Star Wars

Austin Wilson

Elevator pitch

In this project i demonstrate my ability to clean and manipulate data to fit into a machine learning model. I started out with data that was scrambled and i was able to start by shortening it, then i turned that data into column names to be input into my dataset. I was then able to create a model that could predict wether or not someone made more than 50,000 based off of the data i trained the model with.

TECHNICAL DETAILS

GRAND QUESTION 1

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

To do this i first needed to start by replacing all of the blank dat with nans. Then from there i stripped all of the extra spaces and used a dictionary to shorten the columns. Then to finish things off, i made the values, the column names. Below is a part of the starting data, followed by a part of the ending data to show the difference.

Begining Data:

	variable	value
RespondentID	nan	
Have you seen any of the 6 films in the Star Wars franchise?	Response	
Do you consider yourself to be a fan of the Star Wars film franchise?	Response	
Which of the following Star Wars films have you seen? Please select all that apply.	Star Wars: Episode I The Phantom Menace	
Unnamed: 4	Star Wars: Episode II Attack of the Clones	

Ending Data:

seen_any	star_wars_fans	seen__i__the_phantom_menace
Yes	Yes	Star Wars: Episode I The Phantom Menace
No	nan	nan
Yes	No	Star Wars: Episode I The Phantom Menace
Yes	Yes	Star Wars: Episode I The Phantom Menace

seen_any	star_wars_fans	seen__i__the_phantom_menace
Yes	Yes	Star Wars: Episode I The Phantom Menace

GRAND QUESTION 2

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

To accomplish this task i used the seen_any column first. Then from there i checked the additionaly requirment of not having any na values in the star_wars_fans column. I did so using the following code.

```
dat_seen_yes = dat.query("seen_any == 'Yes'")
dat_yes = dat_seen_yes.query("star_wars_fans.notna()")
```

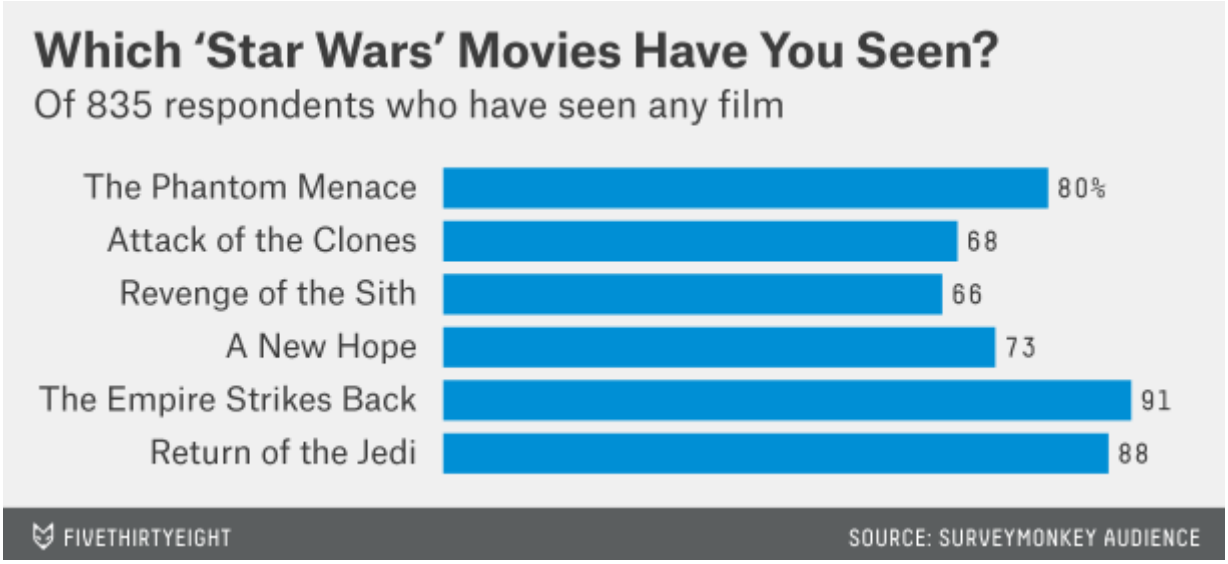
GRAND QUESTION 3

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.

