Census Income Data from mid 90s USA – analysis and classification

Maxwell Bilbow

7082CEM Big Data Management & Data Visualisation

Abstract

This paper analyses US census income data and makes predictions about wages by applying a machine learning algorithm.

Machine learning was applied using PySpark[Section 3], with additional dependencies listed in appendix 6.2.3.

Visualisations were generated using Tableau[Section 4].

All code snippets in this document are inserted word documents. Double clicking on them will open the underlying word document allowing you to copy and paste the code.

Table of Contents

1 INTRODUCTION

In this document we will apply a variety of analytical techniques, including classification via machine learning.

We will initially attempt to clean and investigate the data using Apache Spark (*Apache spark.*). Building on any insights found, we will use data visualisation techniques to produce informative visual representations in Tableau (Milligan, 2020)

1.1 ABOUT THE DATA

The dataset is Census Income Data Set from UCI (*Census income data set.*). It contains 48842 entries in total and various features. Each entry has a categorisation label that describes whether an individual earned more or less than 50,000 USD in a year.

Information about the dataset:

- donated in 1996;
- has 14 multivariate attributes;
- contains categorical and integer attributes;
- contains 48842 instances;
- has missing values.

We will apply a variety of analytical techniques including the creation of a logistical regression machine learning model with the aim of predicting the likelihood of an individual earning more than $50K.

1.1.1 Features

Feature	Type	Additional Information
Age	Continuous	
Workclass	Categorical	
Fnlwgt	Continuous	The actual population the entry represents (Should be ignored for classification). It may be relevant when analysing proportional values.
Education	Categorical	Superseded by Education-num for classification purposes
Education-num	Continuous	Sequential representation of Education.
Marital-status	Categorical	
Occupation	Categorical	
Relationship	Categorical	
Race	Categorical	
Sex	Categorical	Can also be expressed as binary
Capital-gain	Continuous	
Capital-loss	Continuous	
Hours-per-week	Continuous	
Native-country	Categorical	Can be converted to longitude and latitude for visualisation purposes
>50K, <=50K	Classification	Can be converted to binary for simplicity.

1.1.2 Files

The dataset contains two data files and one file containing metadata. Additional preparation was required before the data could be used:

- adult.data – unchanged - CSV file containing training data
- adult.test – unchanged - CSV file containing test data (Note: each line ends with a '.' character which has been removed.
- adult.names – unchanged - File containing metadata
- attributes.csv – created - CSV helper file created from adult.names; used to define schema.

2 INSTALLATION

When deciding on an appropriate toolset and installation process, one of the main considerations was reproducibility. While virtual machines are a good approach, the VM software itself is not always portable. For example, VMWare Player (*VMWare.*) is both free to use and extremely popular however it is not supported on MacOS. VirtualBox (*VirtualBox.*) while supported by Linux, Windows, and MacOS; can be less intuitive to configure.

It was therefore decided that a container basted approach, using Docker, would be more appropriate (*Docker.*).

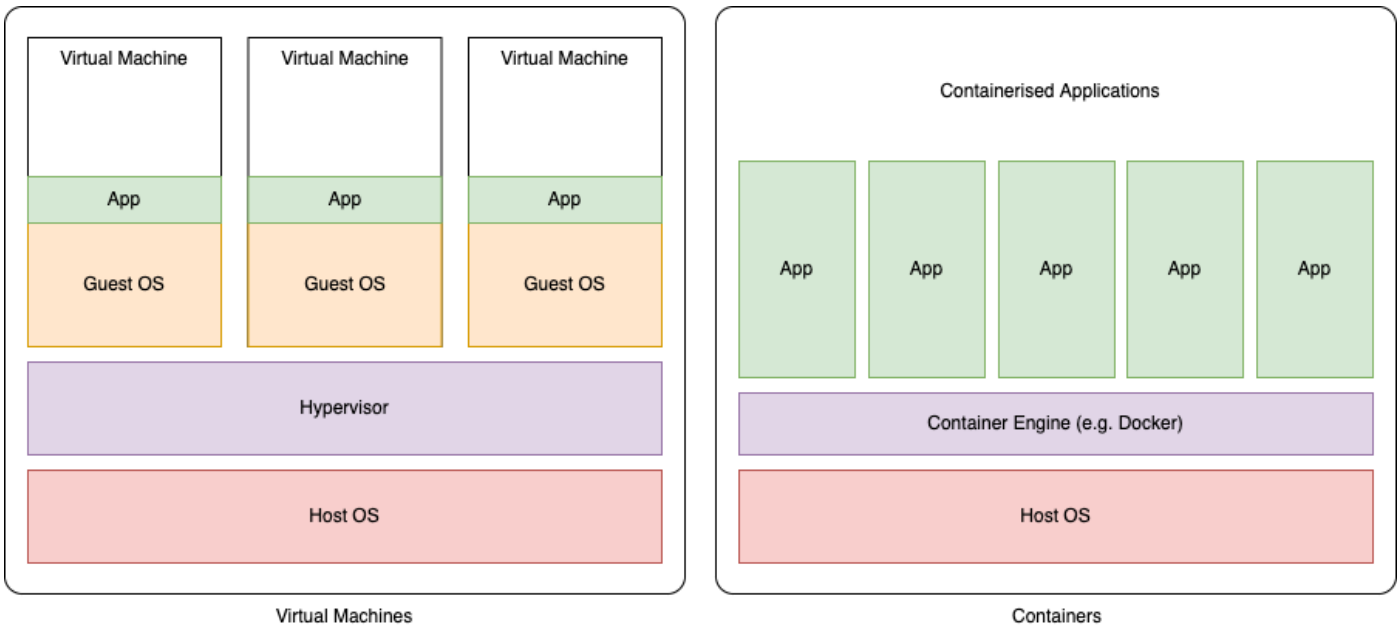


Figure 1 - VM vs Container Architecture

Containerised applications can be more scalable and more lightweight while easily sharing resources. However they can be seen as less secure than Virtual Machines, which are self-contained.

2.1 TOOLS

Two different Docker containers are used:

- Third party Jupyter Notebook (Jupyter/pyspark-notebook docker image.)
- Ubuntu-based image (*Ubuntu docker image.*) built with custom configuration – this will be used to run traditional Python files on Apache Spark

Both containers will use PySpark to interface with Apache Spark.

2.2 INSTALLATION STEPS

In this section, we cover the necessary steps to set up both containers.

2.2.1 Download and Install Docker

The simplest way to install Docker is via their "getting started" page: https://www.docker.com/get-started. MacOS developers with Homebrew installed (*Homebrew.*) may simply enter **brew install --cask docker** on the command line.

Confirm docker is installed and ready by running the following command:

docker run hello-world

2.2.2 Clone the repository

The repository code needed to setup the environments is the appendix[6.1] however, for the sake of simplicity, the repository can be cloned (or simply downloaded) from https://github.com/maxbilbow/7082CEM-big-data-visualisation.

2.2.3 Check active ports

The two containers will map the following ports to localhost: 8888, 4040, 7077, and 6066. If you are currently using any of these then you may need to change the mappings in docker-compose.xml[6.2.4].

2.2.4 Build and Instantiate the Containers

To build and start the containers, navigate to the root of the repository (wherever docker-compose.xml[6.2.4] is located) and run the following command:

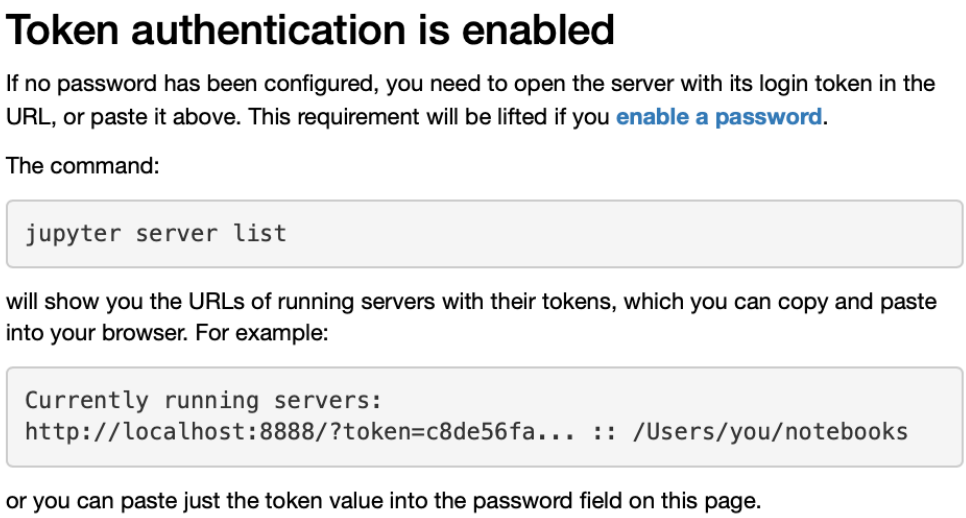
docker-compose up -d

The **-d** flag tells Docker to runt he containers detached. To stop and remove the containers, you can enter:

docker-compose down

2.2.5 Open Jupyter Notebook

Navigate to http://localhost:8888; you should see something like this:



To get the token, you need to access the container via a terminal. Enter the following command:

docker exec -it bdv_notebook /bin/bash

Now you can follow the instructions above and obtain the token:

jupyter server list

You should see something like this:

```
(base) jovyan@0a11c59646d0:~$ jupyter server list
Currently running servers:
http://0a11c59646d0:8888/?token=59224d1e522405acd639cdcbca594c0c40b64fc3e5339e2f :: /home/jovyan
```

Navigate to **localhost:8888/?token={whatever_your_token_is}**, you will not be required to do so again.

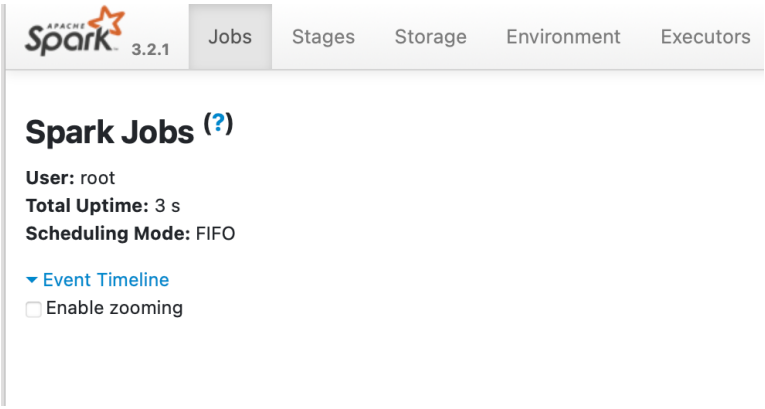
To exit the container's shell, enter the command **exit**.

