Machine Learning with Star Wars

Nate De Kock

Elevator pitch

In 2014, FiveThirtyEight surveyed over 1000 people to write the article titled, America's Favorite 'Star Wars' Movies (And Least Favorite Characters). They have provided the data on GitHub - https://github.com/fivethirtyeight/data/tree/master/star-wars-survey.

This report does the following:

1. Cleans the dataset into tidy data.
2. Creates a machine learning model to predict a person's income based on their interest in Star Wars.

TECHNICAL DETAILS

1. Shorten the column names and clean them up for easier use with pandas.

The following code is an example of one of the ways I was able to shorten the column names by replacing some strings in the column names.

```
var_replace = {
    'Which of the following Star Wars films have you seen\\? Please select all that
apply\\.':'seen',
    'Please rank the Star Wars films in order of preference with 1 being your
favorite film in the franchise and 6 being your least favorite film.':'rank',
    'Please state whether you view the following characters favorably, unfavorably,
or are unfamiliar with him/her.':'view',
    'Do you consider yourself to be a fan of the Star Trek
franchise\\?':'star_trek_fan',
    'Do you consider yourself to be a fan of the Expanded Universe\\?
\x8cæ':'expanded_fan',
    'Are you familiar with the Expanded Universe\\?':'know_expanded',
    'Have you seen any of the 6 films in the Star Wars franchise\\?':'seen_any',
    'Do you consider yourself to be a fan of the Star Wars film
franchise\\?':'star_wars_fans',
    'Which character shot first\\?':'shot_first',
    'Unnamed: \d{1,2}':np.nan,
    ' ':'_',
}

values_replace = {
    'Response':'',
    'Star Wars: Episode ':'',
    ' ':'_'
}
```

Here is an image of how the data has transformed into tidydata with each row being one observation, each column being an individual variable and each cell having its own value.

Interactive - .py | Data Viewer - dat | Data Viewer - dat

dat (1186, 38)

	index	respon…	seen_a…	star_w…	seen__…	seen__…	seen__…	seen__…	seen__…	seen__…	rank__…	rank__…	rank__…	rank__…	rank__…	rank__…	view__…	view__…
0	0	32928799…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	3	2	1	4	5	6	Very fav…	Very fav…
1	1	32928795…	No	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
2	2	32927652…	Yes	No	Star War…	Star War…	Star War…	nan	nan	nan	1	2	3	4	5	6	Somewhat…	Somewhat…
3	3	32927631…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	5	6	1	2	4	3	Very fav…	Very fav…
4	4	32927312…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	5	4	6	2	1	3	Very fav…	Somewhat…
5	5	32927193…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	1	4	3	6	5	2	Very fav…	Very fav…
6	6	32926847…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	6	5	4	3	1	2	Very fav…	Very fav…
7	7	32926637…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	4	5	6	3	2	1	Very fav…	Somewhat…
8	8	32926540…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	5	4	6	2	1	3	Very fav…	Somewhat…
9	9	32926404…	Yes	No	nan	Star War…	nan	nan	nan	nan	1	2	3	4	5	6	Neither …	Very fav…
10	10	32926378…	Yes	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
11	11	32926350…	No	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
12	12	32926092…	Yes	No	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	3	4	5	6	1	2	Somewhat…	Very fav…
13	13	32925969…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	4	5	6	2	3	1	Very fav…	Very fav…
14	14	32925872…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	4	2	5	3	1	6	Somewhat…	Very fav…
15	15	32925830…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	4	6	5	3	1	2	Very fav…	Somewhat…
16	16	32925805…	Yes	Yes	nan	nan	nan	Star War…	nan	nan	4	1	2	3	5	6	Neither …	Somewhat…
17	17	32925728…	Yes	Yes	Star War…	Star War…	Star War…	nan	nan	Star War…	1	2	3	4	5	6	Very fav…	Very fav…
18	18	32925652…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	6	5	2	3	1	4	Very fav…	Very fav…
19	19	32925622…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	6	5	1	4	3	2	Very fav…	Somewhat…
20	20	32925223…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	6	5	4	1	2	3	Very fav…	Very fav…
21	21	32925210…	Yes	No	Star War…	Star War…	Star War…	Star War…	Star War…	nan	3	4	5	1	2	6	Neither …	Neither …
22	22	32925118…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	6	5	4	3	1	2	Very fav…	Very fav…
23	23	32924832…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	6	1	4	2	3	5	Very fav…	Somewhat…
24	24	32924654…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	1	2	6	3	4	5	Very fav…	Very fav…
25	25	32924479…	No	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
26	26	32924201…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	4	6	5	2	3	1	Very fav…	Somewhat…
27	27	32923840…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	6	5	4	2	3	1	Very fav…	Very fav…
28	28	32923804…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	6	5	4	2	1	3	Very fav…	Somewhat…
29	29	32923769…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	4	6	5	2	1	3	Very fav…	Very fav…
30	30	32923667…	Yes	No	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	4	3	5	6	2	1	Neither …	Neither …
31	31	32923393…	Yes	Yes	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	6	5	4	2	1	3	Very fav…	Very fav…
32	32	32923308…	Yes	No	Star War…	Star War…	Star War…	Star War…	Star War…	Star War…	6	1	2	3	4	5	Somewhat…	Somewhat…