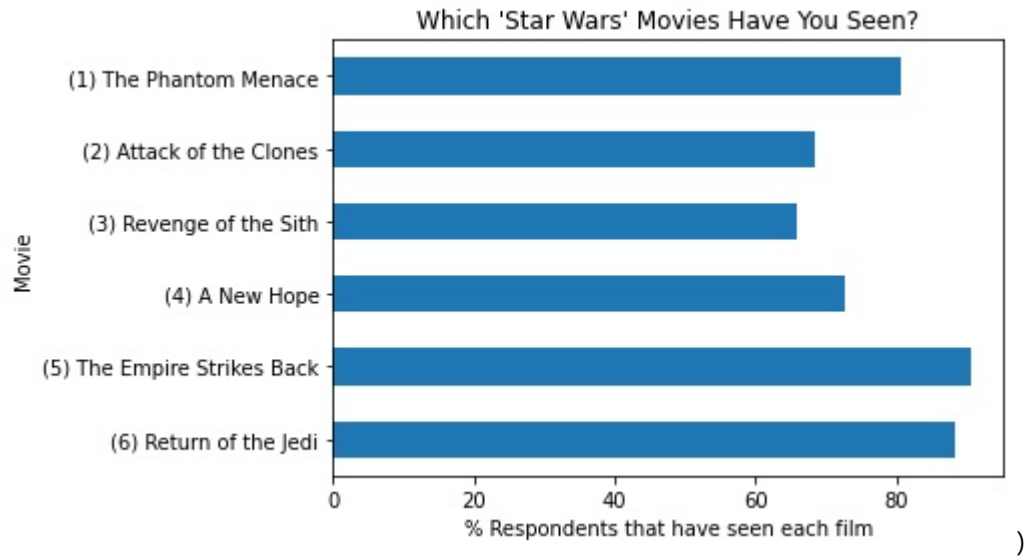
To validate the data i first started by further cleaning and filtering to only get me the data that would be relevant. Then from there i was able to validate the data by dividing each individual variable by the size of the dataframe. I did so for the first graph using the following code.

```
filter_visual_validate_data = dat_yes.filter(like = 'seen__', axis = 1)
visual_validate_data =
pd.melt(filter_visual_validate_data).groupby('value').count().reset_index()
visual_validate_data = visual_validate_data.assign(
    validate_percent = lambda x: ((x.variable/dat_yes.shape[0])*100).round(2))
```

This is the graph that i was trying to recreate. It shows the perentage of people that have watched the films based off the number of people that have seen at least one of the films.