2.2.6 Confirm the Spark Container

In a terminal, enter the following command:

```
docker exec -it bdv_notebook /bin/bash
```

Check that the spark instance works by running entering the command **spark-shell** and navigating to http://localhost:4040. You should see the following:



Exit the spark session by hitting **CTRL+C**.

3 ANALYSIS USING PYSPARK

The following steps, executed sequentially, can be implemented on either environment.

3.1 SELECTING AN ENVIRONMENT:

On the bdv_spark container:

- Add the python script to the repository's 'app' folder;
- Connect to the bdv_spark container[2.2.6]
- Run your script with **spark-submit**, e.g.:

  spark-submit LogisticRegression.py

On the bdv_jupyter (Jupyter Notebook) container:

- Navigate to http://localhost:8888
- Open the 'work' folder
- Create new Python Notebook

The Jupyter Notebook may be preferable if recreating everything step-by-step.

3.2 PREPARING THE DATA

3.2.1 Set Constants

```
import os

ROOT = r"./"
DATA_IN = os.path.join(ROOT, "data", "income-predictor")
SOURCE_DATASET = os.path.join(DATA_IN ,"adult.data")
SOURCE_ATTRIBUTES = os.path.join(DATA_IN, "attributes.csv")
SOURCE_TEST_DATA = os.path.join(DATA_IN, "adult.test")
CLEANED_DATA_OUT = f'{SOURCE_DATASET}.clean'

LABEL_COL='>50K'


print(SOURCE_DATASET)
print(SOURCE_ATTRIBUTES)
print(SOURCE_TEST_DATA)
```

3.2.2 Instantiate Spark Session

```
from pyspark.sql import SparkSession, DataFrame

spark=SparkSession \
    .builder \
    .appName("Income Predictor") \
    .getOrCreate()
```

3.2.3 Create Schema

An attributes file was created from the data's description (*Census income data set.*)

```
from pyspark.sql.types import StructType, StructField, IntegerType,
DecimalType, StringType, BooleanType

attrs = spark.read.csv(SOURCE_ATTRIBUTES, header=True)
attrs.show()

def get_field(t: str):
    if t == "continuous":
        return DecimalType()
    elif t == "string":
        return StringType()
    elif t == "boolean":
        return BooleanType()
    else:
        raise Exception("Not expected: %s" % t)

def to_struct(row) -> StructField:
    return StructField(row['description'], get_field(row['type']),
nullable=False)

struct_fields = attrs.rdd.map(to_struct).collect()

schema=StructType(struct_fields)
```

Output:

```
+-----+----------+--------------+--------------------+
| name|      type|   description|              values|
+-----+----------+--------------+--------------------+
|  f_1|continuous|           age|                null|
|  f_2|    string|     workclass|Private, Self-emp...|
|  f_3|continuous|        fnlwgt|                null|
|  f_4|    string|     education|Bachelors, Some-c...|
|  f_5|continuous| education-num|                null|
|  f_6|    string|marital-status|Married-civ-spous...|
|  f_7|    string|    occupation|Tech-support, Cra...|
|  f_8|    string|  relationship|Wife, Own-child, ...|
|  f_9|    string|          race|White, Asian-Pac-...|
| f_10|    string|           sex|        Female, Male|
| f_11|continuous|  capital-gain|                null|
| f_12|continuous|  capital-loss|                null|
| f_13|continuous|hours-per-week|                null|
| f_14|    string|native-country|United-States, Ca...|
|label|    string|         label|              "<=50k|
+-----+----------+--------------+--------------------+
```

3.2.4 Load the Training Data

We'll also create a function **load_data(file: str)**, among others, which can be used to load the test data later.

```
def load_data(file: str):
    return spark.read.load(file, format="csv", sep=",", header=False,
ignoreLeadingWhiteSpace=True, ignoreTrailingWhiteSpace=True, schema=schema)

df = load_data(SOURCE_DATASET)
df.show(5)
df.printSchema()
df.count()
```

Output:

```
+---+----------------+------+---------+-------------+------------------+-----------------+-------------+-----+------+------------+------------+
|age|       workclass|fnlwgt|education|education-num|    marital-status|       occupation| relationship| race|   sex|capital-gain|capital-loss|
+---+----------------+------+---------+-------------+------------------+-----------------+-------------+-----+------+------------+------------+
| 39|       State-gov| 77516|Bachelors|           13|     Never-married|     Adm-clerical|Not-in-family|White|  Male|        2174|           0|
| 50|Self-emp-not-inc| 83311|Bachelors|           13|Married-civ-spouse|  Exec-managerial|      Husband|White|  Male|           0|           0|
| 38|         Private|215646|  HS-grad|            9|          Divorced|Handlers-cleaners|Not-in-family|White|  Male|           0|           0|
| 53|         Private|234721|     11th|            7|Married-civ-spouse|Handlers-cleaners|      Husband|Black|  Male|           0|           0|
| 28|         Private|338409|Bachelors|           13|Married-civ-spouse|   Prof-specialty|         Wife|Black|Female|           0|           0|
+---+----------------+------+---------+-------------+------------------+-----------------+-------------+-----+------+------------+------------+
only showing top 5 rows
```

```
root
 |-- age: decimal(10,0) (nullable = true)
 |-- workclass: string (nullable = true)
 |-- fnlwgt: decimal(10,0) (nullable = true)
 |-- education: string (nullable = true)
 |-- education-num: decimal(10,0) (nullable = true)
 |-- marital-status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- capital-gain: decimal(10,0) (nullable = true)
 |-- capital-loss: decimal(10,0) (nullable = true)
 |-- hours-per-week: decimal(10,0) (nullable = true)
 |-- native-country: string (nullable = true)
 |-- label: string (nullable = true)

32561
```

3.3 Clean the Data

As it is, the training data is not ready for planned analysis.

3.3.1 Convert Label Column

Our categorisation model will aim to predict whether an individual earns more than $50k per year. Since there are only two categories, we can express this as a binary integer or boolean type.

```
import pyspark.sql.functions as f

def convert_label_col(df: DataFrame):
    df=df.withColumnRenamed('label', LABEL_COL)
    df=df.withColumn(LABEL_COL, f.when(f.col(LABEL_COL) == '>50K',
'True').otherwise('False'))
    return df.withColumn(LABEL_COL, df[LABEL_COL].cast(BooleanType()))

df = convert_label_col(df)

df.select(LABEL_COL).show(10)
```

Output:

```
+-----+
| >50K|
+-----+
|false|
|false|
|false|
|false|
|false|
|false|
|false|
| true|
| true|
| true|
+-----+
only showing top 10 rows
```

3.3.2 Handle Missing Data

Missing values have been replaced with a '?' character. These values may make our model less accurate.

```
def count_rows_with_missing_data(df: DataFrame):
    return df.select([f.sum(f.when(f.col(c) == '?',
1).otherwise(0)).alias(c) for c in df.columns])

count_rows_with_missing_data(df).show()
```

Output:

```
+---+---------+---------+--------------+----------+------------+----+---+------------+------------+-----
|age|workclass|education|marital-status|occupation|relationship|race|sex|capital-gain|capital-loss|hours
+---+---------+---------+--------------+----------+------------+----+---+------------+------------+-----
|  0|     1836|        0|             0|      1843|           0|   0|  0|           0|           0|
+---+---------+---------+--------------+----------+------------+----+---+------------+------------+-----
```

There are a few ways to deal with them:

- Remove entries with missing values (so long as this is not a significant number)
- Impute missing values (if continuous)

```
NUMERIC_COLUMNS = [item[0] for item in df.dtypes if
item[1].startswith('decimal')]
imputer = Imputer(
    inputCols=NUMERIC_COLUMNS,
    outputCols=["{}_imputed".format(c) for c in NUMERIC_COLUMNS]
)
df=imputer.fit(df).transform(df)
```

- Assign them a string value (in this case '?') and include them in the model

Note that none of our numeric columns contain missing data:

```
categoricalColumns = [item[0] for item in df.dtypes if
item[1].startswith('string')]
numericColumns = [item[0] for item in df.dtypes if
item[1].startswith('decimal')]

print(f'Numeric: {numericColumns}')
print(f'Categorical: {categoricalColumns}')

count_rows_with_missing_data(df).select(numericColumns).show()
```

Output:

```
Numeric: ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']
Categorical: ['workclass', 'education', 'marital-status', 'occupation',
'relationship', 'race', 'sex', 'native-country']
+---+-------------+------------+------------+--------------+
|age|education-num|capital-gain|capital-loss|hours-per-week|
+---+-------------+------------+------------+--------------+
|  0|            0|           0|           0|             0|
+---+-------------+------------+------------+--------------+
```

Therefore we do not need to impute these values.

Filtering out all the missing values, we can see the impact this has on the dataset:

```
def remove_rows_with_missing_categorical_data(df: DataFrame):
    for col in categoricalColumns:
        df=df.filter(df[col]!='?')
    return df

total_rows=df.count()
rows_with_missing_data=total_rows-
remove_rows_with_missing_categorical_data(df).count()
percentage=rows_with_missing_data*100/total_rows

print(percentage) // 7.37%
```

Removing these entries should be beneficial for our model but it is probably worth keeping them for other forms of analysis.