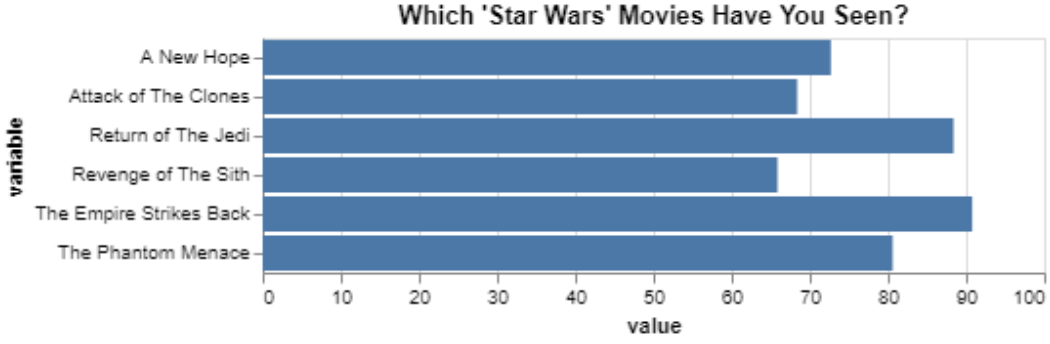
2. Filter the dataset to those that have seen at least one film.

This code was used to filter the dataset to show only those who have seen at least one film.

```
seen = dat['seen_any']=="Yes"
seen_dat = dat[seen].dropna(thresh = 6)
seen_dat
```

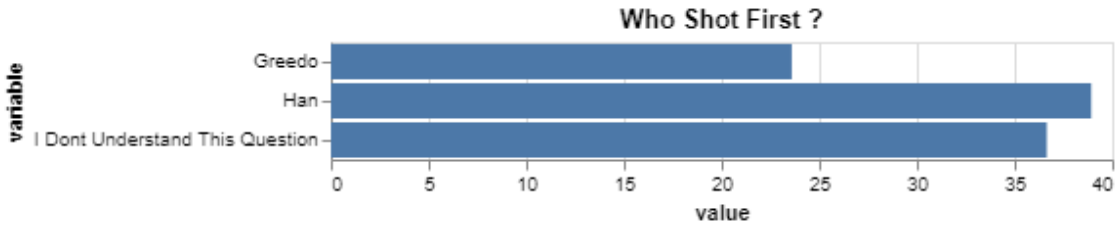
3. Please validate that the data provided on GitHub lines up with the article by recreating 2 of their visuals and calculating 2 summaries that they report in the article.

This visual is a replication of the first visual in the article. The below horizontal bar graph shows that **The Empuire Strikes Back** is the most watched out of all of the Star Wars movies!



The final replicated visual illustrates who shot first in the Star Wars movies according to the survey respondents. This is clearly a split opinion. A potential reason for that is that there are two different first shooters in two

different Star Wars movies. I do not recall which movie but in one of them, Han shot first whilst in the other Greedo shot first. This provides an explanation as to why there is a split opinion.



4. Preparing Dataset for Machine Learning

a) Create an additional column that converts the age ranges to a number and drop the age range categorical column.

In the next few questions, we make valuable use of the *assign* function to create a new column, and the *split* and *strip* functions to clean the values in order to convert them from a string to an integer.

```
seen_dat = (seen_dat.assign(
      new_age = lambda x: x.age.str.split("-", expand = True)))
seen_dat
      #.rename({0:'name_1' , 1:'name_2'}
      #)['name_2'].str.replace(">","")) # the str.strip is taking out spaces at the
beginning and end
seen_dat['new_age'] = seen_dat['new_age'].str.strip(">")
```

b)Create an additional column that converts the school groupings to a number and drop the school categorical column.

