notebook

February 21, 2020

1 Dataiku technical test

Interviewee: Vincent Barbosa Vaz

Dataiku contacts: Judith Müller, Josh Cooper and Adam Jelley

2 Table of contents

3 Introduction

3.1 Instructions

The following link lets you download an archive containing an "exercise" US Census dataset: http://thomasdata.s3.amazonaws.com/ds/us_census_full.zip

This US Census dataset contains detailed but anonymized information for approximately 300,000 people.

The archive contains 3 files:

1. A large training file (csv)
2. Another test file (csv)

3. A metadata file (txt) describing the columns of the two csv files (identical for both)

The goal of this exercise is to model the information contained in the last column (42nd), i.e., whether a person makes more or less than $50,000 per year, from the information contained in the other columns. The exercise here consists of modeling a binary variable.

Work with Python (or R) to carry out the following steps:

1. Load the train and test files.
2. Perform an exploratory analysis on the data and create some relevant visualisations.
3. Clean, preprocess, and engineer features in the training data, with the aim of building a data set that a model will perform well on.
4. Create a model using these features to predict whether a person earns more or less than $50,000 per year. Here, the idea is for you to test a few different models, and see whether there are any techniques you can apply to improve performance over your first results.
5. Choose the model that appears to have the highest performance based on a comparison between reality (the 42nd variable) and the model's prediction.
6. Apply your model to the test file and measure its real performance on it (same method as above).

The goal of this exercise is not to create the best or the purest model, but rather to describe the steps you took to accomplish it.
Explain areas that may have been the most challenging for you.
Find clear insights on the profiles of the people that make more than $50,000 / year. For example, which variables seem to be the most correlated with this phenomenon?
Finally, you push your code on GitHub to share it with me, or send it via email.

Once again, the goal of this exercise is not to solve this problem, but rather to spend a few hours on it and to thoroughly explain your approach.

4 Load files, wait…?!

Import libraries

Spoiler alert, here are the libraries being used in this notebook:

[1]:
```
import pandas as pd
import numpy as np
import math
from importlib import reload

# Machine learning tools
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, classification_report,␣
↪confusion_matrix
```

```python
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.decomposition import PCA

from scipy.stats.mstats import winsorize

# Data visualization
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.figure_factory as ff
import plotly.express as px
from plotly.subplots import make_subplots

from warnings import filterwarnings
filterwarnings('ignore')

import plotly.io as pio

png_renderer = pio.renderers["png"]
png_renderer.width = 900
png_renderer.height = 500

pio.renderers.default = 'jupyterlab+png'
#pio.renderers.default = "png"
```

Loading files

The zip file contains three files:

File	Information
census_income_learn.csv	Train dataset
census_income_test.csv	Test dataset
census_income_metadata.txt	Metadata: feature's information

A first look on **training and test dataset** shows that they **don't have header** (columns names). We will need to get this information from metadata file in order to do feature engineering.

I have created a Python module to read through metadata file (the code is not very interesting from a ML point of view and it adds some scrolling, that's why it is on a module).

[2]:
```python
import read_metadata
reload(read_metadata);
```

[3]:
```python
meta_top = read_metadata.top()
meta_top.head(3)
```

```
Shape: (45, 2)
```

```
[3]:            feature     code
     0               age     AAGE
     1   class of worker  ACLSWKR
     2     industry code   ADTIND
```

```python
[4]: meta_middle = read_metadata.middle()
     meta_middle.head(3)
```

```
Shape: (40, 2)
```

```
[4]:                     feature nunique
     0                       age      91
     1           class of worker       9
     2  detailed industry recode      52
```

```python
[5]: meta_bottom = read_metadata.bottom()
     meta_bottom.head(3)
```

```
Shape: (41, 2)
```

```
[5]:                     feature                                       unique values
     0                       age                                        [continuous]
     1           class of worker  [Not in universe, Federal government, Local go...
     2  detailed industry recode  [0, 40, 44, 2, 43, 47, 48, 1, 11, 19, 24, 25, ...
```

```python
[6]: df_train = pd.read_csv("data/census_income_learn.csv", header=None, na_values='?
      ↪', skipinitialspace=True)
     names = pd.Series(list(df_train.columns), name='feature').astype('category')
```

```python
[7]: merged = pd.concat([names, meta_top.iloc[:,0], meta_middle.iloc[:,0],␣
      ↪meta_bottom.iloc[:,0]], axis=1)
     merged.columns = ['df_train', 'meta_top', 'meta_middle', 'meta_bottom']
     merged.tail(6)
```

```
[7]:    df_train                                     meta_top meta_middle  \
     39       39                          total person income        year
     40       40                 own business or self employed         NaN
     41       41                         taxable income amount         NaN
     42      NaN  fill inc questionnaire for veteran's admin         NaN
     43      NaN                            veterans benefits         NaN
     44      NaN                           weeks worked in year         NaN

               meta_bottom
     39  weeks worked in year
     40                  year
     41                   NaN
     42                   NaN
```

```
43                    NaN
44                    NaN
```

Observations

- Training dataset contains 42 features unnamed.
- Features in metadata file are not described consistenly: length of decribed feature vary.
- It is not straightforward to infer df_train features names from metadata file.
- Features descriptions at the end of metadata file seems to be a better fit (matching length when adding income feature).

Merge of features information:

[8]:
```python
pd.merge(meta_top, meta_middle).merge(meta_bottom).head(5)
```

[8]:
```
                 feature     code nunique  \
0                    age     AAGE      91
1        class of worker  ACLSWKR       9
2              education     AHGA      17
3          wage per hour  AHRSPAY    1240
4    major industry code   AMJIND      24

                                       unique values
0                                       [continuous]
1  [Not in universe, Federal government, Local go...
2  [Children, 7th and 8th grade, 9th grade, 10th ...
3                                       [continuous]
4  [Not in universe or children, Entertainment, S...
```

Note: Feature names vary in file resulting in missing features during merge. An in-depth study would allow us to correctly match all the variables.

4.0.1 How to infer columns name ?

Simple idea

As length of feature description at the end of metadata file (bottom file) closely match with the one of training dataset columns we can assume the order is correct and infer.

More complex idea, a path to autoML

To find the right columns names for our datasets one can use information on features in metadata file. We will only focus on the information from the last descriptions of features (bottom file).

The cell below contains unique values for **second column** from **training dataset** we want to find feature name:

[9]:
```python
list(df_train.iloc[:, 1].unique())
```

[9]:
```
['Not in universe',
 'Self-employed-not incorporated',
```

```
 'Private',
 'Local government',
 'Federal government',
 'Self-employed-incorporated',
 'State government',
 'Never worked',
 'Without pay']
```

The cell below contains unique values for feature **class of worker** from **metadata file**:

[10]:
```
meta_bottom.iloc[1, 1]
```

[10]:
```
['Not in universe',
 'Federal government',
 'Local government',
 'Never worked',
 'Private',
 'Self-employed-incorporated',
 'Self-employed-not incorporated',
 'State government',
 'Without pay']
```

Observations

- Values are similar
- There is a high probability for 2nd column of training dataset to be feature class of worker

How to measure "distance" between metadata file features and every training dataset column?

Jaccard similarity is a distance metric that can be used to measure distance between two lists.