We shall export the data for further analysis and clean the empty values when training our model:

```
df.repartition(1).write.csv(CLEANED_DATA_OUT, header=True, sep=',',
mode='overwrite')
```

3.4 Exploring Data Characteristics

Data cane be grouped in various ways. Here we can see the distribution of instances based on sex, education, and country. We will explore these further in section 4:

```
print('Sex:')
df.groupby('sex').count().show()

print('Education:')
df.groupby('education').count().show()

print('Education_num:')
df.groupby('education-num').count().sort('education-num', ascending=False).show()
```

Output:

```
Sex:
+------+-----+
|   sex|count|
+------+-----+
|Female|10771|
|  Male|21790|
+------+-----+
```

```
Education:
+------------+-----+
|   education|count|
+------------+-----+
|     Masters| 1723|
|        10th|  933|
|     5th-6th|  333|
|  Assoc-acdm| 1067|
|   Assoc-voc| 1382|
|     7th-8th|  646|
|         9th|  514|
|     HS-grad|10501|
|   Bachelors| 5355|
|        11th| 1175|
|     1st-4th|  168|
|   Preschool|   51|
|        12th|  433|
|   Doctorate|  413|
|Some-college| 7291|
| Prof-school|  576|
+------------+-----+
```

```
Education_num:
+-------------+-----+
|education-num|count|
+-------------+-----+
|           16|  413|
|           15|  576|
|           14| 1723|
|           13| 5355|
|           12| 1067|
|           11| 1382|
|           10| 7291|
|            9|10501|
|            8|  433|
|            7| 1175|
|            6|  933|
|            5|  514|
|            4|  646|
|            3|  333|
|            2|  168|
|            1|   51|
+-------------+-----+
```

There are 42 distinct countries in the dataset:

```
distinct_countries_count=df.select(['native-country']).distinct().count()
print(f'Number of distinct countries: {distinct_countries_count}')
df.groupby('native-country').count().show(distinct_countries_count)
```

Output:

```
Number of distinct countries: 42
+--------------------+-----+
|      native-country|count|
+--------------------+-----+
|         Philippines|  198|
|             Germany|  137|
|            Cambodia|   19|
|              France|   29|
|              Greece|   29|
|              Taiwan|   51|
|             Ecuador|   28|
|           Nicaragua|   34|
|                Hong|   20|
|                Peru|   31|
|               India|  100|
|               China|   75|
|               Italy|   73|
|  Holand-Netherlands|    1|
|                Cuba|   95|
|               South|   80|
|                Iran|   43|
|             Ireland|   24|
|            Thailand|   18|
|                Laos|   18|
|         El-Salvador|  106|
|              Mexico|  643|
|           Guatemala|   64|
|            Honduras|   13|
|          Yugoslavia|   16|
|         Puerto-Rico|  114|
|             Jamaica|   81|
|              Canada|  121|
|       United-States|29170|
|  Dominican-Republic|   70|
|Outlying-US(Guam-...|   14|
|               Japan|   62|
|             England|   90|
|               Haiti|   44|
|              Poland|   60|
|            Portugal|   37|
|                   ?|  583|
|            Columbia|   59|
|            Scotland|   12|
|             Hungary|   13|
|             Vietnam|   67|
|     Trinadad&Tobago|   19|
+--------------------+-----+
```

This will be useful for creating a geolocation graph later on[4].

3.5 Preparing the Model

We are going to train a Logistic Regression modal to classify the data. In order to do that, each row must be converted into numeric vector data.

3.5.1 Irrelevant Data

fnlwgt (Final Weight) does not relate to the individual entry; rather it describes the proportion of the population that this entry represents.

We can also remove 'education' as this is already indexed in the column 'education-num'.

It is an artifact of duplicate entries having been removed; it has no use in our analysis and can be removed.

```
def remove_superfluous_cols(df: DataFrame):
    return df.drop('fnlwgt').drop('education')

df = remove_superfluous_cols(df)

df.show(5)
```

Figure 2 - Removal of redundant columns

Output:

```
+---+----------------+------------+------------------+----------------+------------+-----+------+------------+------------+-------
|age|       workclass|education-num|    marital-status|       occupation| relationship| race|   sex|capital-gain|capital-loss|hours-p
+---+----------------+------------+------------------+----------------+------------+-----+------+------------+------------+-------
| 39|       State-gov|          13|     Never-married|     Adm-clerical|Not-in-family|White|  Male|        2174|           0|
| 50|Self-emp-not-inc|          13|Married-civ-spouse|  Exec-managerial|      Husband|White|  Male|           0|           0|
| 38|         Private|           9|          Divorced|Handlers-cleaners|Not-in-family|White|  Male|           0|           0|
| 53|         Private|           7|Married-civ-spouse|Handlers-cleaners|      Husband|Black|  Male|           0|           0|
| 28|         Private|          13|Married-civ-spouse|   Prof-specialty|         Wife|Black|Female|           0|           0|
+---+----------------+------------+------------------+----------------+------------+-----+------+------------+------------+-------
only showing top 5 rows
```

3.5.2 Remove Unknow Values

As previously discussed, we will remove the rows with unknown text values which cannot be imputed.

We can also remove 'education' as this is already indexed in the column 'education-num'.

```
df_clean=remove_rows_with_missing_categorical_data(df)

count_rows_with_missing_data(df_clean).show()
```

Output:

```
+---+---------+-------------+--------------+----------+------------+----+---+------------+------------+-------------
|age|workclass|education-num|marital-status|occupation|relationship|race|sex|capital-gain|capital-loss|hours-per-wee
+---+---------+-------------+--------------+----------+------------+----+---+------------+------------+-------------
|  0|        0|            0|             0|         0|           0|   0|  0|           0|           0|             
+---+---------+-------------+--------------+----------+------------+----+---+------------+------------+-------------
```

3.5.3 Create Feature Vector Data

Now we can:

- convert all string values into numeric index values using a string indexer
- cast the Boolean label column as a binary integer value

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer

indexers = [StringIndexer(inputCol=column,
outputCol=column+"_indexed").fit(df) for column in categoricalColumns ]
pipeline = Pipeline(stages=indexers)


def index_strings(df: DataFrame):
    dfi=pipeline.fit(df).transform(df)
    return dfi.drop(*categoricalColumns).withColumn(LABEL_COL,
df[LABEL_COL].cast(IntegerType()))

df_indexed=index_strings(df_clean)
df_indexed.show(5)
```

Output:

```
+----+----------------+----------------+--------------------+----------------+------------------+-----------
|>50K|workclass indexed|education indexed|marital-status indexed|occupation indexed|relationship indexed|race indexe
+----+----------------+----------------+--------------------+----------------+------------------+-----------
|   0|             4.0|             2.0|                 1.0|             3.0|               1.0|         0.
|   0|             1.0|             2.0|                 0.0|             2.0|               0.0|         0.
|   0|             0.0|             0.0|                 2.0|             9.0|               1.0|         0.
|   0|             0.0|             5.0|                 0.0|             9.0|               0.0|         1.
|   0|             0.0|             2.0|                 0.0|             0.0|               4.0|         1.
+----+----------------+----------------+--------------------+----------------+------------------+-----------
```

3.5.4 Check for Redundant Features

Now that we indexed all our string values, we can create a correlation matrix between each feature.

If two features have a high correlation, we can view one or more of these as redundant and remove it from our machine learning model.

```
import matplotlib.pyplot as plt
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.stat import Correlation


corr_graph=df_indexed.drop(LABEL_COL)

columns = corr_graph.columns
print(columns)
vector_col = "corr_features"
assembler = VectorAssembler(inputCols=corr_graph.columns,
                            outputCol=vector_col)
corr_graph_vector = assembler.transform(corr_graph).select(vector_col)
matrix = Correlation.corr(corr_graph_vector, vector_col)

matrix = Correlation.corr(corr_graph_vector, vector_col).collect()[0][0]
corrmatrix = matrix.toArray().tolist()
# print(corrmatrix)

df = spark.createDataFrame(corrmatrix,columns)

def plot_corr_matrix(correlations,attr,fig_no):
    fig=plt.figure(figsize=(15,15))
    ax=fig.add_subplot(111)
    ax.set_title("Correlation Matrix")
    ax.set_xticks(np.arange(len(attr)))
    ax.set_yticks(np.arange(len(attr)))
    ax.set_xticklabels(attr, rotation=90, fontsize=10)
    ax.set_yticklabels(attr, fontsize=10)
    cax=ax.imshow(correlations,vmax=1,vmin=-1)
    fig.colorbar(cax)
    plt.show()

plot_corr_matrix(corrmatrix, columns, 234)
```

Output:

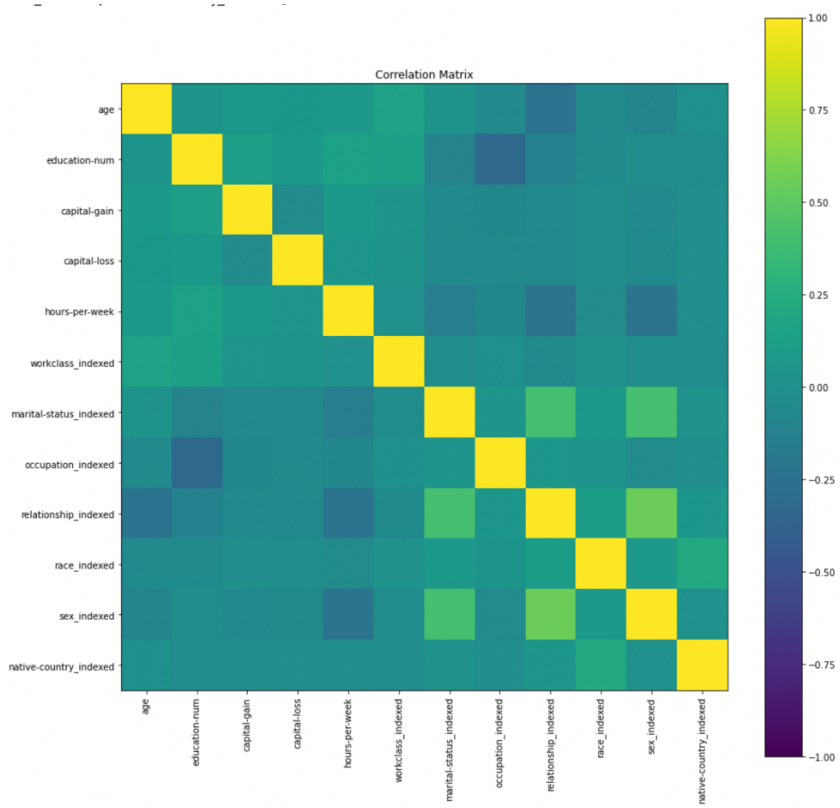


Figure 3 - Feature Correlation Heat Map

As Figure 3 shows, there are a couple of higher correlating features, such as marital-status and relationship, but nothing so striking to warrant removing features from the dataset.

3.5.5 Convert to Features/Label Columns

Before we can build a Logistical Regression model, we must convert our feature columns into a vector.