Here we use the *factorize* function which is more useful because it will assign various answers related to respondents educational qualification to a numeric value.

```
seen_dat['education'], b = pd.factorize((seen_dat['education']))
```

c) Create an additional column that converts the income ranges to a number and drop the income range categorical column.

Importantly, we use the *astype* function to convert income from a string to an integer.

```
seen_dat = (seen_dat.assign(
      new_income = lambda x: x.household_income.str.split("-",expand = True)))
seen_dat['new_income'] = seen_dat['new_income'].str.strip("$")
seen_dat['new_income'] = seen_dat['new_income'].str.replace('+','')
seen_dat['new_income'] = seen_dat['new_income'].str.replace(',','')
seen_dat.new_income.astype(float)
```

d) Create your target (also known as label) column based on the new income range column.

Here is an image of how we have converted categorical data into numerical data in order to prepare it for machine learning. new_age and new_income are newly added columns, whilst education has been factorized so no columns were technically added or dropped but rather converted through factorization.

expanded_fan	star_trek_fan	gender	education	location_(census_region)	new_age	new_income
No	No	Male	0	South Atlantic	18	NaN
NaN	No	Male	0	West North Central	18	0
NaN	Yes	Male	1	West North Central	18	100000
No	No	Male	1	West North Central	18	100000
No	Yes	Male	2	Middle Atlantic	18	25000
...	...	...	...	...	...	...
NaN	No	Female	1	Pacific	45	0
NaN	Yes	Female	1	East North Central	18	0
NaN	Yes	Female	2	Mountain	30	50000
NaN	Yes	Female	1	East North Central	45	100000
NaN	No	Female	3	Pacific	60	50000

e) One-hot encode all remaining categorical columns.

Using the *sklearn* package we utilize the one-hot encode to convert the remaining categorical columns to numerical. This is crucial because our machine learning model can only handle numeric values and not categorical, string type values.

```
#One Hot Encode
from sklearn.preprocessing import OneHotEncoder
from sklearn import preprocessing
```

```
# limit to categorical data using df.select_dtypes()
seen_dat1 = seen_dat.select_dtypes(include=[object])

le = preprocessing.LabelEncoder()

# 2/3. FIT AND TRANSFORM
# use df.apply() to apply le.fit_transform to all columns
seen_dat1_2 = seen_dat1.apply(le.fit_transform)

# 1. INSTANTIATE
enc = preprocessing.OneHotEncoder()

# 2. FIT
enc.fit(seen_dat1_2)

# 3. Transform
onehotlabels = enc.transform(seen_dat1_2).toarray()
onehotlabels.shape
```

Build a machine learning model that predicts whether a person makes more than $50k.

I used a Decision Tree Classifier because I was most comfortable with this type of machine learning model. The model predicts 24% accuracy in predicting whether a person makes more than $50k a year.

	0	1	2	3	4	5	accuracy	macro avg	weighted avg
precision	18.1818	14.9254	18.75	16.1765	35.4545	28.3582	24.1379	21.9744	24.9555
recall	15	18.5185	21.4286	18.6441	33.0508	24.359	24.1379	21.8335	24.1379
f1-score	16.4384	16.5289	20	17.3228	34.2105	26.2069	24.1379	21.7846	24.438
support	4000	5400	2800	5900	11800	7800	24.1379	37700	37700

...