[11]:
```
def jaccard_similarity(list1, list2):
    intersection = len(list(set(list1).intersection(list2)))
    union = (len(list1) + len(list2)) - intersection
    return float(intersection) / union

def compute_similarity(index):
    similarities = []
    for i in range(len(df_train.columns)):
        similarities.append([round(jaccard_similarity(list(df_train.iloc[:, i].
 ↪unique()),
                                                        meta_bottom.iloc[index,␣
 ↪1]),2),
                             meta_bottom.iloc[index, 0],
                             df_train.columns.values[i]
                            ])
        similarities.sort(reverse=True)

    # Select most miningful features
```

```python
    sim = [x[0] for x in similarities]
    ind = [i+1 for i,x in enumerate(sim[1:]) if(sim[0]-x<sim[0]-sim[1]+0.05
                                                and sim[0]-sim[1] <0.1
                                                and sim[0] >0)]
    if(sim[0]>0): ind.insert(0, 0)

    #print("3-closest features:")
    #print(similarities[:5])
    #print()
    if len(ind)>0:
        for i in ind:
            print("\"{}\" match \"{}\" ({}%)".format(similarities[i][1],
                                                     similarities[i][2],
                                                     similarities[i][0]*100))
        print()

for i in range(len(meta_bottom)):
    compute_similarity(i)
```

```
"class of worker" match "1" (100.0%)

"education" match "4" (100.0%)

"enroll in edu inst last wk" match "6" (100.0%)

"marital stat" match "7" (100.0%)

"major industry code" match "8" (100.0%)

"major occupation code" match "9" (88.0%)

"race" match "10" (100.0%)

"hispanic origin" match "11" (82.0%)

"sex" match "12" (100.0%)

"member of a labor union" match "37" (100.0%)
"member of a labor union" match "13" (100.0%)

"reason for unemployment" match "14" (100.0%)

"full or part time employment stat" match "15" (100.0%)

"tax filer stat" match "19" (100.0%)

"region of previous residence" match "20" (100.0%)
```

```
"state of previous residence" match "21" (98.0%)

"detailed household and family stat" match "22" (100.0%)

"detailed household summary in household" match "23" (100.0%)

"migration code-change in msa" match "25" (90.0%)

"migration code-change in reg" match "26" (89.0%)

"migration code-move within reg" match "27" (90.0%)

"live in this house 1 year ago" match "28" (100.0%)

"migration prev res in sunbelt" match "37" (100.0%)
"migration prev res in sunbelt" match "13" (100.0%)

"family members under 18" match "31" (100.0%)

"country of birth father" match "34" (98.0%)
"country of birth father" match "33" (98.0%)
"country of birth father" match "32" (98.0%)

"country of birth mother" match "34" (98.0%)
"country of birth mother" match "33" (98.0%)
"country of birth mother" match "32" (98.0%)

"country of birth self" match "34" (98.0%)
"country of birth self" match "33" (98.0%)
"country of birth self" match "32" (98.0%)

"citizenship" match "35" (67.0%)

"fill inc questionnaire for veteran's admin" match "37" (100.0%)
"fill inc questionnaire for veteran's admin" match "13" (100.0%)
```

The code above measure distances between metadata file and training dataset columns, returning the bests matches.

Conclusion

We introduced Jaccard similarity to infer information on missing data (columns names). One can create such data to automatically deduce the type of data (autoML for feature engineering).

Ultimately we will use the insights given by Jaccard similarity to confirm our pairing.

Pairing method:

1. parse metadata file (last feature descriptions);

2. inpute missing column names from parsed data (same order);
3. manual validation step with Jaccard similarity;
4. construction of names.csv file containing the header for training and test dataset;
5. read training and test files setting names attribute with content from names.csv

Load files (with header)

Now we are ready to load our datasets with the corresponding column names.

[12]:
```
names = pd.read_csv("data/names.csv", sep=',').columns.tolist()

df_train = pd.read_csv("data/census_income_learn.csv", header=None, na_values='?
 ↪',
                       names=names, skipinitialspace=True)

df_test = pd.read_csv("data/census_income_test.csv", header=None, na_values='?',
                      names=names, skipinitialspace=True)
```

4.1 Exploratory analysis

Performing an exploratory analysis on the data and create some relevant visualizations.

[13]:
```
df_train.head(3)
```

[13]:
```
   age                class of worker  detailed industry recode  \
0   73                Not in universe                         0
1   58  Self-employed-not incorporated                         4
2   18                Not in universe                         0

   detailed occupation recode                   education  wage per hour  \
0                           0        High school graduate              0
1                          34  Some college but no degree              0
2                           0                  10th grade              0

  enroll in edu inst last wk   marital stat           major industry code  \
0            Not in universe        Widowed  Not in universe or children
1            Not in universe       Divorced                  Construction
2                High school  Never married  Not in universe or children

                major occupation code  ... country of birth father  \
0                     Not in universe  ...           United-States
1  Precision production craft & repair  ...           United-States
2                     Not in universe  ...                 Vietnam

  country of birth mother country of birth self  \
0           United-States        United-States
1           United-States        United-States
2                 Vietnam              Vietnam
```

```
                       citizenship own business or self employed  \
0    Native- Born in the United States                            0
1    Native- Born in the United States                            0
2  Foreign born- Not a citizen of U S                             0

  fill inc questionnaire for veteran's admin  veterans benefits  \
0                           Not in universe                  2
1                           Not in universe                  2
2                           Not in universe                  2

   weeks worked in year  year    income
0                     0    95  - 50000.
1                    52    94  - 50000.
2                     0    95  - 50000.

[3 rows x 42 columns]
```

[14]:

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 199523 entries, 0 to 199522
Data columns (total 42 columns):
 #   Column                                      Non-Null Count   Dtype
---  ------                                      --------------   -----
 0   age                                         199523 non-null  int64
 1   class of worker                             199523 non-null  object
 2   detailed industry recode                    199523 non-null  int64
 3   detailed occupation recode                  199523 non-null  int64
 4   education                                   199523 non-null  object
 5   wage per hour                               199523 non-null  int64
 6   enroll in edu inst last wk                  199523 non-null  object
 7   marital stat                                199523 non-null  object
 8   major industry code                         199523 non-null  object
 9   major occupation code                       199523 non-null  object
 10  race                                        199523 non-null  object
 11  hispanic origin                             198649 non-null  object
 12  sex                                         199523 non-null  object
 13  member of a labor union                     199523 non-null  object
 14  reason for unemployment                     199523 non-null  object
 15  full or part time employment stat           199523 non-null  object
 16  capital gains                               199523 non-null  int64
 17  capital losses                              199523 non-null  int64
 18  dividends from stocks                       199523 non-null  int64
 19  tax filer stat                              199523 non-null  object
 20  region of previous residence                199523 non-null  object
 21  state of previous residence                 198815 non-null  object
```

```
 22  detailed household and family stat         199523 non-null  object
 23  detailed household summary in household    199523 non-null  object
 24  instance weight                            199523 non-null  float64
 25  migration code-change in msa               99827 non-null   object
 26  migration code-change in reg               99827 non-null   object
 27  migration code-move within reg             99827 non-null   object
 28  live in this house 1 year ago              199523 non-null  object
 29  migration prev res in sunbelt              99827 non-null   object
 30  num persons worked for employer            199523 non-null  int64
 31  family members under 18                    199523 non-null  object
 32  country of birth father                    192810 non-null  object
 33  country of birth mother                    193404 non-null  object
 34  country of birth self                      196130 non-null  object
 35  citizenship                                199523 non-null  object
 36  own business or self employed              199523 non-null  int64
 37  fill inc questionnaire for veteran's admin 199523 non-null  object
 38  veterans benefits                          199523 non-null  int64
 39  weeks worked in year                       199523 non-null  int64
 40  year                                       199523 non-null  int64
 41  income                                     199523 non-null  object
dtypes: float64(1), int64(12), object(29)
memory usage: 63.9+ MB
```

Note

Meaning of value *'Not in Universe'*: According to the IPUMS website (https://cps.ipums.org/cps-action/faq), indicates that the census question was irrelevant to the households or persons to whom the question was asked.

4.1.1 Checking for imbalanced data

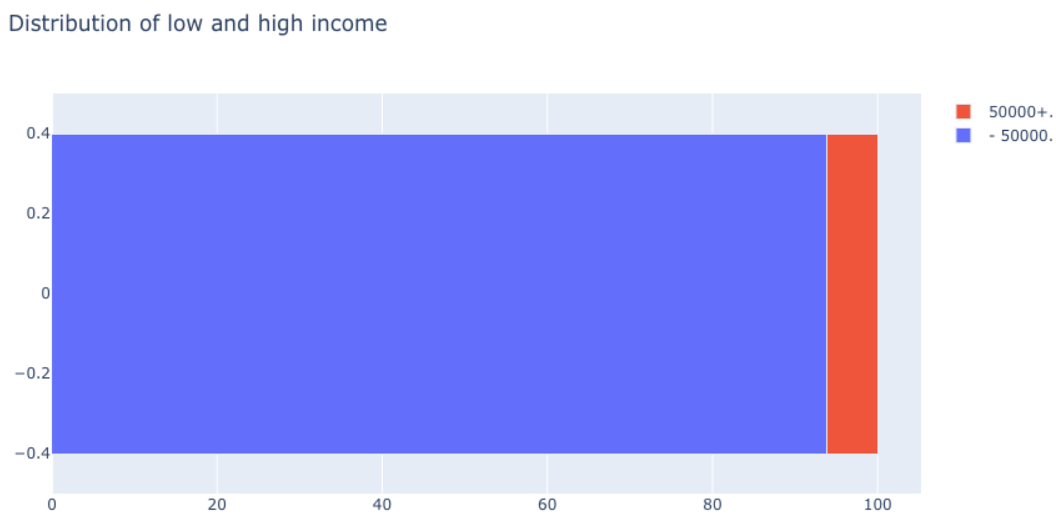
Repartition of income:

[15]:
```
income_rep = df_train['income'].value_counts(normalize=True).mul(100).round(2).
 ↪reset_index()
income_rep
```

[15]:
```
      index  income
0  - 50000.   93.79
1   50000+.    6.21
```

[16]:
```
fig = go.Figure()

fig.add_trace(go.Bar(
    x=[income_rep['income'][0]],
    name=income_rep['index'][0],
    orientation='h'
))
```