```
from pyspark.ml.feature import VectorAssembler, Normalizer
VECTOR_COL='features'

features=list(df_indexed.drop(LABEL_COL).toPandas().columns)

assembler=VectorAssembler(inputCols=features, outputCol=VECTOR_COL)

def create_feature_vector(df):
    df_vector = assembler.transform(df)
    return df_vector.select(*[VECTOR_COL, LABEL_COL])

df_vector = create_feature_vector(df_indexed)
df_vector.show(5)
```

Output:

```
+--------------------+----+
|            features|>50K|
+--------------------+----+
|[39.0,13.0,2174.0...|   0|
|(13,[0,1,4,5,6,8]...|   0|
|(13,[0,1,4,7,8,9]...|   0|
|(13,[0,1,4,6,8,10...|   0|
|[28.0,13.0,0.0,0....|   0|
+--------------------+----+
only showing top 5 rows
```

3.5.6 Normalise Vectors

We will normalize the feature vectors using L^1 norm (Weisstein,)

```
normalizer =
Normalizer().setInputCol(VECTOR_COL).setOutputCol(f'{VECTOR_COL}_n').setP(1.0)


def normalize_vector(df_vector):
    df_vector = normalizer.transform(df_vector)
    return df_vector.drop(VECTOR_COL).withColumnRenamed(f'{VECTOR_COL}_n',VECTOR_COL)

df_vector = normalize_vector(df_vector)
df_vector.show(5)
```

Output:

```
+----+--------------------+
|>50K|            features|
+----+--------------------+
|   0|[0.01712779973649...|
|   0|(13,[0,1,4,5,6,8]...|
|   0|(13,[0,1,4,7,8,9]...|
|   0|(13,[0,1,4,6,8,10...|
|   0|[0.28571428571428...|
+----+--------------------+
only showing top 5 rows
```

3.6 CLASSIFICATION

Logical Regression adapted from example in Spark documentation: https://spark.apache.org/docs/3.0.1/ml-pipeline.html

Configure Logistical Regression Instance

```
from pyspark.ml.classification import LogisticRegression

# Create a LogisticRegression instance. This instance is an Estimator.
lr = LogisticRegression(maxIter=10, regParam=0.01, featuresCol=VECTOR_COL,
labelCol=LABEL_COL)
# Print out the parameters, documentation, and any default values.
print(f"LogisticRegression parameters:")
lr.explainParams()
```

Output:

"aggregationDepth: suggested depth for treeAggregate (>= 2). (default: 2)
elasticNetParam: the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty. (default: 0.0)
family: The name of family which is a description of the label distribution to be used in the model. Supported options: auto, binomial, multinomial (default: auto)
featuresCol: features column name. (default: features, current: features)
fitIntercept: whether to fit an intercept term. (default: True)
labelCol: label column name. (default: label, current: >50K)
lowerBoundsOnCoefficients: The lower bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression. (undefined)
lowerBoundsOnIntercepts: The lower bounds on intercepts if fitting under bound constrained optimization. The bounds v ector size must beequal with 1 for binomial regression, or the number oflasses for multinomial regression. (undefined)
maxBlockSizeInMB: maximum memory in MB for stacking input data into blocks. Data is stacked within partitions. If more than remaining data size in a partition then it is adjusted to the data size. Default 0.0 represents choosing optimal value, de pends on specific algorithm. Must be >= 0. (default: 0.0)
maxIter: max number of iterations (>= 0). (default: 100, current: 10)
predictionCol: prediction column name. (default: prediction)
probabilityCol: Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated pro bability estimates! These probabilities should be treated as confidences, not precise probabilities. (default: probability)
rawPredictionCol: raw prediction (a.k.a. confidence) column name. (default: rawPrediction)
regParam: regularization parameter (>= 0). (default: 0.0, current: 0.01)
standardization: whether to standardize the training features before fitting the model. (default: True)
threshold: Threshold in binary classification prediction, in range [0, 1]. If threshold and thresholds are both set, they must match.e.g. if threshold is p, then thresholds must be equal to [1-p, p]. (default: 0.5)
thresholds: Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have lengt h equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/ t is predicted, where p is the original probability of that class and t is the class's threshold. (undefined)
tol: the convergence tolerance for iterative algorithms (>= 0). (default: 1e-06)
upperBoundsOnCoefficients: The upper bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression. (undefined)
upperBoundsOnIntercepts: The upper bounds on intercepts if fitting under bound constrained optimization. The bound ve ctor size must be equal with 1 for binomial regression, or the number of classes for multinomial regression. (undefined)
weightCol: weight column name. If this is not set or empty, we treat all instance weights as 1.0. (undefined)"

3.6.1 Train the Model

Train the model using the training data we prepared earlier:

```
training_data=df_vector
# Learn a LogisticRegression models. This uses the parameters stored in lr.
model1 = lr.fit(training_data)
print("Model 1 was fit using parameters: ")
model1.extractParamMap()
```

Output:

```
{Param(parent='LogisticRegression 484326172f3e', name='aggregationDepth', doc='suggested depth for
treeAggregate (>= 2).'): 2,
 Param(parent='LogisticRegression 484326172f3e', name='elasticNetParam', doc='the ElasticNet mixing
parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1
penalty.'): 0.0,
 Param(parent='LogisticRegression 484326172f3e', name='family', doc='The name of family which is a
description of the label distribution to be used in the model. Supported options: auto, binomial,
multinomial'): 'auto',
 Param(parent='LogisticRegression 484326172f3e', name='featuresCol', doc='features column name.'):
'features',
 Param(parent='LogisticRegression 484326172f3e', name='fitIntercept', doc='whether to fit an intercept
term.'): True,
 Param(parent='LogisticRegression 484326172f3e', name='labelCol', doc='label column name.'): '>50K',
 Param(parent='LogisticRegression 484326172f3e', name='maxBlockSizeInMB', doc='maximum memory in MB for
stacking input data into blocks. Data is stacked within partitions. If more than remaining data size in a
partition then it is adjusted to the data size. Default 0.0 represents choosing optimal value, depends on
specific algorithm. Must be >= 0.'): 0.0,
 Param(parent='LogisticRegression 484326172f3e', name='maxIter', doc='max number of iterations (>= 0).'): 10,
 Param(parent='LogisticRegression 484326172f3e', name='predictionCol', doc='prediction column name.'):
'prediction',
 Param(parent='LogisticRegression 484326172f3e', name='probabilityCol', doc='Column name for predicted class
conditional probabilities. Note: Not all models output well-calibrated probability estimates! These
probabilities should be treated as confidences, not precise probabilities.'): 'probability',
 Param(parent='LogisticRegression_484326172f3e', name='rawPredictionCol', doc='raw prediction (a.k.a.
confidence) column name.'): 'rawPrediction',
 Param(parent='LogisticRegression 484326172f3e', name='regParam', doc='regularization parameter (>= 0).'):
0.01,
 Param(parent='LogisticRegression 484326172f3e', name='standardization', doc='whether to standardize the
training features before fitting the model.'): True,
 Param(parent='LogisticRegression 484326172f3e', name='threshold', doc='Threshold in binary classification
prediction, in range [0, 1]. If threshold and thresholds are both set, they must match.e.g. if threshold is
p, then thresholds must be equal to [1-p, p].'): 0.5,
 Param(parent='LogisticRegression 484326172f3e', name='tol', doc='the convergence tolerance for iterative
algorithms (>= 0).'): 1e-06}
```

3.6.2 Prepare the Test Data

In a separate file, additional values, with their classifications, are stored.

We will load this data and prepare it as we did our training data:

```
test_data=load_data(SOURCE_TEST_DATA)
test_data=remove_superfluous_cols(test_data)
test_data=convert_label_col(test_data)
test_data=index_strings(test_data)
test_data=create_feature_vector(test_data)
test_data=normalize_vector(test_data)

test_data.show(5)
```

Output:

```
+----+--------------------+
|>50K|            features|
+----+--------------------+
|   0|[0.28735632183908...|
|   0|(13,[0,1,4,8],[0....|
|   1|(13,[0,1,4,5,6,8]...|
|   1|(13,[0,1,2,4,6,8,...|
|   0|[0.24657534246575...|
+----+--------------------+
```

3.6.3 Make Predictions

Finally we can use our model to make predictions:

```
model=model1
# Make predictions on test data using the Transformer.transform() method.
# LogisticRegression.transform will only use the 'features' column.
prediction = model.transform(test_data)

prediction.select(['>50K', 'prediction' ]).show()
```

Output:

```
+----+----------+
|>50K|prediction|
+----+----------+
|   0|       0.0|
|   0|       0.0|
|   1|       0.0|
|   1|       1.0|
|   0|       0.0|
|   0|       0.0|
|   0|       0.0|
|   1|       1.0|
|   0|       0.0|
|   0|       0.0|
|   1|       1.0|
|   0|       1.0|
|   0|       0.0|
|   0|       0.0|
|   1|       1.0|
|   1|       0.0|
|   0|       0.0|
|   0|       0.0|
|   0|       0.0|
|   1|       0.0|
+----+----------+
```

3.6.4 Evaluate the Model

Finally, we can evaluate the success of our model with a classification evaluator:

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# obtain evaluator.
evaluator = MulticlassClassificationEvaluator(metricName='accuracy',
labelCol=LABEL_COL)
# compute the classification error on test data.
accuracy = evaluator.evaluate(prediction)

if not evaluator.isLargerBetter():
    accuracy = 1 - accuracy

print("Test Accuracy = %g" % accuracy)
```

Output:

```
Test Accuracy = 0.800135
```

4 Data Visualisation with Tableau

The data contains information that can be interpreted and extrapolated through visualisation.

4.1 Additional Measures

To help with visualisation, some additional measures will be created from existing data:

- % >50K
- Longitude/Latitude

4.1.1 Percentage earning more than $50K

Given that the primary task for the dataset was to categorisation based on the '>50k' column, we shall create a new value in Tableau called '%>50K'.

This will allow us to visualise the initial categorical hypothesis against various population groups.

