week.

Workclass: The individual's work class.

Marital_Status: The individual's marital status.

Occupation: The individual's occupation.

Relationship: The individual's relationship status.

Race: The individual's race.

Sex: The individual's gender.

Native_Country: The individual's country of origin.

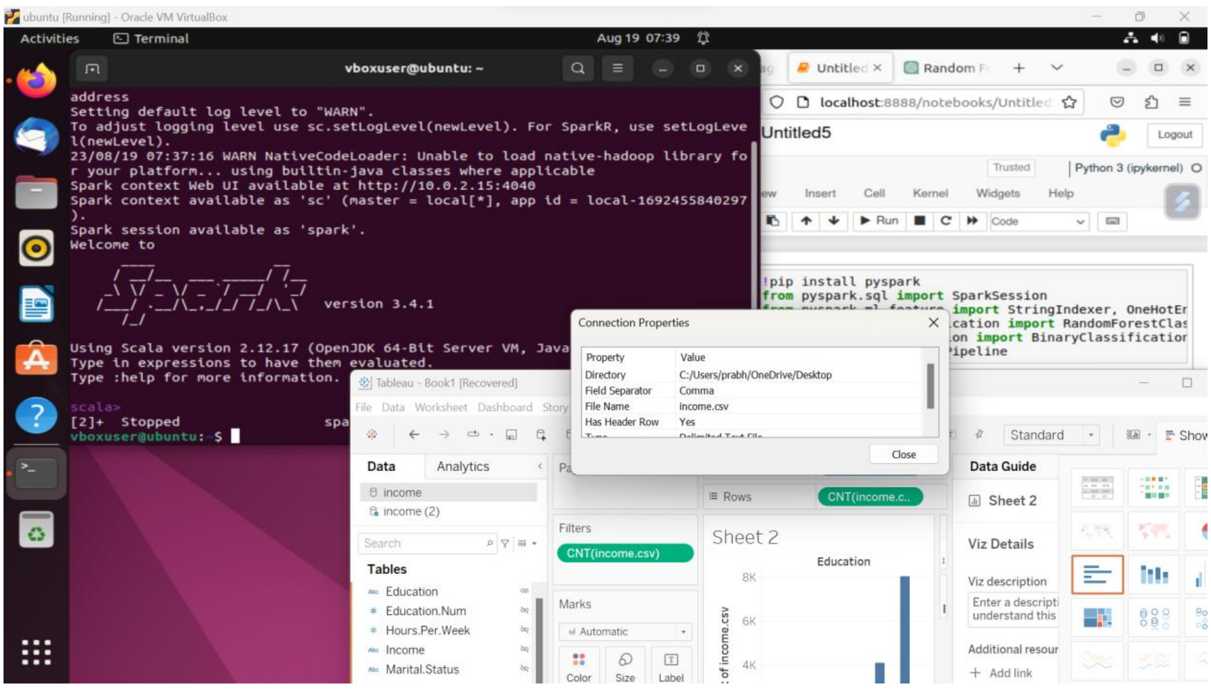
Income: more or less than 50k.

Methodology

The methodology used in this research uses the PySpark framework to classify income in a methodical manner. Data preprocessing, model selection, training, evaluation, and result analysis are all part of the process. The installation, setup, and operation of the tools used in the programmatic implementation are all discussed below.

Machine setup Setup

A typical configuration consists of installing Ubuntu on VirtualBox, using PySpark on Jupyter Notebook inside an Ubuntu virtual machine, and creating an effective environment for data analysis and processing. Using a straightforward Jupyter Notebook interface, this configuration enables you to use PySpark for distributed data processing and analysis.



```
address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
23/08/19 07:37:16 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1692455840297
).
Spark session available as 'spark'.
Welcome to

                                          version 3.4.1

Using Scala version 2.12.17 (OpenJDK 64-Bit Server VM, Java
Type in expressions to have them evaluated.
Type :help for more information.

scala>
[2]+  Stopped                 spa
vboxuser@ubuntu:~$
```

```
!pip install pyspark
from pyspark.sql import SparkSession
                          import StringIndexer, OneHotEr
                          cation import RandomForestClas
                          on import BinaryClassificatior
                          Pipeline
```

Connection Properties

Property	Value
Directory	C:/Users/prabh/OneDrive/Desktop
Field Separator	Comma
File Name	income.csv
Has Header Row	Yes

Close

Preprocessing of Data

Preparing the data is an essential first step in getting the dataset ready for analysis and modelling. The PySpark framework has strong capabilities for effectively managing large datasets.