```
fig.add_trace(go.Bar(
    x=[income_rep['income'][1]],
    name=income_rep['index'][1],
    orientation='h'
))

fig.update_layout(barmode='stack', title = "Distribution of low and high income")
fig.show()
```



Observation: *'income'* has two unique values *'50 000+'* and *'- 50 000'* with a ratio 1:9

Let's analyze the distribution of income for *'race'* feature:

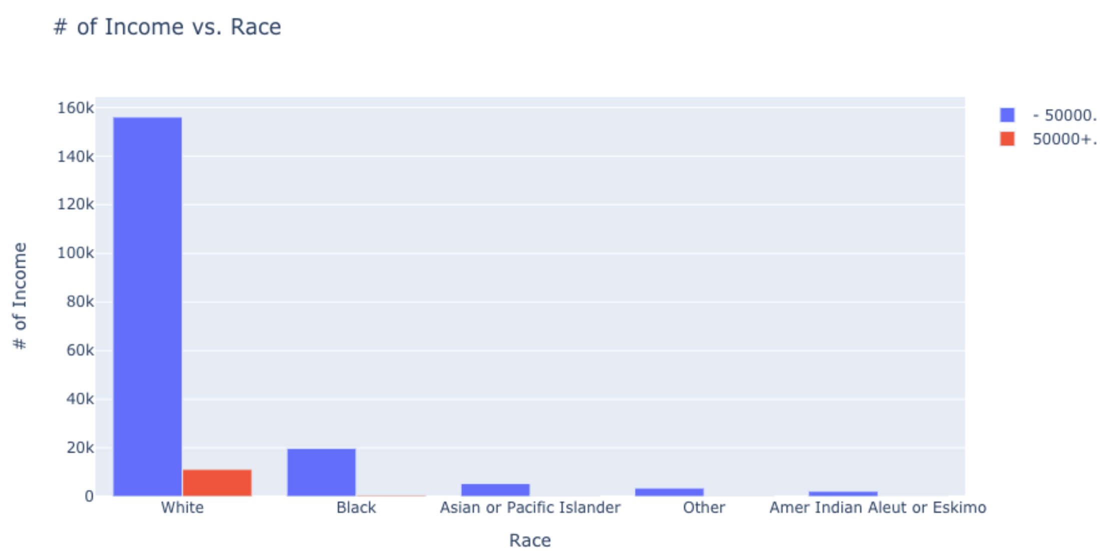
[17]:
```
crosstab = pd.crosstab(df_train['race'], df_train['income'])
crosstab = crosstab.sort_values(crosstab.columns[0],axis=0,ascending=False)
crosstab
```

[17]:

income	- 50000.	50000+.
race		
White	156093	11272
Black	19875	540
Asian or Pacific Islander	5405	430
Other	3566	91
Amer Indian Aleut or Eskimo	2202	49

[18]:
```
fig = go.Figure()
```

```
for i in range(len(crosstab.columns)):
    fig.add_trace(go.Bar(x=crosstab.index, y=crosstab.values[:,i], name=crosstab.
 ↪columns[i]))

fig.update_layout(
    title= "# of Income vs. Race",
    xaxis_title = "Race",
    yaxis_title = "# of Income",
    font=dict(size=12)
)

fig.show()
```



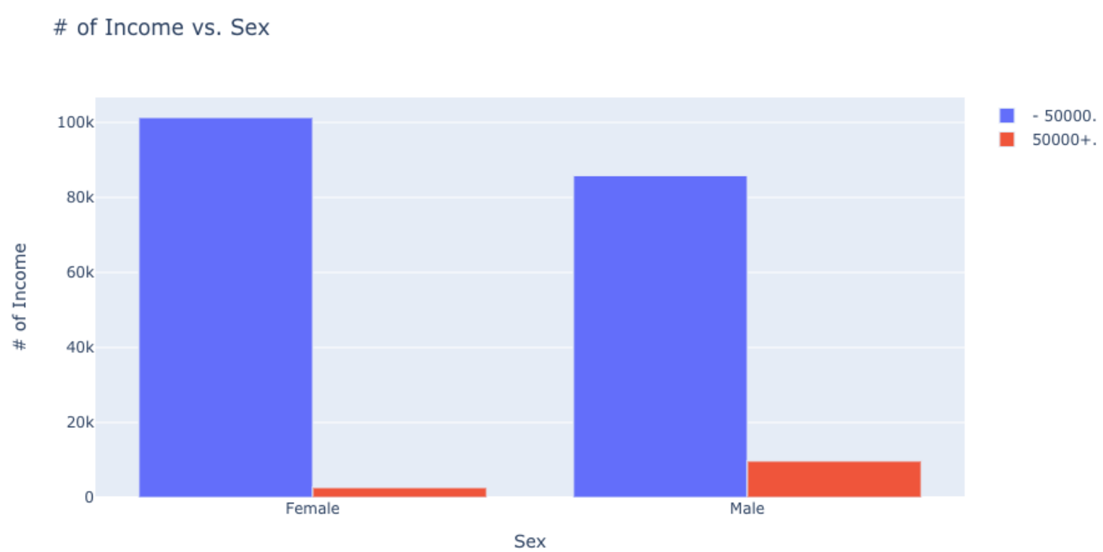
Observation: Ethnicity is imbalanced, white people are overrepresented

Let's create a function for further plotting:

[19]:
```
def countplot(data, x, hue, values=None, aggfunc=None):
    """
    Implementation of sns.countplot for Plotly.
    """

    crosstab = pd.crosstab(df_train[x], df_train[hue], values=values,␣
 ↪aggfunc=aggfunc)
    crosstab = crosstab.sort_values(crosstab.columns[0],axis=0,ascending=False)

    fig = go.Figure()
```

```
    for i in range(len(crosstab.columns)):
        fig.add_trace(go.Bar(x=crosstab.index, y=crosstab.values[:,i],␣
↪name=crosstab.columns[i]))

    fig.update_layout(
        title= hue.capitalize() + " vs. " + x.capitalize() ,
        xaxis_title = x,
        yaxis_title = hue,
        font=dict(size=12)
    )