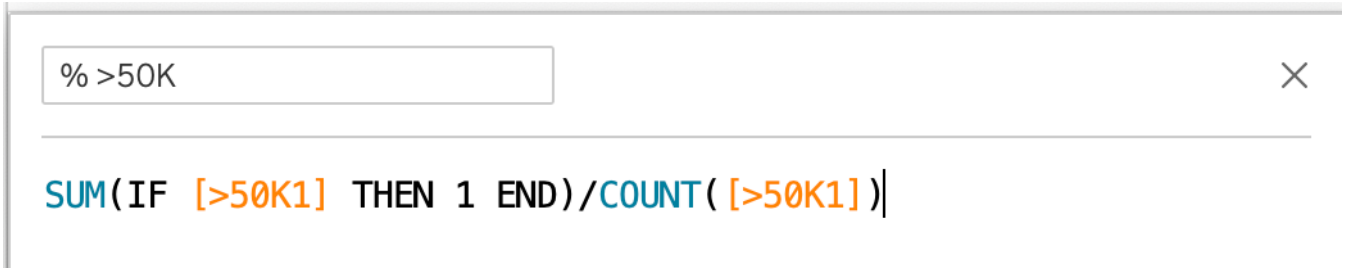


Figure 4 - '%>50K' Calculated Field

The new calculated field is added using the GUI in Figure 4.

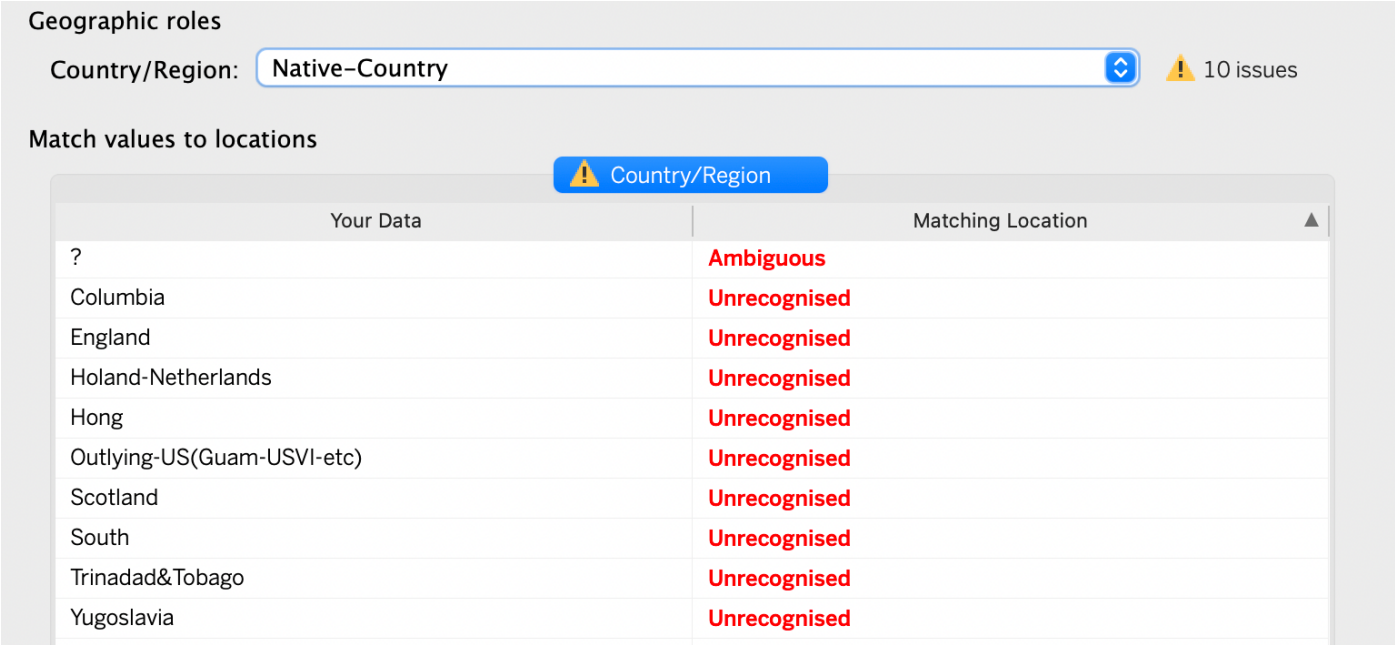
4.1.2 Applying Weight

When looking at proportional values, it is important to consider the population that each instance represents. For this reason the fnlwgt column can be applied to proportional values (Figure 5):

Figure 5 - %>50K fnlwgt

4.2 Geographical Data

Tableau can automatically compute longitude and latitude values based on the column 'native-country', with a few remaining issues:



There are a few different issues here:

- Spelling mistakes (e.g. Colu**m**bia)
- Unexpected names/characters (e.g. Holand-Netherlands, Trinadad&Tobago)
- Missing words: (e.g. Hong [kong], South)
- Country sub-regions that can be grouped together (i.e. England, Scotland)
- Former countries that have since changed their name and borders (i.e. Yugoslavia)

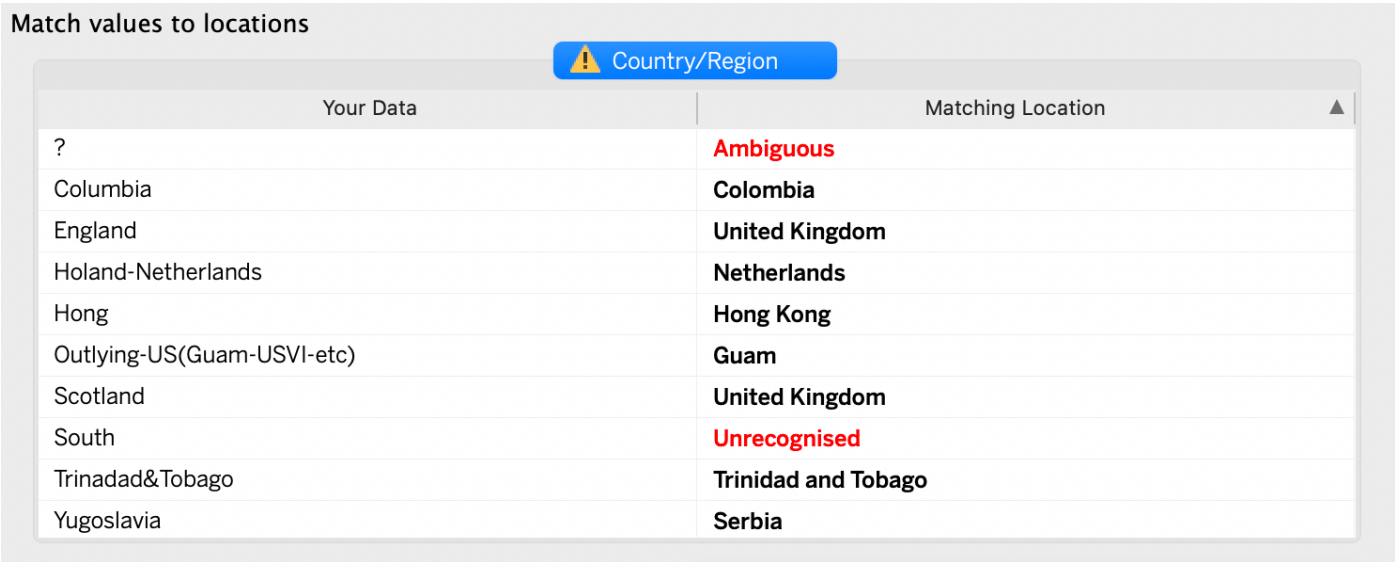


Figure 6 - Region corrections applied

The corrections above should suffice the mapping of Yugoslavia to Serbia is not totally accurate since Yugoslavia was broken up into several smaller nations. For our analysis, this is OK but should be noted in case it becomes relevant.

Missing values, or undeterminable values will have to be ignored.

4.2.1 Percentage earning > 50K by Country

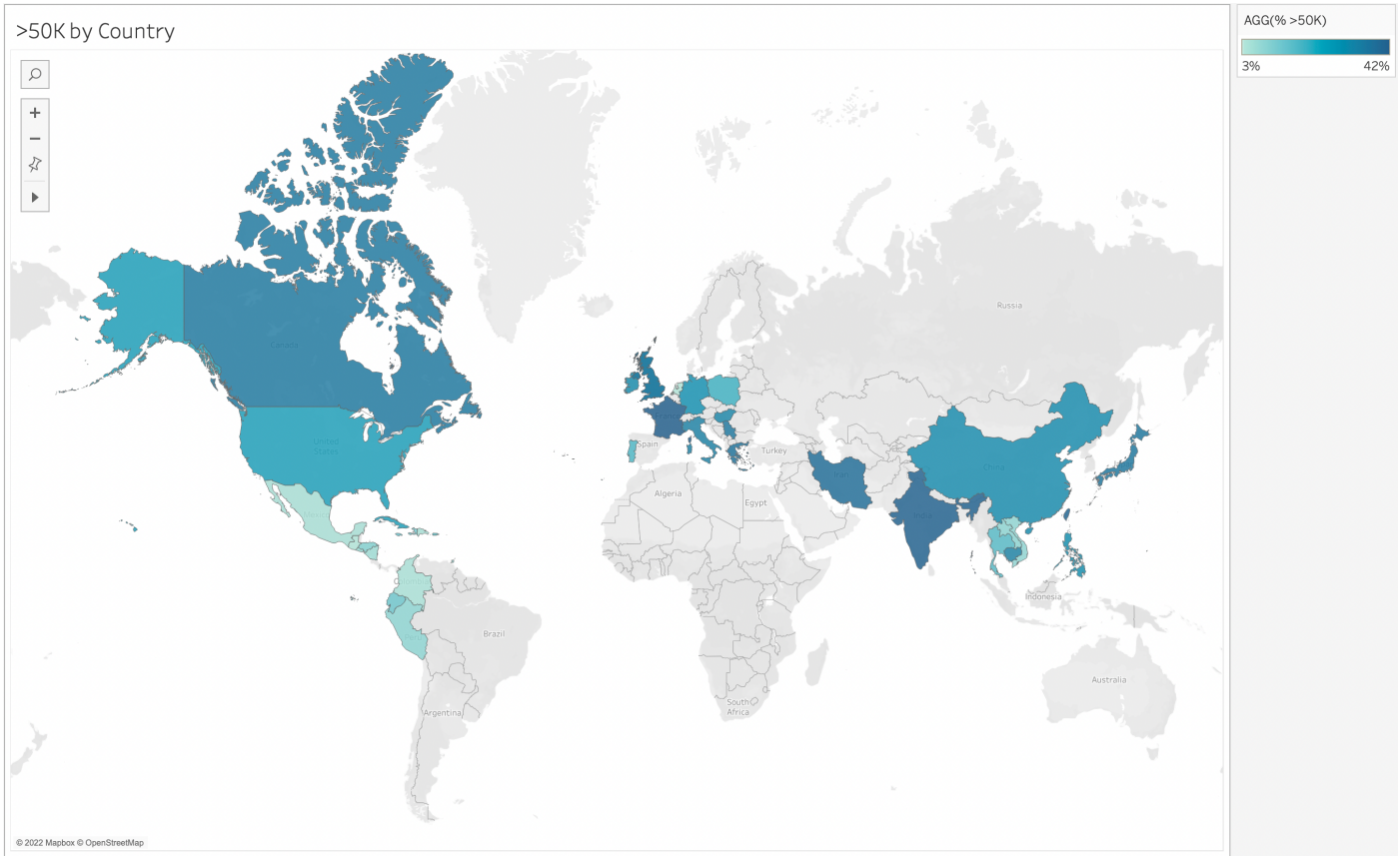


Figure 7 - Percentage earning above $50K by Native Country

The map in Figure 7 shows the percentage of individuals earning more than $50K in each native country, with darker hues indicating a higher percentage. There are some simple observations that we can make here:

- Earnings from many foreign nations appear to higher than within the US
- The majority of foreign nations with lower earnings are on the US' southern boarder
- Notable exceptions being Vietnam and the surrounding areas.