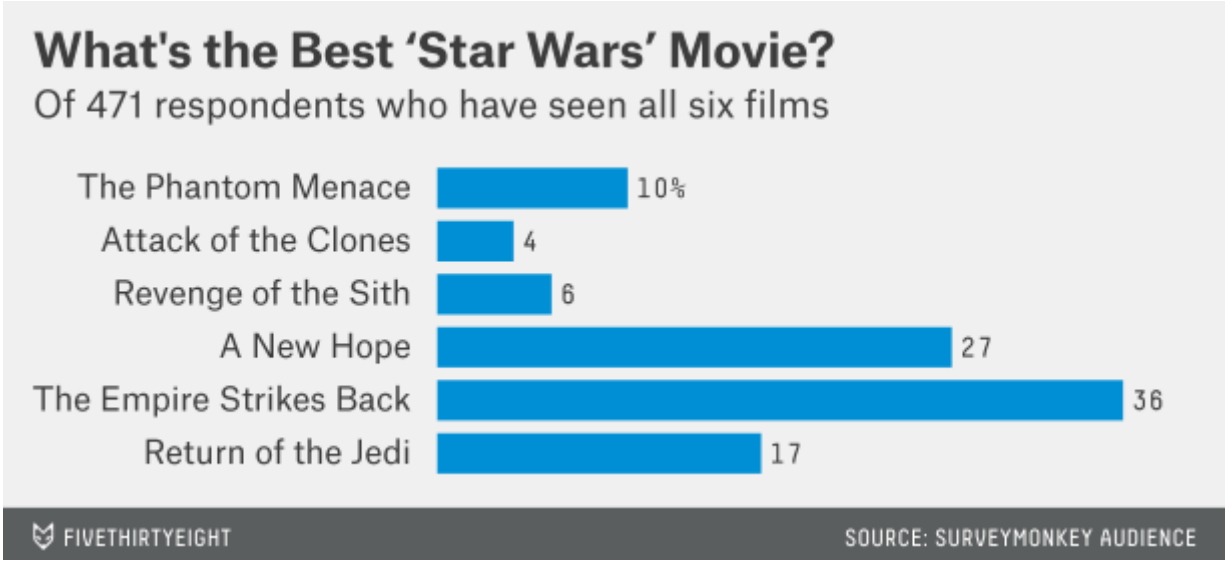


This was my resulting graph

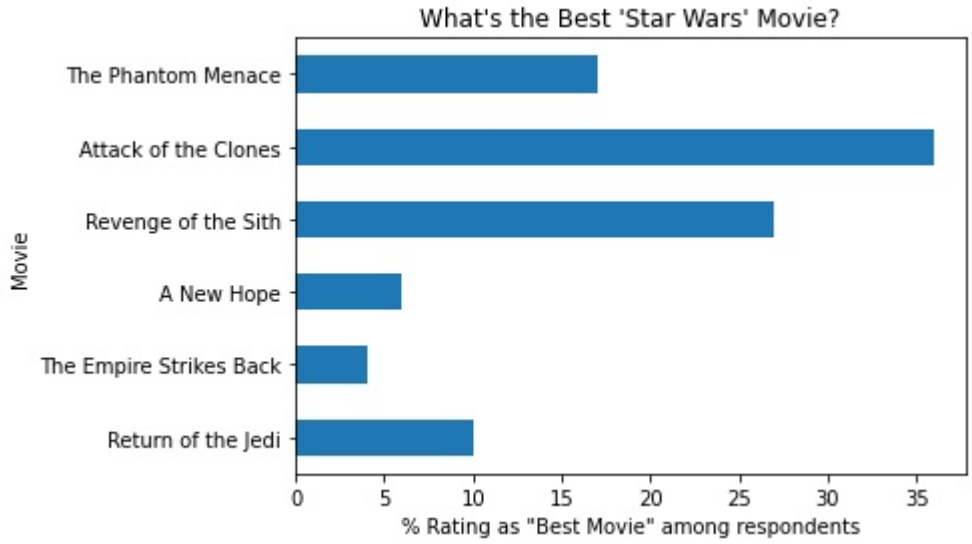


)

The next visual i was trying to recreate was showing the ranking of all star wars films based off the number of people who had seen all of the films.



This was my resulting graph



Next i needed to check wether or not some of other numbers were right in my database. I first calculated the correct percentage of people that had seen at least one movie. I did that by using the following code.

```
at_least_one_film = (len(dat_yes)/len(dat))*100
```

I then checked wether or not my data was correct by validating wether or not by specific amount of men compared to women who have watched at least film was correct. I did so with the following code.

```
dat_yes.groupby("gender").size() / dat.groupby("gender").size()
```

GRAND QUESTION 4

Clean and format the data so that it can be used in a machine learning model. Please achieve the following requests and provide examples of the table with a short description the changes made in your report.