    return fig
```

[20]:
```
fig = countplot(data=df_train, x='sex', hue='income')
fig.update_layout(
    title= '# of Income vs. Sex',
    xaxis_title = 'Sex',
    yaxis_title = '# of Income',
    font=dict(size=12),
    height=500
)
fig.show()
```



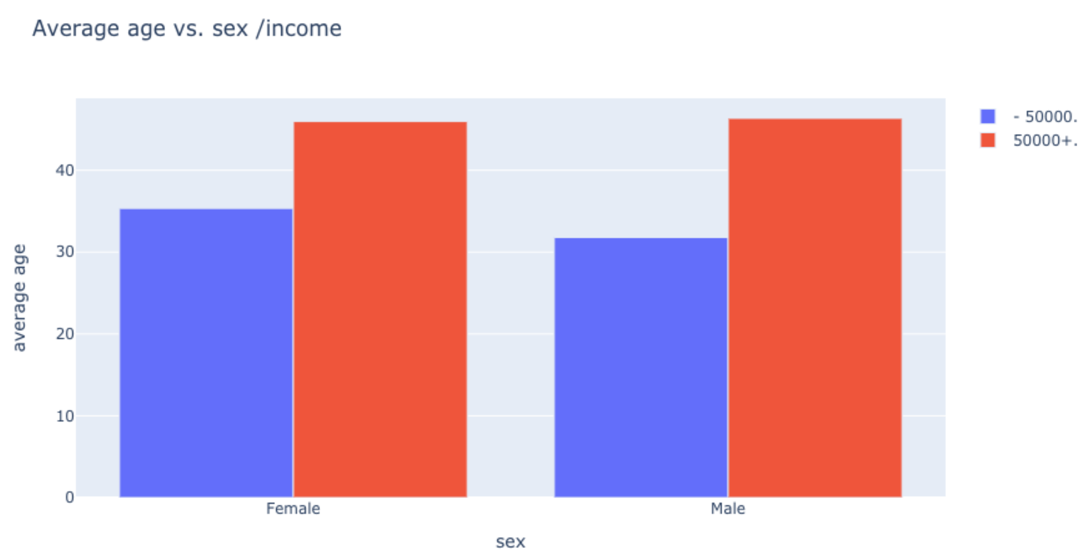
Observations

- Female and male are balanced (total number of income equal)

- Nevertheless, fewer men make less than $50 000 comparing to women

4.1.2 More analysis

[21]:
```
fig = countplot(data=df_train, x='sex', hue='income', values=df_train['age'],␣
↪aggfunc=np.mean)
fig.update_layout(
    title= 'Average age vs. sex /income',
    xaxis_title = 'sex',
    yaxis_title = 'average age',
    font=dict(size=12),
    height=400
)
fig.show()
```



Observations

- Revenue for men and women is unequal
- The average age for women making less than $50 000 is 35 while it is 32 for men
- Men start earning more money earlier than women do

Tree map and geographical plots

Let's focus on the "state of previous residence" feature.

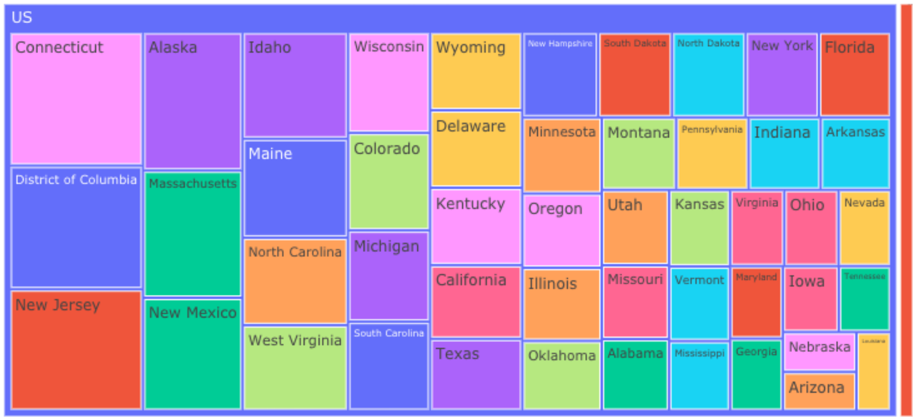
We would like to determine which state most of the high-income earners come from. State of previous residence may be involved in tax income (related to a higher or lower income).

[22]:
```
states_count = pd.crosstab(df_train["state of previous residence"],␣
↪df_train["income"])
states_count = states_count.sort_values(crosstab.
↪columns[0],axis=0,ascending=False)

states_count = states_count.reset_index()
states_count.rename(columns={'state of previous residence':'state',
                             '- 50000.':'under50000',
                             '50000+.':'above50000', }, inplace=True)
states_count["pct"] = states_count.apply (lambda row: row.above50000/(row.
↪above50000 + row.under50000)*100, axis=1)


states_count = states_count[~states_count.state.str.contains("Not in universe")]
states_count.head(3)
```

[22]:

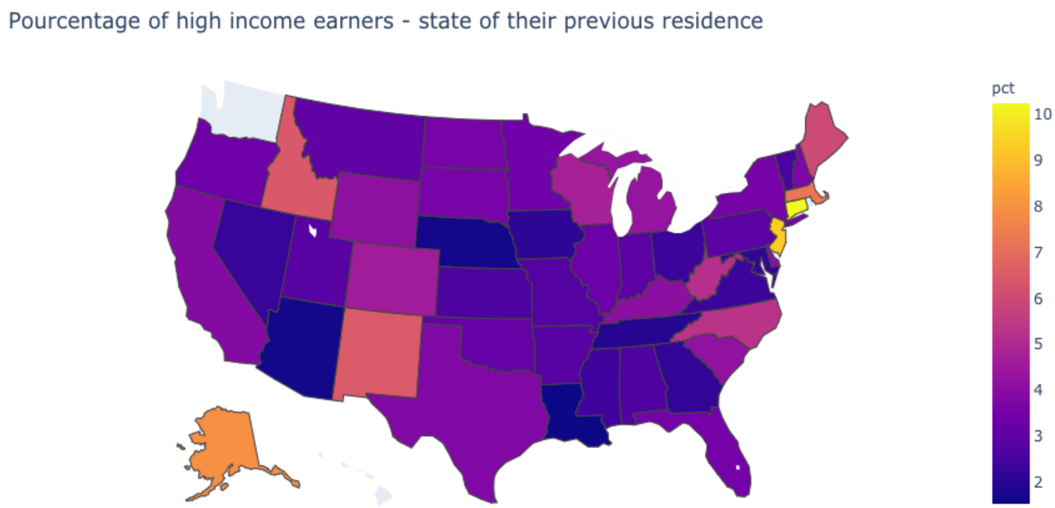
income	state	under50000	above50000	pct
1	California	1647	67	3.908985
2	Utah	1032	31	2.916275
3	Florida	819	30	3.533569

[23]:
```
is_US = ["not US" if (x == 'Abroad') else "US" for x in states_count['state']]

tree = px.treemap(states_count,
                path = [is_US,"state"],
                values="pct",
                color= "state")


tree.show()
```



In the previous treemap, we group the US states in a same "box". Connecticut, New Jersey, Alaska, Massachusetts and Columbia are home to the high income earners (previous residence state). But is there any geographical correlation ?

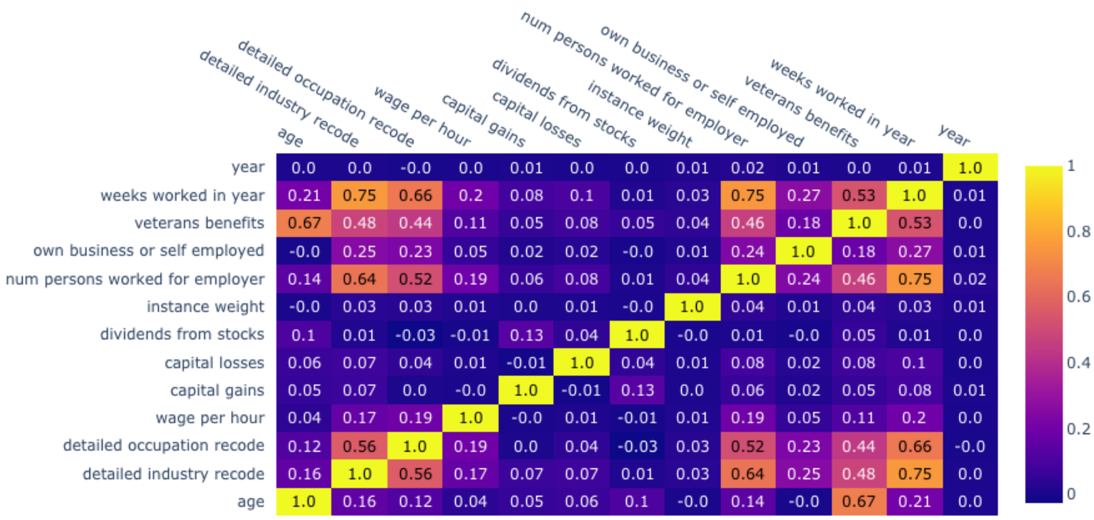
[24]:
```python
us_states = pd.read_csv("data/us_states.csv")
us_states.rename(columns={'State':'state'}, inplace=True)


merged = states_count.merge(us_states, on="state")

fig = px.choropleth(merged,
                    locations="Code",
                    color="pct",
                    hover_name="state",
                    locationmode = 'USA-states')
fig.update_layout(
    title_text = 'Pourcentage of high income earners - state of their previous residence',
    geo_scope='usa',
)