Looking at Figure 8, we can confirm this to be true.

4.2.2 Interpretation and Historical Significance

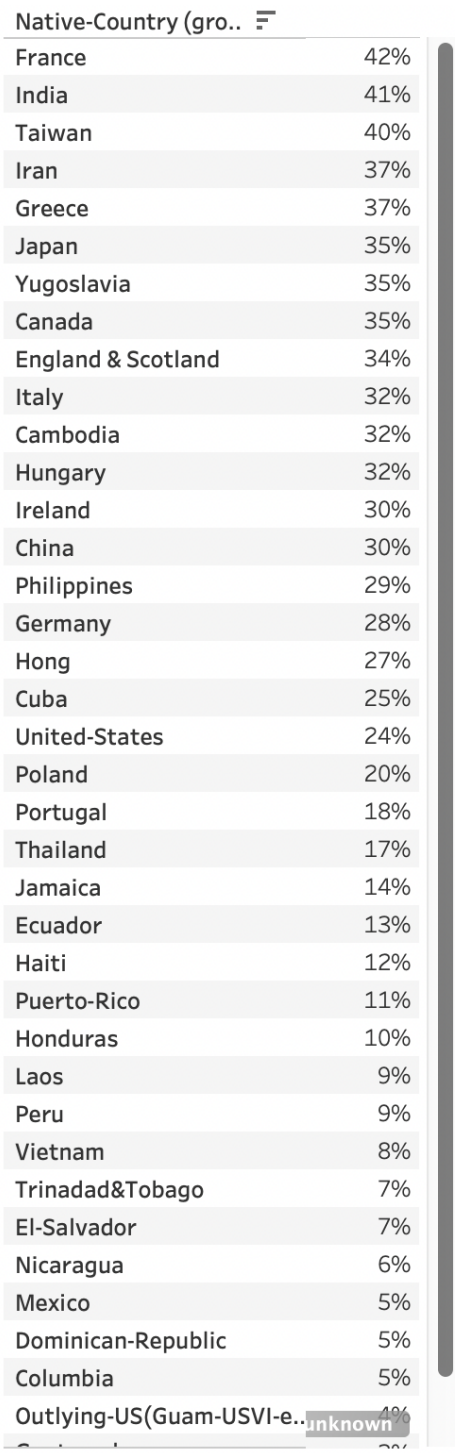
The United States of America has traditionally been a nation of immigrants. However its policy of immigration has changed significantly over time.

In modern America, immigration is largely controlled. Immigrants would have to meet certain economic or educational requirements to be eligible for a 'Green Card'. Therefore we might expect this to translate to the higher earnings displayed in Figure 7.

Changes to immigration policy in the 1960s resulted in increased migration from Asia and Latin America, many of whom without authorisation (Massey & Pren, 2012) .

Following the same period, many immigrants arrived as refugees and asylum seekers from countries like Vietnam and other struggling nations (*Vietnamese immigrants in the united states* .).

Without drawing any firm conclusions, it is apparent that the map in Figure 7 can be placed in a historical context to reinforce a particular narrative.



Native-Country (gro..	
France	42%
India	41%
Taiwan	40%
Iran	37%
Greece	37%
Japan	35%
Yugoslavia	35%
Canada	35%
England & Scotland	34%
Italy	32%
Cambodia	32%
Hungary	32%
Ireland	30%
China	30%
Philippines	29%
Germany	28%
Hong	27%
Cuba	25%
United-States	24%
Poland	20%
Portugal	18%
Thailand	17%
Jamaica	14%
Ecuador	13%
Haiti	12%
Puerto-Rico	11%
Honduras	10%
Laos	9%
Peru	9%
Vietnam	8%
Trinadad&Tobago	7%
El-Salvador	7%
Nicaragua	6%
Mexico	5%
Dominican-Republic	5%
Columbia	5%
Outlying-US(Guam-USVI-e..	4%

Figure 8 – Native Countries ordered by %>50K

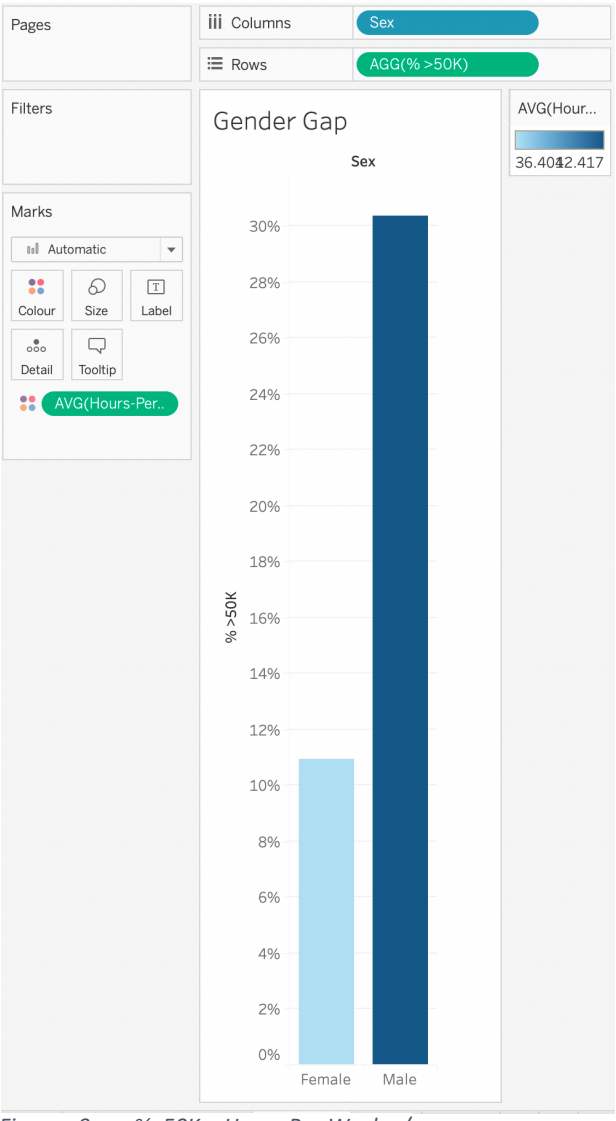


Figure 9 - %>50K, Hours-Per-Week / Gender

4.3 The Gender Pay Gap

Figure 9 shows the disparity in wages between men and women. To provide additional context, the mean hours worked has been overlayed as a colour gradient.

On first look, it would appear that there is a correlation between the number of hours worked and earnings. However, this is misleading as the difference is very minor in reality, making this a particularly misleading use of gradient.

If we use a Gnatt Bar with median distribution for hours worked (Figure 10), we can clearly see that the difference is minor compared with the disparity in wages.

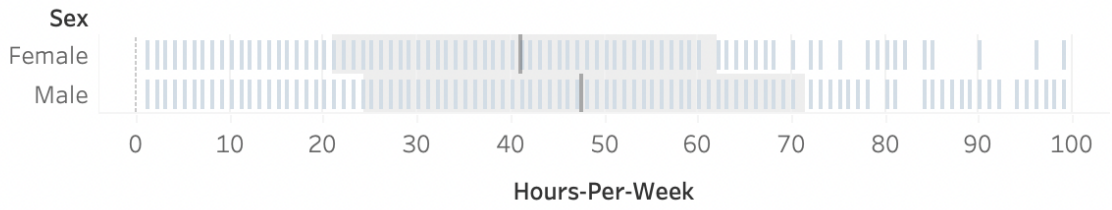


Figure 10 - Hours Worked M/F with Median Distribution

4.4 Race Pay Gap

Ordered by %>50K, Figure 11 demonstrates the disparity between the wages of different ethnicities. As in section 4.3, salary has been contrasted against hours worked to clarify the disconnect between them.