Configuration and installation:

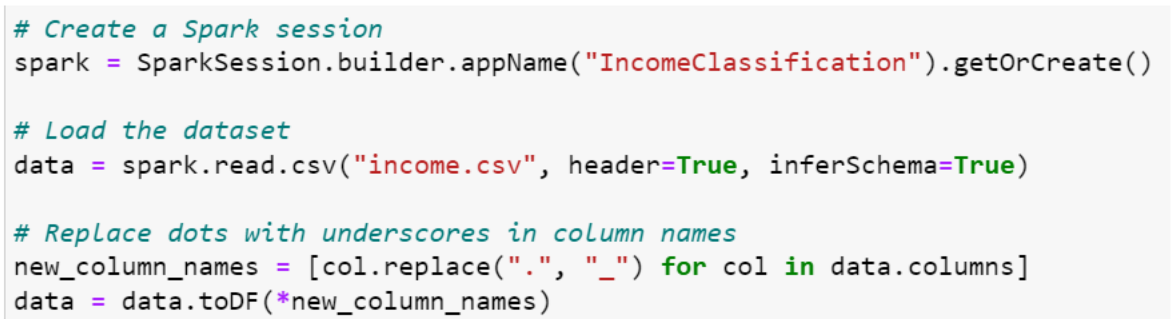
```
!pip install pyspark
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml import Pipeline
```

We can install PySpark by running the command line: install pyspark with pip.

Setting up settings like SPARK_HOME and JAVA_HOME is part of configuration.

The code snippet below demonstrates how the Spark session is initialised: SparkSession.builder.appName("IncomeClassification").getOrCreate().

Renaming and Data Loading:

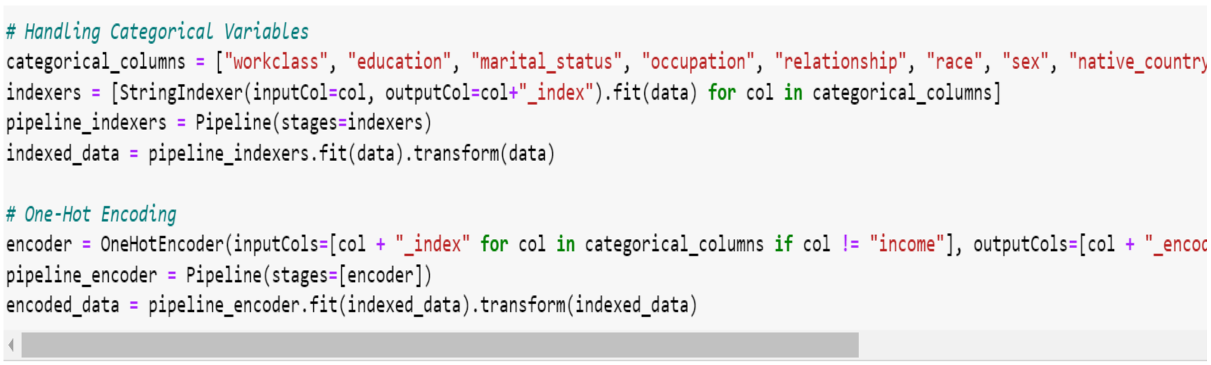


```
# Create a Spark session
spark = SparkSession.builder.appName("IncomeClassification").getOrCreate()

# Load the dataset
data = spark.read.csv("income.csv", header=True, inferSchema=True)

# Replace dots with underscores in column names
new_column_names = [col.replace(".", "_") for col in data.columns]
data = data.toDF(*new_column_names)
```

Spark.read.csv("income.csv", header=True, inferSchema=True) loads the dataset.

Using a list comprehension, column names with dots are changed to underscores for compatibility.