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

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

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

To answer a-c i used one piece of code to change each of the columns to numeric columns. On age and house_hold income this meant splitting the data on "-" and then firther cleaning to make sure any additionaly symbols were not present so that they could be converted to floats. With the education column i used a dictionary to go through and change each of them manualy to a number of years for schooling. I did so with the following code.

```

```

```
dat_numeric = pd.concat([
   (dat.age
      .str.split("-", expand = True)
      .rename(columns = {0:'age_min', 1:'age_max'})
      .age_min
      .str.replace(">","")
      .astype('float')),
   (dat.household_income
      .str.split("-", expand = True)
      .rename(columns = {0: 'income_min', 1: 'income_max'})
      .income_max
      .str.replace("\$|,|\+", "")
      .astype('float')),
   (dat.education
      .replace({
      'Less than high school degree':'9',
      'High school degree':'12',
      'Some college or Associate degree':'14',
      'Bachelor degree':'16',
      'Graduate degree':'19'})
    .astype('float'))],
   axis = 1
)
```

This is an example of the starting data

	age	household_income	education
0	18-29	nan	High school degree
1	18-29	$0 - $24,999	Bachelor degree
2	18-29	$0 - $24,999	High school degree
3	18-29	$100,000 - $149,999	Some college or Associate degree
4	18-29	$100,000 - $149,999	Some college or Associate degree

This is an example of the resulting data after the code was run

	age_min	income_max	education
0	18	nan	12
1	18	24999	16
2	18	24999	12
3	18	149999	14
4	18	149999	14

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

I created a column that is a one-hot encoded column that will tell wether or not the income is above 50,000. I did so with the following code.

```
dat_ml['above_50000'] = np.where(dat_ml['household_income'] > 50000, 1, 0)
```

(e) One-hot encode all remaining categorical columns. To one hot encode the rest of the data what i needed to do was first remove the age, household_income, and education columns. This was so that i could replace the range data with my numeric data that i created previouslyn. Then i used the pd.get_dummies to one-hot encode the data. Following that i added back the data that had been converted previously to numeric data. I did so using the following code.

```
filter_dat_one_hot_1 = dat.drop(['age', 'household_income','education'], axis=1)
dat_ml = pd.get_dummies(filter_dat_one_hot_1)
dat_ml['age'] = dat_numeric['age_min']
dat_ml['household_income'] = dat_numeric['income_max']
dat_ml['education'] = dat_numeric['education']
```

GRAND QUESTION 5

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

Because i had already one hot encoded all my data and changed it all to numeric, this was fairly easy. What i did have to do before creating my model was to convert all of my data to float16. Then from there i plugged the data into the model to predict wether a person makes more than 50,000.

```
X_pred = dat_ml_1.drop(['household_income', 'above_50000'], axis = 1)
y_pred = dat_ml_1.above_50000
X_train, X_test, y_train, y_test = train_test_split(
    X_pred,
    y_pred,
    test_size = .34,
    random_state = 76)
```

Running the model i got the following results.

	Predicted no	Predicted Yes
Actual no	32	50
Actual Yes	28	93

	precision	recall	f1-score	support
0.0	0.53	0.39	0.45	82
1.0	0.65	0.77	0.70	121
accuracy			0.62	203
macro avg	0.59	0.58	0.58	203
weighted avg	0.60	0.62	0.60	203

What this data shows is that more data would be needed to accuratly predict wether or not someone will make more than 50,000.