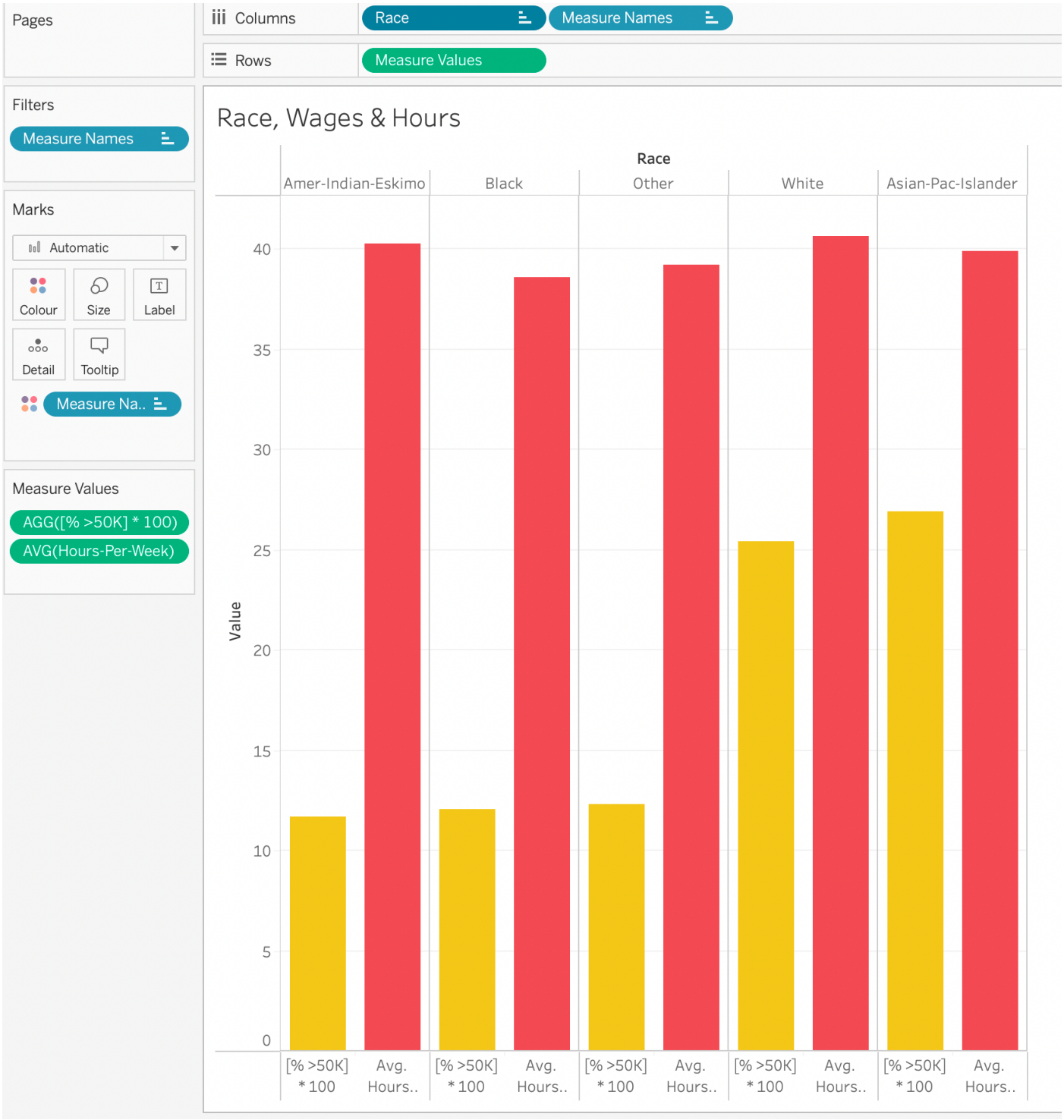


Figure 11 - Wages and Race

4.5 EDUCATION VS. WAGES

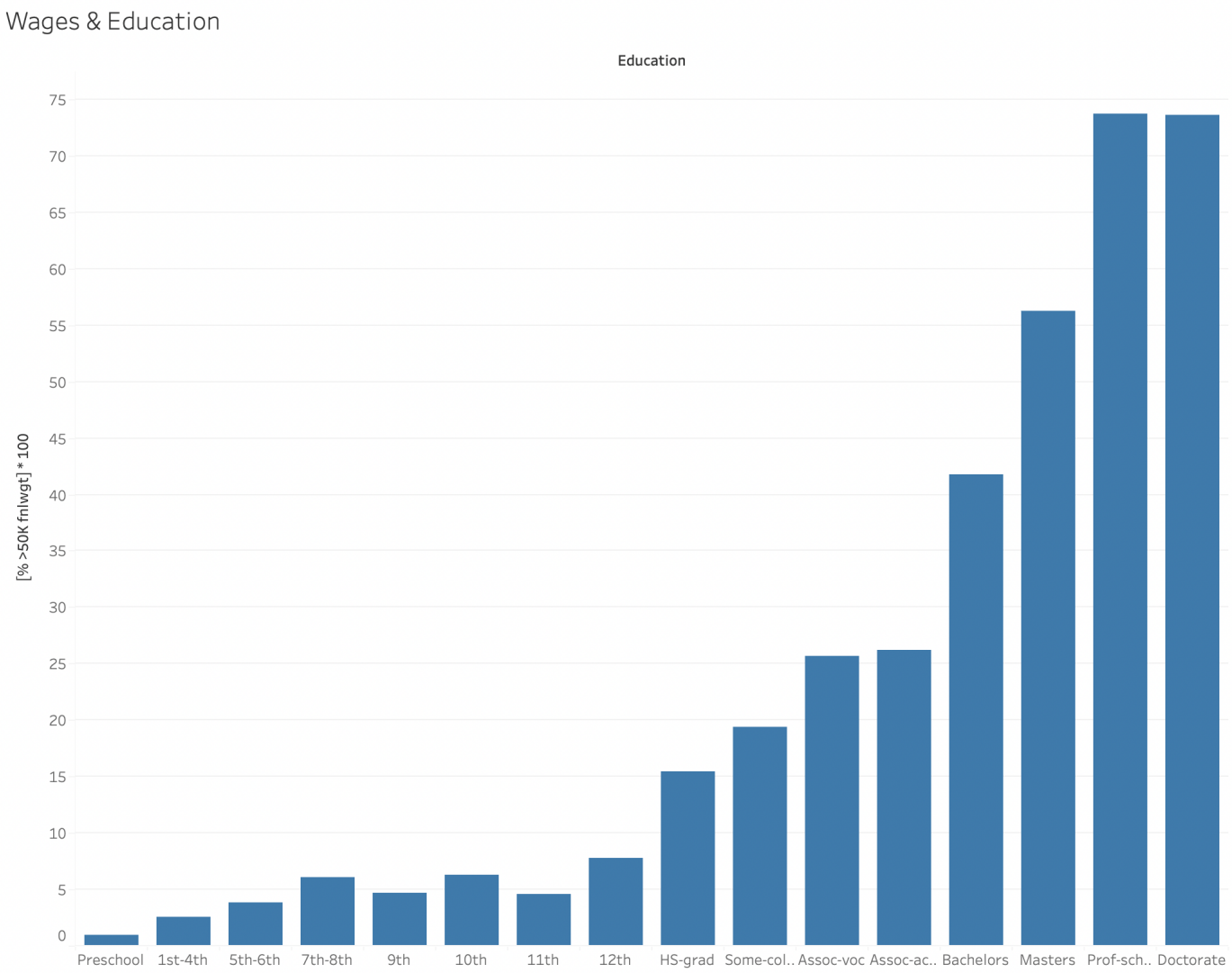


Figure 12 - Education vs. Wages (ordered by education-num)

The level of education plays a key role in whether an individual earns more than $50K. Figure 12 displays the percentage of individuals earning more than $50K, with columns ordered by 'education-num'. Figure 13 shows this same information in tabular form and size.

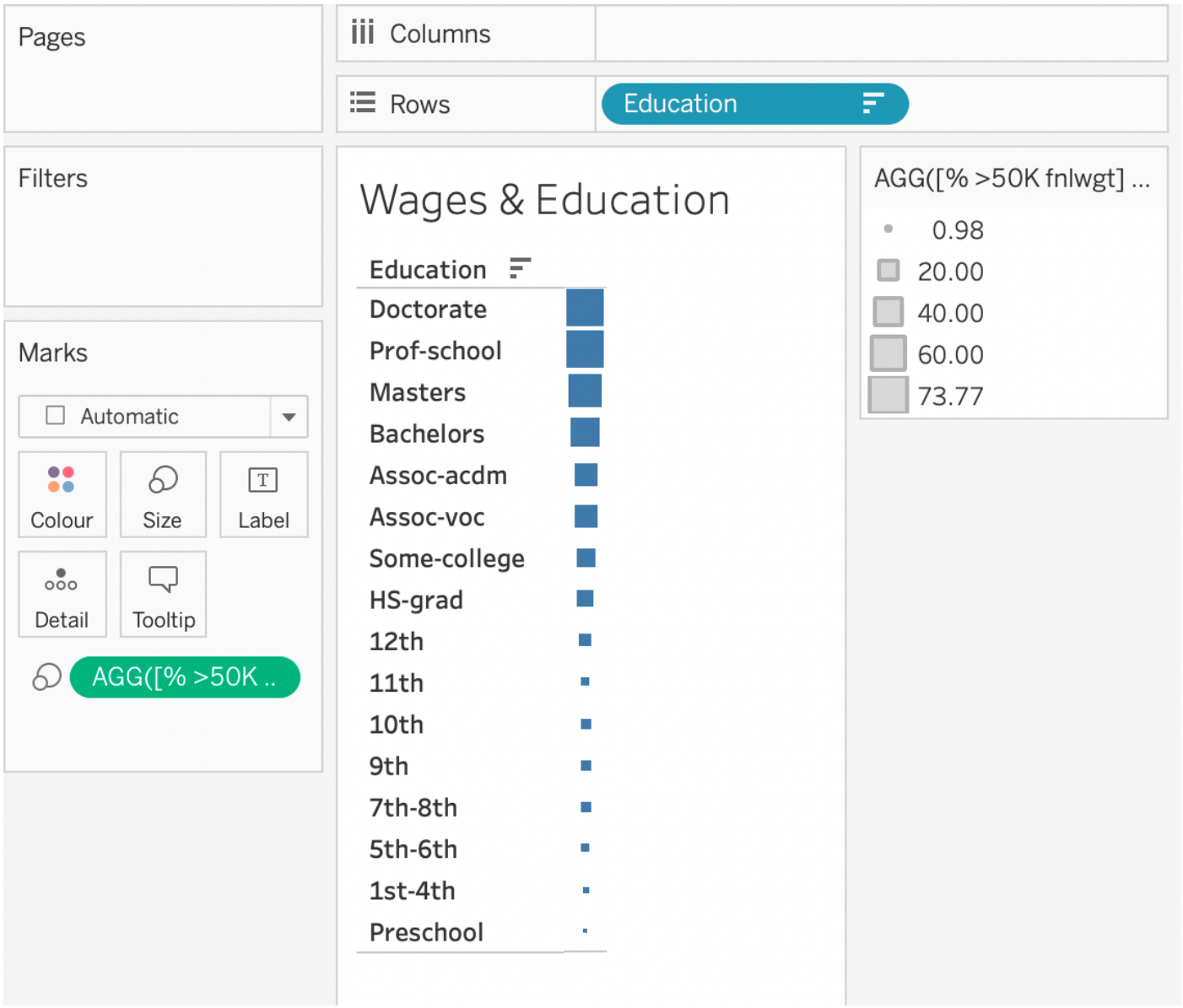


Figure 13 - Education vs. Wages - size table

5 DISCUSSION

The analytics performed with PySpark[Section 3] confirmed that it was indeed possible to make predictions about the individuals' earnings. With this in mind, a selection of features were visually examined [Section 4] based on assumptions about race, ethnicity, education, and sex.