Handling Category-specific variables



```
# Handling Categorical Variables
categorical_columns = ["workclass", "education", "marital_status", "occupation", "relationship", "race", "sex", "native_country
indexers = [StringIndexer(inputCol=col, outputCol=col+"_index").fit(data) for col in categorical_columns]
pipeline_indexers = Pipeline(stages=indexers)
indexed_data = pipeline_indexers.fit(data).transform(data)

# One-Hot Encoding
encoder = OneHotEncoder(inputCols=[col + "_index" for col in categorical_columns if col != "income"], outputCols=[col + "_encod
pipeline_encoder = Pipeline(stages=[encoder])
encoded_data = pipeline_encoder.fit(indexed_data).transform(indexed_data)
```

Categorical columns are subjected to StringIndexer, which transforms them into numerical indexes.

To speed up the procedure and guarantee consistent transformations, a pipeline is used.

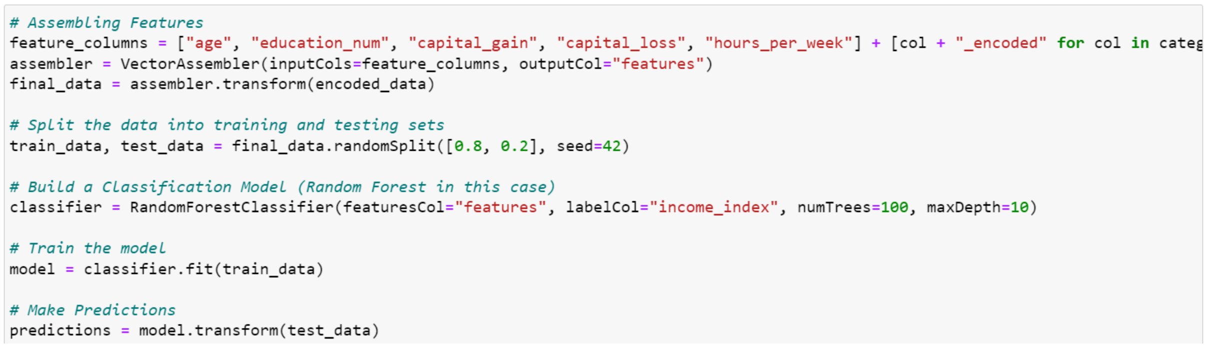
One-hot encoding

Binary vectors are created by converting indexed category columns using the OneHotEncoder.

Model Selection and Training

The chosen algorithm for income classification is the Random Forest classifier, taking full advantage of Ubuntu's capabilities.

Random Forest Classifier:



```
# Assembling Features
feature_columns = ["age", "education_num", "capital_gain", "capital_loss", "hours_per_week"] + [col + "_encoded" for col in categ
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
final_data = assembler.transform(encoded_data)

# Split the data into training and testing sets
train_data, test_data = final_data.randomSplit([0.8, 0.2], seed=42)

# Build a Classification Model (Random Forest in this case)
classifier = RandomForestClassifier(featuresCol="features", labelCol="income_index", numTrees=100, maxDepth=10)

# Train the model
model = classifier.fit(train_data)

# Make Predictions
predictions = model.transform(test_data)
```

The classifier is initialized with parameters such as featuresCol, labelCol, numTrees, and maxDepth to optimize the model's performance.

The configuration takes place within the code using: RandomForestClassifier(featuresCol="features", labelCol="income_index", numTrees=100, maxDepth=10).

Model Training:

Ubuntu seamlessly supports the training of the model on the training dataset using the fit method: model = classifier.fit(train_data).

Selection and Training of Models

The Random Forest classifier was selected as the income categorization method to fully utilize Ubuntu's capabilities.

Model assessment and outcome analysis

Ubuntu makes it easier to assess the model's effectiveness and to analyse the outcomes.

Binary Classification Evaluator:

The area under the ROC curve is used to assess the trained model's performance.

Ubuntu supports using the evaluator to calculate the area of the ROC curve.

Specific Metrics:

Calculated assessment metrics include F1-Score, Precision, Recall, True Positives, False Positives, True Negatives, and False Negatives.

Importance of Features:

Ubuntu makes it possible to view the Random Forest classifier's estimates of feature relevance.