fig.show()
```



By mapping the data in a choropleth map, we can see that most of those states are neighbors, and in the East coast of the United States. For a deeper analysis, we can study the migration code-change features.

Plotting correlation heatmap

[25]:
```
corrs = df_train.corr()
figure = ff.create_annotated_heatmap(
    z=corrs.values,
    x=list(corrs.columns),
    y=list(corrs.index),
    annotation_text=corrs.round(2).values,
    showscale=True)
figure
```



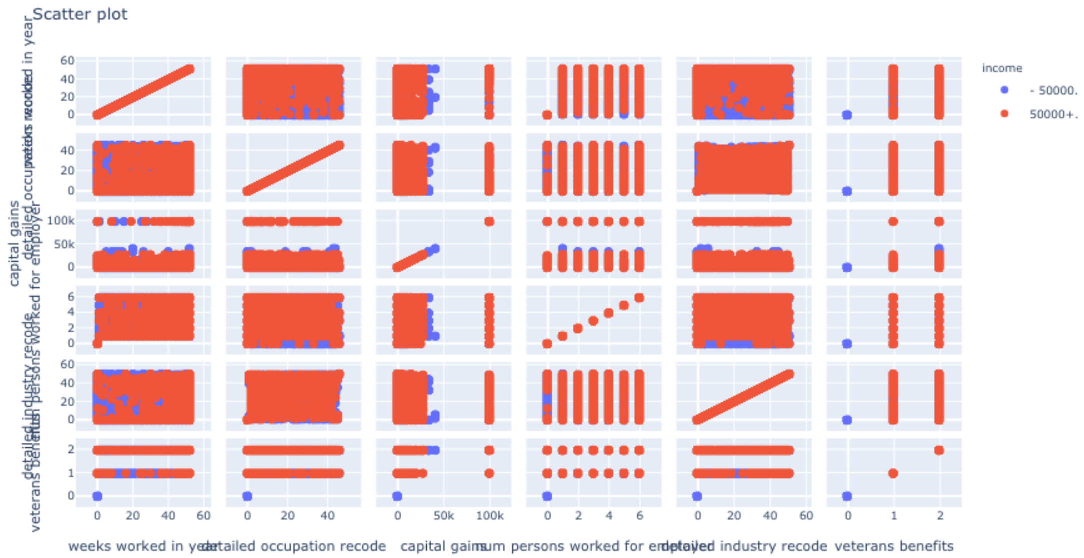
Observations

- As we see above *'weeks worked in year'* and *'detailed industry recode'* **have high correlation** (0.75). *'veterans benefits'* and *'age'* (0.67), *'detailed occupation recode'* and *'weeks worked in year'* (0.66), *'num persons worked for employer'* and *'detailed industry recode'* (0.64) too.
- *'detailed industry recode'* and *'detailed occupation recode'* have **medium correlation** between *'veterans benefits'*.
- The reminder columns have **low correlation**.

Scatter plot

[26]:
```
fig = px.scatter_matrix(
    df_train,
    dimensions=["weeks worked in year",
                "detailed occupation recode",
                "capital gains",
                "num persons worked for employer",
                "detailed industry recode",
                "veterans benefits"
               ],
    color="income")