APPENDIX A (PYTHON SCRIPT)

```
#%%
import pandas as pd
import altair as alt
import numpy as np

url = 'https://github.com/fivethirtyeight/data/raw/master/star-wars-
survey/StarWars.csv'

dat = pd.read_csv(url,skiprows=2,encoding = "ISO-8859-1",header=None )


dat_names = (pd.read_csv(url, encoding = "ISO-8859-1", nrows = 1).melt())

dat_names.head(10)
#dat.head()


# Replace untidy data
# %%
dat_names = (dat_names.replace('Unnamed: \d{1,2}', np.nan, regex=True)
.replace('Response', "")) ## made blanks in rows 2,3 in value columns
dat_names

# Create third column
# %%
dat_names = (dat_names.assign(
      clean_variable = lambda x: x.variable.str.strip())) # the str.strip is taking
out spaces at the beginning and end
dat_names

# Clean Value column
# %%
dat_names =(dat_names
   .assign(
      clean_variable = lambda x: x.variable.str.strip()
         .replace(
            'Which of the following Star Wars films have you seen? Please select all
that apply.','seen'),
      clean_value = lambda x: x.value.str.strip()
      ))
dat_names

# Fill Nan column
# %%

dat_names =(dat_names
   .replace('Unnamed: \d{1,2}', np.nan, regex=True)
   .replace('Response', "")
   .assign(
      clean_variable = lambda x: x.variable.str.strip()
         .replace(
            'Which of the following Star Wars films have you seen? Please select all
```

```
that apply.','seen'),
      clean_value = lambda x: x.value.str.strip()
      )
   .fillna(method = 'ffill')) ## all the N
dat_names

# Creating Final Column
# %%
dat_names =(dat_names
   .replace('Unnamed: \d{1,2}', np.nan, regex=True)
   .replace('Response', "")
   .assign(
      clean_variable = lambda x: x.variable.str.strip()
         .replace(
            'Which of the following Star Wars films have you seen? Please select all
that apply.','seen'),
      clean_value = lambda x: x.value.str.strip()
      )
   .fillna(method = 'ffill')
   .assign(
      column_name = lambda x: x.clean_variable.str.cat(x.clean_value, sep = "__")
   )## this combines the clean_variable column with the clean_value column seperated
by __
)

dat_names

dat_names.column_name

# %%
## lets shorten some columns up by replacing some strings in the column names

variables_replace = {
    'Which of the following Star Wars films have you seen\\? Please select all that
apply\\.':'seen',
    'Please rank the Star Wars films in order of preference with 1 being your
favorite film in the franchise and 6 being your least favorite film.':'rank',
    'Please state whether you view the following characters favorably, unfavorably,
or are unfamiliar with him/her.':'view',
    'Do you consider yourself to be a fan of the Star Trek
franchise\\?':'star_trek_fan',
    'Do you consider yourself to be a fan of the Expanded Universe\\?
\x8cæ':'expanded_fan',
    'Are you familiar with the Expanded Universe\\?':'know_expanded',
    'Have you seen any of the 6 films in the Star Wars franchise\\?':'seen_any',
    'Do you consider yourself to be a fan of the Star Wars film
franchise\\?':'star_wars_fans',
    'Which character shot first\\?':'shot_first',
    'Unnamed: \d{1,2}':np.nan,
    ' ':'_',
}

values_replace = {
    'Response':'',
    'Star Wars: Episode ':'',
```

```
    ' ':'_'
}

#New DataFrame
#%%

dat_cols_use = (dat_names
    .assign(
        value_replace = lambda x:  x.value.str.strip().replace(values_replace,
regex=True), # replacing stuff in the value column with what we want
        variable_replace = lambda x:
x.variable.str.strip().replace(variables_replace, regex=True) # replacing stuff in
variable column
    ))
dat_cols_use

#Clean NA's
# %%

dat_cols_use = (dat_names
    .assign(
        value_replace = lambda x:  x.value.str.strip().replace(values_replace,
regex=True),
        variable_replace = lambda x:
x.variable.str.strip().replace(variables_replace, regex=True)
    )
    .fillna(method = 'ffill')
    .fillna(value = ""))
dat_cols_use

#Final Column
# %%

dat_cols_use = (dat_names
    .assign(
        value_replace = lambda x:  x.value.str.strip().replace(values_replace,
regex=True),
        variable_replace = lambda x:
x.variable.str.strip().replace(variables_replace, regex=True)
    )
    .fillna(method = 'ffill')
    .fillna(value = "")
    .assign(column_names = lambda x: x.variable_replace.str.cat(x.value_replace, sep
= "__").str.strip('__').str.lower()) # combines variable_replace + value_replace
    )# the .str.strip('__').str.lower() is getting rid of the __ in the last few
column
dat_cols_use

#Join new column to DataFrame
# %%

dat.columns = dat_cols_use.column_names.to_list()# replacing all the column names in