Based on the analysis made in section 4, we may be inclined to view the following features as deterministic:

- Race
- Sex
- Education
- Native country

We can test this hypothesis very easily by making a single alteration to our classification code[3.6]. We simply change the function in Figure 2 to the following:

```
def remove_superfluous_cols(df: DataFrame):
    return df.select(['sex', 'race', 'native-country', 'education-num', LABEL_COL])
```

When we run all our code again, we have a new classification model with a different result: an accuracy of ~76%; nearly 4% less than our original model. Therefore it is clear that we must be careful when making assumptions about which attributes are more or less deterministic.

Conversely, we can alter the function to remove these columns:

```
def remove_superfluous_cols(df: DataFrame):
    return df.drop('fnlwgt').drop('education').drop('sex').drop('race').drop('native-
country').drop('education-num')
```

Interestingly we see a higher result, approximately the same as the original. However, this does not mean that not all attributes are valid; it is more likely that Logistical Regression was the wrong technique for classification of this data.

As expected with big data and machine learning algorithms, the classification is neither simple nor intuitive. While it is possible to make observations based on comparing and contrasting features, making accurate predictions resides within the realm of machine learning; where big data is involved.

6 CONCLUSION

A classification accuracy of ~80% would not usually be considered sufficient. While a 100% accuracy is unrealistic (Valverde-Albacete & Peláez-Moreno, 2014), a good benchmark would be greater or equal to 90% (*Failure of classification accuracy for imbalanced class distributions.*).

Logistic Regression is good for categorical data (Which machine learning algorithm should I use?) but, if the purpose of this assignment were to make the most accurate model, more time might be invested in exploring different techniques such as decision trees, which may yield better results.

The dataset itself provided sufficient challenge withing the classification task however more could have been made of the visualisation if it contained a wider variety of data types, such as date and time.

However it became apparent that visualisation is extremely powerful, making it extremely dangerous. Tableau makes light work of creating powerful visuals that can support a narrative and provide powerful persuasive ammunition. Because of this it is extremely easy to, either by accident or design, create misleading or subversive visualisations that can be harmful to society. This was demonstrated in section 4.3.

Also apparent is the importance of avoiding bias in machine learning (Kumar et al., 2020). At any stage of the process, it is essential that human assumptions do not drive data preparation.

APPENDIX

6.1 REPOSITORY

Repository: https://github.com/maxbilbow/7082CEM-big-data-visualisation

```
7082CEM-big-data-visualisation
├ app
│ ├ data
│ │ ├ income-predictor
│ │ │ ├ Index
│ │ │ ├ README.md
│ │ │ ├ adult.data
│ │ │ ├ adult.data.clean.csv
│ │ │ ├ adult.names
│ │ │ ├ adult.test
│ │ │ ├ attributes.csv
│ │ │ └ old.adult.names
│ ├ LogisticRegression.ipynb
│ ├ LogisticRegression.py
│ ├ Reduced-features.ipynb
│ ├ Reduced-features.py
│ ├ Reduced-features2.ipynb
│ ├ Reduced-features2.py
├ spark
│ ├ Dockerfile
│ ├ log4j.properties
│ └ requirements.txt
├ .gitignore
├ 50K Visualisation.twb
├ LICENSE
├ README.md
└ docker-compose.yml
```

6.2 Docker Setup

6.2.1 Spark Container

./spark/Dockerfile

```
# Based on ubunto Docker image
FROM ubuntu:latest as builder

# Install necessary packages
RUN apt-get update
RUN apt-get install -y wget
RUN apt-get install -y default-jdk-headless
RUN apt-get install -y pip

# Specify environment variables
ENV SPARK_VERSION=3.2.1
ENV HADOOP_VERSION=3.2
ENV SPARK_HOME=/opt/spark
ENV JAVA_HOME=/usr/lib/jvm/default-java
ENV PATH=${JAVA_HOME}/bin:${SPARK_HOME}/bin:${SPARK_HOME}/sbin:${PATH}

# Download and install Apache Spark
RUN wget --no-verbose -O apache-spark.tgz
"https://archive.apache.org/dist/spark/spark-${SPARK_VERSION}/spark-
${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz" \
  && mkdir -p /opt/spark \
  && tar -xf apache-spark.tgz -C /opt/spark --strip-components=1 \
  && rm apache-spark.tgz

# Create working directory
RUN mkdir /workspace

FROM builder as apache-spark

# Copy log4j configuration into spark config folder
COPY log4j.properties /opt/spark/conf/log4j.properties

# Set starting directory
WORKDIR /workspace

# Copy Python requirements file
COPY requirements.txt requirements.txt

# Install requirements
RUN pip3 install -r requirements.txt

# Expose ports
EXPOSE 4040 6066 7077

# Ensure container remains active
ENTRYPOINT ["tail", "-f", "/dev/null"]
```

6.2.2 Logging Config

./spark/log4j.properties (Based on the template file found in the Apache Spark config folder)

```
# Set everything to be logged to the console
log4j.rootCategory=WARN, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p
%c{1}: %m%n

# Set the default spark-shell log level to WARN. When running the spark-
shell, the
# log level for this class is used to overwrite the root logger's log
level, so that
# the user can have different defaults for the shell and regular Spark
apps.
log4j.logger.org.apache.spark.repl.Main=WARN

# Settings to quiet third party logs that are too verbose
log4j.logger.org.spark_project.jetty=WARN
log4j.logger.org.spark_project.jetty.util.component.AbstractLifeCycle=ERROR
log4j.logger.org.apache.spark.repl.SparkIMain$exprTyper=INFO
log4j.logger.org.apache.spark.repl.SparkILoop$SparkILoopInterpreter=INFO
log4j.logger.org.apache.parquet=ERROR
log4j.logger.parquet=ERROR

# SPARK-9183: Settings to avoid annoying messages when looking up
nonexistent UDFs in SparkSQL with Hive support
log4j.logger.org.apache.hadoop.hive.metastore.RetryingHMSHandler=FATAL
log4j.logger.org.apache.hadoop.hive.ql.exec.FunctionRegistry=ERROR
```

6.2.3 Python Dependencies

./spark/requirements.txt

```
pyspark==3.2.1
numpy==1.22.2
pandas==1.4.1
```

6.2.4 docker-compose.xml

```
version: "1"
services:
  jupyter:
    container_name: bdv_notebook
    image: jupyter/pyspark-notebook # Docker image to download and run
    ports:
      - 8888:8888
    volumes: # Map local app folder to notebook container
      - ./app:/home/jovyan/work
  spark:
    container_name: bdv_spark
    build: ./spark # Location of custom docker image to build and run
    ports:
      - 4040:4040
      - 7077:7077
      - 6066:6066
    volumes: # Map local app folder to spark container
      - ./app:/workspace
```

6.3 LogisticRegression.py

```
#!/usr/bin/env python
# coding: utf-8

# # Prepare the Data
# ## Set Constants
# Set constants

# In[1]:


import os

ROOT = r"./"
DATA_IN = os.path.join(ROOT, "data", "income-predictor")
SOURCE_DATASET = os.path.join(DATA_IN ,"adult.data")
SOURCE_ATTRIBUTES = os.path.join(DATA_IN, "attributes.csv")
SOURCE_TEST_DATA = os.path.join(DATA_IN, "adult.test")
COMBINED_DATA_OUT = f'{SOURCE_DATASET}.full'

LABEL_COL='>50K'


print(SOURCE_DATASET)
print(SOURCE_ATTRIBUTES)
print(SOURCE_TEST_DATA)


# ## Instantiate Spark Session

# In[2]:


from pyspark.sql import SparkSession, DataFrame

spark=SparkSession     .builder     .appName("Income Predictor")
.getOrCreate()


# ## Create Schema
# An attributes file was created from the [data's
decription](http://archive.ics.uci.edu/ml/datasets/Census+Income)

# In[3]:


from pyspark.sql.types import StructType, StructField, IntegerType,
DecimalType, StringType, BooleanType

attrs = spark.read.csv(SOURCE_ATTRIBUTES, header=True)
attrs.show()

def get_field(t: str):
    if t == "continuous":
        return DecimalType()
    elif t == "string":
        return StringType()
    elif t == "boolean":
        return BooleanType()
    else:
        raise Exception("Not expected: %s" % t)
```

7 REFERENCES

References

Apache spark. http://spark.apache.org/

Census income data set. http://archive.ics.uci.edu/ml/datasets/Census+Income

Docker. https://www.docker.com/

Failure of classification accuracy for imbalanced class distributions. https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/

Homebrew. https://brew.sh

Jupyter/pyspark-notebook docker image. https://hub.docker.com/r/jupyter/pyspark-notebook

Kumar, M., Roy, R., & Oden, K. D. (2020). Identifying bias in machine learning algorithms: CLASSIFICATION WITHOUT DISCRIMINATION. *The RMA Journal, 103*(1), 42-48. https://search.proquest.com/docview/2438199688

Massey, D. S., & Pren, K. A. (2012). Unintended consequences of US immigration policy: Explaining the post-1965 surge from latin america. *Population and Development Review, 38*(1), 1-29. https://doi.org/10.1111/j.1728-4457.2012.00470.x

Milligan, J. N. (2020). *Learning tableau 2020* (Fourth edition ed.). Packt Publishing.

Ubuntu docker image. https://hub.docker.com/_/ubuntu

Valverde-Albacete, F. J., & Peláez-Moreno, C. (2014). 100% classification accuracy considered harmful: The normalized information transfer factor explains the accuracy paradox. *PloS One, 9*(1), e84217. https://doi.org/10.1371/journal.pone.0084217

Vietnamese immigrants in the united states . https://www.migrationpolicy.org/article/vietnamese-immigrants-united-states

VirtualBox. https://www.virtualbox.org

VMWare. https://www.vmware.com/uk/products/workstation-player/workstation-player-evaluation.html

Weisstein, E.*L^1-norm.* https://mathworld.wolfram.com/L1-Norm.html

Which machine learning algorithm should I use? https://blogs.sas.com/content/subconsciousmusings/2020/12/09/machine-learning-algorithm-use/