APPENDIX A (PYTHON SCRIPT)

```
# %%
# Imports libraries
import pandas as pd
import altair as alt
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.core.frame import DataFrame
from sklearn.preprocessing import OneHotEncoder
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

# %%
# Imports data
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

# %%
# 1 - prints original data to markdown
print(dat_names.head(5).to_markdown())

# %%
# 1 - Replace unnamed data with Nans, replaces response with blank spaces, removes
extra spaces on strings to clean data in new columns,
# creates a new column that combines the two cleaned data columns seperated by "_"
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 = "__")
   )
)
dat_names

# %%
# 1 - creates two dictionaries for each column that can be used to replace long
strings with shorter strings
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 ':'',
    ' ':'_'
}

# %%
# 1 - Using the created dictionaries, this will go through and replace all of the
strings with the correct response in new columns,
# it will then fill in the missing values with spaces and will create a new column
that combines the altered data
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())
    )
dat_cols_use

# %%
# 1 - replaces the column names in dat with the data from the
dat_cols_use.column_names created previously
dat.columns = dat_cols_use.column_names.to_list()
dat

# %%
# 1 - prints adjusted data to markdown
print(dat.head(5).filter(['seen_any','star_wars_fans','seen__i__the_phantom_menace
']).to_markdown())

# %%
# 2 - creates two seperate data frames for people that have seen at least one film
and those who have not
dat_seen_yes = dat.query("seen_any == 'Yes'")
dat_yes = dat_seen_yes.query("star_wars_fans.notna()")
dat_no = dat.query("seen_any == 'No'")
dat_yes

# %%
# 3 - creates a one-hot encoded dataframe from dat_yes
dat_yes_one_hot = pd.get_dummies(dat_yes)

# %%
# 3 - validates the data by showing the actual percent of people that watched each
movie,
# also renames the columns in the data to be shorter in the visuals
filter_visual_validate_data = dat_yes.filter(like = 'seen__', axis = 1)
visual_validate_data =
pd.melt(filter_visual_validate_data).groupby('value').count().reset_index()
visual_validate_data = visual_validate_data.assign(
    validate_percent = lambda x: ((x.variable/dat_yes.shape[0])*100).round(2)
)
visual_validate_data['value'].replace({"Star Wars: Episode I  The Phantom Menace":
"(1) The Phantom Menace",
"Star Wars: Episode II  Attack of the Clones": "(2) Attack of the Clones",
"Star Wars: Episode III  Revenge of the Sith": "(3) Revenge of the Sith",
"Star Wars: Episode IV  A New Hope": "(4) A New Hope",
"Star Wars: Episode V The Empire Strikes Back": "(5) The Empire Strikes Back",
"Star Wars: Episode VI Return of the Jedi": "(6) Return of the
Jedi"},inplace=True)
visual_validate_data = visual_validate_data.sort_values('value', ascending=False)
visual_validate_data

# %%
# 3 - creates first visual showing the star wars movies that have been seen
validate_data_visual_1 = visual_validate_data.plot(x='value',
y='validate_percent', kind='barh')
plt.gca().get_legend().remove()
plt.xlabel("% Respondents that have seen each film")
```

```
plt.ylabel("Movie")
plt.title("Which 'Star Wars' Movies Have You Seen?")
plt.savefig('star_wars_seen_visual.jpg',bbox_inches='tight')

# %%
# 3 - creates second visual showing the best movie for people who have watched all
6 movies
best_movie = {'Movie': ['Return of the Jedi','The Empire Strikes Back','A New
Hope',
            'Revenge of the Sith','Attack of the Clones','The Phantom Menace'],
        '% Rating as "Best Movie" among respondents': [10,4,6,27,36,17]
        }
best_movie_data = pd.DataFrame(best_movie, columns = ['Movie', '% Rating as "Best
Movie" among respondents'])
validate_data_visual_2 = best_movie_data.plot(x='Movie', kind='barh')