fig.update_layout(
    title="Scatter plot",
    font=dict(
        size=9
    )
```

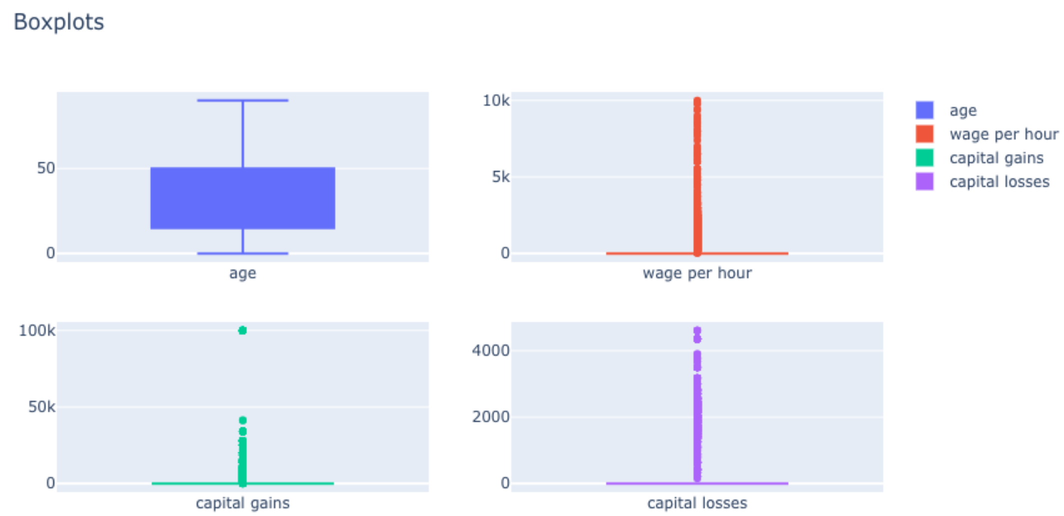
```
)
fig.show()
```



4.1.3 Checking for outliers

To identify outliers we will use boxplots:

[27]:
```
fig = go.Figure(make_subplots(rows=2, cols=2))
fig.add_trace(go.Box(y=df_train['age'], name="age"), row=1, col=1)
fig.add_trace(go.Box(y=df_train['wage per hour'], name="wage per hour"), row=1,
 ↪col=2)
fig.add_trace(go.Box(y=df_train['capital gains'], name="capital gains"), row=2,
 ↪col=1)
fig.add_trace(go.Box(y=df_train['capital losses'], name="capital losses"),
 ↪row=2, col=2)
fig.update_layout(title='Boxplots')
fig.show()
```

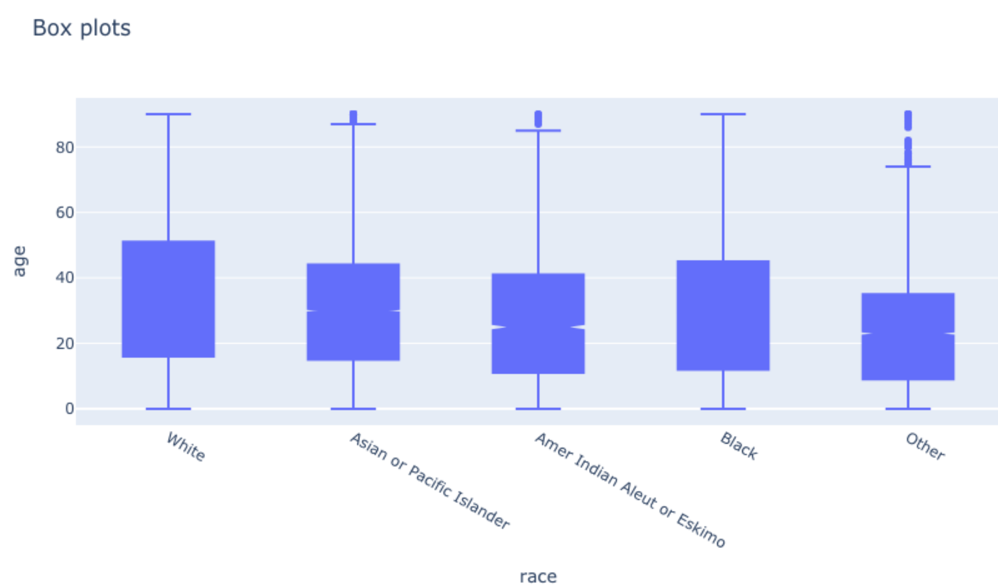


Observations

- Feature **age** does not have outliers, we will not apply outliers ridding techniques
- Other features plots does

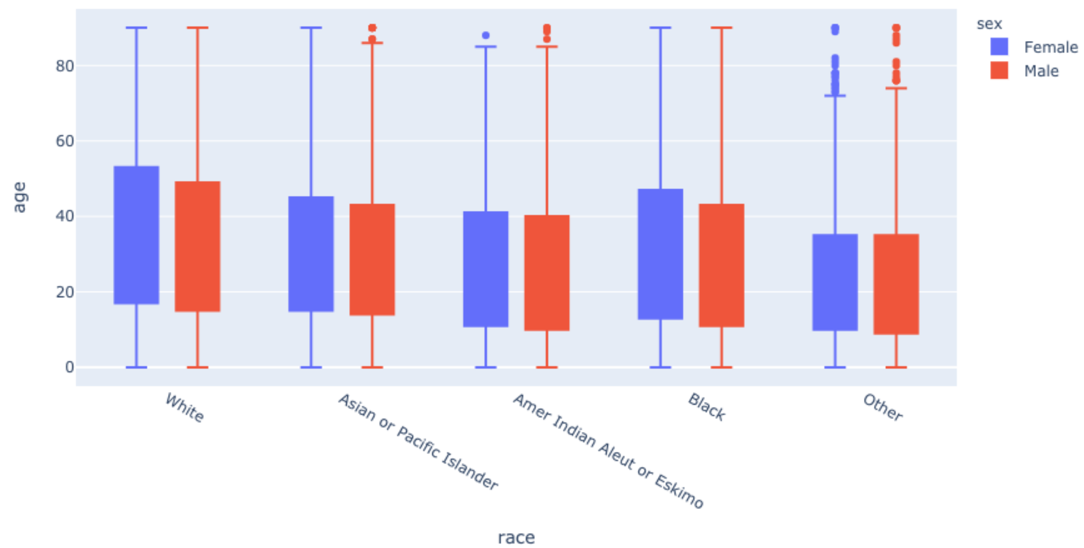
In-depth analysis:

[28]:
```
fig = px.box(df_train, x="race", y="age", title="Box plots", notched=True,␣
↪height=400)
fig.show()
```

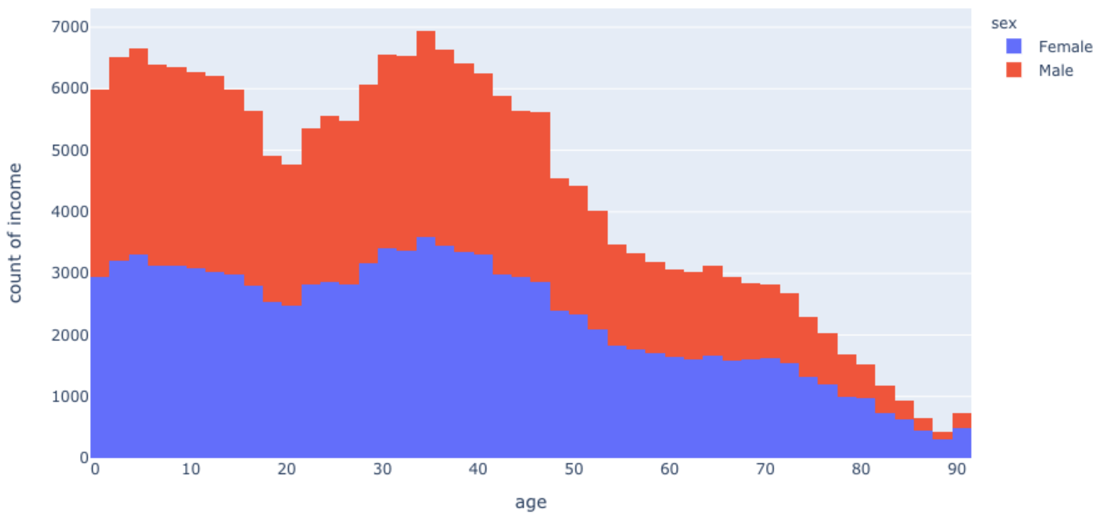


In-depth analysis:

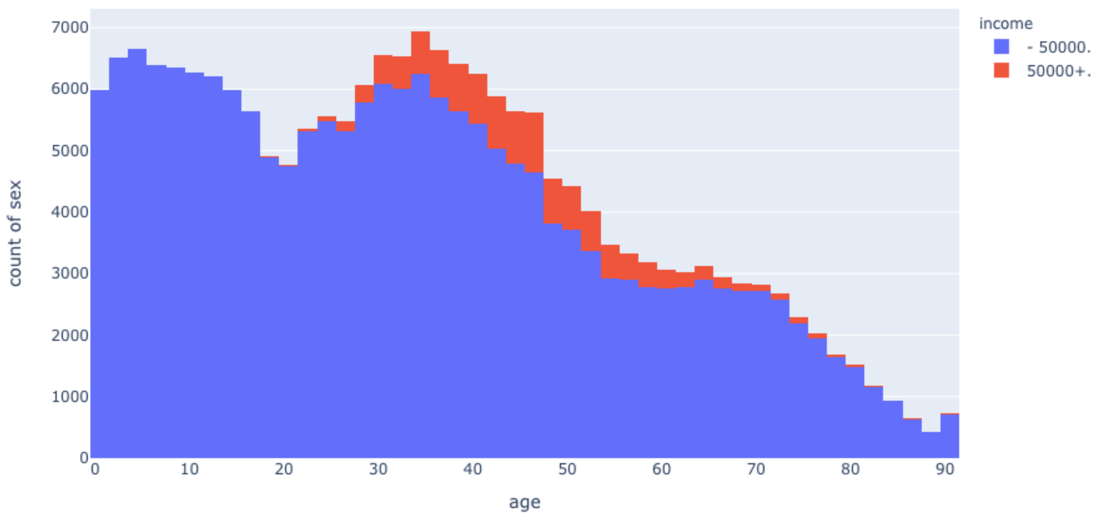
[29]:
```
fig = px.box(df_train, x="race", y="age", color="sex", height=400)
fig.show()
```



[30]:
```
fig = px.histogram(df_train, x="age", color="sex", y="income", nbins=50,␣
 ↪height=400)
fig.show()
```



[31]:
```
fig = px.histogram(df_train, x="age", color="income", y="sex", nbins=50)
fig.show()
```



4.2 Cleaning and feature engineering

4.2.1 Data cleaning

Checking for missing data Numerical features:

[32]:
```
df_train.select_dtypes(include=['number']).isna().sum(axis = 0)
```

[32]:
```
age                              0
detailed industry recode         0
detailed occupation recode       0
wage per hour                    0
capital gains                    0
capital losses                   0
dividends from stocks            0
instance weight                  0
num persons worked for employer  0
own business or self employed    0
veterans benefits                0
weeks worked in year             0
year                             0
dtype: int64
```

Categorical features:

[33]:
```
df_train.select_dtypes(include=['object']).isna().sum(axis = 0)
```

[33]:
```
class of worker                                0
education                                      0
enroll in edu inst last wk                     0
marital stat                                   0
major industry code                            0
major occupation code                          0
race                                           0
hispanic origin                              874
sex                                            0
member of a labor union                        0
reason for unemployment                        0
full or part time employment stat              0
tax filer stat                                 0
region of previous residence                   0
state of previous residence                  708
detailed household and family stat             0
detailed household summary in household        0
migration code-change in msa               99696
migration code-change in reg               99696
migration code-move within reg             99696
live in this house 1 year ago                  0
migration prev res in sunbelt              99696
```

```
family members under 18                          0
country of birth father                       6713
country of birth mother                       6119
country of birth self                         3393
citizenship                                      0
fill inc questionnaire for veteran's admin       0
income                                           0
dtype: int64
```

Observations

- There is no missing data for numerical features
- Nearly 100 000 NaN values for *migration code* features

We will transform and fill missing values for categorical variables.

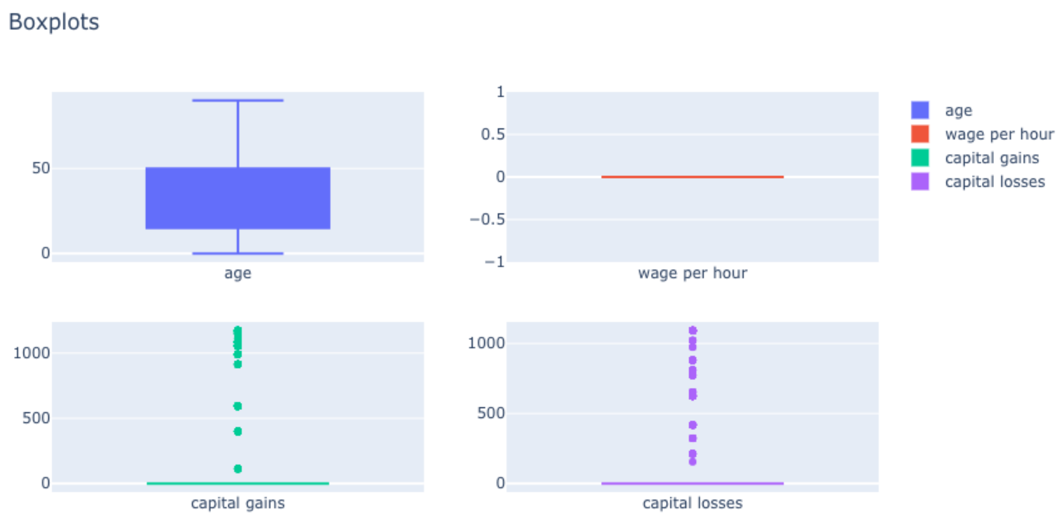
Removing outliers Based on our previous analysis we will remove outliers.

Apply winsorization:

[34]:
```
df_train["capital gains"] = winsorize(df_train["capital gains"],(0,0.035))
df_train["capital losses"] = winsorize(df_train["capital losses"],(0,0.019))
df_train["wage per hour"] = winsorize(df_train["wage per hour"],(0,0.057))
```

Re-plot (with winsorization):

[35]:
```
fig = go.Figure(make_subplots(rows=2, cols=2))
fig.add_trace(go.Box(y=df_train['age'], name="age"), row=1, col=1)
fig.add_trace(go.Box(y=df_train['wage per hour'], name="wage per hour"), row=1,␣
 ↪col=2)
fig.add_trace(go.Box(y=df_train['capital gains'], name="capital gains"), row=2,␣
 ↪col=1)
fig.add_trace(go.Box(y=df_train['capital losses'], name="capital losses"),␣
 ↪row=2, col=2)
fig.update_layout(title='Boxplots')
fig.show()
```



4.2.2 Feature engineering

PCA: numerical features

[36]:
```
df_train_num = df_train._get_numeric_data()