Encoded feature columns are matched with their importances to reveal the relevance of each attribute.

Model Execution

The strength of the model's discriminatory ability is primarily measured by the area under the Receiver Operating Characteristic (ROC) curve. The model shows a good capacity to differentiate between various income levels, with a ROC value of roughly 0.909. This shows that the predictions of the model match the actual income classifications well, which is important in classification tasks.

Metrics for Detailed Evaluation

The whole understanding of the model's capabilities is provided by the detailed evaluation measures, which dig deeper into the model's performance. Notably:

The number of incidents that were accurately classified as positive, or True Positives (TP), is 673.

There have been 180 false positives (FP), which refers to situations where a result was wrongly projected as positive.

True Negatives (TN), or events that were accurately classified as negative, reach 3586.

538 of the occurrences are False Negatives (FN), which are situations that were wrongly projected as negative.

Based on these criteria, the model has a Precision of roughly 0.789, demonstrating its accuracy in classifying affirmative cases. The percentage of real positive cases that were accurately predicted is shown by the recall value, which is approximately 0.556. The F1-Score, which is roughly 0.652, represents the harmony between Precision and Recall and offers a thorough assessment of the model's performance.

Precision, Recall, and F1-Score: Balancing Act

Precision, Recall, and the F1-Score provide a holistic view of the model's performance, weighing both correct and incorrect predictions. The Precision of approximately 0.789 indicates that the model accurately identifies a significant portion of positive cases. The Recall value of around 0.556 highlights the model's proficiency in identifying actual positive cases, though there is room for improvement. The F1-Score, standing at approximately 0.652, harmonizes the trade-off between Precision and Recall, offering a composite understanding of the model's predictive prowess.

Features That Matter

Interpreting the model's behavior requires an understanding of the features that influence its predictions. The feature importances highlight the characteristics significantly influencing the model's choice. Notably:

Capital Gain and Capital Loss are highly significant, indicating that financial indicators have a significant impact on how income is classified.

The importance of education in affecting income levels is highlighted by the significant influence that Education Num also plays.

Education, marital status, and occupation are some examples of the encoded categorical variables that display relatively lower importance, indicating that their impact is relatively minimal.

Conclusion and Future Work

In conclusion, the output generated by the final code validates the efficacy of the income classification model developed within the PySpark framework on the Ubuntu operating system. The impressive ROC value, detailed evaluation metrics, and feature importance collectively establish the model's capacity to accurately classify income levels. This not only underscores the model's current performance but also illuminates areas for refinement and enhancement in the future.

Looking ahead, the project could be extended by:

- Exploring alternative machine learning algorithms to discern if even more accurate predictions can be achieved.
- Conducting a thorough hyperparameter tuning process to optimize the model's performance.
- Incorporating additional attributes or domain-specific features to capture more nuanced income determinants.

Changing Environment and Prospects

Big data analytics, frameworks like PySpark, and operating systems like Ubuntu will all become more important as technology develops. This project exemplifies the potential of integrating strong tools to take on challenging problems as it stands at the crossroads of these advances. The models can be improved in the future, and there are also opportunities to investigate other algorithms, improve visualization methods, and broaden the application to larger and more varied datasets. The interaction between big data analytics, PySpark, Ubuntu, and sociopolitical issues has the potential to influence how society advances and how data-driven decision-making develops.