"dat" with tha list of column names from the dat_cols_use.column_names column
dat.head()
# %%
```

```
dat

# Q2- Filtered DF to only those who have seen at least one film
# %%
seen = dat['seen_any']=="Yes"
seen_dat = dat[seen].dropna(thresh = 6)
seen_dat

# Q3 - Recreate Charts and Summaries




#movie_data1.rename(index=str,columns={'seen__i__the_phantom_menace_Star Wars:
Episode I The Phantom Menace':'Phantom_Menace',
   #'seen__ii__attack_of_the_clones_Star Wars: Episode II Attack of the Clones':
'Attack_of_The_Clones',
   #'seen__iii__revenge_of_the_sith_Star Wars: Episode III Revenge of the
Sith':'Revenge_of_the_Sith',
   #'seen__iv__a_new_hope_Star Wars: Episode IV A New Hope': 'A_New_Hope',
   #'seen__v_the_empire_strikes_back_Star Wars: Episode V The Empire Strikes Back':
'The_Empire_Strikes_Back',
   #'seen__vi_return_of_the_jedi_Star Wars: Episode VI Return of the Jedi':
'Return_of_The_Jedi'}, inplace=True)
#movie_data1

# %%

movie_data = seen_dat.filter(['seen__i__the_phantom_menace',
    'seen__ii__attack_of_the_clones',
    'seen__iii__revenge_of_the_sith',
    'seen__iv__a_new_hope',
    'seen__v_the_empire_strikes_back',
    'seen__vi_return_of_the_jedi'])
movie_data

movie_data1 = pd.get_dummies(movie_data)
movie_data1


for movie in movie_data1:
       print(movie_data1[movie].sum())


movie_dict = pd.DataFrame({'The Phantom Menace': [673/835],
   'Attack of The Clones':[571/835],
   'Revenge of The Sith': [550/835],
   'A New Hope':[607/835],
   'The Empire Strikes Back': [758/835],
   'Return of The Jedi': [738/835]})

movie_dict = movie_dict * 100

movie_dict = movie_dict.melt()
```

```
Chart = (alt.Chart(movie_dict,title = "")
  .encode(
    x= alt.X('value', axis = alt.Axis(format = "d")),
    y='variable')
  .mark_bar()
)
Chart

#Chart No.2

shot = dat.filter(['shot_first'])
# %%
shot_1 = pd.get_dummies(shot)
# %%
shot_1 = pd.DataFrame({'Greedo': [197/835],
    'Han' : [325/835],
    'I Dont Understand This Question': [306/835]})
shot_1 = shot_1 * 100
shot_1 = shot_1.melt()
# %%
# Chart for 3b
Chart = alt.Chart(shot_1, title = 'Who Shot First ?').encode(
   x = alt.X('value'),
   y = alt.Y('variable')
).mark_bar()
Chart


# %%

seen_dat = (seen_dat.assign(
      new_age = lambda x: x.age.str.split("-", expand = True)))
seen_dat
      #.rename({0:'name_1' , 1:'name_2'}
      #)['name_2'].str.replace(">",""))) # the str.strip is taking out spaces at the
beginning and end


seen_dat['new_age'] = seen_dat['new_age'].str.strip(">")
seen_dat


seen_dat['education'], b = pd.factorize((seen_dat['education']))


seen_dat = (seen_dat.assign(
      new_income = lambda x: x.household_income.str.split("-",expand = True)))


seen_dat['new_income'] = seen_dat['new_income'].str.strip("$")
seen_dat['new_income'] = seen_dat['new_income'].str.replace('+','')
seen_dat['new_income'] = seen_dat['new_income'].str.replace(',','')
seen_dat.new_income.astype(float)
```

```
seen_dat

seen_dat.dropna(subset = ['new_income'])



#Drop Columns
seen_dat.drop(columns = ['age','household_income'], inplace = True)


#One Hot Encode
from sklearn.preprocessing import OneHotEncoder
from sklearn import preprocessing

# limit to categorical data using df.select_dtypes()
seen_dat1 = seen_dat.select_dtypes(include=[object])

le = preprocessing.LabelEncoder()


# 2/3. FIT AND TRANSFORM
# use df.apply() to apply le.fit_transform to all columns
seen_dat1_2 = seen_dat1.apply(le.fit_transform)



# 1. INSTANTIATE
enc = preprocessing.OneHotEncoder()

# 2. FIT
enc.fit(seen_dat1_2)

# 3. Transform
onehotlabels = enc.transform(seen_dat1_2).toarray()
onehotlabels.shape




# Machine Learning
#%%
import seaborn as sns
import altair as alt
from sklearn import ensemble
from sklearn.metrics import classification_report
#%%
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import metrics

#%%
```

```
X_pred = seen_dat1_2.drop(['new_income'], axis = 1)
y_pred = seen_dat1_2.new_income
X_train, X_test, y_train, y_test = train_test_split(
    X_pred,
    y_pred,
    test_size = .45,
    random_state = 76)


# now we use X_train and y_train to build a model.
# %%

#Model 1 - Tree Decision Model
model1 = tree.DecisionTreeClassifier()
model2 = model1.fit(X_train, y_train)
result = pd.DataFrame(model2.predict(X_test))

report = metrics.classification_report(y_test,result,output_dict=True)
df = pd.DataFrame(report)
print(df.to_markdown())

# Model 2 -
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
model4 = classifier.fit(X_train,y_train)
result1 = pd.DataFrame(model2.predict(X_test))

report1 = metrics.classification_report(y_test,result,output_dict=True)
df1 = pd.DataFrame(report1)
print(df1.to_markdown())
```