df_stand = StandardScaler().fit_transform(df_train_num)
pca = PCA(n_components=0.9, whiten=True)
df_pca = pca.fit_transform(df_stand)
```

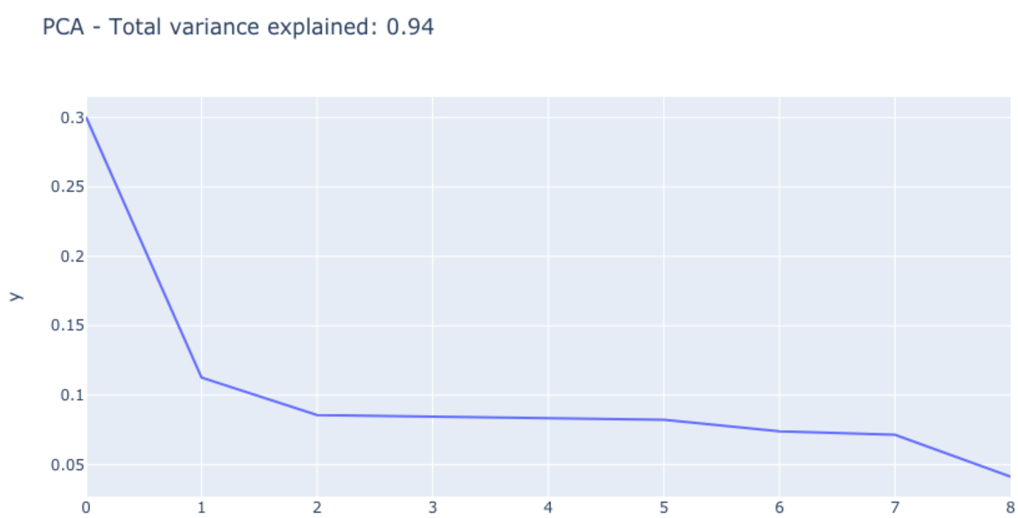
[37]:
```
print(pca.explained_variance_ratio_)
```

```
[0.30050763 0.11272997 0.0857723  0.08434398 0.08344548 0.08219555
 0.07399952 0.07151394 0.04134661]
```

[38]:
```
print('Original number of features', df_stand.shape[1])
print('Reduced number of features', df_pca.shape[1])
```

```
Original number of features 13
Reduced number of features 9
```

[39]:
```
fig = px.line(y=pca.explained_variance_ratio_,
              title='PCA - Total variance explained: {0:.2f}'.format(pca.
 ↪explained_variance_ratio_.sum()),
              height=500)
fig.show()
```



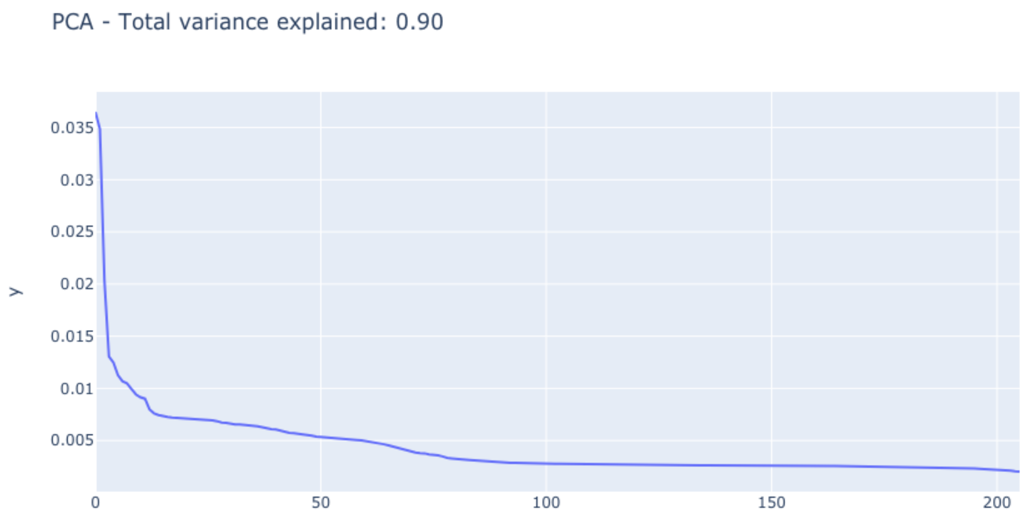
PCA: all features

[40]:
```
df_dummies = pd.get_dummies(df_train.drop(columns=['income']).dropna())

df_stand = StandardScaler().fit_transform(df_dummies)
pca = PCA(n_components=0.9, whiten=True)
df_pca = pca.fit_transform(df_stand)
```

[41]:
```
print('Original number of features', df_stand.shape[1])
print('Reduced number of features', df_pca.shape[1])
```

```
Original number of features 386
Reduced number of features 206
```

[42]:
```
fig = px.line(y=pca.explained_variance_ratio_,
              title='PCA - Total variance explained: {0:.2f}'.format(pca.
 ↪explained_variance_ratio_.sum()),
              height=500)
fig.show()
```



5 Models

[43]:
```
X = pd.get_dummies(df_train.dropna().drop(columns=['income']))
y = df_train.dropna()['income']
```

[44]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

[45]:
```
logreg = LogisticRegression(max_iter = 100)
model_1 = logreg.fit(X_train, y_train)

pred_1 = model_1.predict(X_test)

print(classification_report(y_test, pred_1, target_names = ["+50 000", "-50␣
↪000"]))
```

```
              precision    recall  f1-score   support

     +50 000       0.95      0.99      0.97     17849
     -50 000       0.59      0.17      0.26      1044

    accuracy                           0.95     18893
   macro avg       0.77      0.58      0.62     18893
weighted avg       0.93      0.95      0.93     18893
```

[46]:
```
randf = RandomForestClassifier(n_estimators = 100)
model_2 = randf.fit(X_train, y_train)

pred_2 = model_2.predict(X_test)

print(classification_report(y_test, pred_2, target_names = ["+50 000", "-50␣
↪000"]))
```

```
              precision    recall  f1-score   support

     +50 000       0.96      0.99      0.98     17849
     -50 000       0.70      0.31      0.42      1044

    accuracy                           0.95     18893
   macro avg       0.83      0.65      0.70     18893
weighted avg       0.95      0.95      0.95     18893
```

[47]:
```
gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=0.5,␣
↪max_depth=1)
model_3 = gbc.fit(X_train, y_train)

pred_3 = model_3.predict(X_test)

print(classification_report(y_test, pred_3, target_names = ["+50 000", "-50␣
↪000"]))
```

```
              precision    recall  f1-score   support

     +50 000       0.96      0.99      0.98     17849
     -50 000       0.68      0.33      0.45      1044

    accuracy                           0.95     18893
   macro avg       0.82      0.66      0.71     18893
weighted avg       0.95      0.95      0.95     18893
```

6 Pipeline

[48]:
```
names = pd.read_csv("data/names.csv", sep=',').columns.tolist()

df_train = pd.read_csv("data/census_income_learn.csv", header=None, na_values='?
↪',
                       names=names, skipinitialspace=True)

df_test = pd.read_csv("data/census_income_test.csv", header=None, na_values='?',
```