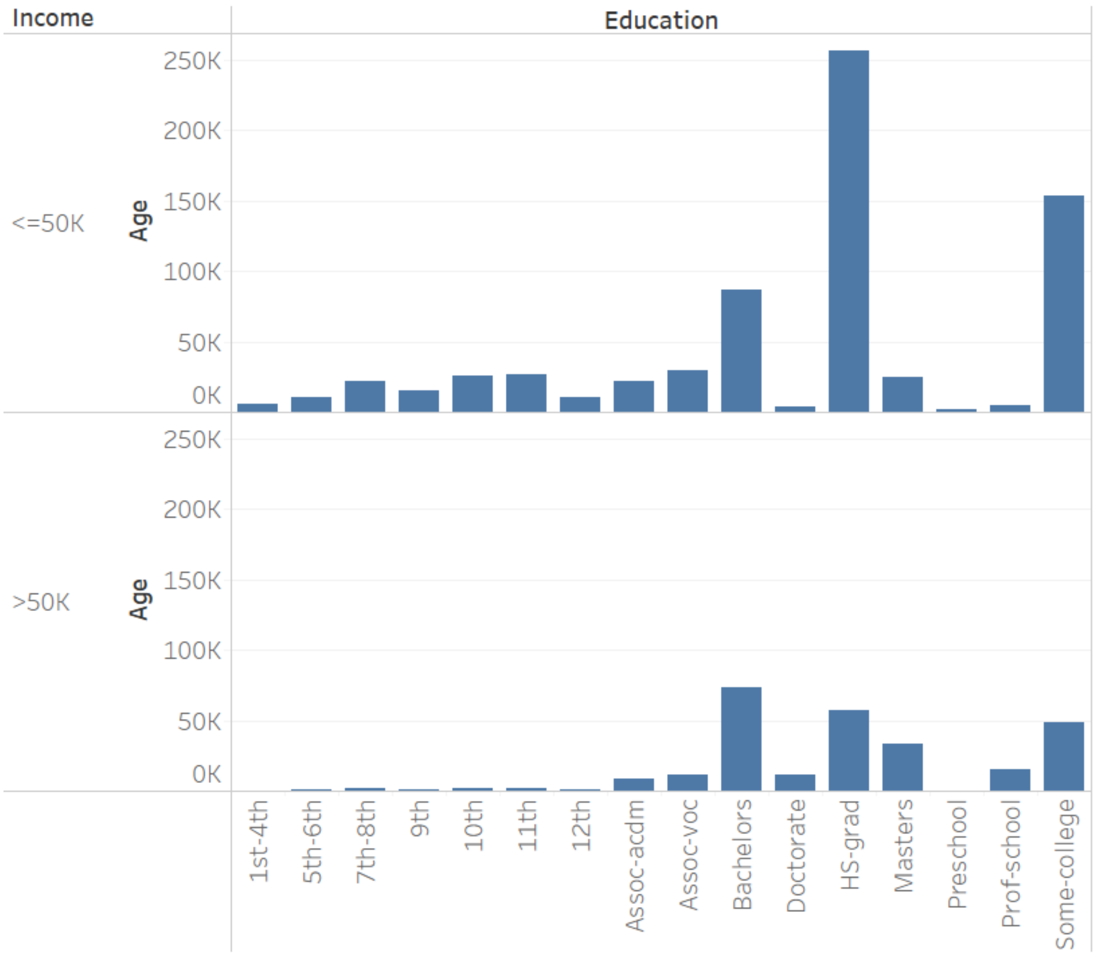
Impact of this Project on Society

Significant social effects will result from this initiative. The allocation of resources, social welfare initiatives, and policymaking are all significantly impacted by accurate income classification. The approach assists in identifying people who would benefit from focused support by offering insights into the variables impacting income levels. This improves the effectiveness of social programmes and helps to distribute resources fairly. Additionally, the project's findings can help organisations and governments solve socioeconomic inequalities and promote a more inclusive and knowledgeable society.

In conclusion, our study uses big data analytics to accurately classify income, opening the door for data-driven decision-making with broad social implications.