plt.gca().get_legend().remove()
plt.xlabel('% Rating as "Best Movie" among respondents')
plt.ylabel('Movie')
plt.title("What's the Best 'Star Wars' Movie?")
plt.savefig('star_wars_favorite_movie_visual.jpg',bbox_inches='tight')

# %%
# 3 - shows how many people have watched at least one movie
at_least_one_film = (len(dat_yes)/len(dat))*100
# %%
# 3 - shows how many people have watched star wars seperated by gender
dat_yes.groupby("gender").size() / dat.groupby("gender").size()

# %%
# 4 - a,b,c,d - prints old data to markdown
print(dat.head(5).filter(['age','household_income','education']).to_markdown())

# %%
# 4 - a,b,c,d - converts age, household_income, and education to to numbers
dat_numeric = pd.concat([
   (dat.age
      .str.split("-", expand = True)
      .rename(columns = {0:'age_min', 1:'age_max'})
      .age_min
      .str.replace(">","")
      .astype('float')),
   (dat.household_income
      .str.split("-", expand = True)
      .rename(columns = {0: 'income_min', 1: 'income_max'})
      .income_max
      .str.replace("\$|,|\+", "")
      .astype('float')),
   (dat.education
      .replace({
      'Less than high school degree':'9',
      'High school degree':'12',
      'Some college or Associate degree':'14',
      'Bachelor degree':'16',
      'Graduate degree':'19'})
```

```
    .astype('float'))],
   axis = 1
)

# %%
# 4 - a,b,c,d - prints new data to markdown
print(dat_numeric.head(5).to_markdown())

# %%
# 4 - e - converts all the remainging data to one hot encoded
filter_dat_one_hot_1 = dat.drop(['age', 'household_income','education'], axis=1)
dat_ml = pd.get_dummies(filter_dat_one_hot_1)
dat_ml['age'] = dat_numeric['age_min']
dat_ml['household_income'] = dat_numeric['income_max']
dat_ml['education'] = dat_numeric['education']

# %%
# 4 - d - creates new column that shows wether or not household_income is more
than 50000
dat_ml['above_50000'] = np.where(dat_ml['household_income'] > 50000, 1, 0)

# %%
# 5 - filters data to only relevant data for machine learning model
dat_ml_1 =
dat_ml.filter(['rank__i__the_phantom_menace','rank__ii__attack_of_the_clones','ran
k__iii__revenge_of_the_sith',
'rank__iv__a_new_hope','rank__v_the_empire_strikes_back','rank__vi_return_of_the_j
edi','age','education','household_income',
'above_50000'])
dat_ml_h_subset = dat_ml_1.sample(500)

# %%
# 5 - creates pairplot for correlation
sns.pairplot(dat_ml_h_subset, hue = 'above_50000')
dat_ml_corr = dat_ml_h_subset.drop(columns = 'above_50000').corr()

# %%
# 5 - saves heatmap for correlation
sns.heatmap(dat_ml_corr).figure.savefig('dat_ml_correlation.jpg')

# %%
# 5 - converts data to floats
dat_ml_1 = dat_ml_1.astype('float16')
dat_ml_1 = dat_ml_1.dropna()

# %%
# 5 - creates graph of correlation of education and income
(alt.Chart(dat_ml_1, title = 'Education level to Income Comparison')
    .encode(
        x = alt.X('household_income', scale = alt.Scale(zero = False)),
        y = alt.Y('education', scale = alt.Scale(zero = False)),
        color = alt.Color('above_50000:O' ,scale = alt.Scale(scheme='redblue')))
        .mark_circle()).save('income_to_eduction_correlation.png')
```

```
# %%
# 5 - Creates machine learning model
X_pred = dat_ml_1.drop(['household_income', 'above_50000'], axis = 1)
y_pred = dat_ml_1.above_50000
X_train, X_test, y_train, y_test = train_test_split(
    X_pred,
    y_pred,
    test_size = .34,
    random_state = 76)

# %%
# 5 - creates confusion matrix
clf = GradientBoostingClassifier()
clf = clf.fit(X_train, y_train)
predict_p = clf.predict(X_test)
model_confusion_matrix_results = pd.DataFrame(metrics.confusion_matrix(y_test,
predict_p))
print(model_confusion_matrix_results.to_markdown())
metrics.plot_confusion_matrix(clf, X_test, y_test)

# %%
# 5 - shows model results
model_results = metrics.classification_report(y_test, predict_p)
print(model_results)
```