```
                    names=names, skipinitialspace=True)
```

[49]:
```
X = df_train.drop(columns=['income'])
y = df_train['income']
```

[50]:
```
categorical_features = X.select_dtypes(include=['object']).columns
numerical_features = X.select_dtypes(include=['number']).columns
```

[51]:
```
numerical_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

[52]:
```
preprocessor = ColumnTransformer(transformers=[
    ('drop_columns', 'drop', ['instance weight']),
    ('num', numerical_transformer, numerical_features),
    ('cat', categorical_transformer, categorical_features)
])
```

[53]:
```
model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('RandomForestClassifier', RandomForestClassifier())
])
```

[54]:
```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)
```

[55]:
```
model_pipeline.fit(X_train, y_train)
```

[55]:
```
Pipeline(memory=None,
         steps=[('preprocessor',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('drop_columns', 'drop',
                                                  ['instance weight']),
                                                 ('num',
                                                  Pipeline(memory=None,
                                                           steps=[('scaler',
StandardScaler(copy=True,
  with_mean=True,
  with_std=True))],
                                                           verbose=False),
                                                  Index(['age', 'detailed
```

```
industry recode', '...
                 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                        class_weight=None, criterion='gini',
                                        max_depth=None, max_features='auto',
                                        max_leaf_nodes=None, max_samples=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_samples_split=2,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators=100, n_jobs=None,
                                        oob_score=False, random_state=None,
                                        verbose=0, warm_start=False))],
         verbose=False)
```

[56]:
```
y_pred_train = model_pipeline.predict(X_train)
```

[57]:
```
accuracy_score(y_train, y_pred_train)
```

[57]:
```
0.9999812051272413
```

[58]:
```
y_pred_val = model_pipeline.predict(X_val)
```

[59]:
```
accuracy_score(y_val, y_pred_val)
```

[59]:
```
0.95323894248841
```

6.1 Prediction for test file

[60]:
```
X_test = df_test.drop(columns=['income'])
y_test = df_test['income']
```

6.1.1 Data check

Let's have a look on test dataset even if **we will not use this information to tune our model**.

[61]:
```
df_test.head(1)
```

[61]:
```
   age class of worker  detailed industry recode  detailed occupation recode  \
0   38         Private                         6                          36

                  education  wage per hour enroll in edu inst last wk  \
0  1st 2nd 3rd or 4th grade              0            Not in universe

                      marital stat          major industry code  \
0  Married-civilian spouse present  Manufacturing-durable goods
```

```
                   major occupation code  ... country of birth father  \
0  Machine operators assmblrs & inspctrs  ...                  Mexico

  country of birth mother country of birth self  \
0                  Mexico                Mexico

                           citizenship own business or self employed  \
0  Foreign born- Not a citizen of U S                              0

  fill inc questionnaire for veteran's admin  veterans benefits  \
0                            Not in universe                  2

   weeks worked in year  year    income
0                    12    95  - 50000.

[1 rows x 42 columns]
```

Note: Great, columns in test dataset seems to match with header from train dataset. Luckily!

[62]:
```
df_test.select_dtypes(include=['number']).isna().sum(axis = 0)
```

[62]:
```
age                                  0
detailed industry recode             0
detailed occupation recode           0
wage per hour                        0
capital gains                        0
capital losses                       0
dividends from stocks                0
instance weight                      0
num persons worked for employer      0
own business or self employed        0
veterans benefits                    0
weeks worked in year                 0
year                                 0
dtype: int64
```

[63]:
```
df_test.select_dtypes(include=['object']).isna().sum(axis = 0)
```

[63]:
```
class of worker                                   0
education                                         0
enroll in edu inst last wk                        0
marital stat                                      0
major industry code                               0
major occupation code                             0
race                                              0
hispanic origin                                 405
```

```
sex                                              0
member of a labor union                          0
reason for unemployment                          0
full or part time employment stat                0
tax filer stat                                   0
region of previous residence                     0
state of previous residence                    330
detailed household and family stat              0
detailed household summary in household          0
migration code-change in msa                 49946
migration code-change in reg                 49946
migration code-move within reg               49946
live in this house 1 year ago                    0
migration prev res in sunbelt                49946
family members under 18                          0
country of birth father                       3429
country of birth mother                       3072
country of birth self                         1764
citizenship                                      0
fill inc questionnaire for veteran's admin       0
income                                           0
dtype: int64
```

Note: Same feature repartition for NaNs. Numerical features don't have missing values (our model don't support numerical filling NaNs yet). Luckily!

6.1.2 Prediction

Fitting model with entire train dataset.

[64]:
```
model_pipeline.fit(X, y)
```

[64]:
```
Pipeline(memory=None,
         steps=[('preprocessor',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('drop_columns', 'drop',
                                                  ['instance weight']),
                                                 ('num',
                                                  Pipeline(memory=None,
                                                           steps=[('scaler',
StandardScaler(copy=True,
  with_mean=True,
  with_std=True))],
                                                           verbose=False),
                                                  Index(['age', 'detailed
```

```
industry recode', '...
                 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                        class_weight=None, criterion='gini',
                                        max_depth=None, max_features='auto',
                                        max_leaf_nodes=None, max_samples=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_samples_split=2,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators=100, n_jobs=None,
                                        oob_score=False, random_state=None,
                                        verbose=0, warm_start=False))],
         verbose=False)
```

[65]:
```
y_pred_train = model_pipeline.predict(X)
```

[66]:
```
accuracy_score(y, y_pred_train)
```

[66]: 0.9999799521859635

[67]:
```
y_pred_test = model_pipeline.predict(X_test)
```

[68]:
```
accuracy_score(y_test, y_pred_test)
```

[68]: 0.9543112608007057

6.2 Conclusion

Columns names, metadata file

It is the first time I encounter datasets with header missing. Usually, metadata files come with training and test sets with further information for context understanding and feature engineering. It gave me the idea to build a matching feature algorithm that could be used for autoML purposes.

Results

We got 0.95 accuracy on test data, impressive! (Did I do something wrong ?)

What to do next?

Compare other ML models, use cross validation etc. Validate the model for production purposes (performances).

6.3 Sources

- https://www.kdnuggets.com/2018/08/make-machine-learning-models-robust-outliers.html
- https://stackoverflow.com/questions/14720324/compute-the-similarity-between-two-lists

6.4 Appendix

Cosine similarity implementation to compute distance between two lists. Alternative to Jaccard distance.

```
def counter_cosine_similarity(c1, c2):
    terms = set(c1).union(c2)
    dotprod = sum(c1.get(k, 0) * c2.get(k, 0) for k in terms)
    magA = math.sqrt(sum(c1.get(k, 0)**2 for k in terms))
    magB = math.sqrt(sum(c2.get(k, 0)**2 for k in terms))
    return dotprod / (magA * magB)

def cosine(listA, df):
    similarities = []

    for i in range(len(df.columns)):
        counterA = Counter(listA[1])
        counterB = Counter(list(df.iloc[:, i].unique()))

        sim = counter_cosine_similarity(counterA, counterB)
        similarities.append([round(sim, 2), listA[0], df.columns.values[i]])
        similarities.sort(reverse=True)
    return similarities

cosine(meta_bottom.iloc[1, :], df_train)[:15]
```

Alternative to Crosstab is Pivot Table. For this purpose Crosstab has a more succinct syntax. These two lines are equivalent:

```
crosstab = pd.crosstab(df_train['race'], df_train['income'])
pivot = pd.pivot_table(df_train[['income', 'race']], index='race',␣
 ↪columns='income', aggfunc=len, fill_value=0)
    Plotly alternative to plot histogram categorical variables:
x0 = df_train[df_train["income"]=="50000+."]["race"]
x1 = df_train[df_train["income"]=="- 50000."]["race"]

fig = go.Figure()
fig.add_trace(go.Histogram(x=x0))
fig.add_trace(go.Histogram(x=x1))

fig.show()
    Plotly alternative to plot histogram:
fig = go.Figure()
fig.add_trace(go.Histogram(x=df_train[df_train["income"]=="- 50000."]["age"]))
fig.add_trace(go.Histogram(x=df_train[df_train["income"]=="50000+."]["age"]))

fig.update_layout(barmode='overlay')
```

```
fig.update_traces(opacity=0.95)
fig.show()
    Plotly static:
img_bytes = fig.to_image(format="png")
from IPython.display import Image
Image(img_bytes)
```