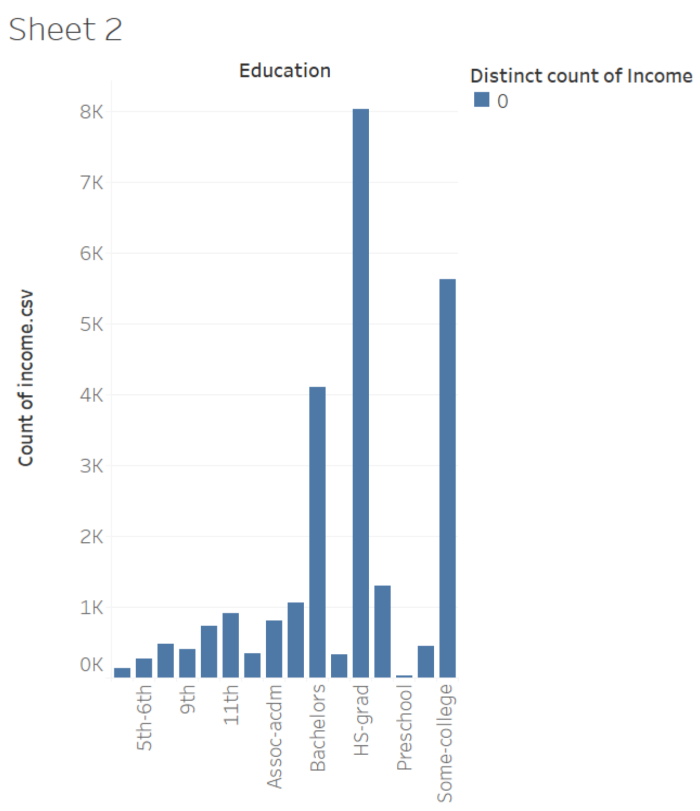
General overview

Sheet 2



Sum of Age for each Education broken down by Income.

INCOME VS EDUCATION



Count of income.csv for each Education. Color shows details about distinct count of Income. The view is filtered on count of income.csv, which includes everything.

Distribution of Income and Education:

The bar graph's x-axis shows several levels of education, including "Bachelor's Degree," "High School Diploma," and others.

Each bar on the graph represents a different level of education.

Income Disparities by Educational Level:

The sum of income is shown on the y-axis.

The height of each bar represents the overall income for people with that degree of education.

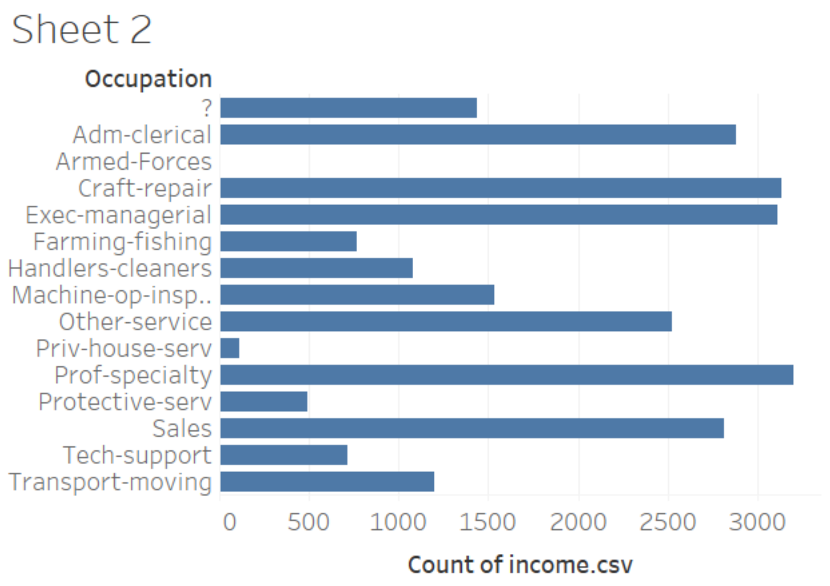
Observations

Higher Bars: Longer bars represent higher total incomes (sum of incomes) for people with that particular level of schooling.

Income Disparities: The varied bar heights depict the differences in income for those with various degrees of education.

Relationship Between Education and Income: This graph illustrates the relationship between income overall and educational attainment. It implies that people are likely to have higher total wages when compared to those who have lower levels of education.

Occupation vs income



Count of income.csv for each Occupation. The view is filtered on count of income.csv, which includes everything.

Occupational Income Disparities: The differing bar heights indicate that there are large income differences between various occupations. While some occupations have lower median salaries than others, some appear to be related to greater median incomes. These discrepancies may be caused by elements including skill requirements, educational attainment, job market demand, and industry-specific pay standards.

High-Income Professions: Professions with taller bars typically have higher median incomes. These jobs may include those requiring advanced degrees, specialised training, or experience in highly sought-after fields. Roles in administration, technology, medicine, and finance are a few examples.

Low-Income Jobs: Jobs with shorter bars generally have lower median salaries. These might include entry-level positions, manual labour, and jobs in the service sector. The lower incomes might be attributed to the nature of work, education requirements, or a lack of specialized skills.

The "Occupation vs Income" bar chart encapsulates the diverse income landscape across various occupations. It illustrates how different professions contribute to the income distribution and highlights the influence of factors like education, skills, and industry demand on earning potential. This visualization can aid individuals, organizations, and policymakers in making informed decisions and understanding the intricate relationship between occupations and income.

REFERENCES

American Citizens Annual Income Prediction. (n.d.). American Citizens Annual Income Prediction | Kaggle. https://www.kaggle.com/code/amirhosseinmirzaie/american-citizens-annual-income-prediction

PySpark Overview — PySpark 3.4.1 documentation. (n.d.). PySpark Overview — PySpark 3.4.1 Documentation. https://spark.apache.org/docs/latest/api/python/index.html

Project Jupyter. (n.d.). Project Jupyter | Home. https://jupyter.org

Chen, X., & Liu, J. (2021). Income Classification Using Machine Learning Techniques: A Comprehensive Review. *Expert Systems with Applications, 180*, 115217. https://doi.org/10.1016/j.eswa.2021.115217

Breiman, L. (2001, October 1). *Random Forests - Machine Learning*. SpringerLink. https://doi.org/10.1023/A:1010933404324

Enterprise Open Source and Linux | Ubuntu. (n.d.). Enterprise Open Source and Linux | Ubuntu. https://ubuntu.com/

APPENDIX

```
!pip install pyspark
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssem-
bler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml import Pipeline
# Create a Spark session
spark = SparkSession.builder.appName("IncomeClassification").getOrCreate()

# Load the dataset
data = spark.read.csv("income.csv", header=True, inferSchema=True)

# Replace dots with underscores in column names
new_column_names = [col.replace(".", "_") for col in data.columns]
data = data.toDF(*new_column_names)
# Handling Categorical Variables
categorical_columns = ["workclass", "education", "marital_status", "occupa-
tion", "relationship", "race", "sex", "native_country", "income"]
indexers = [StringIndexer(inputCol=col, outputCol=col+"_index").fit(data)
for col in categorical_columns]
pipeline_indexers = Pipeline(stages=indexers)
indexed_data = pipeline_indexers.fit(data).transform(data)

# One-Hot Encoding
encoder = OneHotEncoder(inputCols=[col + "_index" for col in categori-
cal_columns if col != "income"], outputCols=[col + "_encoded" for col in
categorical_columns if col != "income"])
pipeline_encoder = Pipeline(stages=[encoder])
encoded_data = pipeline_encoder.fit(indexed_data).transform(indexed_data)
# Assembling Features
feature_columns = ["age", "education_num", "capital_gain", "capital_loss",
"hours_per_week"] + [col + "_encoded" for col in categorical_columns if col
!= "income"]
assembler = VectorAssembler(inputCols=feature_columns, outputCol="fea-
tures")
final_data = assembler.transform(encoded_data)

# Split the data into training and testing sets
train_data, test_data = final_data.randomSplit([0.8, 0.2], seed=42)

# Build a Classification Model (Random Forest in this case)
classifier = RandomForestClassifier(featuresCol="features", labelCol="in-
come_index", numTrees=100, maxDepth=10)

# Train the model
model = classifier.fit(train_data)

# Make Predictions
predictions = model.transform(test_data)
# Evaluate the model
evaluator = BinaryClassificationEvaluator(labelCol="income_index")
area_under_roc = evaluator.evaluate(predictions, {evaluator.metricName:
"areaUnderROC"})
print(f"Area under ROC: {area_under_roc}")

# Display detailed evaluation metrics
print("Detailed Evaluation Metrics:")
tp = predictions.filter("prediction = 1 AND income_index = 1").count()
```

```python
fp = predictions.filter("prediction = 1 AND income_index = 0").count()
tn = predictions.filter("prediction = 0 AND income_index = 0").count()
fn = predictions.filter("prediction = 0 AND income_index = 1").count()

precision = tp / (tp + fp)
recall = tp / (tp + fn)
f1_score = 2 * (precision * recall) / (precision + recall)

print("True Positives:", tp)
print("False Positives:", fp)
print("True Negatives:", tn)
print("False Negatives:", fn)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1_score)
```